Micronix Operating System

Version 1.61

MORROW

IMPORTANT WARRANTY INFORMATION

LIMITED WARRANTY

Morrow, Inc. warrants its products to be free from defects in workmanship and materials for the periods indicated below. This warranty is limited to the repair and replacement of parts only.

This warranty is void if, in the sole opinion of Morrow Inc., the product has been subject to abuse or misuse, or has been interconnected to other manufacturer's equipment for which compatibility has not been established in writing.

Circuit boards - Parts, including the printed circuit board, purchased as factory assemblies, are warranted for a period of ninety (90) days from the original invoice/purchase date.

Electro-mechanical peripherals - Peripheral equipment such as floppy or hard disk drives, etc., not manufactured by Morrow, Inc., are included in the limited warranty period of 90 days from the original invoice date when sold as part of a Morrow system.

Exception - Expendable items such as printer ribbons, software media, and printwheels are not covered by any warranty.

Software/Firmware - Morrow, Inc. makes no representations or warranties whatsoever with respect to software or firmware associated with its products and specifically disclaims any implied or expressed warranty of fitness for any particular purpose or compatibility with any hardware, operating system, or software/firmware. Morrow, Inc. reserves the right to alter or update any program, publication or manual without obligation to notify any person of such changes.

LIMITATION OF LIABILITY: THE FOREGOING WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL MORROW, INC. BE LIABLE FOR CONSEQUENTIAL DAMAGES EVEN IF MORROW, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

WARRANTY RETURN PROCEDURE

Should a buyer experience a defect in either workmanship or materials during the warranty period, any Morrow Authorized Service Center will replace or repair the product at its expense only if the product is promptly returned to the dealer or Service Center with dated proof of purchase.

Should factory repair be necessary, the Service Center shall contact Morrow Customer Service for a Return Materials Authorization (RMA) number.

DISCLAIMER

No representations or warranties, express or implied, are made with respect to the contents hereof, including, but not limited to, the implied warranty of merchantability or fitness for a particular purpose. Further, Morrow, Inc. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation to notify any person of such revision.

Morrow
600 McCormick St.
San Leandro, CA 94577

TRADEMARKS USED IN THIS BINDER

UNIX is a product of Bell Telephone Laboratories.

Micronix, Decision, MicroDecision, and Correct-It are products of Morrow Inc.

CP/M 2.2 and CP/M 86 are products of Digital Research, Inc.

IBM and PC-DOS are trademarks of International Business Machines, Inc.

WordStar is a product of MicroPro Corporation.

LogiCalc is a product of Software Products, International.

BASIC-80 (MBASIC) is a product of Microsoft, Inc.

BaZic is a product of Micro-Mike's, Inc.

Personal Pearl is a product of Relational Systems, Inc.

XEROX 820 is a product of Xerox Corporation.

Osbourne is a trademark of Osbourne Computers, Inc.

FOREWORD TO

THE MICRONIX MANUALS

If you hate technical manuals as a matter of principle, we can't blame you. This one, at first glance, looks very thick, it only has a handful of pictures, and it's not even typeset.

Well, think of it as a beat up old Buick. It may not be pretty, but it's comfortable and reliable.

Writing a set of manuals that covers an operating system is a difficult task. When the operating system is one that emulates Bell Lab's UNIX, the task becomes almost overwhelming. The standard Bell Lab's documentation is a 6" thick stack of technicalese, daunting to even the most hardy of individuals. Their documentation was written over a period of ten years, by many different people, in a variety of styles.

Our documentation has to be somewhat different. Instead of tens of programmers contributing, Gary Fitts wrote the Micronix operating system and Len Edmondson wrote most of the programs. They wrote many of the entries in the Reference Manual, and provided the technical editing necessary for accuracy in the User's Guide.

The User's Guide to Micronix was written in an atmosphere of friendly chaos by John VanderWood and myself. Part of the chaos revolved around a change of attitude that made it possible to rewrite the old manual in a much more amiable style. That made us happy. But, naturally, this project was scheduled to be finished before it even began. Add to this the simultaneous improvements to the existing software by Gary and Len, and the feeling we had was one of running a race in shifting sand. The goal, a complete and easy to understand User's Guide, was like a mirage shimmering in the distance.

Result: the majority of the User's Guide is pretty good, with the possible exception of Tutorials, which we haven't had time even to look at yet. The Reference Manual still needs some work; at least it's as accurate as human imperfection allows.

Most of this manual was written in the stimulating and distracting environment of Morrow's engineering department. We received a lot of support from Bob Groppo, the S-100 project

engineer, and further assistance and distraction from John Zalabak, Dave Block, Don Mowry, Howard Fullmer and Ken Toland. Customer service got into the act, mainly in the person of Norm Tilbury, who read and corrected drafts. Dana Tilbury and Lawrence Curtis, who together have installed (at the "factory"), more Micronix systems than anybody in the world, also read and commented on drafts of the User's Guide.

We hope that this edition of the Micronix Manuals enlightens more than frustrates. There should be a great big "Under Construction, Hard Heads Required" sign at the beginning of this manual. We've tried all of the examples provided, and can only pray that the distribution software that you're using is the same as what we work with. If you find parts that confuse you, discover better examples, or can't find something you need to know about, write us a note, care of Documentation, at Morrow.

Take care, and much good fortune,

Rik Farrow, June 30, 1983

Micronix Operating System
user's manual

MORROW

ORIENTATION

TABLE OF CONTENTS

1. USING THE MANUALS

You received quite a mass of paper with your Micronix system. Let's see if we can sort it out a little, eliminating what's unnecessary at this point.

All of Micronix documentation can be divided into three parts:

The User Manual

The Micronix Reference Manual

The Technical Reference Manuals

1.1. Confusion Relief

Before launching into a discussion of what information can be found where, we should clear up a couple of potential sources of confusion.

Confusion Factor 1: The User Manual and Micronix Reference Manual are in the same binder, with no obvious separation between them. This is pretty easy to get used to. The first set of tabs, **orientation, installation, maintenance & administration,** and **tutorials,** comprise the User's Guide. Everything else, from **Programs** on back, is the Micronix Reference Manual.

Confusion Factor 2: There are all these other reference manuals in addition to the Micronix Reference Manual. Well, these all refer to the individual hardware components that make up the Decision computer. And unless you're one of that strange breed of computer freak who can't keep his mitts out of the circuitry, you can just put these Technical Reference Manuals away. From now on, when we say "Reference Manual", we mean the Micronix Reference Manual.

Confusion Factor 3: Finally, you have the manuals for the software that is included with Micronix (a word processor, for example). Since this isn't really part of Micronix as such, we'll overlook it here. The same is true of the C and Pascal manuals, if you purchased those separately-priced compilers.

Now then. The next sections give an overview of the contents behind each of the tabs in the User's Guide and Reference Manual.

1.2. The User's Guide

This is the manual that you will use the most while you are getting to know your way around Micronix. It tells you how to install the hardware and software, how to run it, and how to keep it running.

1.2.1. Orientation

The section you are now reading covers the documentation and attempts to convey the concepts underlying the Micronix file system.

1.2.2. Installation and Operation

This section runs you through hooking up all of the cables, setting up your terminals and printer, checking out the preinstalled software, and reinstalling it if it doesn't check out properly. It also has instructions for an introductory session with Micronix, including how to turn the system on and off properly. This section ends with a discussion on installing CP/M software.

1.2.3. Maintenance and Administration

Here you will learn about all of the exciting and wonderful tasks that await you as system manager. We'll tell you how to check the file systems, maintain free disk space, customize the environment, copy diskettes, provide system security, back up files, add new user accounts, and respond to error messages.

1.2.4. Tutorials

The last section of the User's Guide provides elementary training in common Micronix operations that are not unique to the system manager. These include changing passwords, modifying the "search path", mounting disks, using the CP/M emulator **upm**, and taking advantage of some of the trickier features of Micronix.

1.3. The Reference Manual

This manual makes up most of the bulk of the Micronix binder. Once you get used to Micronix, you will probably find the Reference sections to be the most frequently accessed, when you have mental lapses and the like. All of this information is also available online, by way of the "help" command. Just pick the name of a program, file, system call, device, or subroutine, enter the command **help so-and-so**, and lo and behold! there is the information you wanted, right there on your screen. For example, to learn how to use the concatenation program, enter **help cat**. (No, this won't tell you how to get your kitty out of a tree.)

These sections of the Reference Manual are arranged by topic, in alphabetical order. Leaf through them to get an idea of what's in there and how to find it. Go ahead, do that right now.

1.3.1. Programs

This will probably soon become the most dog-eared section of the manual, since you will refer to it constantly to check on the usage and syntax of Micronix commands. It also contains valuable tips on practical applications of the programs. While many of the commands are discussed in the context of installation or administration of Micronix, this section is the only central depository of Micronix's capabilities. So it's a good idea to just browse around in it while you're waiting for a disk to copy or a file to print. You'll probably spot something that you've been wanting but didn't know was in there.

1.3.2. System Calls

We're getting into some fairly advanced stuff here, aimed mostly at programmers. System calls and subroutines are the building blocks upon which Micronix is constructed. This section is intended for those of you who wish to make (Version 6 UNIX) system calls to Micronix from C or assembly language programs.

1.3.3. Subroutines

These are frequently-used mini-programs that programmers can embed into their own creations with a simple C invocation.

1.3.4. Devices

This section gives terse descriptions of the interface between Micronix and the hardware components of the Decision computer. Some of it is very technical, but other parts are quite accessible to the average user and helpful when modifying the hardware configuration (see especially **cables**, **printers**, and **ports**). Those wishing to access memory or I/O mapped devices of their own will be interested in the mem and io devices.

1.3.5. Files

These are tools for cutting and forming the surfaces of objects (just making sure you're still with us).

Again, this is a pretty technical discussion of file formats used internally by Micronix. Non-programmers will still want to be familiar with the layout of the **banner**, **motd**, **passwd**, **rc**, **signon**, and **ttys** files.

2. THINGS YOU ABSOLUTELY MUST KNOW

I. Whenever you are instructed to enter some command in the pages that follow, always follow it with a carriage return.

II. When you see a screenful of data reproduced in this manual, the part that is printed in boldface is the part you type. The rest is sent out by the computer.

III. If you get stuck in some procedure and you can't figure out what to do to escape, the **delete** key, (sometimes labeled RUB instead) is your best bet. Sometimes it might take a moment for Micronix to respond to the delete, so if you're in a panic, go ahead and press delete a couple more dozen times while you wait. Delete also interrupts the output of Micronix programs like **type** when you decide that you've seen enough.

When you're working inside some application program that is not part of Micronix as such (e.g. Personal Pearl), there will be a different keystroke that performs the abort. Look for that in the relevant documentation (Good luck).

UNDER NO CIRCUMSTANCES should you abort a program by turning off the power or by resetting the system. Check out the section called "Desperate Measures" in the **Maintenance and Administration** division for additional procedures.

IV. When you tell Micronix to list something out on your screen with the **type**, **form**, **cat**, **ls**, **far**, and other commands, and the info is whizzing up the screen faster than even Evelyn Wood could handle it, press the **ESCape** key or **control** and **s** keys simultaneously to freeze the listing. When you're ready to resume, press any other key. You can use escape or "control-s" over and over in the same operation to stop-start output.

3. SYSTEM OVERVIEW

What is Micronix, anyway?

Every computer manual has its obligatory, patronizing, and futile attempt to explain just what an operating system is. Let's hope we do a little better.

Think of the computer hardware as the North American continent, sans the blessings of civilization. Now think of your marvelous computer program (like WordStar) as a sleek, gleaming Mercedes waiting to launch you from equestrian barbarism into the rolling pushbutton thunder of the latter twentieth century.

Now there's something missing here, right? Your gullwing isn't going to get too far without paved roads and bridges. So you can think of an operating system as that system of highways and byways that brings the vistas of automation within your reach. As for Micronix, well, it's the Interstate of microcomputer operating systems.

Just as a road must follow the contours of the terrain (within limits), an operating system must conform itself to the ideosyncrasies of the hardware on which it is running. On the other hand, you would expect your car to perform equally well on a variety of topographies without having to modify it. Thus operating systems enable an application program to run on a range of hardware without having to monkey around with the code.

As you travel down that highway, you stop and exchange information, drop off reports, make phone calls and connect with other programs. This is known as input/output, or i/o. The operating system provides the facilities.

Now that we've belabored this tenuous metaphor sufficiently, we can sum up by saying that the operating system creates a network of functions that give the computer its personality. Without any operating system, a computer makes a fine bookend. With one o.s. versus another, a computer will be able to handle a different set of chores, at different levels of proficiency and efficiency.

Micronix is your o.s. It is rather elegant, as microcomputer operating systems go. The following sections describe some of the features of Micronix, but as you will see, reading this is no substitute for getting in there and putting it through its paces.

3.1. Micronix Software Features

> **Mi-cro-nix** \mi'-cro-nix\ n. an interactive timesharing operating system designed to run on Morrow's Decision computer. It is compatible with Bell lab's UNIX version 6, containing many of the same features, including:

* **a hierarchical file system** - Unlike the "flat" file systems (such as the one used by CP/M), a hierarchical file arrangement allows you to organize files in a logical and convenient manner.

* **multi-tasking (background processes) for each user** - This feature makes it possible for you to start some time-consuming program and at the same terminal, go on to something else while the first "process" runs merrily along.

* **compatible file, device and interprocess** I/O - Whoa! Jargon city! This means that you can send the output of a program to a terminal, to a printer, to a disk file, or to the input of another program without having to reformat it. Likewise, you can get the input for a program from a terminal, a disk file, or from the output of another program.

* **mountable disks** - You can extend the reaches of your file system to other hard disks or to floppy disks with a simple mount command. Without this feature, you would have to copy the files on these devices over into the file system in order to access them for reading or writing. That would be a clunky, stagnant, and altogether distasteful state of affairs.

* **a choice of shells (command line interpreters)** - You can run Micronix in several "modes", that is, if you are in the Micronix shell, the computer will recognize Micronix commands; if you switch to the upm shell, it will recognize CP/M commands. Finally, you can configure a user so that he runs only one application program, say, Logicalc. This user then can be said to possess a Logicalc shell, since the shell will recognize only Logicalc commands.

* **nearly 100 software tools for program development, document preparation, file handling and inter-system communication** (including an emulator that runs CP/M object code unchanged)

* **all UNIX 6 system calls (except "ptrace") and a complete standard I/O library.** Existing UNIX programs can be compiled generally unchanged under Micronix.

3.1.1. Memory Management

The Micronix "kernel" (the part that stays in memory instead of being called in now and then) takes up 64K bytes. Each "process" can occupy from 16 to 68K. A process, in normal terms, is a program that you are running from your terminal. More strictly, a process can exist in background mode and even exist independent of any terminal. It is a block of memory occupied by a program and its data. When the total demand for memory exceeds the amount available, Micronix swaps selected processes out to disk storage in order to make room.

3.1.2. Languages

Most of the code in Micronix is written in the C language. This language is both highly structured and fairly unrestrictive, making it efficient and easy to use. Programmers will find that C language is good for a wide variety of programs.

Other languages supported by Micronix include Pascal, RATFOR, BASIC and Intel assembly language. More specific details concerning software with Micronix are not contained in this manual, since they are separately packaged products.

3.2. The Micronix Shell

The shell or "command line interpreter" is your channel of communication with Micronix. It displays a prompt (# or %) telling you to give it something to do. Then it reads what you type, and checks to make sure you typed it JUST RIGHT. Of course, it only checks for correct syntax. If you accidentally tell it to erase the book you spent the last six months penning, it will do so without asking you if it's okay. (We'll preach your ears off about the importance of backup files later.)

The shell has some fancier functions too. It has a set of built-in commands, for moving around in the file system and displaying its contents. You can use "wildcards" in filenames to make life easier for yourself. The shell handles interpreting wildcard symbols, as well as setting up background processes, piped data flow, and i/o redirection. All of these handy features are discussed in more detail in the **Tutorials** division of the User's Guide.

3.3. The File System

One of the most beneficial features of Micronix is its file system. If you are new to hierarchical file structures, you may have to acquire a taste for it. Once you do, though, those flat directories will never be good enough again. We're going to make a heroic effort here to explain how the Micronix file system works, but for really getting the hang of it you should take a tour of the file system as described under "First Time Use of Micronix" in the **Installation** division of the User's Guide.

Okay, here we go.

Traditionally, a file system like the one used in Micronix is compared to an inverted tree. There is a root directory at the top, with other files and directories descending, ramifying, multiplying amoeba-like into the infinite depths below.

Alternatively, you could envision it as a starfish, or a wagon wheel, a snowflake (no two file systems are exactly the same), or perhaps a mine shaft.

Here are a few things that the file system is NOT like:

a doughnut
a tricycle
a jazz band
an armadillo

We're going to try a different paradigm.

But first, we need to be clear on what's a file and what's a directory. Let's assume you know enough to have at least a foggy idea of what a file is: it could be some text you generated, some numbers in a data base, a program's coding; it doesn't matter. The point is that it's NOT a directory. A directory is a list of files and where to find them. Now we get to the hierarchical part - a directory can also contain other directories, cannily referred to as **subdirectories**.

Imagine you have awakened in a spacious circular room, chrome-and-glass motif, ferns atop filing cabinets, earth tone sculptured carpet - the office of the future. Woven into the carpet is the cryptic symbol " / ".

Surrounding you is a panorama of doors and filing cabinets. Where is the coffee machine, you wonder. This is the office of the future, remember, and all of the coffee bushes in the world have been laid waste in the Great Java Blight. Don't you know anything? Here, drink this Petro-delite and listen.

You are in the **root directory**, a clean and wholesome place. The doors you see before you are entrances to other directories, like **bin** and **usr**. The filing cabinets are the files themselves,

like **micronix** and **finstall**. The files are the important things. Directories are just pathways that you go through to get to the rest of the filing cabinets.

Egad! You have just been transported deep into the file system. Another room, but with over a hundred filing cabinets and only one door, mysteriously labelled "..". Emblazoned on the carpet is the ominous message " **/usr/man/man1** "

Kirk: Kirk to Enterprise! Kirk to Enterprise! Come in, Mister Scott!

Scotty: (brogue-ish) Scott here, Cap'n.

Kirk: What in the blazes is going on, Scotty? Are you fiddling with the transporter's controls or what?

Scotty: Beg pardon, sir. I was just testin' the transporter's new operatin' system with this "cd /usr/man/man1" command.

Kirk: (aside) Explanation, Mister Spock?

Spock: It would seem that the expression /usr/man/man1 somehow represents this location, when one considers the unlikely coincidence of us materializing here (thanks to Mister Scott's tinkering) and the fact that those very characters appear there on the floor. Perhaps "cd" is some sort of directive, meaning "coordinate definition" or "consign to destination".

McCoy: Or "capture and detain", or "chafe and decapitate"!

(Menacing theme music crescendos, cut to commercial)

While the commercials are on, we will put you at ease by assuring you that the good doctor's anxieties are amiss. "cd" stands for **change directory**, which means pass through a door, or a series of doors, depending on the "argument" that follows cd.

Therefore, the Enterprise brass have in effect passed from the root directory, through the **usr** door, on through the **man** door, and finally into the **man1** room. The complete path of this journey is "/usr/man/man1", which is the argument. By the way, this room contains the online documentation for section 1 of the Reference Manual, hence "man1". But the show's back on.

Spock: Captain, there is a door to your left with two dots on it.

Kirk: So I see. No doubt a secret message of some sort. Now, if I can remember back to my days in the signal corps, two dots is Morse code for "I". Hmmm. But "I" is English for the Latin "ego" which sounds like "y-go" which is sort of French for "Go there"!

Spock: (muttering) Which I was about to suggest.

Kirk: Scotty, give us a "cd ..".

Scotty: But sir, why doon't ya just walk o'er there?

Kirk: Do what I tell you, Mister Scott, or I'll have you keelhauled! Besides, it's such a MOVING experience. (winks at McCoy).

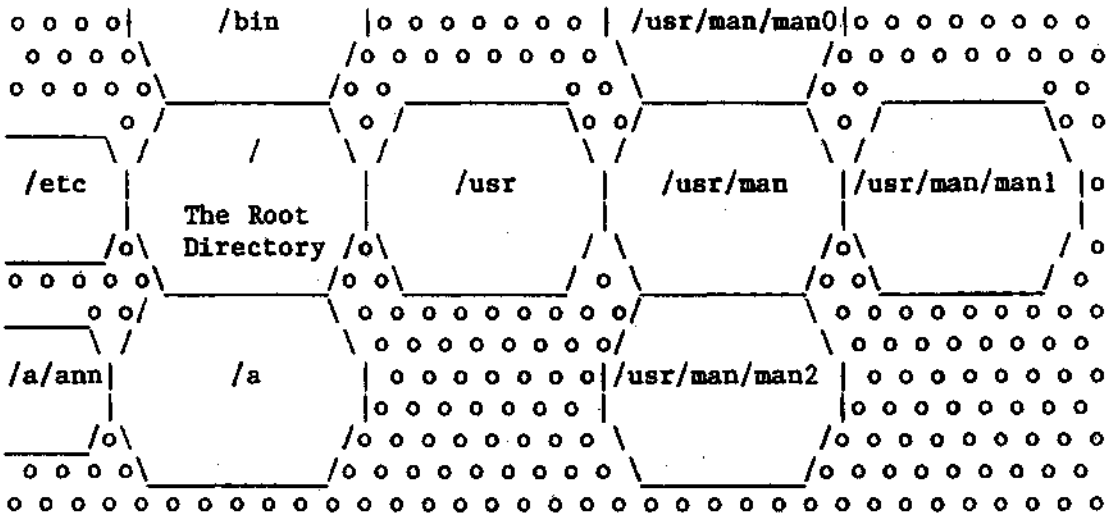
(whirring and whining of transporter - protagonists appear on the other side of the ".." door. No filing cabinets this time; instead, there are seven doors labeled "man0", "man1", and so on up to "man5", plus another strange ".." door. This floor's carpet is decorated with the insignia "/usr/man".)

Spock: Fascinating. Our previous location was /usr/man/man1. After passing through the double-dot door, our present location is /usr/man. Over there is a door marked "man1". I posit that these are one and the same door, viewed from opposite sides. Furthermore, this second double-dot door should lead us into a chamber known as "/usr", wherein will be a "man" door, leading back into this very place. It is quite conceivable that in the /usr room we would discover yet another dot-dot door, being an entrance to the "/" room, where we began.

McCoy: Why you contemptible conglomeration of collagen and chloroplasts! What the devil are you talking about?

(Spock raises eyebrow, exchanges knowing look with smiling Kirk - up on happy music, freeze, superimpose credits)

More than likely you're in a similar state of befuddlement at this point. The foregoing episode serves mainly to give you a feel for the labyrinthine nature of the file system, a glimpse from the inside as opposed to the external view, which we will present next.



This is a partial overhead view of the file system. The thing to notice is the way that each directory cell has a "pathname" that is determined by its position relative to the root directory. The farther down into the file system you go, the longer the pathname becomes.

Whenever you are using Micronix, you will be "in" some directory. You will have access to the files in that directory (unless they are restricted) by simply typing the file's name after some command; for example, if you are in **/usr/man/man1**, you could look at file **cptree.1**, which is in that directory, by entering **type cptree.1**.

If you were in some other directory, though, you would have to include the whole pathname, as in **type /usr/man/man1/cptree.1**

A major potential confusion factor is that when you look at the contents of a directory using the **dir** command, you'll get a list of filenames and subdirectory names with no apparent distinction between the two types. Once you know your way around, this isn't a problem. But early on you'll be making the harmless mistake of trying to change directory (**cd**) to a path ending at a file, which won't work, and attempting file manipulations on some directory, which won't work either.

There are two ways to get your bearings within the file system:

1. Use the command **pwd**, meaning pathway to directory. Micronix will respond with your current pathname, for example, /a/ann/letters.

2. The **dir** command displays the files and subdirectories that are in your current directory. Hopefully something will look familiar enough to clue you in as to where you are.

Now for a little mental exercise to help assimilate what's gone before. . .

You are in the root directory. You get there either by logging in as the user named "root" or by typing **cd /**.

You want to get to the home directory of user "bob", which means the directory that Bob automatically lands in when he logs in. (The setting up of "home directories" is discussed in **Maintenance and Administration.**)

The pathname of Bob's home directory is /a/fulltime/bob.

There are four ways of getting there, but only one does it in a single step. See if you can come up with the right one. Don't peek. Cover the answers below with your hand. (Hint: the answer is painfully obvious.)

Answer #1:
```
cd a
cd fulltime
cd bob
```

Answer #2:
```
cd a/fulltime
cd bob
```

Answer #3:
```
cd a
cd fulltime/bob
```

Answer #4:
```
cd a/fulltime/bob
```

If you produced any of the first three answers, you're catching on, but your sense of efficiency is a little suspect.

You may be wondering why Answer #4 doesn't have a slash in front of the "a" in the pathname, since the pathname was given as /a/fulltime/bob. This is because you were already in the root directory (symbolized by /, giving slash a double meaning). That means subdirectory a was in your current directory, so anything that comes before it in the pathname is superfluous.

Now suppose you are in directory /a/fulltime/oglethorpe, and you want to get over to Bob's home directory. One way to accomplish this is by entering **cd /a/fulltime/bob**. The slash has to be up front there since you weren't in the root directory at the time.

The alternative route demonstrates backing out of directories a step at a time, symbolized by the dot-dot doors in our earlier fable:

```
cd ..      to back up from /a/fulltime/oglethorpe to /a/fulltime
cd bob     to advance from /a/fulltime to /a/fulltime/bob
```

And finally, these two commands may be condensed to **cd ../bob**.

We're about finished with this file system discussion. It's doubtful that you feel inspired and enlightened, because as we said earlier, reading about it is no substitute for tapping on that keyboard.

3.4. Directory Terminology

Just to make sure you're clear on some of the terminology you'll be running into elsewhere in the manual -

"Root Directory" is the center of the file system, if you like the starfish analogy; if you prefer the inverted tree, it is the top. For the office-building scenario, consider it the lobby. All pathnames to files in the system begin at the root directory. The root directory contains a few files (unless you put more in it) but mostly it contains subdirectories. The root directory is symbolized by a slash, so the command **cd /** will put you there.

"Home Directory" is the position in the file system where a user is placed when he logs in. This choice is under the control of the system manager. The home directory of the user "root" (usually the login name for the system manager) is the root directory. Other users would typically have home directories like **/a/henry**. No matter where a user happens to be in the file system, he can always return to his home directory with the command **cd**, with no argument following it.

"Current Directory" is whatever directory you happen to be in at present. It could be the root directory, your home directory, or some other subdirectory. The pathname of your current directory can be subtracted from the pathname of some file farther down in the file system if you want to do something with that file. For example, if you are presently at /a/sylvia and you want to print the file /a/sylvia/letters/mom, you could enter **lpr letters/mom**. You can find out the pathname of your current directory by typing **pwd**. To see the contents of your current directory, enter **dir**.

The ".." symbol means "one step back in the file system". You can use more than one of them in a string. For example, to move from /f/john/micronix/chapter1 back to /f, you could enter **cd ../../..** Starting at the same place, if you wanted to get to /f/john/cpm/intro, you can mix ..'s with a normal path in **cd ../../cpm/intro**. Remember that cd commands have to end up at a directory name, not a file name; therefore "intro" and "chapter1" above are directories.

3.5. Where Files Come From

Your Micronix system comes with a basic file system made up of the Micronix programs and the online documentation. You will be adding new directories with the **mkdir** (make directory) command, and adding files by copying things into the system (such as when you install your word and data processing software), and by running that software (it creates files for its own use). Other ways of generating new files are by using the **edit** command on a file that doesn't yet exist, by sending the output of some operation to a file that isn't there yet, and by duplicating existing files with the **cp** and **cptree** programs.

No matter how extensive or intricate your file system eventually becomes, it will still always lead back to the original root directory.

A couple of tips on setting up your own file space:

1. As a regular user (instead of system manager), you will probably have a home directory of /a/yourname. This is created by the **accounts** program. It will be an empty directory at that point. When you use **mkdir directoryname** to make a subdirectory, capitalize the name or at least its first letter to distinguish it as a directory instead of a file.

2. Micronix has certain restrictions on the file names that you can come up with. File (and directory) names can be any combination of printing characters up to 14 characters long, with one exception. The slash character, /, has a special meaning for Micronix (it separates directory names), so it can't appear in file or directory names.

   A different set of restrictions on file names is imposed when using CP/M. Micronix files for CP/M must be all lower case letters or numbers, with a maximum length of 8 characters, possibly followed by a period and three more characters.

4. WHERE TO GO FROM HERE

Well, if you haven't installed your system yet, by golly, you can't do too much else until you get that overwith. The **INSTALLATION** division covers this subject pretty well.

Then you should take the tour of the file system as described toward the end of the **INSTALLATION** division.

Next, refer to **MAINTENANCE AND ADMINISTRATION** for instructions on setting up user accounts. Flip back to PROGRAMS and try running a few of them, just to see what happens. Remember that the **DELete** or **RUBout** key will normally get you out of confusing situations. **CTRL** and **D** simultaneously can be helpful too, since it logs you out. Now dig in, and enjoy yourself.

Micronix Operating System

user's manual

installation & operation

MORROW

INSTALLING MICRONIX

TABLE OF CONTENTS

Appendices

INSTALLING YOUR MICRONIX SYSTEM

Congratulations on getting your new system. This section of the Micronix manual explains how to connect your terminal(s) and printer(s) and how to get the software running. Rather than trying to do everything at once, then trying to figure what went wrong, we will proceed a single step at a time. This may seem a little slow if you are anxious, but it is merely honoring Murphy's law:

If anything can go wrong, it will.

Your Micronix system was connected with a terminal and tested for 12 hours before it was packed at Morrow. Twelve hours may not seem like much, but most problems from faulty hardware show up within the first half hour of use. Because of the test, we know that your Micronix system worked great for us; it should also work for you. Just follow the instructions, please. We will use checkpoints at every step so that you can be sure that you have succeeded in understanding and following instructions.

We'll also do our best to explain briefly and simply what you are doing to your system as you install the software. This should provide you with a better understanding of how Micronix works.

1. INSTALLATION OVERVIEW

Before we get started, this section provides a brief description of the procedures involved in installing Micronix on a Decision. This section may be all that you need if you have previously installed a Micronix system. For those of you first-timers, the installation steps are explained in greater detail in the sections that follow. Here goes...

1. Unpack and check for external damage; remove the inserts from the floppy drives;

2. Choose a suitable location for the computer and its peripherals;

3. Plug the Decision into a grounded socket; plug a terminal into a power socket on the Decision or into another grounded socket;

4. Turn on the power: are the fan and switch panel light on?

5. Set the terminal to 9600 baud, 2 stop bits and no parity;

6. Connect this terminal to the RS-232 connector located in the lower right-hand corner of the back of the Decision; there are (at least) three of these connectors: use the one nearest the right side (when viewed from the back.)

7. Insert the CP/M floppy disk, and boot up CP/M by pressing the RESET switch (or flicking the key to RESET)

8. Run the CP/M **CHECK** program to check the contents of the hard disk.

The **CHECK** program is an important milestone in this process, since its outcome determines whether you will have to repeat a time-consuming (though easy) procedure that was already done when your system was tested. If it is successful, you are ready to boot and play around with Micronix and can install whatever software packages you may have purchased. We will traverse these bridges upon arrival.

2. STEP BY STEP INSTALLATION PROCEDURE

2.1. STEP 1: UNPACK AND INSPECT

The instructions for unpacking your system are essentially the same as for unpacking a piece of stereo equipment: carefully remove the cabinet from the box and save all packing materials in case you ever want to ship the system again. Besides the cabinet containing the Decision, you should have also found:

- o two binders, one with these instructions,
- o a packet of diskettes,
- o warranty cards, and
- o a power cord.

The packet of diskettes contains a CP/M diskette for testing and "booting" your system, copies of the Micronix software that has already been placed onto the hard disk, and a set of application diskettes for word and data processing.

The warranty cards should be read, filled in, and mailed.

THIS IS IMPORTANT: There are cardboard inserts in your floppy drives to protect them during shipping. **You must remove them before applying power to the system.** Lift the door latch and slide them out. Leave the doors open. Easy enough?

The power cord is the only external part that is packed with the Decision. Since you are going to connect at least one terminal to the system, you also need an RS-232 cable with two male ends for each terminal you are connecting. Hopefully, your dealer was savvy enough to realize this and sold you one with each terminal. Or, if you are daring, and competent technically,

you can build your own cable with parts purchased at Radio Shack or an electronics store. The section on installing printers has more details on RS-232 cables. If you bought a Morrow terminal, a RS-232 cable will be in the box with the terminal.

Keep your carton and packing materials! Don't give them to the kids for playing camp-out.

If you do see blatant physical damage to the Decision, notify the carrier immediately for filling out a damage claim. Then contact Morrow's Customer Service Department (408-430-1970) and ask for a Return Authorization Number. They will not accept returned equipment without that number and it must be shipped in the original carton.

Those of you with Decisions that have the keyswitches on the front can ignore the next subsection and proceed to STEP 2.

2.1.1. Checking for Hidden Damage

This section can be skipped by owners of Decisions that have an ON/OFF keyswitch on the front panel; this should include just about all of you. However, if you do run into some intractable problem later in the installation, the following procedures may prove helpful, regardless of which cabinet your Decision uses.

By the time you have removed the Decision cabinet from the packing materials, you should have discovered any gross damage suffered during shipping. What we want you to do now is check for hidden damage. The hidden damage will probably involve loose, rather than broken, parts that you can reattach yourself. And by "hidden", we mean inside the Decision cabinet. Opening the cabinet is simple and not dangerous, so don't be afraid.

If you have already plugged in your Decision, please unplug it. There are two types of Decision cabinets: a molded plastic one and a metal one. To look inside the plastic cabinet, remove two screws in back, and slide the top of the Decision toward the back and lift it off. The cover to the metal cabinet is attached by four screws which you need to remove before you can slide off the cover. The cover slides off toward the front.

Just under the cover, on the left hand side, there is a piece of foam rubber. Please remove the foam. This foam was put into your Decision after testing to protect the pc boards during shipping and it will interfere with air cooling if you leave it in. (Non-keyswitch models only.)

When you have removed the foam, you will have revealed the printed circuit boards. Each board looks like a green or blue rectangle, about 5 by 10 inches in size, covered with black integrated circuits on one side. The boards should be neatly standing in a row, with white plastic guides holding the boards upright, and a black plastic connector fastening them along their lower edge.

By looking down between the pc boards (there is room for several more boards right in the middle) you will see another, much larger, green printed circuit board that extends from the front to the rear of the cabinet. This board is called the WUNDERBUS I/O, and holds 14 black connectors, each about 7 inches long. These connectors work identically to one another, that is, they provide the same data and signals to each printed circuit board that is plugged into them. To work properly, the lower edges of the smaller pc boards must be firmly attached to the black connectors. This connection, between the smaller printed circuit boards and the WUNDERBUS I/O, is often a problem when shipping computers.

Checkpoint:

We are going to check for two things: that the boards are held between the white plastic guides, and that all the boards are firmly connected.

1. To see whether each pc board is still held between the two white plastic guides, look along the left and right ends of the boards. The white guides begin 3/4" (2 cm.) below the top of the boards, and its easy to see whether each board is resting in the guides, unless you keep your computer under the table or in a dark room.

2. It's pretty hard to see the connection between the boards and the WUNDERBUS, but you can do two things. You can see that the top edge of each board is about 3/8" (.8 cm.) above the metal sides, and that the tops of all the boards appear to be about the same height. And you can press down on the tops of the boards to reseat them. To reseat boards, press gently with your thumbs on both ends of the top of a board. You can use up to about 10 lbs. (4 kilos) of force pressing down, and can alternate the force from your left thumb to your right to gently rock the board into place. Even if all the boards look seated, try this with at least one board.

If you find a board that is completely unseated, well, reseat it. It doesn't matter which slot on the WUNDERBUS you fit it into, and it will only go in one way. If you find a loose cable, it should be readily apparent where it connects. Chapter 11 of the D120 manual (the hardware manual for the Decision that is in the other binder) shows the proper connection for most cables.

If you can't figure out where to connect a cable, get on the horn to Morrow Customer Service. These situations are very unlikely, but we want you to be as self-reliant as possible in case they occur.

2.2 STEP 2: SELECT A LOCATION

The choice of the location for your Decision is up to you. However, there are three requirements that must be followed. Your Decision should be installed where it will remain level, stable and dry. The level and stable requirements refer mainly to the hard and floppy disk drives. These both contain moving parts that are designed to be operated on a level surface for maximum life expectancy. And, if you drop or jar a hard disk, the head alignment inside the sealed unit may change, making everything on the disk unreadable. Even worse, dropping an operating hard disk will probably force the heads into contact with the spinning disk, destroying many sectors. Treat your hard disk like a priceless Ming vase: keep it where it won't be dropped.

The definition of "dry" for electronic equipment is that the environment must be non-condensing. That is, the humidity should never be so high that water condenses on the metal parts of your computer. Obviously, rain, or spilled coffee, is too wet.

And, if it is really dry in your environment, you may have a problem with static. The static-charge that you can build up in your body walking across a rug might amount to thousands of volts, but very little current. What this means is that what feels like a little shock to you is like elctro-convulsive therapy for your computer. Computer-brain damage will be the likely result. Problems with static can be vanquished by purchasing a rug that has copper wires woven into it. Strange though it may seem, rugs with metal content are not too uncommon, and you may be able to buy a small one from a rug store. Or, you can get an anti-static rug from an office supply or computer retailer.

One final suggestion about location. Like a stereo system, all of the connections for your system are located on the back of the cabinet. If you can arrange the system so that you can stand behind it, at least while installing it, everything will be a lot easier. Once the physical installation is completed, you can move the system so that the back faces a wall. Remember to leave at least 4" (10 cm.) at the back and over the top of the cabinet for ventilation.

2.3 STEP 3: PLUG IN THE POWER CORDS

... but don't just jump right in and do that. Read this section first, and **be sure that the power switches to the Decision and its peripherals are off.** Some Decisions have large red or white rocker switches on the front or rear panels for controlling power. To confuse matters further, they can be mounted vertically or horizontally. Off is always DOWN for vertical switches and LEFT for horizontal ones (when facing the switch).

Whew. If your Decision has a keyswitch on the front panel, ignore the above and turn it fully COUNTERCLOCKWISE.

Your Decision comes configured from the factory for either 110 or 220 volt operation, as specified by the dealer who ordered it. Just to be absolutely sure that your available current matches what the computer needs, check the sticker on the rear of the Decision. The sticker covers the power socket and tells which voltage level the computer expects.

Your Decision can be plugged into any grounded outlet. Grounded outlets for 110 volts look something like this:

They will receive the three-pronged plug that comes with the Decision. (If you have a 220 volt system, you will also have a three pronged socket, but in a different shape.) If grounded receptacles are not available, use an adapter to plug the power cord into the wall, and plug accessories, such as your terminal and printer, into the outlets on the back of the Decision. When using a three pronged adapter, be sure to fasten the metal tab or green wire from the adapter to the screw that secures the socket cover to the wall. And **don't just break the ground lead off of the power plug.** This is a computer, please, not a sabre saw.

To make matters worse, not all three pronged sockets are actually grounded. In older houses and buildings, all the wiring was done with two wires - no ground wire! Later, three pronged sockets were added by someone and "grounded" to the metal box they were installed in. If you want to check your ground (the third hole, which is round, or the screw that you attach the adapter to), use a volt-ohmmeter and check the resistance between your alleged ground and a cold water pipe. The resistance should hover near zero.

If you need more outlets, get a power strip and plug everything into the power strip. Computer systems with their accessories are like stereos in that they work best if they are all using the same ground. If you are going to have terminals that are located a distance away from the Decision, use the full RS-232 specification cables which include a protective ground on pin 1.

Besides a shared ground, the Decision and its peripherals, (terminals, printers, etc.) require a steady power supply. The minimum suggested voltage for the Decision is 105 volts, and the maximum is 125 volts. For 220 volt systems, the tested voltage range is 208 to 265 volts. In the United States and most of Europe, the power companies are pretty reliable at delivering electricity in this range, so most of you won't

have trouble with this.

One problem that you need to try to avoid is voltage variations due to the startup of large electrical appliances like air conditioners, space heaters, and freezers. If an electric lamp connected to the outlet you want to use dims visibly when one of these appliances turns on, you will probably be plagued with erratic performance from your computer. The only solutions are running a dedicated power line from your main service panel to either the computer or the appliance, or getting an uninterruptable power supply.

2.4. STEP 4: TURN THE POWER ON, FIRST TIME

Now, ready for the big step? Look on the back of the Decision cabinet. The socket for power will be covered by the sticker with the voltage level on it. Remove the sticker. The socket is a hole with three prongs in it. Plug the female end of the power cord into this socket. The socket is designed so that there is only one way the plug will fit in, so don't worry about doing it wrong. Plug the other end of the power cord into the outlet you have selected. Now, apply power by either reversing the position of the rocker switch or by turning the keyswitch one click clockwise. The power is now ON to the Decision.

Checkpoint 1:

A pilot lamp should now be glowing to notify that power has been applied. In the keyswitch models, the panel next to the keyswitch illuminates; in other models the Reset button lights up. The fan inside of the the back of cabinet should also be on. If you have a 5 and 1/4 inch hard disk, known as a mini-wini or mini-winchester drive, the red indicator light on the drive will also be on. The two grounded power outlets on the rear of the Decision now have power applied as well. If everything checks out here, go on to the next section on connecting the first terminal.

Troubleshooting 1:

If neither the pilot light nor the fan came on, check:

1. The connection of the power cord to the back of the Decision. This socket is about 3/4 (2 cm.) deep, and the female end of the power cord must be fully inserted for a good connection.

2. The grounded outlet. Plug a desk lamp into the outlet to make sure that you have power.

3. The circuit breaker on the back of the Decision (keyswitch models only). It is of the pushbutton type seen on stereo receivers and speakers. Push it all the way in and release with a snap.

4. The power switch itself. Rock it back and forth (or flick the keyswitch) a couple of times.

If the fan comes on, but the pilot light doesn't come on, the Decision may well be operational with the exception of the lamp itself. In **NON-keyswitch models**, there is an all-too-frequent problem with a loose connection on the wires that connect the RESET button to the WUNDERBUS I/O. Unplug your Decision, take the top off and, with good lighting, look into the cabinet between the back of the cabinet and the rearmost pc board. Located just to the rear of a black connector on the WUNDERBUS are two pins with the word PRESET printed next to them. A small black (or brown) plastic connector with a black wire and a blue wire coming from it should be attached to the two pins, but is instead lying nearby. The black and blue wires are tied down near to the two pins, so that if the connector is loose it will be easy to find. It is a tight fit for large hands, but you can maneuver the connector over the pins by holding the connector between two fingers and slide it back on. The blue wire goes on the pin nearest the power switch, the black wire on the pin nearest the fan.

2.5. STEP 5: SETTING UP THE FIRST TERMINAL

This section covers attaching ONLY the first terminal. If you're going to have more, these procedures apply in principle, but there is further information under "Terminals" in the **MAINTENANCE AND ADMINISTRATION** division of this manual.

The first terminal connected to a Micronix system is called the console. On mainframe (large) computer systems, the console is used by the system operator. Micronix's console is used both by the system administrator and ordinary users. When you first bring up your Micronix system, the console is also the ONLY terminal that you can use. In fact, whenever the system is reset, only the console will be operative. (Making other terminals usable is known as "going multiuser".)

2.5.1. Terminal Settings

Terminals have many switch-selectable features, of which only a few concern us at present. The first, and most important, is the baud rate, that is, the transmission rate in bits per second. Both the CP/M system that we will use to check out the hard disk and Micronix initially expect that you will be using a baud rate of 9600. 9600 is the most common baud rate used for terminals connected directly to computers. Look up the baud rate selection, or option switch settings, in the operation manual for the terminal you are using, and set the terminal for 9600 baud. While you are looking at this section of your terminal's manual, also set the switches for:

- full duplex - since the Decision echos the characters

you type back to your screen, if you see two of everything later, you should double check this setting.

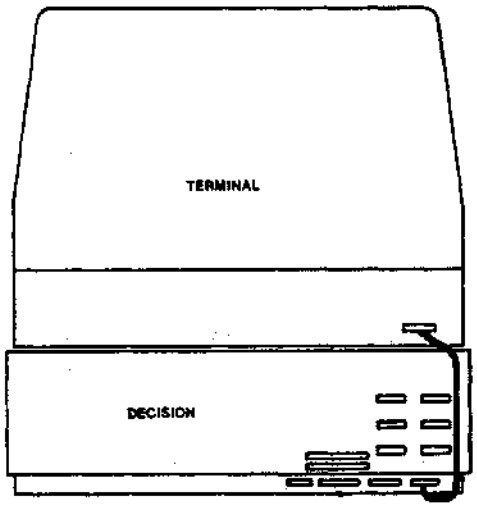
- 8 bits - the number of bits used to send a character;
- 2 stop bits - the pattern that marks the end of a character;
- no parity - parity is an option that is used for checking transmission accuracy (usually not a problem at 9600 baud); and
- on line - as opposed to local, means that the keys you type are transmitted to the computer; sometimes this switch will be on the keyboard.

The odds are that if you have already been using a terminal with a different computer, it is set up correctly for use with Micronix. Once again, the most likely cause of trouble here is the baud rate. The usual sign of using the wrong baud rate is that the sign-on messages (after booting or resetting) will be a nonsensical collection of assorted characters and the letters that you type will be displayed as other letters on your terminal. For example, if you type an "r" and the characters "~p" appear instead, your terminal is set to 4800 baud. The Decision has software-controlled baud rates, so you must set your terminal to 9600 baud initially in order to use the software to change the baud rate, if you so desire.

2.6. STEP 6: CONNECTING THE RS-232 TERMINAL CABLE

Now, look at the back of your terminal. There will be one or two trapezoid shaped female connectors there, (like the capitol letter "D" lying on its rounded side), the same size and shape of the end of the RS-232 cable we told you you'd be needing. If there is only one connector on the back of the terminal, great. If there are two, use the one labeled "modem", "RS-232", "main port", or "P1", not the one labeled "printer" or "aux". The second connector is not designed to be connected to a computer, and you don't need it. Connect one end of the RS-232 cable (either end is fine) to the correct connector on the back of the terminal.

The other end of the RS-232 cable is attached to the connector in the lower right hand corner of the back of the Decision. There are two additional connectors of the same shape and size to the left of this connector, when looking at the cabinet from the back. Don't use them now. Use the rightmost connector, nearest to the side of the cabinet, for the console.



Now, plug in the terminal and turn it on. You can use one of the two sockets on the back of the Decision to plug in your terminal, so that the console will turn on when the Decision is turned on. When the terminal is turned on, it may "beep", and you should be able to hear it warming up (sounds like a TV set, a high-pitched whine.) After warming up, (5 seconds or so), you should see a marker on the screen of your console called the cursor. The cursor marks the place where the next character will appear on the screen. (The cursor precurses the next character.) It will probably be in the upper left-hand corner of your screen.

If you don't see the cursor on the screen, you may need to adjust your terminal's brightness control. Before you leave this paragraph, find your cursor. If you can't find the cursor, something is wrong with your terminal, so you might as well just mark your place in this manual, and return when your terminal is okay. Thanks.

2.7. STEP 7: BOOTING UP CP/M

Now that we've got the basic connections made, a terminal and power, we're ready to load CP/M into your Decision. Although there is a complete Micronix operating system already on the hard disk, we'd like you to check things first by using the less sophisticated CP/M. Then you will run the **CHECK** program mentioned earlier. There is a lengthy primer on diskettes embedded in this procedure which the more experienced of you may skip.

Look in the packet of diskettes that came with your system and take out the disk labeled

COLD BOOT LOADER (CP/M)

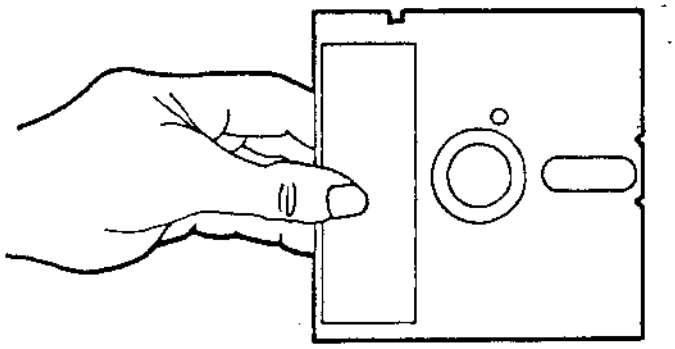
With the power on to your system and the console, insert this disk in the floppy disk drive. If you are familar with floppy disks, this should be no problem and you should skip the next section. For those of you unfamiliar with the use of floppy disks, the next section is for you.

2.7.1. TUTORIAL: Care and Feeding of Floppy Disks

There are two sizes of floppy disks available with Decision systems: 5 and 1/4 inch and 8 inch. Disks of both sizes consist of a black plastic envelope with a circular piece of recording material on the inside. The recording material is visible in the center hole of the disk, and through an oblong slot between the center and one edge of the diskette. Floppy disks are read and written through this oblong slot.

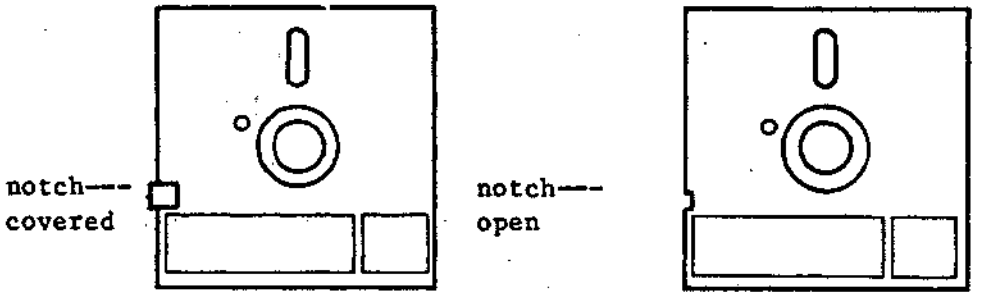
Be careful never to touch the recording material revealed though the oblong slot. The oil naturally occuring on your fingertips will ruin the magnetic material. The recording material can also be damaged by: being scratched, heated, folded or getting wet. Floppy disks are cheap and convenient, but somewhat fragile, so be careful when you handle them.

The top side of a floppy disk is the side with the label on it. The label is always next to the edge opposite the oblong slot, so the label is the best place to hold floppy disks. When inserting a floppy disk, **hold the disk with your thumb on the label and your other fingers underneath.** (It may happen that you have bought a bulk quantity of blank floppies without labels. In this case, the labels would go on the smooth side, the one without the seams folded over on it.)



Now for the difference between 5 and 1/4 and 8 inch diskettes. We'll talk about 5 and 1/4 inch diskettes first. The major difference is their size: 5 and 1/4 inch diskettes are 5 and 1/4 inch square. The other important difference is something called "write-protect". When a diskette is write-protected, it may be read, but not written on. Write-protection is a physical mechanism that cannot be overridden in software, so it provides a good way of protecting valuable data on diskettes from accidental erasure. Cassette tapes in stereo systems have a similar protection scheme.

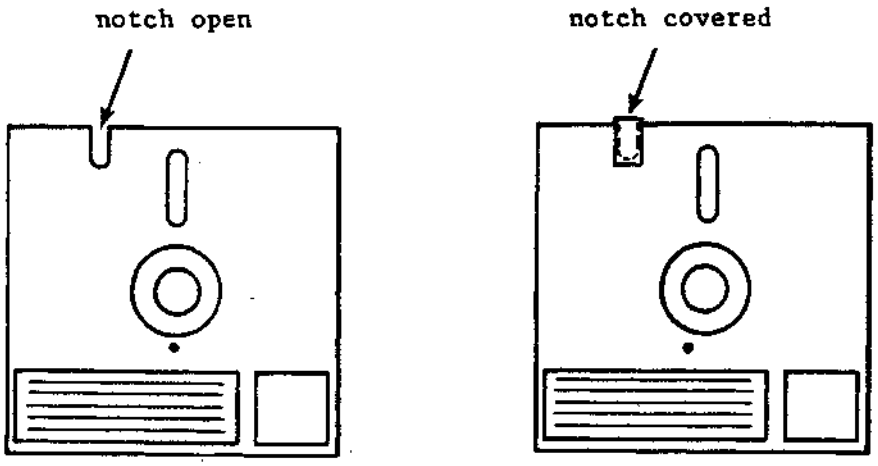
The write-protect notch on a 5 and 1/4 inch diskette is on the left hand edge when your are holding the diskette properly (by its label). When this notch is covered, the disk is write-protected. Left uncovered, the diskette can be written on or erased.



A WRITE-PROTECTED 5 1/4" DISKETTE

AN UNPROTECTED 5 1/4" DISKETTE

Eight inch floppy disks have the write-protect notch in a different position. The write-protect notch is on the edge of the diskette nearest the oblong opening. And, to confuse the issue, 8" diskettes use the opposite scheme from 5 and 1/4 inch diskettes: when the notch is open, the disk is write protected; when the notch is covered, it is write enabled. Make sure you've got that straight, or forget it if you don't use 8" diskettes.



A WRITE-PROTECTED 8" SINGLE-SIDED DISKETTE

AN UNPROTECTED 8" DOUBLE-SIDED DISKETTE

NOTE: The following discussion of formatting applies only to new diskettes, or those you wish to recycle. **Do not format any of the diskettes that came with your system.**

Before new floppy diskettes can be used the first time, they must be formatted. Formatting adds a timing pattern to a disk that must be there before an ordinary read or write can occur.

Even if you bought blank preformatted diskettes, they should be reformatted on your Decision before you use them. The fdj program should be used for this. The Decision accepts the following diskette formats:

5 and 1/4"-

North Star compatible, single and double density, single and double-sided, 35 and 40 tracks. For use exclusively with Micronix, use: double density, double-sided, 40 tracks. Soft-sectored Micro Decision format is also supported.

8"-

IBM 3740 compatible, single and double density, single and double sided. For use exclusively with Micronix, select double density, double sided, 512 byte sectors when using the fdj program. Micronix can handle the other format options as well, but this is the most efficient.

If you buy a diskette with software on it, it is already formatted, SO DON'T FORMAT IT or you will erase the software. The diskettes provided with your system all have software on them and don't need formatting.

If you're curious about the meanings of some of the terms used above, here we'll discuss **single versus double sided.** There is no difference in appearance between single and double sided 5 1/4" floppy diskettes. However, the 5 1/4" diskettes that are labeled and sold as double sided have passed a test for double-sidedness; the diskettes labeled as single-sided FAILED this test. **You should buy 5 1/4" diskettes certified as double-sided.**

There is a physical difference between single and double sided 8" diskettes. If you look at the drawings of 8" diskettes on the previous page, you will notice that one is labeled "SINGLE-SIDED", and the other is labeled "DOUBLE-SIDED". The difference between the two is the location of the index hole. The index hole is a small (1/4" or .6 cm.) hole located between the large hole in the middle of the diskette and the label. The index hole for single-sided 8" diskettes is located at about 6:30, a little off to the left below the center hole. In double-sided 8" diskettes, this hole is located further to the left, around 7 o'clock. This physical difference allows the hardware and software to distinguish between single and double sided 8" diskettes. If you want to use double-sided 8" diskettes, you must buy them this way.

The last point to discuss about floppy diskettes is how to open and close the drive doors. There are at least three types of drive doors. In every case, closing the door causes the diskette to be clamped in place through the hole in its middle. Please close the door gently, so that you don't mangle the edges of the center hole in the diskette.

The first type of door is a flap that covers part of the slot where you slide in a diskette. To open the door, lift up the flap. Slide the diskette in as far as it will go and close the door by pushing the flap down gently.

The second type of door is like a latch handle. To open the latch, turn the latch counterclockwise so that it pivots to a position above the slot. Slide the diskette in as far as it will go and close the latch by turning it clockwise until it is vertical. Simple enough?

The third type of door is used on some 8" drives. To open the door, press the rectangular button with the red indicator light in it. The door will pop open revealing the slot. Slide the diskette in as far as it will go. This time, there will be a "click" when the diskette is in far enough. If it is not, it will pop out like toast from a toaster. Just slide it in again, gently, until it clicks. Then, press down gently on the gate above the slot until it too clicks into place. Now you're all set.

You should NEVER close the door of a drive without a diskette in it. Closing the drive door moves the reading and writing heads into position for contacting the diskette. Without a diskette in place, the heads are in position for crashing together, possibly cracking or scratching each other. This is why there was a piece of cardboard in the drive when your system was shipped.

The red light that is on the front of disk drives is called an activity light. When the activity light is ON, it means the diskette is being read or written. NEVER open a drive door when the red light is ON.

2.7.2. Getting Ready to Boot. . .

We're back on track now. We assume that your Decision and the console terminal are up and running with their communications cable connected and the baud rate set to 9600.

After power is applied to the Decision, the computer is ready to be booted. The word "booting" refers to the phrase "to pull oneself up by one's own bootstraps". For computers, booting means loading in the operating system by reading in the first sector on the disk. The first sector contains enough instructions to read in the first two tracks on the diskette. On the disk labeled COLD BOOT LOADER (CP/M), the operating system is completely contained on these first two tracks.

Resetting the computer tells it to reload the operating system from the diskette into memory and to produce a prompt on the console screen to tell you that it is ready for further instructions.

Now, with the CP/M diskette in your floppy disk drive, close the door and press the red RESET switch. On keyswitch models, give the key a quick flick clockwise; a spring will push it back to the ON position (away from RESET) similar to a car's ignition switch. We have now arrived at checkpoint 2.

Checkpoint 2:

After you RESET the system, the red activity light on the front of the floppy disk drive should begin to blink. After a pause, the screen may be cleared, depending on your terminal, and the CP/M cold boot sign-on message and a prompt are displayed:

```
Morrow Designs 48K CP/M 2.2 E4
AB: DJ/DMA 5 1/4",  CD: DJ/DMA 8", E: HDC/DMA M5

A>[]
```

Once you see the CP/M prompt, "A>", CP/M has been successfully booted. Those of you familar with CP/M may like to take a look around by typing DIR <RETURN>. The files that include the word LOAD can be used to create a diskette that will boot Micronix directly. But we're jumping ahead. If you have the CP/M prompt on the console, you're doing great! Please skip ahead to STEP 8: CHECKING THE HARD DISK.

Troubleshooting 2:

1. The red activity light turns on and stays on. Please re-read the section on floppy disks because you have most likely inserted the diskette incorrectly.

2. The red activity light turned on, stayed on for several seconds, and turned off without anything appearing on your screen. When the red activity light has turned off, the first tracks of the diskette have been read successfully, so the problem is in the connection between the Decision and your terminal. Please go back to the section on connecting the first terminal.

3. Gibberish appeared on the console, instead the sign-on message. The baud rate is set incorrectly on your terminal. The correct baud rate for the console is 9600 baud.

4. The red activity light on the drive never turned on and nothing appeared- This indicates a hardware problem. The simplest answer is that a printed circuit board or cable was shaken loose during shipping. Please go through the section on CHECKING FOR HIDDEN DAMAGE again, and pay particular care to checking the cables. Then, turn on the computer and try again. If you still have a problem (no activity light), you will need technical assistance.

5. A cryptic looking message such as F0000, F8000, or M0BAD appears on the console instead of the sign-on message. This could indicate a loose board or cable. Refer to the section on CHECKING FOR HIDDEN DAMAGE for the things that you can try. If you still have problems, make a note of the error message that appears so you can tell your dealer or Morrow Customer Service what it said. (These messages are produced by the Monitor program in the MPZ80 board.)

6. You reset the system, the activity light goes on but turns off after several seconds, and the CP/M prompt doesn't appear- You are using the wrong diskette. You need the diskette whose label states COLD BOOT LOADER; there will be another diskette that starts out "CP/M Copyright Morrow Designs, 1979, 1980...". This diskette should have been included in your packet of diskettes. Please locate it, insert it, and RESET again.

7. You're sure you are using the correct diskette, but get no "A>" prompt- Well, there are two possiblities here: one, you've got the diskette in upside down and the system tracks are on the other side; two, the system tracks are unreadable. Hopefully, you forgot to insert the diskette label side up and close the door. If the diskette is in correctly and the activity light goes on and off after you reset, then we have a problem. You'll need to replace the diskette with another Micronix COLD BOOT LOADER (CP/M) diskette. Try your dealer.

2.8. STEP 8: CHECKING THE HARD DISK

We've finally reached the most important checkpoint. The reason for using CP/M is that it is somewhat simpler than Micronix, and possibly a lot more familar to you. Now that you have reached this step, you know that your terminal works and that you can read floppy disks. The moment of truth has arrived. Is the software that was carefully copied on the hard disk by the technicians at Morrow still readable?

The CP/M program that checks the hard disk is called, appropriately enough, CHECK. All CHECK does is attempt to read every sector on your hard disk. If CHECK is successful, you can probably boot up the hard disk and Micronix. If not, we'll explain to you how to initialize the hard disk for Micronix.

Ready? Okay, type CHECK and carriage return. The CHECK program will be loaded by CP/M and ask you which drive to check.

```
A>CHECK

     A) Drive 0. (if you have only one drive)
     B) Drive 1.
     C) Drive 2.
     D) Drive 3.

Drive: A

     A) m5
     B) m10
     C) m16

Selection: []
```

We want you to check Drive 0, so type an "A". Next, you are given the choice of three drive capacities: 5, 10 and 16 megabyte. Most Decisions are shipped with 10 megabyte miniwinchester hard disks. If you aren't sure which disk you bought, check your invoice (your copy of the bill).

The final selection required is the type of system: either CP/M or Micronix. Please enter a "B" for Micronix.

CHECK prints a line on the screen for each unreadable sector that it encounters. When it is finished, the total number of unreadable sectors is displayed. If you have more than 25 unreadable sectors, you will have to reinstall Micronix. If CHECK finishes (as signaled by the return of "A>") with zero or less than 25 bad sectors, the hard disk should be alright, and you are ready to boot Micronix.

If your hard disk failed CHECK, skip the next section.

2.8.1. Booting Micronix

Once CHECK has passed your hard disk, you are ready to boot Micronix. This is done by running a CP/M program on the same diskette as the CHECK program. There are four different boot programs, one for each size of hard disk:

M5BOOT	5 megabyte
M10BOOT	10 megabyte
M16BOOT	16 megabyte
HDBOOT	for systems with 8" hard disks

You boot Micronix by typing the name of one of these CP/M programs, and pressing **RETURN**. Since most Decisions have 10 megabyte miniwinchesters, most of you will be typing

```
A>M10BOOT
Micronix loader for the hard disk
Files
```

and Micronix will go through the process of loading itself. (Use the other boot programs if you have other capacity hard disks.) If you are curious about the messages that you see on the screen at this time, there is an explanation of the messages during loading in Section 3.2, Using Standalone Micronix.

Booting Micronix is complete when you have the superuser prompt, #, on your screen with the cursor waiting beside it. You can proceed to the narrative **FIRST TIME USE OF MICRONIX.** Please do not follow the instructions in the next chapter, INSTALLING MICRONIX ON THE HARD DISK, because Micronix is already there.

Then look through the division on **MAINTENANCE AND ADMINISTRATION** for information on adding new users, copying diskettes, backing up your hard disk, adding more terminals, configuring Micronix to your terminals and maintaining a healthy file system. Installing a printer is the last section of the Installation portion of this manual.

2.8.2. If CHECK FAILS...

The following discussion assumes that your hard disk failed the CHECK test. Don't be upset if it did. It used to be that we would tell you to reinstall Micronix regardless of the condition of the data on the hard disk, so you really haven't been set back.

If your hard disk is accessible but has 25 or more unreadable sectors, you must reformat your hard disk and reinstall Micronix. The chapter called **INSTALLING MICRONIX ON THE HARD DISK** is what you need to proceed with the reinstallation. How do you know that the hard disk is accessible? Two ways. The red activity light on the hard disk should be on, or flicker during the CHECK program. And if the CHECK program reports unreadable sectors, it is reporting the status from the hard disk controller, so the disk is accessible. If CHECK can't read the status, it will tell you so and you should return to the section on **CHECKING FOR HIDDEN DAMAGE.**

CHECK's inability to read the hard disk status points to possible hardware problems that might be an unseated board or poor cable connections. If you find a loose board or cable, you may want to try CHECK again after reseating the board or connecting the cable. If CHECK still fails, then you need to install Micronix, the subject of the next chapter.

```
**************************************************
*                                                *
*                                                *
*    PLEASE DO NOT FORMAT YOUR HARD DISK IF      *
*    YOU PASSED CHECK WITH LESS THAN 25 ERRORS.  *
*    THE FOLLOWING CHAPTER IS FOR THOSE WHO      *
*    MUST RE-INSTALL MICRONIX ONLY.              *
*                                                *
*    You are, of course, welcome to read this    *
*    Chapter for the information it contains.    *
*                                                *
**************************************************
```

3. INSTALLING MICRONIX ON THE HARD DISK

You will need to follow the steps in this section (the next 10 pages or so) **only if your system failed the CHECK program** as described above. These procedures are also used to restore Micronix on a hard disk when the root file system was destroyed. However, there is a considerable amount of useful information about Micronix in general contained herein, so you may wish to read through it in any case.

This section explains how to get a fresh copy of Micronix and all its software onto a hard disk. During this process, everything that was previously copied on the hard disk is overwritten. If you have been using your hard disk and have files on it, these will need to be restored from the backups that we hope you made. (If you are adding an additional hard disk, all you need to do is make a file (mkfs) system on it.)

There are three stages in installing Micronix:

1. Formatting the hard disk,

2. Using the Standalone Micronix diskette to build a skeletal system on the hard disk, and

3. Booting up the skeletal hard disk Micronix and copying the rest of the software to it.

These steps are carried out by using one software command for each stage. The remainder of this section explains the three software commands necessary and how to tell if they are working correctly.

3.1. FORMATTING THE HARD DISK

This procedure assumes that you are using a 5 1/4" miniwinchester hard disk as your root device. It is possible, though not likely, that you have an 8" or 14" hard disk instead. In these cases, you should run the CP/M program **formathd** instead of the program **formatmw** described below.

You will need to know the storage capacity of your hard disk. The 5 1/4" models are either 5, 10, or 16 megabytes, as indicated by a number in their model designation (for example, "m5").

Boot up CP/M (the COLD BOOT LOADER) as described earlier. Leave the boot diskette in the floppy drive.

Enter the capacity and sector size (always 512 bytes) on the command line as shown:

formatmw m# size 512 test

where m# is the drive type. For example, to format the 16 megabyte drive, with the CP/M boot diskette in the drive and with the A> prompt displayed, you should type:

formatmw m16 size 512 test

This will take around 45 minutes. If you run the formatmw program without the test option, it will only take about five minutes, instead of 45 minutes. The difference is that the test option reads and writes various test patterns, while formatmw without the test option writes only the byte E5 (hex) and checks to see if the sector can be read. Using formatmw without the test option may hide a problem that involves a pattern that is different than a sector filled with E5's. The decision, to use test or not, is up to you.

The information displayed on your screen during this process will look like

```
A>formatmw m16 size 512 test
Formatting.
Checking format.

Testing.
Bad sector report.
No bad sectors detected.

A>[]
```

If FORMATMW Seems To Be Having Problems:

In this example, everything went correctly and no problems occured. What type of problems can occur? Well, two types of problems are the most likely if any happen: the formatmw program can't access the controller or drive, or bad sectors are discovered while verifying the drive.

When formatmw has trouble accessing the HD-DMA controller board it provides a terse but unequivocal report:

Controller does not respond.

This report means that formatmw issued a command to select the drive and nothing happened within a reasonable length of time. You should check to see that the HD-DMA board is well seated. If the controller is working, formatmw next tries to check the drive status, and reports a failure as

Can't read drive status.

This may mean that the cables that connect the drive to the controller are loose, or the drive is not functioning correctly. If you have an extra hard disk located in an external cabinet, make sure that the power to it is on.

Lastly, if the formatmw program succeeds in reading the drive status, but finds that it is not ready, you will be told

Drive not ready.

The last two messages, "Can't read drive status" and "Drive not ready", are signals to you that there are problems with the hard disk. You may be able to solve the problem yourself. Prime places to check are the two cables between the controller and the drive, and the the four wires that run from the power supply to the drive. Look in the HD-DMA Technical Reference Manual for illustrations on how these cables are connected.

The second type of problem occurs after the drive has been formatted. After displaying the message "Checking format", the formatmw program attempts to read every sector on the disk. If there are any problems, a report is produced stating the type of error, the track, head and sector number. This report is called the bad spot table. Unless there are excessive bad spots (more than 128), this table will be written to the drive automatically.

If, in the bad spot table report, there are many errors labeled as "Soft", you may wish to try to run the formatmw program a second time. Soft errors represent marginal sectors, that is, sectors that can be read after several attempts. The sectors with Soft errors are set aside along with the sectors that have "Hard" errors, "Hard" meaning completely unreadable. Reformatting sometimes fixes Soft errors, so these sectors are not consigned to the bad spot table.

3.2 USING STANDALONE MICRONIX

The Standalone Micronix diskette contains the Micronix operating system and the absolute minimum of the programs that are needed to install Micronix 1.6 on the hard disk.

Trying to fit Micronix on a floppy disk isn't easy; besides, floppy disks operate at a much slower speed than hard disks. You don't want to, or need to, use the Standalone Micronix floppy disk except for special occasions. As soon as you can, make a backup copy of the Standalone diskette and stash it away in a safe place. (See Copying Micronix Diskettes, Maint. and Adm.).

There is one other thing we need to mention about the Standalone Micronix floppy diskette: it should be write-protected when used. 5 1/4" floppies should have a write-protect tab, while 8" floppies should have a no tab in place. The Standalone diskette you received will already be write-protected. Write-protection will prevent you from inadvertently damaging your precious Standalone Diskette. (The ability to write-protect Standalone Micronix diskette is a feature added with the release of Micronix 1.61).

Insert the "Stand Alone" diskette for Micronix 1.6 into the floppy drive and press Reset. Your Decision is now primed for action. Your display should now be:

```
Booting

micronix.mw

Micronix 1.61
Copyright 1983 Gary Fitts

256k memory
14 cache blocks
20 processes
disks:  djdma hddma
root dev:  djdma/76
swap dev:  nodev/0

Root device is Read-only

Welcome to Micronix.  The stand-alone floppy system that
you are now running is designed to initialize your hard
disk, and to perform surgery on it if necessary.  Please
do not attempt to run multiple users yet.  To initialize
your hard disk:

     IF YOU HAVE                        TYPE
an m5 with the HDDMA controller      source m5init
an m10 with the HDDMA controller     source m10init
an m16 with the HDDMA controller     source m16init
any drive with the HDC controller    source hdinit

Then boot micronix from the hard disk.  If you have any
questions, please read the installation section of the
Micronix User's Manual.
```

Note: All standard Micronix configurations will include HDDMA controllers instead of HDC controllers.

Okay, does this all look familiar? If everything went right, you should have seen the lines above displayed on your terminal. We'll give a quick explanation of these lines mean.

Booting means that the program that starts Micronix has been read sucessfully. If your Stand Alone diskette is unreadable, you will never get the message "Booting". Make certain that you have inserted the diskette correctly. The booting program next tries to load the file **micronix.mw**. If this fails, the message "BAD LOAD" will be displayed. You may want to try again, because if you can't load this diskette, you'll need to acquire a replacement from your dealer (or Morrow). The next line displays the release level of Micronix, 1.61 in this example, and a Copyright notice.

After the Copyright notice, the next six lines give a report on the system you are using. The report for your system will be different than this one if you are using more memory, the HDC controller, etc. The last line of this report states: "swap dev: nodev/0". In English, this line is telling you that your swap device, used for swapping between users, is no device at all. So don't ever type the word "exit" (which causes Micronix to become multi-user) while you are using Stand-Alone Micronix, or you'll find yourself swapped out into space!

Now, on to business. The message beginning "Welcome to Micronix..." should still be on your screen. It states the same warning as in the paragraph above (do not attempt to run multiple users). In fact, you don't want to do anything until your hard disk is ready. Just follow instructions. Next to "#", type:

source m5init<return>

for 5 megabyte miniwinchesters,

source m10init<return>

for 10 megabyte miniwinchesters,

source m16init<return>

for 16 megabyte miniwinchesters, or

source hdinit<return>

for any hard disk using the HDC controller. This is the single software command that you use to initialize your hard disk. The source program works something like CP/M's SUBMIT program. The source command expects the name that follows it to be the name of a file containing other commands to be followed. And, indeed, each of these four files, m5init, m10init, m16init and hdinit, contains the necessary commands to prepare your particular hard disk.

Once you have pressed <return>, the process of initialization is automatic, and takes about 30 minutes to complete. Presuming that you have successfully formatted your hard disk just previously to this, everything will happen automatically. If you get any errors during this process, you should proceed with these instructions anyway, since almost all of the files on the Stand Alone floppy are duplicated on the distribution volumes that you will now install.

3.3. BUILDING A SKELETAL MICRONIX

While you are waiting for initialization to complete, we thought you might appreciate an explanation of what's happening. So here goes. (Most of the commands used during initialization are explained in MAINTENANCE AND ADMINISTRATION, so the descriptions here are brief and not detailed.)

mkfs — make a file system- this command adds organization to your recently formatted hard disk; specifically, mkfs writes the information necessary for an empty Micronix file system to your hard disk. This takes about a minute.

badspots — copies the bad spot table from the hard disk to a file that is used by fsck in the next step.

fsck — file system check- this command is the very effective file system repair and maintenance program. It checks the new file system for any errors, and fixes them if it finds any. (You should use this daily.) The fsck command is used here with the "-t" option, which means that fsck will "test" every sector in the file system. This takes from 15 to 25 minutes.

era — erases files.

mount — mount a file system- mount is used to add, in this case, the new file system on the hard disk to the file system on the Stand-Alone floppy disk.

chmod — change file access mode- adjust the access permissions for the new file system on the hard disk.

cptree — copy directory trees- copies groups of files from one part of a file system to another; in this case, from the floppy portion to the hard disk part of the file system (remember, mount added the hard disk's file system to the Stand-Alone floppy file system.) Takes about 3 minutes.

ln — link names together- a way to have two (or more) filenames for the same file.

ddt — dynamic disk tool- allows the editing of object (machine code) files; this particular command modifies the file "micronix" so that Micronix uses the hard disk instead of the floppy disk, and adds the hard disk as the swap device.

umount — unmounts file system- finishes writing to the mounted file system before it is removed.

echo simply echoes whatever follows it.

sync synchronize- force all disk buffers to be written to the disk they belong on. Sync is ALWAYS used before turning off Micronix.

You have read up to this point, but the system has not completed initialization. Great! This gives us a little more time to talk about "sync". Do you remember any of the old war movies, where the sargeant looks his men in the eye and says: "Ready? Synchronize your watches... Now!" The idea was to set all of the watches to the same time so that if anyone was late he couldn't blame his watch.

Synchronization is a little different with Micronix. What is synchronized in Micronix is sectors in memory with the same sectors on disk. Any operating system uses some memory as a temporary storage place for information to be read from or written to disks. Micronix, as a matter of fact, has room for 14 such places (remember the "14 cache buffers" report during the loading process?). Besides the cache buffers, which are used for data, Micronix also keeps information about the file system and open files in memory. This speeds up the operation of Micronix a lot, memory being much faster than disks. This also means that the information in memory can be changed without changing the corresponding information on disk.

So, sync forces all the buffers and information in memory that have been changed since the last sync to be written to the corresponding place on the disk. Then, memory and disk have been synchronized, that is, they are the same. Just imagine the trouble you could have without sync! Suppose, for example, you had just finished writing a really tremendous resume', or an encryption program the National Security Agency is sure to outlaw. In your excitement, you shut off your system without typing sync! Fudge! Great Jehosaphat! Even though the editor program said that it was finished writing, your file is still (or was still) in the cache buffers when you turned the system off. Too bad.

There are other, even worse things that can happen if you shut down or reset prematurely. What if you are running multi-user? Suppose that you type sync, and then as you reach for the power switch another user makes a change on the disk? You want to be sure that no other programs are running. This is all taken care of by the **down** command. So please make it a habit to type

```
# down
```

before turning off the power or resetting the system. Note that down is rather draconian - it pulls the rug out from under everyone (but in an orderly fashion). For this reason, it only works for the super-user, who presumably knows how to warn everyone first (See **TURNING YOUR MICRONIX SYSTEM ON AND OFF**).

On the other hand, anyone can use sync at any time. The system itself does a sync at the end of every command, and also one every 30 seconds for good measure when running multi-user.

Okay. When the initialization process has completed, a succinct message,

```
Disk initialization complete.
# []
```

will be displayed on your screen, followed by the "#" prompt. We are going to reset the system and load Micronix from the hard disk. What do you think you should do before resetting? Please type

```
# down

Micronix is down.
[]
```

After the activity light on the floppy drive stops flashing, remove the Stand-Alone Micronix diskette and put the COLD BOOT LOADER (CP/M) diskette back in, and reset with the reset button or the keyswitch. You are now ready for the final stage of initialization.

3.4. BOOTING A SKELETAL HARD DISK MICRONIX

The previous stage, initializing the hard disk, created a file system and copied enough software to it to get things started. In this, the final stage, you will copy the rest of the software (and on-line documentation) that is a part of Micronix to the hard disk. But first, let's boot the hard disk.

Booting Micronix on the hard disk is essentially the same as booting the Stand-Alone Micronix floppy. After you have booted CP/M, use the boot command for your hard disk to load Micronix. The boot commands are

m5boot	Loads Micronix from the 5 megabyte miniwini
m10boot	Loads Micronix from the 10 megabyte miniwini
m16boot	Loads Micronix from the 16 megabyte minimini
hdboot	Loads Micronix for the HDC disk controller

In this example, we are booting a 10 megabyte miniwinchester with the m10boot command. You, of course, should substitute the correct command for your hard disk. Type "m10boot<RETURN>", and you should see the following display:

```
A>m10boot
Micronix Loader for the HD-DMA.
Files:
micronix
Loading
```

and so on until the "#" prompt appears. You are now ready for stage three.

3.5 ADDING THE SOFTWARE

The Micronix system on your hard disk has a bare minimum of software on it now. The remainder of the software is on the diskettes labeled Volume I, Volume II, etc. You may also have diskettes labeled "Whitesmith's C" or Pascal if you purchased them. Each of these diskettes are added to the file system by the use of a single software command, repeated for each diskette. The CP/M software (such as Logicalc and Personal Pearl) that was included with your system is installed with a different set of commands.

Please remove the CP/M Cold Boot Loader diskette from the floppy drive if you haven't done so already, and insert the diskette labeled Volume I. This diskette should be write-protected, as it is your master copy of the software. These diskettes, (the Volumes), contain a portion of the Micronix software in fp format. The software command you need to type is source finstall<RETURN>. After typing it, your display looks like

```
# source finstall
x bin
x bin/form
   ...
x bin/date
"beep!"
# []
```

Copying Volume I takes about 3 minutes and is done by the commands in the file finstall, which stands for first installation. If you remember from the last stage, the source command takes the name that follows it as the name of a file with commands to be executed. The finstall file has a command that extracts files from the volume (fp x). The last line in finstall will echo a "beep" to your terminal to signal completion. When the first volume is finished, remove it and insert the second volume. This time type

```
# !s
source finstall
x bin/ddt
  ...
x man/man1/account.1
"beep!"
# []
```

The "!s" is shorthand for "re-execute the last command that began with s". The Micronix shell, the command line interpreter, remembers the last twenty commands you have used.

The only problems you may have during this stage will involve the floppy diskettes. The fp command may produce some error messages if it encounters difficulty while reading a diskette. This may result in only a portion of the diskette being copied. For example, if finstall has trouble halfway through a diskette, you will get a message like

```
x bin/wall
Read error block 280 on disk djdma/12
bin/wall: I/O error
x bin/upm
...
```

In this example, the finstall script failed to copy bin/wall. Sometimes there will be several unreadable files. Please copy the names of the files labeled with the message "I/O error" so you can get good copies from your dealer or Morrow.

If finstall can't read the diskette at all, you will get a message that is like

```
# source finstall
Volume not in fp format.
"beep!"
# []
```

You may want to remove the diskette, and reinsert it and try again. Make certain that you are inserting the diskette correctly. If the diskette is unreadable, you can try to get a replacement from your dealer or, get one from Morrow.

After each volume has been copied, remove it and insert the next volume, until all diskettes have been copied, including "C" and Pascal if you have them. The same command, "!s" for source finstall, is used each time to start the copy. Each diskette will take between three and seven minutes to copy. If you are an experienced Micronix user, you should run the **fsck** command after copying the last diskette. For the majority of you, the next section explains how to run fsck, and will familiarize you with Micronix.

4. FIRST TIME USE OF MICRONIX

This chapter of the **INSTALLATION** division explains what you should do the first time you use Micronix. We will ask you to run one more check, then give you a tour of the file system. During the tour, you will be provided with the commands for moving about the file system on your own, and a map of the file system to help you find your way.

The next division of the manual, **MAINTENANCE AND ADMINISTRATION**, provides in-depth explanations of how the file system check program works, how to format and copy diskettes and how to back up the hard disk. By the time you have finished installing and checking Micronix you may have had enough of computers for the day. However, you should start backing up your hard disk the same day as you start creating files of your own on it. So don't delay reading and using the section on backing up the hard disk.

4.1. CHECKING THE FILE SYSTEM

Possibly you are a little tired of all the checks you have run on your system. Well, the check you are about to use is the one that you will use every day to ensure the health of your file system. Let's get it started at once, and you can read the explanation while it runs. Type

```
# fsck /dev/root <return>
```

and read on.

Having a system programmer on call to repair your file system would be quite an expensive luxury. Instead, the file system check command is your system programmer simulator. The fsck command explores the entire data structure that makes up a file system, checking for damage using the knowledge and experiance of the system programmer that wrote fsck. You see, the Micronix (and UNIX) file systems are made up of several parts that cross-reference one another. By comparing the references in one part of the file system to the other parts, the fsck program can determine if all the references are consistant with one another.

If references in parts of the file system disagree, fsck corrects the differences by repairing the file system. Often, this repair work will be unnoticable to users of the file system. Occasionally, fsck will discover damage to a file or files. When this occurs, a report of the damaged files, called "fsck.victims", is created that contains the names of the damaged files and the nature of the repairs made to them.

There are two ways a file system can become damaged: through a hardware malfunction and by shutting down the system improperly. A hardware malfunction that damages the file system involves an error in writing to the disk, or the loss of one or more sectors on the hard disk. Shutting down the system improperly occurs when the power is turned off, or lost suddenly, while a program or disk access is in progress. While you may never shut off your system improperly, you can't be certain that there won't be a power failure or that the hardware will function perfectly forever. This is one reason why you should run fsck every day.

The other reason to make running fsck a daily habit is that small, minor file problems can grow and multiply if not corrected early on. Fsck can detect problems as subtle as a single block being claimed by two different files. It will attempt to correct the situation and will let you know the outcome.

By now, fsck should be finished. If everything went well, your display looks something like:

```
# fsck /dev/root
Checking /dev/root
** Checking I-list, first pass
** Checking the free list
** Checking the I-list, second pass
** Checking connectivity
380 files, 47 special, 28 directories, 250 small, 100 large, 0 huge
30188 blocks, 732 I-list, 130 indir, 0 in2dir, 4502 data
4632 used, 24822 free, 0 bad
# []
```

Some of the statistics reported on the last three lines may be different for your system. There are other lines that may have been displayed when you ran fsck. The free list may have been rebuilt, for example. The only case where you need to take action is if the line

**** Hunting up filenames of casualties**

is displayed. The names of the casualties are also kept in the file fsck.victims. If you have any victims, you should replace them from backups. (See Fsck.victims in **MAINTENANCE AND ADMINISTRATION.**)

One more note about fsck, before moving on. This time we ran fsck as the first step in exploring Micronix. In fact, **running fsck is the first thing you should do every day, before anyone else is using the system.** If any files are being used while fsck is being run, the files involved will be changing and fool fsck into thinking that some damage has occured. In this case, fsck may cause the damage. You may use fsck only while in single user mode, or on unmounted file systems. Mounting is explained later, but just remember: RUN FSCK BEFORE STARTING ANYTHING ELSE.

4.2. SETTING THE DATE

The Decision that you are using has a clock in it that will need setting every time the system is turned on. The clock supplies date and time information for the file system. Files in Micronix keep a record of the last date they were modified, which can be used to select files for backup on a basis of when they were last changed. The only way this will work is if you remember to set the clock after running fsck. To set the clock, you simply type "date", followed by the date and time in the requested format. For example, to set the clock to 11:03 a.m. on April 14, 1983, you would type

```
# date apr 14 1983 11:03
Thu Apr 14 11:03:41 1983
# []
```

The clock in the Decision is a 24 hour clock, so that 2:00 p.m. is represented as 14:00. To convert an afternoon or evening hour to the 24 hour clock representation, add 12 to the hour. For example, to reset the time of the clock to 4:12 p.m., you can type

```
# date 16:12
Thu Apr 14 16:12:41 1983
# []
```

The date command recognizes the different parts of the date through the use of hints. For example, one or two digits followed by a colon (:) means the time of day. A one or two digit number preceded by the first three letters of a month is expected to be the date. A four digit number without a colon is considered to be the year. So, the hints are: a colon in the number means time, the first three letters of the month means a date, and a four digit number without either of these hints is the year. Here are a few examples:

```
# date dec 31 23:59:59 1983      -the last second before 1984
# date 17:00                     -sets the clock to 5:00 p.m.
# date 1984                      -sets the year to 1984
# date apr 1                     -sets the date to April 1st
```

Notice that you never specify the day of the week. The date command will always know which day it is, if you provide the date.

4.3. MOVING AROUND THE FILE SYSTEM

Now that you've taken care of some details, how would you like a look around? One of the nicest features of the Micronix file system is that it allows the use of multiple directories. Each directory can have a name that relates to the files that are kept in that directory. This helps you to organize your hard disk, and may help you in remembering where you have left certain files when the file system gets large.

To help you get a feel for file systems, we would like you to visualize a group of rooms. In each room, there are file cabinets and doors which lead to other rooms. The file cabinets and the doors all have names on them. The file cabinets in this analogy represent files, and the rooms represent directories. The doors that lead to other rooms are the pathways to the other rooms, that is, directories. In the discussion that follows, the words "room" and "directory" will be used interchangeably.

There are three commands that will assist you in moving around the file system, and visiting different rooms. These commands are:

cd change directory, lets you move between rooms,

dir directory list, lets you see what's in a room, and

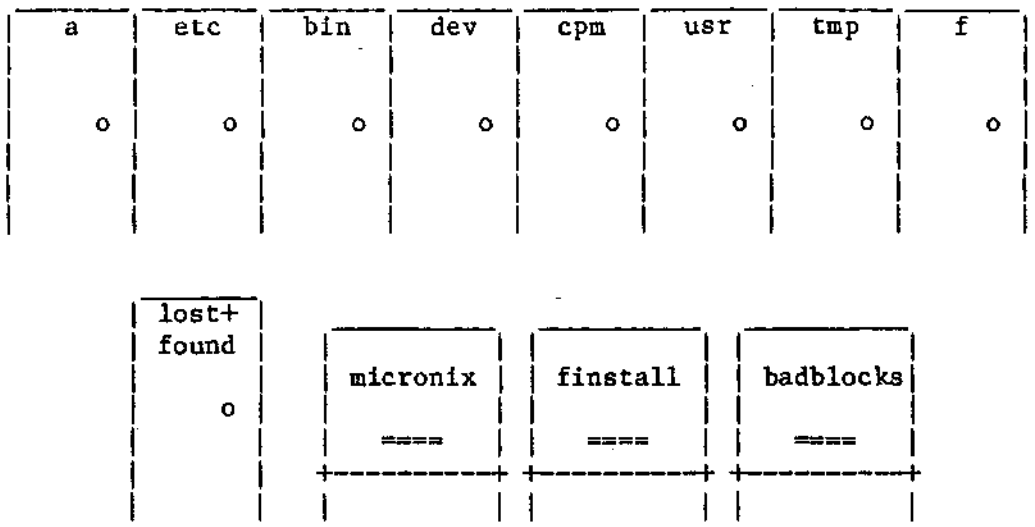
pwd path way to directory, tells you what room you are in.

Whenever you enter the Micronix file system, you start out in the room that is determined by your user name. During this first visit, there is only one user name available and it is "root". Later, you can add other user names. The root directory is the beginning of the file system. The doors in this room, in other words, lead to the rest of the file system. Let's take a look around by typing **dir <return>**.

```
# dir
a             cpm           f             micronix
badblocks     dev           finstall      tmp
bin           etc           lost+found    usr
# []
```

Most of the names in this directory are probably unfamiliar to you now except for micronix, and possibly finstall. As it turns out, micronix, finstall and badblocks are the only files in the root directory. The rest of the names are the names of other directories written on the doors. Visualizing this room, the

root directory looks like:



To save us some time, we will tell you that there's nothing behind some of the doors. The tmp directory will be used later for temporary files. The lost+found room is used by fsck as a place for keeping "orphaned" files and directories. The "a" directory will have user directories branch off from it. The "f" directory is set aside for adding file systems on floppy disks to the hard disk file system with the **mount** command.

Before we leave root, you should become aware that the root directory has an unusual name. If you type **pwd <return>**, you can see root's pathname. Pathnames are the sequence of doors that you must go through starting at "root" and ending at a particular file.

```
# pwd
/
# []
```

The slash (/) by itself represents the root directory. You will see later that this slash is at the beginning of all other full pathnames, as the root is the ancestor of all other files and directories.

4.3.1 The BIN Directory

Now we are going to walk through the door labeled "bin" and visit the /bin directory. The command,

```
# cd bin
# []
```

moves us there, and

```
# dir
account     df          icheck      netact      stty
badspots    diablo      kill        netcp       su
cal         diff        last        netdaemon   sum
cat         down        lines       netmail     tail
chars       du          ln          od          td
chmod       echo        login       owner       tee
clean       edit        lpr         passwd      touch
cmp         far         ls          paste       tree
comm        fdj         mail        pilot       tty
cp          field       make        print       umount
cptree      file        man         ps          unique
create      find        mkdir       pwd         update
cu          form        mkfs        recon       upm
cxr         fp          mknod       rm          wall
dar         fsck        more        rp          wc
date        grep        mount       sh          who
dc          group       msgs        sleep       word
dcheck      hd          mv          sort        write
ddt         help        ncheck      split
```

gives us a look around. Quite a lot of files here, eh? The bin directory contains most of the Micronix commands in the form of files. In fact, the name "bin" refers to binary (executable) files.

Some of the files here may look somewhat familiar. For example, the **date** command is near the bottom of the first column, and **fsck** is near the bottom of the second column of filenames. Some command names are pretty cryptic, while others are actually English words related to the action of the commands. Later, you will be given the knowledge of how to create "aliases" for the command names that you'd like to change.

4.3.2 The DEVICE Directory

Let's move on to another directory. We are currently in /bin, as shown by using the path way to directory command:

```
# pwd
/bin
# []
```

We want to go through the door in the root room labeled "dev", but currently we are in the room behind the door labeled "bin". If you use the change directory (**cd**) command from here with the name of the "dev" door alone, it won't work:

```
# cd dev
dev: no such file or directory
# []
```

It looks like "dev" isn't there, but that is only an illusion caused by looking for the dev door while were visiting the bin room. What you need to do is specify that you are looking for the door to dev from the root room, named slash (/). To do this, you use the change directory (cd) command and specify that the dev room is found by starting at the root:

```
# cd /dev
# []
```

This works because we spelled out the path way to the directory that must be followed: starting at root (/), look for dev. More on path ways to directories later. Let's examine the **dev** room.

```
# dir
cent0       flb         ml0c        mem         ttyC
cent1       flc         ml0d        mfa         ttyD
cent2       fld         ml6a        mfb         ttyE
cent3       hda         ml6b        mfc         ttyF
console     hdb         ml6c        mfd         ttyG
diab0       hdc         ml6d        null        ttyH
diab1       hdd         m5a         root        ttyI
diab2       io          m5b         swap        ttyJ
diab3       ml0a        m5c         ttyA        ttyK
fla         ml0b        m5d         ttyB        ttyL
# []
```

The device directory is filled with the names of the devices used by Micronix. For example, **console** is the first terminal device, **ttyB** is the second terminal, **ttyC** is the third terminal, **cent0** is the first centronics type device, and **ml6a** is the first miniwinchester of 16 megabyte capacity.

All the files in dev are called **special files.** Each file has, instead of data, two numbers that refer to the software driver for the device built into the Micronix operating system. Information sent to the file /dev/ttyC is routed through the

driver to a device, for example, a printer, by Micronix. Don't worry if "software driver" is a foreign phrase to you. You won't be tested on this.

So, you could say the the /dev room is the place in the file system where the connections to the devices, like terminals and hard disks, are found. Let's move on to the /etc room.

4.3.3 The ETC Directory

It is easy to remember the name of this directory by thinking of it as the "et cetera room". Files that are used in system administration are stored here. Let's move into /etc and look around.

```
# cd /etc
# dir
banner          motd            passwd          terminals
group           mtab            rc              ttys
init            netmap          signon          utmp
# []
```

The files in /etc are all text files, except for **init**. For example, the **passwd** file contains user names and their encrypted passwords. The **group** file contains similar information about groups. **Motd** stands for message of the day. You can type out motd by typing the command:

```
# type motd
```

You will later edit the motd file to display reminders or inspirational quotations to yourself and others as users log in. The **signon** file contains the message that appears on the console after a system reset. The banner file is an interesting one. Take a look at it by typing

```
# type banner
```

The division of this manual called **MAINTENANCE AND ADMISTRATION** has more to say on editing these files.

4.3.4 The USR Directory

The acronym "usr" stands for "user service routines". What's actually kept there aren't so much routines as directories. Let's take a look at /usr:

```
# cd /usr
# dir
adm             man             spool
bin             pub             upm
# []
```

Immediately, the "bin" entry looks familiar. Don't confuse this bin directory with the one described earlier though. This brings

up an important point: **You can have two files or directories with the same name, so long as they aren't both at the end of the same path.** for example, you could have two files named "john" on the system, in different directories such as /a/john and /b/john.

The **/usr/bin** directory contains more executable binary files. The **adm** directory is used by Micronix as a storeplace for login times. The **pub** directory contains only one file: a table of characters and their ASCII equivalent in octal and hexadecimal. The **spool** directory is used as a temporary storage place for print spooling. Spool is also used by the mail and networking programs. The upm directory is used by upm for terminal emulation information.

The **man** directory leads us in the direction of a vast repository of information. Change directory to man and take a look:

```
# cd man
# dir
man0            man2            man4
man1            man3            man5
# []
```

The man directory contains 6 subdirectories. Each subdirectory has files for all the pages in a section of the manual. The **man** command, as in MANual, accesses these pages, so that information about all Micronix programs, subroutines, system calls, devices and files are on-line and readily available. The **help** command uses these same files in the man directory.

While we're here, let's examine the path way to the man directory. We started at the root, /, changed directory to usr, and changed to man. So the path way we wind up with should contain each of these names, /, usr and man. And, indeed it does:

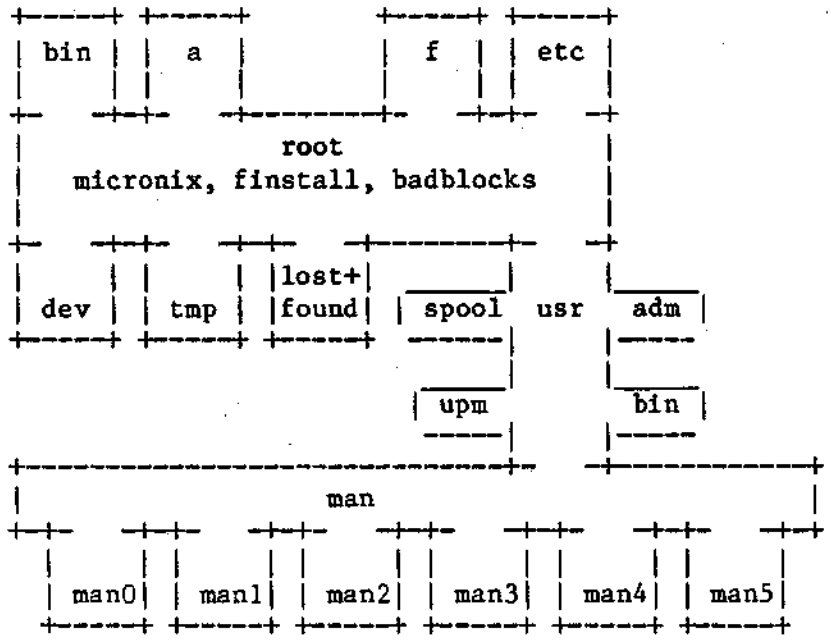
```
# pwd
/usr/man
# []
```

The initial slash (/) stands for the root directory. The second slash is a divider between subsequent directories. If we had changed to the man2 subdirectory of man, the path way to directory would be /usr/man/man2. You can visualize getting to man2 as starting in the root room, walking through the door labeled usr, going through the door marked man, finally into man2 room.

4.3.5 A Map Of The File System

At the beginning of this tour, we promised a map of the file system. After you have used the Micronix system for a while, you will become familiar with the various names and path ways, and won't be needing a map. For now, here is a map showing the file

system as it currently exists.



You can add new files and directories where you like to the root, or any other directory. The initial path ways to these directories are important, however, so don't go renaming "man" to "manuals", or "bin" to "commands" because you will make these directories impossible to find for the programs that know where they are now.

4.4 ADDING A PASSWORD FOR ROOT

You are now approaching a momentous occasion: you are about to add some security to your Micronix system. As the system stands, anyone can become the root by resetting the system and booting Micronix. The root is also known as the superuser because there is absolutely no protection from the person logged in as root. The root user can examine any file, or remove any file. The reason for the root's power of unlimited access is that a user must exist that can access all the files for backup, or to use fsck.

You can easily prevent this unlimited access by using the **passwd** command to install a password for the root. Once you have installed a password, you will have to remember what it is because you will use it every time you reset the system or log in as root. If you ever forget or lose the root password, you will have to "operate" on the passwd file from the Standalone Micronix disk and remove the root's password. Okay. Let's choose a password for root. You should probably write it down, and keep it someplace safe. Suggested passwords are nonsense words, or, even better, combinations of letters and numbers at least eight characters long. As an example, let's use "az804me2".

```
# passwd
New Password (or RETURN for no password):
Once again to be sure:
# []
```

The characters "az804me2" were typed after both colons in the example. The passwords you use will never appear on the screen for security reasons. The reason for typing the password twice is to check that you typed it correctly the first time.

4.5. GOING MULTIUSER

The transition from single-user to multi-user Micronix is very simple. Just type

```
# exit
```

and press <RETURN>. Micronix will then automatically make the changes necessary to become multi-user. After going multi-user, the Micronix banner will be displayed, and a log-in name requested, as in

```
Name:
```

Since this is still, presumably, your first use of Micronix, you should use the name "root". Next, you will be asked for a password. Type in the password that you added for root in the last section. The message of the day will then be displayed, and the "#" prompt. After you have added other user names, you will be able to log-in using names other than the root.

Even if you plan to be the only user on your system, you should probably bring your Micronix system up multi-user. Until your system is multi-user, only the console can be used. If you have a program crash while you are working at the console, the console could still be under the control of the crashed program. Then, the only way to regain control is to reset the system, risking possible data loss and damage to the file system. But, if you are operating in multi-user mode, you can plug your terminal into the second serial port, log-in, and kill the offending program that has captured the console. See the section named Desparate Measures in Maintenance and Administration.

The other important advantage of going multi-user is the waking up of the update daemon. The update daemon sleeps in the background, and wakes up every thiry seconds to execute the sync command. The periodic and automatic syncs by the update daemon serve to keep the disk current with memory without any intervention on your own part. This can be a file system saver if the system goes down unexpectedly.

The update daemon is started during execution of the script in the /etc/rc file. If you have other commands that you want to

execute routinely while going multi-user, you can edit /etc/rc and add the commands here. For example, you could run fsck as part of this routine without actually ever keying in the command. See edit in the Programs section of the Reference Manual, and rc in the Files section.

4.6. TURNING YOUR MICRONIX SYSTEM ON AND OFF

We are sure that you understand that you can't turn Micronix on and off like a desk lamp. As we have mentioned, improper shutdowns are a prime cause of file system damage. Since your system is currently on and running multi-user, let's turn it off the right way.

The software command for bringing down your Micronix system is named "down". The down command first calls the sync command to synchronize the disks with memory. Then, down goes out and kills any processes that are still alive. When it is finished, the message

```
Micronix is down
```

is displayed on the console. After this message has appeared, you can turn the power off to your hard disk, (if it is in a separate cabinet), and then turn the power off to the Decision and other peripherals (terminals, printers, modems, etc.).

The down command can only be used by the superuser. You are the superuser when you are logged in as root, and have the superuser prompt, "#". This is because only the superuser has permission to kill all active processes. Before typing "down" as superuser, you should certainly inform everyone else who is using the system that you intend to bring it down. If the other users aren't in the same room as you, you can use the **wall** (write all) command to send them a warning:

```
# wall
I am turning the system off in 1 minute!!!
<CONTROL-D>
# I am turning the system off in 1 minute!!!
[]
```

The <CONTROL-D> means hold down the control key and type a d. Control-d is used to signal end of file to Micronix, in this case, end of message.

To turn your system on, you start by turning the power on to the hard disk, if it is in a separate cabinet from the Decision, then turn on the Decision. After you turn on the hard disk, you will hear its drive motor come up to speed, and possibly the sound of the stepper after several seconds.

Insert the CP/M Cold Boot Loader diskette and press Reset or flick the keyswitch to the Reset position. If the system passes its startup diagnostics tests, the CP/M prompt (A>) and sign-on message appear on the console. There are other possiblities that indicate a hardware problem, which were discussed back in section 2.7.2. When CP/M presents its prompt, you need to type the name of the loader program for your system:

m5boot	for 5 megabyte hard disks
m10boot	for 10 megabyte hard disks
m16boot	for 16 megabyte hard disks
hdboot	for hard disks with the HDC controller

For example, to load Micronix on a 10 megabyte hard disk, you should type

A>m10boot<return>

and Micronix will begin loading from the 10 megabyte hard disk. If you have installed a password on the root, you will be required to log-in as root and use the password. After logging in, there are two things to do:

- o run fsck on the /dev/root device, and
- o set the date.

Both fsck and date have been explained previously.

It is possible to load Micronix without using CP/M. Reading the section in **MAINTENANCE AND ADMINISTRATION** about the SYSGEN command will explain how to create a diskette that loads Micronix directly.

5. CONNECTING THE PRINTER

Before you attempt to connect a printer, please finish the rest of the installation procedures for Micronix. If you are successfully running a multi-user Micronix, you are ready for printer installation.

Micronix and the Decision come equipped to handle four types of printer interfaces. The interface refers to the physical connection, the cable, between the printer and the Decision. The interface also refers to the method used by your printer to stop the transmission of characters to be printed when it gets behind in printing, and how it restarts transmission. This is called handshaking. The four types of interfaces are:

1. Serial printers (or "RS-232") with X-ON and X-OFF handshaking,

2. Daisy-wheel printers with 50 pin cables, (Diablo standard interfaces),

3. Serial printers with hardware handshaking, and

4. Parallel printers with centronics-style interfaces.

If you have either the first or second type of printer, you're in luck. You can skip the rest of this section and start reading PLUGGING IN THE PRINTER. If you don't have one of these two types, or are unsure of which type you are using, read on.

With parallel printers, the handshaking is built into the parallel interface. Serial printers use either software handshaking or hardware handshaking. Software handshaking means that when the printer's buffer is full it sends a character to the computer to stop the computer from outputting more characters. The printer sends a different character when it is ready to receive again. The character that Micronix recognizes for stopping transmission is X-OFF. Other aliases for X-OFF are control-S, DC3 and 13 hex. The character that restarts output is X-ON, also known as control-Q, DC1 and 11 hex. If you have a serial printer that doesn't use X-ON and X-OFF, for example, it uses ETX and ACK instead, you'll have to try hardware handshaking.

Hardware handshaking makes use of one of the wires in an RS-232 cable to start and stop output from the printer. When the printer wants to stop transmission, it lowers the voltage on this wire. To continue transmission, the voltage is raised again. If your printer uses hardware handshaking, please read the section on cables in the DEVICES section of the Micronix technical manual because, unfortunately, there is no such thing as a real life standard for either printers or computers. Hardware handshaking between a printer and the Decision will probably require modification of the RS-232 cable. After modifying your cable, read the next section, called PLUGGING IN YOUR PRINTER.

Parallel printers using Centronics-style interfaces will also need to have a special cable made for them. Please read the section on cables in the DEVICES division, or buy the cable from Morrow then return to the next section and follow instructions for parallel printers. The software for handling Centronics interfaces is built into Micronix.

5.1. PLUGGING IN YOUR PRINTER

Okay, here's where you'll need to be able to see the back of your Decision again. There are probably two unused connectors of the type you need on the back of the Decision, so it will be much easier to be able to see what you are doing.

If you are using a serial printer, the connector you need is just like the one the console is connected to: a trapezoid (like a D lying on its rounded part) that matches the end of the RS-232 cable. The console is connected to the rightmost of three connectors, looking at the Decision from the back; you should connect the serial printer cable to the LEFTMOST connector. The connector in the middle is reserved for a second user.

There is a diagram of the Decision back panel on a green page at the beginning of INSTALLATION division of the User Guide.

For those of you with parallel printer cables, you need the 50 pin connector immediately above the serial printer connector. There are two 50 pin connectors on the back of your Decision; the upper one is for connection to an 8" disk drive. Obviously, you will connect your printer to the lower one.

The 50 pin connector is not "keyed": that is, it is quite possible to plug in your parallel cable upside down. Well, read on for some guidance. On many cables, pin 1 is labeled by having a red stripe on that edge. Or, if you constructed your own cable, you should know where pin 1 is. The 50 pin connector on the back of your Decision has pin 1 on the left hand side when looking at the back of the cabinet. Plug the parallel cable into the Decision with the red stripe, and pin 1, toward the left (while looking at the back of the Decision).

Parallel printer cables that are built by Morrow have an additional way of telling you which way to plug the cable in. The red stripe will be on the left from the back AND the cable will hang down without covering the connector. So, if you put the connector on upside down, the cable will be folded over and cover the connector. Install the connector so that the cable hangs down without obscuring the connector.

5.2. TESTING YOUR PRINTER

Everyone have your printer connector plugged in? Turn the printer ON, make sure that it is ON-LINE, and we'll try it out. (There are several different expressions used for "on-line" by printer manufacturers. Most often, though, there will simply be a switch labeled online, and usually a light will go on to indicate on-line status.) There are three different test commands for printers: one for serial printers, and two for parallel printers. The difference between the commands is in the way the device is named. For serial printers, set the printer to 1200 baud, and try typing

```
% echo A > /dev/ttyC
```

If your printer can't be set to 1200 baud, you can try this command anyway, but, at best, you'll get the wrong letter. The next section explains how to change your baud rate. Skip ahead, change your baud rate, and try out the "echo" command again.

For parallel daisy-wheel printers, try

```
% echo A > /dev/diab0
```

and for Centronic style parallel printers, use

```
% echo A > /dev/cent0
```

Everybody get an "A" on your printer? Those of you who got an "A", get an A in interfacing, and should start reading the next section, ADJUSTING MICRONIX FOR YOUR PRINTER. For the rest of you, interfacing printers IS non-trivial. Just in case our problem is a trivial one, please check to see if:

- your printer is turned on (and plugged in),
- the printer is set to ON-LINE,
- there is paper in it (many printers won't print unless they can sense a piece of paper in them; the same is true for printer ribbons),
- the cable is plugged in at both ends (at the correct connector; re-read the beginning of this section, please.)

If you have a parallel cable, you can also try plugging it in upside down. After trying all of the suggestions, try the suggested command again, please. If you get a different letter than an "A", there are some wires scambled in your parallel cable, or your serial printer is set to the wrong baud rate.

If none of these suggestions helps, your cable is suspect. Home built cables should be rechecked and store bought cables checked for flaws. RS-232 cables, whether home-made or

purchased, can be checked by using them to replace the cable connecting your console. Parallel cables can be checked for continuity using an ohmmeter or continuity tester.

It is possible that other problems with your printer will surface later. The most common of these is missing letters or words after several sentences, or even many lines, have been printed successfully. When this happens, it looks like everything is going great, until the printer starts pounding out nonsense. What has happened is this: Your printer has a small buffer built into it that stores letters until they can be printed. If the letters are received slightly faster than they can be printed, everything works well until the buffer gets completely full. Then the printer signals the computer to stop sending by way of handshaking, but the handshaking doesn't work. The garbled output is the result of the handshaking not working, and being hidden because the printers buffer can almost keep up.

When you received some correct letters and/or words, the problem is the handshaking. Handshaking is the most common difficulty faced by owners of new printers or systems. If handshaking is your problem, you have two choices available: try to solve your handshaking puzzle, or reduce the baud rate at your printer and Micronix. It may be that your cable has one wrong wire that a diligent search will uncover. Then again, reducing your baud rate may turn out to affect the speed of your printer very little because it was accepting characters much faster than it could print them anyway.

5.3. ADJUSTING MICRONIX FOR YOUR PRINTER

If you have a serial printer with software handshaking operating at 1200 baud, you can forget this section and go on to USING YOUR PRINTER WITH MICRONIX. For the rest of you, the changes you need to make are simple. The file that determines which printer port and handshaking to use is called /etc/ttys. You can either use the **recon** program or edit /etc/ttys. The recon program is pretty much self-documenting. All you need to do is type "recon" while you are logged in as root.

We will edit the /etc/ttys file so that it agrees with the type of printer you are using and you can see what it is that recon does. (There is an example of using recon in the section on Additional Printers several pages ahead, information on recon in the Terminals section of MAINTENANCE AND ADMINISTRATION, and more information on /etc/ttys in the reference manual in the Files division.) Please log in as root and type

```
# edit /etc/ttys
```

The editor responds with the number of lines in the /etc/ttys file, and a colon (:) prompt on the next line. North Star compatible diskettes are hard sectored, with ten sectors. Hard sectored means that there is a hole punched in the diskette material marking the beginning of each sector. There is also an extra

hole that marks the first sector. Micro-Decision diskettes are soft sectored. Soft sectored diskettes have only the single hole that marks the first sector.

You can check to see if a diskette is hard sectored or not by looking through the e

```
ttyC  300  lpr lst  shake
.                                   finish with a dot
:w                                  write out the file
"/etc/ttys", writing 14 lines
:q                                  quit
# []
```

Easy enough. Now, Micronix has been configured to use a 300 baud serial printer with hardware handshaking. If you make a mistake while typing, the backspace key (or leftarrow key) lets you backspace over the error. If you feel like you may have really messed up, do not enter the write command (w), but enter q! instead and start all over.

For those of you with **parallel printers**, you will be changing a line in the file, and removing the words "lpr" and "lst" from the serial printer line. If you are using a daisy-wheel printer, you are changing a line beginning with "diab0". For centronics-style printers, the line begins with "cent0". The following script assumes you are using a centronics so substitute "diab0" for "cent0" if you are need to. Begin with the **edit** command given above.

```
:/lpr/                              find the lpr entry
ttyC  1200 lpr lst
:c                                  change the line
ttyC
.                                   period finishes change
:/cent0/                            find your printer
cent0
:c                                  change it by
cent0   lpr  lst                    retyping the line
.                                   a dot ends the change
:w                                  write the file
"/etc/ttys", 14 lines
:q                                  quit
% []
```

That's it. You have modified Micronix for use with your printer. Unless you had to make your own cable, it was a pretty painless process, wasn't it? Now, for the moment of truth: let's see if it works.

NOTE: Those of you that are using daisy-wheel printers and the diab0 interface should read the entry in the Programs section for diablo. The diablo program provides the extra control codes required for micro-justification that parallel daisy-wheel printers require.

5.4. USING LPR, THE PRINTER COMMAND

When we first tested your printer, we used a command, "echo", and directed its output directly to the printer. In Micronix, all output directed to the file with the name of your printer interface (/dev/ttyC, /dev/diab0, or /dev/cent0) will go directly to your printer. The problem with this is: what if two people try to use the printer at the same time? The answer is that both people get to use the printer simultaneously and the resulting output is a real mess.

The lpr program provides a solution to this problem. The lpr program copies files to be output to a special directory called /usr/spool/lpr. Many people can be using the lpr program simultaneously because it copies each person's output into a separate file in the /usr/spool/lpr directory. And, if there is no printing going on at the moment, lpr awakens a daemon. The daemon handles printing out the line (queue) of printer files one at a time, allowing you to go onto something else instead of waiting for the printout to finish.

The lpr daemon is assigned to printing out the files in the /usr/spool/lpr directory. The lpr program always checks first to see if there is already a daemon awake, so that there are never two lpr daemons around at the same time. Once a daemon is awakened, it functions as a process independent of any user or terminal. It is as if there were a ghost in the machine that handles printing. When there are no more files to print, the lpr daemon goes back to sleep.

The changes that you made in the file "/etc/ttys" inform the lpr daemon which printer interface to use. Let's make a test of your changes to /etc/ttys, by printing it out using the lpr daemon.

```
% lpr /etc/ttys
%
```

The second % prompt should appear before the file /etc/ttys starts printing. This is because the daemon works independently of the lpr command that you type at your terminal. As soon as the lpr command finishes copying /etc/ttys to /usr/spool/lpr, control is returned to you. The technique used with lpr is called "spooling". The name spooling refers to using software for collecting printer output and passing the output along to the printer at a rate the printer can handle. The entry for the lpr command in the Programs section of this manual explains the options available with lpr.

People using parallel daisy-wheel printers will need to use a different way of invoking lpr, because they must add the micro-justification codes for their printers. To use lpr, you should type the line

```
% type /etc/ttys | diablo | lpr
```

5.5 USING YOUR PRINTER WITH CP/M

The Micronix CP/M emulator, **upm**, is configured so that output sent to the CP/M LST: device goes directly to your printer interface file (/dev/ttyC, /dev/cent0, or /dev/diab0). For serial printers this was already set up for you by the presence of the letters "lst" on the ttyC line in file /etc/ttys. Otherwise you put "lst" on the diab0 or cent0 line yourself a page or two back.

If you are the only user of the Micronix system, this may be acceptable. There are situations where you will use this LST linkup, such as using the PRINT command under WordStar. However, there will be times when you'd rather use the **lpr** command for printing instead. As mentioned previously, if multiple users send output to the LST device at the same time, the printer will intermingle their words in the most avant garde fashion. Also, the printer users all have to wait until the printing is finished before starting something else. The lpr daemon eliminates these problems.

If you're not interested in using lpr and you are the sole user of a printer, you can skip this section. However, you should read the next section on configuring WordStar.

5.5.1 Using LPR With CP/M

Using the lpr daemon from upm is a two step process. The first step is changing the connection made between the LST: device and the ttys file entry for your printer. You want to change this so that your output for the printer is directed to a temporary file in your directory. Assuming you are in the upm emulator (see **upm** in the Programs division of the Reference manual), enter

```
A>LST:prntfile
```

Now, everything that CP/M sends to list goes into **prntfile**. To get the daemon working on printing prntfile, you need to temporarily escape from upm, and start up the daemon. This is done by using the exclamation point (!) to escape from upm:

```
A>!lpr prntfile
```

This command sends the contents of **prntfile** to the printer's spooling file, where it waits for the daemon to get it printed. (The exclamation point causes upm to pass the command onto Micronix). The data is still there in prntfile, though, and if you send more stuff to printfile via LST, it just gets tacked onto the end of what's already there. So when you go to print it (by entering "!lpr prntfile") you'll reprint what you printed a little while ago. Sooooo. . ., you have to empty printfile out before using it as the recipient of further LST: output.

Typing this command empties prntfile and prepares it for more listing:

```
A>LST:prntfile
```

This is basically a return to step one. The command empties the contents of prntfile without deleting its name from the directory. The next time you want to start the daemon again, go back to the second step, "!lpr prntfile".

5.6 ADDITIONAL PRINTERS

This section explains how to configure Micronix to use an additional printer. To make it easier on us, we will assume you are connecting a serial printer to interface /dev/ttyB. ttyB corresponds to the center RS-232 connector on the rear of the chassis, between the console and first printer connections. If you are adding a parallel printer, the earlier sections will provide enough information for altering the information in this section to fit your parallel configuration, if you extrapolate a little. For example, instead of using ttyB, you would use cent0 for a Centronics style printer interface.

The first thing to do is change the /etc/ttys file and add in the name you have chosen for your new printer daemon. That's right, if you have two printers, you can have a daemon for each of them, and they must have different names. We will use the name tpr for our second printer daemon, to keep it straight from lpr (our other daemon's name). This is the name that we need to add to the entry for ttyB, using the recon program:

```
# recon
Micronix IO configuration

   1  Help
   2  Configure login terminals
   3  Configure printers
   4  Quit and ABANDON all changes
   5  Quit and KEEP all changes

Choose 1 to 5 --> 3
At present, the printer configuration is
   - A printer called lpr is connected to parallel port 0.
     It uses a Centronics parallel interface.
   - The default CP/M list device is the lpr printer.
Do you want to make any changes? (y or n) --> y
How many printers do you want to connect?
Choose 1 to 9 --> 2
Please specify a name for each printer.
The  name is arbitrary, but it traditionally ends in 'pr'.
There should be one printer named lpr .

Name for printer number 1 --> lpr
Name for printer number 2 --> tpr
```

```
Please specify the type of interface for each printer.
The options are Serial, Centronics-parallel, or Diablo-parallel.

Interface for the lpr printer (S, C, or D) --> C
Interface for the tpr printer (S, C, or D) --> S

Serial printers require a baud rate.
The most common are 110 300 600 1200 2400 4800 9600

Baud rate for the tpr printer --> 1200

Serial printers also need a handshaking method.
The options are XON-XOFF and CLEAR-TO-SEND.

Handshaking for the tpr printer (C or X) --> C
Finally, you must specify an IO port for each printer.
The options for serial printers are ttyB through ttyL.
The options for parallel printers are port0 through port3.

Parallel port for the lpr printer --> 0
Serial port for the tpr printer --> B
Last but not least, you should specify a default CPM list device.

Name of printer to use as LST: device --> lpr
The proposed printer configuration is
   - A printer called tpr is connected to serial port ttyB.
     It runs at 1200 baud and uses CLEAR-TO-SEND handshaking.
   - A printer called lpr is connected to parallel port 0.
     It uses a Centronics parallel interface.
   - The default CP/M list device is the lpr printer.
Press RETURN to continue -->
   1  Help
   2  Configure login terminals
   3  Configure printers
   4  Quit and ABANDON all changes
   5  Quit and KEEP all changes

Choose 1 to 5 --> 5
# []
```

The handshaking options given here, XON-XOFF or CLEAR-TO-SEND, correspond to software handshaking and hardware handshaking, respectively. If this isn't clear to you, reread the section on Connecting the First Printer.

One more thing. The first time you invoke tpr, Micronix will still respond "Command not found". The secret is to try a second time. Micronix will rebuild its internal listing of commands (hash table) if a previously unknown command is invoked twice, and, of course, if it's really in there.

6. INSTALLING YOUR CP/M SOFTWARE

One of the wonderful things about your Micronix system is that you can use most of the software that was developed for the CP/M operating system. The CP/M emulator, upm, creates an environment for CP/M programs that is almost indistinguishable from an ordinary CP/M operating system. (For details on how upm differs from "ordinary" CP/M, check the description of upm in the Programs section of the Reference Manual. It will also tell you how such CP/M details as disk drive assignments relate to the Micronix file system).

Included with your Micronix system are several packets of diskettes which have CP/M software on them. These are essentially the same programs distributed with Morrow's CP/M systems. The programs include:

- NewWord, the Mercedes-Benz of wordprocessing,
- Microsoft's MBASIC (BASIC 80),
- Micro Mike's baZic (Z-80, North Star compatible BASIC)
- Logicalc, a financial modeling and report generation package, commonly refered to as a spreadsheet program,
- Personal Pearl, a database program designed for ease of use and complete management of data, and
- Correct-it, a spelling correction program.

You might also have purchased some of the Whitesmith software, such as the C programming language and Pascal. The Whitesmith's software is not on CP/M formatted diskettes, but should be added with the finstall command. (See Chapter 3, Adding the Software).

These software packages are not included on your hard disk when it is delivered mainly because of licensing restrictions. There is an additional reason for not adding all the software packages to your hard disk: altogether, they would take over 2 megabytes of space on your hard disk. What you need to decide is which of these packages you are going to use. Then, you can copy and install the chosen ones on your hard disk. Of course, there is nothing to keep you from installing all of them aside from disk space conservation.

Please peruse your documentation for the software packages and decide which ones you want. Except for the BASIC's which are combined on one diskette, each software package comes on its own diskette (or set of diskettes). Select the diskettes with the software you want to use in preparation for the next step.

6.1 Copying CP/M Diskettes to the Hard Disk

The Micronix program for copying CP/M diskettes is named far (short for floppy archiver). The far program possesses a working knowledge of the layout of CP/M diskettes, and how to read and write them. The CP/M emulator, upm, does not know what to do with CP/M formatted diskettes (it uses the Micronix file system, which differs radically from the one used in CP/M) so you need to use far to provide the interface between Micronix's file system and CP/M floppy disks.

Your Micronix file system has a directory set up for CP/M files, named cpm. There is nothing sacred or special about /cpm; it's just a convenient space for copying your software to. Later, you can move or copy files anywhere you want them. To get ready to copy diskettes, let's move into the /cpm directory with the command

```
# cd /cpm
# []
```

Now that you're in position, insert a diskette that you'd like to copy into a floppy drive. The floppy disk drives are known by several names, depending on the drive type. Here are the three most common floppy drive device names:

```
mfa  - 5 1/4" hard sectored diskettes, or MicroDecision soft
       sectored, single or double sided, first drive
mfa2 - 5 1/4" soft sectored, double-sided diskettes, non-
       Morrow formats, first drive,
fla  - 8" IBM 3740-formatted diskettes, also first drive
```

We'll consistently use mfa, since that's what most of you will be using, and expect the rest of you to substitute mfa2 or fla where ever we use mfa. The application software distributed by Morrow comes on soft-sectored 5 1/4" diskettes. (If you have an older Decision, you may need to upgrade the DJ/DMA with new PROMs so it can read soft-sectored diskettes).

The command that will copy an entire diskette to your current directory is:

```
# far mfa -xv
x  BAZIC08.COM
x  ...
# []
```

The x option stands for extract files, and the v means "verbose" (that is, "Tell me what's going on as you perform the copy.") This command will continue unassisted until the entire diskette is copied unless your hard disk file system becomes full. If you get an out-of-space error, you will need to clean out your hard disk first.

You can repeat the far command with each diskette that you want to copy to the hard disk. If it was the last command that you used, remember that you can use the shorthand form, !f, as in

```
# !f
far mfa xv
lc.com x
lc.set x
...
# []
```

and the shell will expand this into the last command that you used (beginning with "f") and extract the files from the next diskette.

6.2 Customizing Software

Much of the software that is written these days is designed to be interactive. And, most of the terminals sold are somewhat intelligent, that is, they can move their cursor about, clear their screens, and maybe even insert in the middle of a line. Since there are so many degrees of intelligence and precious little standardization in the codes they use, the terminal is what you customize the software for. Customizing software normally requires that you run an installation program that comes with the software to incorporate your terminal type into the software. You may also be asked to specify a printer model.

We really can't provide information about all the customizing procedures that exist. We would never finish the book about them, because the number of programs is constantly growing. What we will do is give you some hints about how to customize your software for Micronix, and describe two program customizations in detail.

Here are the general guidelines for running the installation programs:

1. You will find the programs in the /cpm directory, if that is where you copied the floppies to.

2. With the current distribution software, their names are **nwinst.com** for NewWord, **lcset.com** for LogiCalc, **welcome.com** for Personal Pearl, and **kins.com** for Quest.

3. Pick a terminal from the menus you see. Since upm has a built-in terminal emulator, you may be choosing one of these three: ADM-3a, Soroc IQ-120 or ADM 31. There is more information on terminal selection in the section on Terminal Emulation and upm.

4. If you are offered a printer driver menu, always select the CP/M LST: device driver.

5. If you are offered a printer selection menu, and you don't see your printer mentioned, you are always safe by specifying "TTY-like printer."

6. If you encounter a selection of communication protocols, answer "none", since this will be handled by Micronix.

7. If you are technically astute, there are usually many installation options that you can specify beyond terminal and printer type. Most of these have minor impacts on the operation of the software, and thus are not required. Refer to the manual for the software to locate these detailed options.

What you mustn't do is use special S-100 video boards and custom drivers with Micronix. If an installation program offers a custom terminal driver, you must reject the offer. The same goes for adding an I/O or video board of your own and adding a custom driver for it. Micronix jealously guards its privilege of handling input/output, and will quash any attempts to do it directly. (Actually, there is an indirect way for doing this by using the io or mem devices which will work in some cases. See the Devices section of the Reference Manual. You can also purchase a version of Micronix that allows you to add custom drivers, but it's for system programmers only.)

The same warning goes for printer installation. WordStar, for example, offers to patch in its own drivers for you. You must firmly refuse such offers because they won't work. Micronix has what is called an interrupt driven I/O system which is superior to the techniques that can be accomplished with a software only approach. Always select the LST: device for printer output.

Some software packages expect that you will be using floppy disks, and will have to divide their package between two drives. Since you are using a hard disk this won't be necessary. What might be necessary is if you go out and buy software, you may have to dig into its manual and find out how to change the default drive specifications. The software that comes with the Decision has already been modified for this.

When you become familiar with the workings of **.upm** files, you will see that there are advantages to having several versions of the same software package installed at various places in the file system. For example, if you have three users, each with a different level of terminal, you would have to run the WordStar installation program three times, creating WordStar command files in the /cpm directory with three different names. Or, you could put the command files into the users' respective home directories. Or, use upm's terminal emulation.

For more information, refer to the writeup on the upm program in the Programs section of the Reference Manual and to the section on .upm files under Customizing the Environment in the MAINTENANCE AND ADMINISTRATION division of the User's Guide (whew!).

6.3 Terminal Emulation and upm

The upm program has the capability of emulating different terminals. A terminal emulator converts the control characters that your terminal and a terminal-configured piece of software use to communicate "special effects". These special effects are things like moving the cursor around, clearing the screen or highlighting parts of the screen. By converting control characters you can make several terminals all look the same for software. This may allow you to use the same configured version of software for all your terminals.

Terminal emulation may also be of use to you if you have an unusual terminal that is not listed in software installation programs. You would simply add your terminal specifications to the Micronix file /etc/terminals, and upm will emulate one of the more common types of terminals for you. Then, you install software for the terminal that upm is emulating.

There are three different terminals that upm can emulate: an ADM-3A, a Soroc IQ-120, and an ADM-31. These three terminals represent increasingly sophisticated levels of terminal "intelligence". The ADM-3A, the least intelligent, can clear the screen and move the cursor. The Soroc IQ-120 represents the next level. The Soroc can do everything the ADM-3A can do, plus it can display characters in high or low intensity and has erase to end of screen and end of line functions.

The most intelligent terminals have all the capabilities previously mentioned, plus delete and insert line and character functions. The ADM-31 has all of these capabilities and represents the "highest" level of terminal intelligence.

6.3.1 Turning Terminal Emulation On and Off

Terminal emulation is activated by adding the terminal type to a line in the /etc/ttys file. upm notices that the ttys file has been changed, and searches for the line corresponding to the port that the terminal running upm is attached to. If this line has a terminal name on it, upm then searches the file /etc/terminals for the matching terminal. If there is no terminal name on the line in the ttys file, or a matching terminal in the terminals file, upm turns off terminal emulation.

Terminal names are added to the /etc/ttys file with the recon program. The recon program displays the names of terminals in the /etc/terminals file, asks for the name of terminal connected to each login port, and adds the name to the line for

that port. If you examine the ttys file after using recon, you would see that terminal names have been added after the word "term=", as in

```
ttyA      9600      login      term=mor20
```

In this case, the terminal name is mor20. If you prefer not to use the recon program to add terminal names, you can add the names yourself with an editor. Be careful not to leave a space between "term=" and the terminal name. The terminal name must be one of the abbreviated mnemonic names listed in the /etc/terminals file. (See the ttys (5) section in the back of this binder).

recon will also tell you which of the three terminals (ADM-3A, Soroc, or ADM-31) that upm will emulate. This is the terminal that you must install your software for. When you use terminal emulation, the control codes sent to the emulator must be one of the three terminals emulated. The emulator, in turn, translates these control codes into the ones needed by your terminal. This only works if you install your software properly. Just remember to install your software for the terminal that upm is emulating, as reported by recon.

6.3.2 Terminals File

The /etc/terminals file contains the names and capabilities of the terminals that you will be using on your system. Initially, the terminals file has only about twenty different terminal names in it, so you may not find your terminal in the file. The terminals file is purposely short in an attempt to reduce the time spent scanning through the file looking for terminal names and capabilities.

Each terminal definition in /etc/terminals begins with an abbreviation for the name of the terminal. The abbreviated names of terminals are the ONLY items in /etc/terminals that begin in column 1, the leftmost column. After the abbreviation, the full name of the terminal may appear. On the lines that follow the terminal's name are the definition of the characters that communicate special effects.

If your terminal name is not listed, you can add it by following the directions given in the entry in the Reference Manual for Files, terminals (5), in the back part of this binder. An explanation of what terminal characteristics are required to qualify a terminal at a particular level is also explained in terminals (5).

6.3.3 Upm's Notepad: The upmttyX File

The process of checking the /etc/terminals and /etc/ttys files to see if changes have been made takes upm a noticeable amount of time. In fact, the longer the /etc/terminals file is, the longer it will take upm to search through it. To reduce the

time added by including terminal emulation, a special file is created for each login port when upm is executed. If upm finds this file and neither /etc/ttys or /etc/terminals has been modified more recently than this file, then upm uses the information in this file for terminal emulation. This file is in the /usr/upm directory, and is named upmttyX, where X is a letter representing the port, for example, A for the console (upmttyA).

If you do not have a /usr/upm directory, upm will never be able to create the special file upmttyX, and will take 10 seconds or so to start working every time. Once a upmttyX file has been created, upm will start working much quicker. If your file system lacks this directory, you can make one by typing

```
# mkdir /usr/upm
# []
```

upm will change the upmttyX file if it discovers that either the files /etc/ttys or /etc/terminals have been changed since the upmttyX file was created. This means that you don't need to signal a change in terminal type to upm because upm will always check for it.

6.3.4 EXAMPLE: Installing NewWord

As an example of what to do when customizing a CP/M program, we will point out what you need to do to customize NewWord. The **nwinstal** program handles installation of NewWord, and contains enough explanations to guide you through. And, if you need more help using nwinstal, you should refer to the NewWord installation manual (Nuts and Bolts), for assistance. We will quickly step through the procedure to point out the potential trouble spots.

You begin installing NewWord by executing nwinstal and providing answers to the first two prompts:

```
# nwinst
Copyright NewStar Inc., 1983

Name of file to install? nwu
File to hold installed NewWord? nwl
```

The name of the file to hold the installed NewWord is actually up to you. We suggest that you choose a name that reflects the terminal (or printer) that you are installing. In this example, we are using the name nwl to stand for NewWord installed for a level 1 terminal. (See terminal emulation).

There are two things that you must do next: select your terminal, and select your printer. To select a terminal, enter menu selection A. A list of terminals will be presented. Either select your own terminal, if you only have one terminal, or select one of the three emulated terminals if there are going to

be several terminals. Please refer to the previous explanation of terminal emulation.

The next step is to select a printer. Menu selection B allows you to pick a printer. If your printer is on this list, all you have to do is enter the correct letter. If not, chose either the draft printer or the typewriter printer. Draft printers are not expected to have any special features; typewriter printers have the ability to backspace, whcih makes special effects, such as **boldface print** easier to print.

NewWord always sends its output via the LST: device, so your modifications to Micronix will correctly handle printer output.

At this point, you are finished with normal installation of NewWord. We would like you to make a few more modifications at this point. Of course, you are free to modify other installable features as you see fit.

There are five delays used by NewWord. We would like you to disable all of them by setting them to 0. From nwinstal's Main Menu, select Menu D. Then change the Short, Medium, Long and Cursor Position delays to 0. This will improve the performance of NewWord considerably for use with Micronix.

The changes made to NewWord utilizing the nwinstal disable most of the internal timing used by the program. Some of the best software available for CP/M was written to simulate multi-processing on single processing systems. NewWord is very good at this, but Micronix is, of course, better. If you have a program that allows two activities to occur simultaneously, such as typing and printing, or typing and screen rewriting, you'll want to disable, if possible, the internal timing loops that drive these functions.

Internal timing loops are short programs that run rapidly in a circle, simulating a stopwatch. The problem with this is you don't want to waste the power of your CPU by making it pretend to be a clock. Your Decision has a perfectly good clock in it already. Most of the timing delays required by a program will occur naturally as a result of multi-user Micronix's normal processing. If the removal of timing loop delays adversely affects the use of a program, put the needed one back. An example of a needed delay is the medium delay in NewWord. Without the medium delay, you will immediately be presented with a Help Screen whether you need it or not (unless you disable all Help).

6.3.5 ANOTHER EXAMPLE: Customizing Personal Pearl

The Pearlsoft people want to make things as easy as possible for you. Unfortunately, there's no mention of an Installation program anywhere in the manual. There is, however, on page 6, a mention of a "Welcome Disk". After looking carefully through

your packages of diskettes for a "Welcome Disk" for Personal Pearl, you dug out this manual and hoped we would provide you with a few clues. Well, we're sorry about the lack of a "Welcome Disk". Your Pearl Software is on three diskettes labeled 1 of 3, 2 of 3 and 3 of 3. We suggest that you follow the instructions previously offered for copying the diskettes to your hard disk.

Once you have done this, you can use the **welcome** program that is now in the /cpm directory, and proceed with installation of Pearl. You can skip the section on copying (creating working) diskettes: you've already copied all of your software to the hard disk. Select the Personalize option to add in your terminal type. There is more information on Installation in Pearl's Appendix C, in the Advanced Features section of the manual.

A. Installing WordStar

For those of you who have purchased (or own) WordStar, we have included this section on information specific to installing WordStar. The **instws** program handles installation of WordStar, and contains enough explanations to guide you through. And, if you need more help using instws, you should refer to the WordStar Installation Manual (Section 3), for assistance. We will quickly step through the procedure to point out the potential trouble spots.

The first phase of the instws program allows you to select your terminal, which should be easy. (Either select your own terminal, if you only have one terminal, or select one of the three emulated terminals if there are going to be several terminals. Please refer to the previous explanation of terminal emulation). The next step is to select a printer. WordStar will modify itself to provide any special control characters your printer may desire. Simple enough.

Now, we come to the potential trouble-maker, the Communications Protocol Menu. WordStar graciously offers to handle the protocol for you, which you don't need. Please type N for "NONE required" (or protocol handled outside of WordStar, which it is). If you capitulate to this offer, your very first attempt to access your printer through WordStar will result in WordStar being halted, and the message

```
attempt to execute illegal instruction: core dumped
```

being displayed. Micronix traps programs that attempt to perform I/O, and assumes that a mistake has been made. A special file, named **core**, will be created under the assumption that you are debugging a program. Since your mistake is of another nature, just remove (erase) the core file. (Core files are described in the Files section of the Reference Manual. See upm in the Programs section for a list of illegal instructions).

Your next choice is the selection of Printer Drivers. Always select the LST: device, since the actual handling of I/O will be performed by Micronix. The instws program will tell you to use the STAT command to assign a device to LST:, but this is unnecessary for upm users like yourself. The section on Connecting Printers has more information on using your printer with Micronix.

At this point, you are finished with normal installation of WordStar, and you will be asked if modifications are complete. We would like you to bravely state that they are not by typing N. This brings you to the PATCHER routine, which lets you select individual bytes to be changed. The following table contains a list of the bytes that you should patch for maximum efficiency:

LOCATION	OLD VALUE	NEW VALUE	COMMENTS
NMOFUS:	01	FF	Number of users
NMOFUS:+1	01	FF	Also number of users
DELCUS:	10	00	Cursor delay
DELMIS:	05	00	Cursor delay
DEL4:	64	00	Sign-on delay
DEL5:	09	00	Refresh delay

There may be other bytes that you wish to change at this time to personalize WordStar. You can find more information about this in section 8 of the Installation Manual for WordStar. When you are finished, enter 00 as the address to change.

Note to MD20 terminal users: Morrow's MD20 terminal almost emulates an ADM 31 terminal. (If you have chosen to use upm's terminal emulator, you should ignore this note). Select the ADM 31 terminal from the terminal menus. Then, while you are using the PATCHER, change the order in which the row and column are transmitted by WordStar with

CB4LFG:	00	FF	Column before line flag

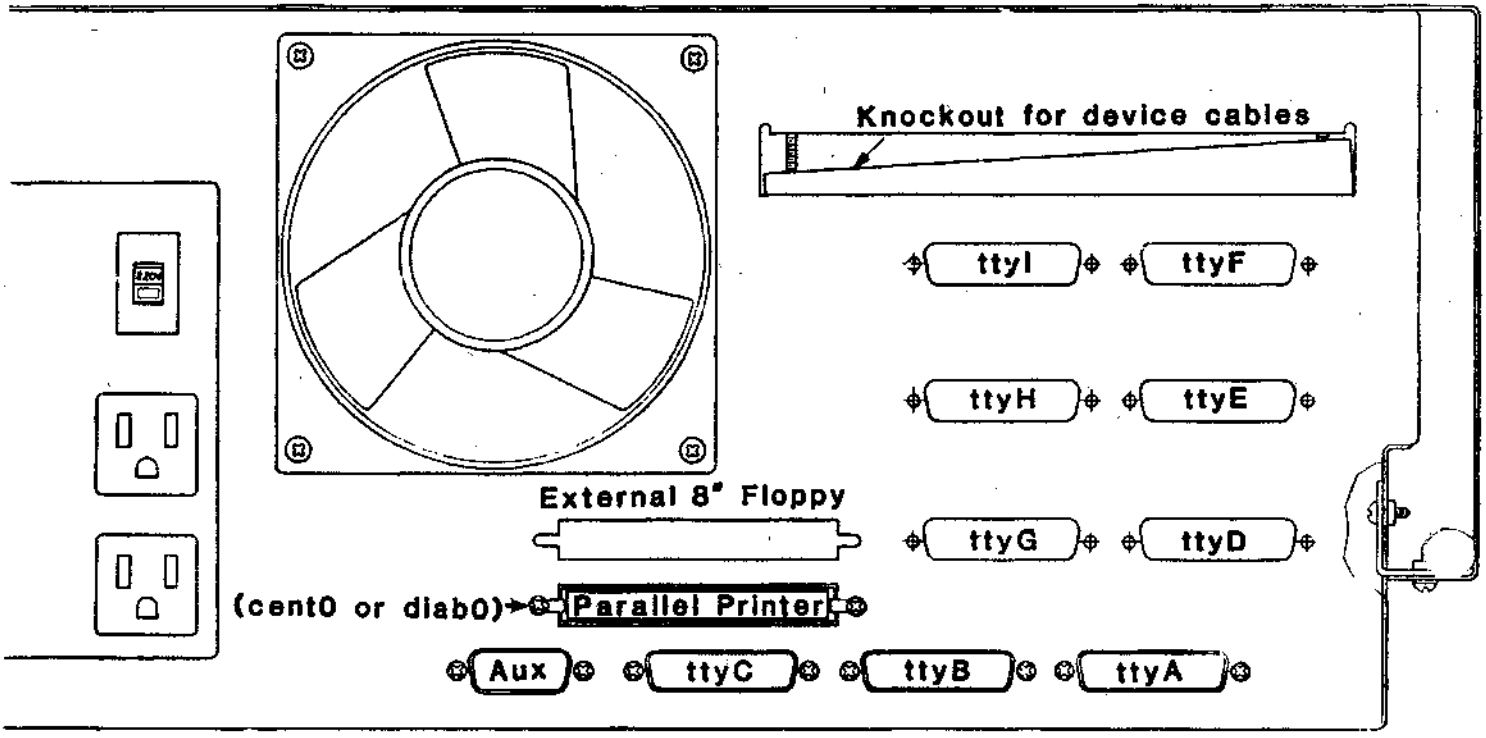
The MD20 will now work correctly with Word Star.

The changes made to WordStar utilizing the PATCHER disable most of the internal timing used by the program. Some of the best software available for CP/M was written to simulate multi-processing on single processing systems. WordStar is very good at this, but Micronix is, of course, better. If you have a program that allows two activities to occur simultaneously, such as typing and printing, or typing and screen rewriting, you'll want to disable, if possible, the internal timing loops that drive these functions.

Internal timing loops are short programs that run rapidly in a circle, simulating a stopwatch. The problem with this is you don't want to waste the power of your CPU by making it pretend to be a clock. Your Decision has a perfectly good clock in it already. Most of the timing delays required by a program will occur naturally as a result of multi-user Micronix's normal processing. If the removal of timing loop delays adversely affects the use of a program, put the needed one back. An example of a needed delay is DEL3: in WordStar. Without DEL3:, you will immediately be presented with a Help Screen whether you need it or not (unless you disable all Help).

BACK PANEL CONNECTIONS

(KEYSWITCH MODELS)



NOTES:

For systems without MULT/IO boards:
Connect your console terminal to ttyA. Connect your serial printer to ttyC. Connect an optional second terminal or printer ttyB.

For systems WITH MULT/IO board(s):
The only restriction on your configuration is that the console must be connected to ttyA.

Micronix Operating System
user's manual

maintenance &
administration

MORROW

MAINTENANCE AND ADMINISTRATION

TABLE OF CONTENTS

Appendices

L I S T O F F I G U R E S

MAINTENANCE AND ADMINISTRATION

This section of the Micronix User's Guide explains what you need to know and do to keep your system up and running. In most cases, we will tell you the complete command lines to use. In others, we will just describe what commands to use, and you will fill in some blanks to fit the command to your customized system.

Administering a multi-user system with a hard disk is not as simple as caring for a microcomputer with floppy disks. With the microcomputer, administration consists of backing up diskettes by copying them. But with a hard disk, you need about an hour plus twenty-five or so diskettes for each backup. Not a very practical solution. Don't get upset, because this section details an improved, systematic backup scheme that minimizes the need for time and media.

As system manager, you also have to concern yourself with creating new user accounts, removing old ones, keeping the hard disk from becoming full, adding new printers or programs, and managing the system in general. The file system itself is more complex, which is both a blessing and a potential problem.

Well, the good news is that the Micronix system comes with a complete set of the tools that you need for maintaining and administering your system. You can do a thorough job of administration with a handful of commands and fifteen minutes a day, with an additional half hour once a week. This will allow you to back up, check for free disk space, and maintain your file system's integrity with very little effort.

The bad news isn't really "bad". It is simply that there is no mechanism that will encourage you to maintain your file system. You can run your Micronix system without backups or checks, until something goes wrong. Then, you may lose several days' or several months' work. We can only try to reinforce the notion that an hour and a half per week spent on routine maintenance is vastly preferable to the aggravation of trying to replace lost files.

Maintenance and Administration covers the topics shown below, in that order.

- **A day in the life of a system administrator** - A fable to introduce you to life in the fast lane
- fsck - the file system check program, and what to do when it turns up something unpleasant
- df & du - the "disk free" and "disk usage" programs that help you manage disk space

- **Customizing the Environment** - adding and removing users, adding hardware, plus a garden of wonderful miscellaneous tips that exercise the finer points of Micronix to your benefit

- **System Security** - implementing passwords and protecting files

- **Copying Diskettes** - Formatting, copying with a single drive, "SYSGENing" a loading diskette, CP/M vs. Micronix file systems

- **Backing up and Restoring the Hard Disk** - step-by-step backup procedures, and how to use the backups when the need arises

- **Desperate Measures** - getting out of tight spots like forgetting the root password, freeing a locked-up terminal, and repairing a wounded root directory.

- **Console Error Messages** - These are messages that get sent to the console as warnings that something undesirable just happened or is about to happen to the system. You'll find the appropriate responses listed here as well as explanations of the messages. Not particularly fun reading, but handy.

1. A DAY IN THE LIFE OF A SYSTEM ADMINISTRATOR

Before you go delving into the intricacies of system maintenance and administration, we'd thought that you would appreciate sharing the experience of an experienced system administrator. So, we installed a hidden camera in the office of one our Beta-test sites, and filmed the entire day's activities. The version presented here was edited, of course, to the bare essentials, and all embarassing scenes removed. We fabricated names to protect the innocent, and to make it impossible for the subject to file an invasion of privacy suit.

Our experienced administrator's name is Clair. The company he works for has been using a Micronix system for over a year, and Clair has been responsible for maintaining the system every day of that period, except for his 2 week vacation at Lake Tahoe. We chose Clair for this study because he has never called and asked us to replace a lost data file that wasn't backed up, (a hopeless task). We are sorry that we can't give Clair the recognition that he truly deserves.

1.1. Turning on the System

Our saga begins with Clair walking into his office and sitting down at his desk. He swivels his chair to face the Decision cabinet, and takes a key out of his pocket. Clair inserts the key in the Decision, and turns it all the way to the right, so the Decision is now on and reset. The light on the floppy disk drive starts to flash, and Clair realizes that he needs to find the Micronix Load diskette.

Clair is a meticulous person, and manages to keep everything in its place. It's just that he's a little groggy this morning. He scratches his head, and looks on the table where the Decision sits for the Load diskette. And sure enough, there's a diskette with the word "LOAD" printed on the label in shaky letters. Clair slips the diskette into the drive, and closes the drive door just as the light flashes off. The next time the light turns on, the solenoid in the drive clicks and the Loading message appears on the console.

Clair turns back to his desk and takes another sip of coffee from a foam cup. In another half hour, he seems to be thinking, the guys in engineering will have a pot of strong French roast coffee going. (Clair drinks about three cups of coffee before his eyes are fully open most mornings.)

The hard disk has quit humming and whirring, so Clair swivels to face the console again, and logs in as "root". Clair smiles to himself while entering the password, as if the root password, known only to himself and one other person, has some special meaning for him. Then, he enters the correct time and date. According to his 10 function LCD diver's watch, it is exactly 8:17:31 on July 6.

```
# date July 6 8:17:55
Wed Jul 6, 1983 8:17:56
# []
```

So, Clair enters a date several seconds in the future, and waits to hit the return at the end of the **date** command until the time syncronizes perfectly with his watch. This isn't really necessary, but Clair likes to be picky about little things.

Next, Clair starts up the file system check program. He does this by simply typing

```
# fsck /dev/root
Checking /dev/root
  ...
```

and turning back to his desk. Clair knows that **fsck** takes about 5 minutes to complete, so he can get on with the business of the day. He looks first at this appointment calendar, sighs, gets up and walks out of the room.

1.2. Going Multi-user

The clock built into our hidden camera indicates that about 14 minutes have elasped since Clair wandered out of his office. He appears to be slightly flustered; must be a new secretary in the office. Clair glances at the console. Fsck has finished, and, as usual, there weren't any "victims". Good. Clair types the word "exit" after the prompt, sending Micronix into the business of becoming a multi-user system. As the hard disk begins to hum and whizz, he faces back to his desk, and reaches for the top of his "in" stack. Several minutes later, the Micronix banner and the log-in prompt (Name:) appear on Clair's terminal, and on the other terminals that have been turned on around the office. Micronix has "gone multi-user" and is ready for the day's work.

1.3. Routine Physical Maintenance

Several hours have passed since Clair began his workday. At this moment, Clair is returning to his office after lunch (we can tell from the mustard smeared on his tie). Before settling down at his desk, Clair elects to perform the physical maintenance necessary to keep the Decision in top working condition. He reaches around to the back of the Decision cabinet, and grabs something that is located right in the middle of the back.

As his hand comes back into view, you can see that he is holding a black square of foam rubber, a little dusty on one side. This is the fan filter, which sits in a holder outside the cabinet. Clair walks out of the room to the washroom. The fan filter is washed in lukewarm, mild soapy water, blotted dry, and allowed sit until it is completely dry.

Fifteen minutes later, Clair re-enters his office with the

fan filter in hand and a smile on his face. He seems to appreciate the extra goof off time that being system administrator provides him with. He reaches around to the back of the Decision, folds the foam filter in half, and fits it back into its enclosure.

Monthly maintenance completed, Clair goes back to work. The only other physical maintenance that can be done is to wipe off the cabinet with a damp cloth.

1.4 End of the Day Backup

As the end of day approaches, our camera discloses a tired and wired looking Clair: he really should do something about his caffeine addiction. Maybe acupuncture, or a few sessions with a hypnotist. Nevertheless, dependable Clair prepares to perform the daily backup ritual. He reaches for a couple of formatted floppy diskettes, inserts the first one into the floppy disk drive, and closes the door.

Clair has been logged in as an ordinary user all day, so he first needs to become the superuser, root. He uses the su (switch-user) program to change his user identity.

```
% su
Password:
# []
```

Now, he has the power to access any file or directory on the system.

Clair has already checked to see that all the other persons using the system have logged off with the who command. If he were to backup a file while someone was writing to it, that file would not be correctly backed up. It isn't really necessary that everyone be logged off, just so no one change any file during the backup process. Then, he types

```
# fp cril /
Enter volume name:  System backup by Clair W.
   ...
# []
```

Clair knows that those letters after the fp command will create a new fp volume, replace those files that have been modified since the last backup (i) and request a label for the volume. Fp runs pretty fast, but Clair is getting a little impatient to go home.

The backup process itself is automatic, except for inserting diskettes and pressing return. Clair appears to be grumbling about something, but he has conscientiously backed up the system for over a year now without complaining to anyone. Maybe if he were to share the root password with someone besides that secretary, he wouldn't need to leave so late every night.

When the backup program finishes, Clair carefully labels the diskette (it only required one today) with the day's date, puts it in the bottom drawer of his desk, and carefully locks his desk.

1.5. Turning the System Off

Well, that's it for today. Clair, still logged in as the superuser after backing up, types

```
# down

Micronix is down[]
```

to bring down the operating system gracefully. Then, he turns the key on the front of the Decision to the left to turn off the power, removes the key, and sticks it in his pocket. Another day, another dollar. Clair gets up, and moves out of view.

2. CHECKING THE FILE SYSTEM

By now, you are aware that the file system is both the organization of data and the data itself on a hard disk or diskette. Assuming 100% performance of hardware, software, and the personnel using them, your file system will last forever, and never have a problem. In the real world, the people, hardware, and software operate at perhaps 95% of perfection. This means that there will be occasions that create problems for your file systems.

2.1. Things NOT To Do To Your File System

These are the things that are most likely to give your file system a headache:

1. Resetting the system without first giving the **down** command
2. Interrupting power to the computer or the disk drive
3. Copying a file to a "/dev/" disk device file that contains a file system
4. Hardware problems within the disk drive

It is a good thing to be able to recognize a hardware problem when you see one, because running fsck on faulty hardware may do more harm than good. The symptoms produced by the first

two improprieties above are: the computer will be off (no power), in a halt state (you can't get it do anything, even though it has power), or it will be in a typical post-reset condition.

The third action results in a "not a file system" error message whenever you try to mount the device. Copying or redirecting data directly to a /dev file can be disastrous, but usually hard disks are protected from such a snafu attempted by anyone but the superuser (root). If you copy to a floppy /dev file, with or without a file system on it, the diskette becomes worthless (of course, you can always reformat it and use it again).

The fourth difficulty, hardware problems, are the most difficult to deal with. A hardware problem may raise its ugly head by not allowing you to boot from the disk properly. Or, it may show up during operation. The console terminal may become flooded with read or write error messages for the disk drive (please refer to the Console Error Messages section for a translation of the messages). So, if you can't boot, or if you suddenly get a jump in the frequency of read/write error messages, suspect your hardware and have it fixed BEFORE running fsck. Otherwise, fsck will be zipping around trying to fix nonexistent problems.

2.2. FSCK: Tender Loving Care for Your File System

The Micronix file system is an elegant beast. Therefore, rather than teaching you to be a Micronix systems programmer, and forcing you to hack away at blocks on the disk, we have provided you with fsck, your "systems programmer program". Fsck, which stands for file system check program, takes advantage of the cross-referencing inherent to Micronix for checking the system and repairing it whenever possible.

Fsck also looks for logical inconsistencies in a file system, such as blocks that are claimed by two or more files. Fsck settles conflicts over multiply-claimed blocks by making a copy of the block and giving it to all the files that claim it. The names of the (possibly) affected files are reported to you, so that you can take the appropriate responses. The section named Fsck Victims explains how to handle your damaged files.

You can use fsck on Micronix file systems on both hard and floppy disks. Fsck will perform identically on each, taking much longer with hard disks because of their greater storage capacity. The next sections tell you how to use fsck.

2.3. Before Using Fsck: Inactive File Systems

There is a very important pre-condition for using fsck: the file system you use it on must not be active. Active file systems are being used, that is, they have files "opened" by a program for reading or writing. You can be sure that a file

system is inactive by unmounting it, or checking it before it is mounted. (See **mount** in the Programs section of the Micronix Reference Manual, and Section 8.4 of Maintenance.) Immediately after resetting your Micronix system, there are no mounted file systems. This makes the time immediately after a reset an excellent time for running fsck.

If you are not certain which file systems are mounted, you can use the mount command to see what's mounted.

```
# mount
ml6b on /b
# []
```

tells you that "ml6b" is mounted on directory /b. So, if you want to check device ml6b, you must first unmount it by typing

```
# cd /
# umount ml6b
# []
```

If someone is using the file system, the umount command will tell you "device busy". A device is busy when files on the device are opened by a program, or someone has moved his current directory to somewhere on the device.

The root file system is an **exception to the rule** about checking inactive file systems. The root file system is always active when Micronix is running. You can use fsck on it when no one else is running programs, i.e., when you are in single-user mode after resetting the system. Fsck can fix most problems on the root file system. However, problems with the root directory (/) will be recurrent: fsck will fix them, but they'll come back. The solution to this dilemma is found in the section called Desperate Measures. (You will need to repair the root file system from the Standalone Micronix floppy.)

2.4. Using Fsck

The fsck command expects the name of a disk drive device as an argument. The disk in the drive, of course, must contain a file system to check. Disk device names are listed in the /dev directory. To make things simpler for you, a partial listing of disk device names and descriptions of the devices is included here:

/dev/root	always the name of the root device (virtually always your internal hard disk)
/dev/ml6a	the name of the first 16 megabyte mini-winchester hard disk device (# 0, which may be the same as your root device)
/dev/ml6b	the name of the second 16 megabyte mini-winchester hard disk device (# 1)

/dev/ml0a	the first 10 megabyte drive (# 0)
/dev/hdca	the first drive using the HDCA controller
/dev/mfa	the name of the first 5 1/4" (mini) floppy drive (# 0)
/dev/mfb	the second 5 1/4" drive (# 1)
/dev/fla	the name of the first 8" floppy drive (# 0)
/dev/flb	the second 8" floppy drive (# 1)

These are the device names that you will most often be using with fsck. There are also three options available with fsck, -t, -n, and -b:

-t fsck test-reads every block on the device

-n fsck reports problems, but doesn't fix anything

-b fsck will take the list of block numbers that follows the b flag and add these blocks to files in /badblocks, effectively removing them from the system.

2.4.1. The -t Option

Test-reading all the blocks on a hard disk might take as long as half an hour (typically 20 minutes for 16 megabytes). Generally speaking, you should run fsck with the -t option once a week at least, and more frequently if you start getting read/write error messages on the console. Running fsck with -t on a floppy disk is fairly quick, and a good way to check floppy disks.

2.4.2. The -n Option

The -n option will report the problems it finds in a file system without doing anything about them. This is useful for surveying the extent of the damage to a file system. Generally speaking, you want to fix a file system, not just "talk" about fixing it, so you won't be using the -n option very often.

2.4.3. The -b Option

The -b option is designed for use with virgin file systems on hard disks. Hard disks come from the factory with a known and acceptable percentage of unusable areas (badspots), which are kept in a special map so they don't get used for reading and writing. When you create a new file system on a hard disk, the information in the hard disk's badspot map must be added to the file system. The badspots program can read this map, and create a list suitable for the -b option. Here's how you would use the

-b option with badspots on a new file system on your third 10 megabyte hard disk:

```
# fsck -b `badspots /dev/m10c` /dev/m10c
Checking device m10c
...

# []
```

2.5. EXAMPLE of Using FSCK

Let's take a look at an example of using fsck. On this occasion, we were checking a file system on a 5 1/4" diskette in the first drive, with the -t option:

```
# fsck -t /dev/mfa
Checking /dev/mfa:
Hunting for bad blocks
** Checking I-list, first pass
** Checking the free list
** Checking I-list, second pass
** Checking connectivity
3 files, 0 special, 1 directories, 2 small, 1 large, 0 huge
760 blocks, 17 I-list, 1 indir, 0 in2dir, 203 data
204 used, 537 free, 0 bad
# []
```

After displaying the message "Hunting for bad blocks", fsck proceeded to read every sector on the disk. If it had found an unreadable sector, it would have reported it, and marked it for attempted repair by a later phase of fsck. For this double-sided floppy, the test took about 1 minute.

The lines beginning with "**" mark the four normal phases of fsck. Each phase looks at the selected file system in a different way. Any problems discovered by fsck are reported on the terminal and marked by fsck for repair. Whenever fsck does something that affects a file, it makes an entry in the file **fsck.victims**. The next section tells you how to deal with the victims.

Here are a few more examples of fsck. Each of the following example commands is described in the right hand column:

fsck /dev/hda	test the first hard disk using the HDCA
fsck /dev/root	checks the root file system
fsck /dev/f1a	examines a disk in the first 8" drive
fsck /dev/mfb	looks at a disk in the second 5 1/4" drive

If you are curious about what fsck actually does, you can find out more by reading about fsck victims.

2.6. Fsck Victims

It's Monday morning, and, as part of your well-established routine, you are running the fsck program right after turning on the Decision and resetting Micronix. As your eyes begin to focus, you notice that there are signs of a party in the computer room; the empty wine bottles certainly weren't lying about when you left on Friday afternoon. This might be cause for concern, if the revelers neglected to shut down the system properly.

Unfortunately, this appears to be the case. As fsck proceeds through the file system, it uncovers and reports on several problems. You wait for the fsck program to complete its checks. After the usual summation of files and block usage, the lines

```
** Hunting up filenames of casualties.
A copy of this list will be saved in the file "fsck.victims"
```

appear, followed by a list of the affected files. As the "system administrator", your work is cut out for you. You need to salvage as many of the files and directories in the victims list as possible. In some cases, total recovery is possible. In others, the files will have to be restored from backups if possible, or an obituary sent to the owner, if not.

Fsck has done its best to fix your file system; it has located the source and nature of problems and handled them in such a way as to make the system usable without further damage. There is still some detective work involved, though, that requires your vastly superior human intellect.

The first step in recovering victims is to print out a list of the victims. This will save you some time, and provide a space for making notes about the victims. Fsck creates the file **fsck.victims** in whatever directory you happen to be in when you execute fsck. Send the file to the lpr daemon,

```
# lpr -i fsck.victims
# []
```

and get the listing from the printer. The listing we are using looks like this:

```
/dev/root:
a/david/init/exec.bak: File's data possibly corrupt
a/len/Util/Edit/YY: Possibly truncated
lost+found/dir342: Orphaned directory
lost+found/file1329: Orphaned file
a/dir261: Directory's name changed
a/rik/ws/invent: File's data lost
a/jayne/Euro/file2284: File's name changed
```

What luck! This printout provides us with one example of each of the 7 possible types of fsck victims. Notice that the filenames

don't start with a slash (for the root directory); the filenames in the victims file will begin with the names of the subdirectories of the root file system being checked.

2.6.1. Victim 1: File's data possibly corrupt

A file's data is corrupted if a Read Error occured while reading a file block when using fsck with the -t option. Fsck does its best to copy the unreadable block to a new block, for the sake of avoiding further corruption. You'll have to look at the contents of the file to see how much of the data was successfully copied. If the file isn't a textfile, you may want to restore it from a backup copy just to be safe.

A second way that a file's data may become corrupt is because of a duplicate block. If you recall, fsck checks to see that each block in a file system is used only once. Blocks that are used more than once are called duplicate blocks. When more than one file claims possession of the same block, all files are given a copy of the block, but only one file will be correct (the other files will have a strange 512-byte segment in them that just doesn't belong). Let's look at the message from our example victims file:

```
a/david/init/exec.bak: File's data possibly corrupt
```

Right off, the filename provides us with some important information. The file with corrupt data is in a subdirectory of /a/david. The /a/david directory is the home directory of the user named "david". So, the user david would be the person who decides whether the file is really corrupt or not.

There is a second clue in this name that can help. The filename ends in ".bak", suggesting that it is a backup file. Let's look in the /a/david/init directory and see if there is another "exec" file:

```
# ls -l /a/david/init/exec*
rw-rw-rw-   1 david    605 Mar  1 13:12 /a/david/init/exec.bak
rw-rw-rw-   1 david    609 Mar  3  9:21 /a/david/init/exec.c
rw-rw-rw-   1 david    639 Mar  3  9:22 /a/david/init/exec.o
# []
```

The "ls -l" program produces long directory listings of the files beginning with "exec". You can see that there is a more recent (March 3) version of "exec.c", so having corrupt data in the exec.bak file probably doesn't hurt anything.

If the file with corrupt data was very important, for example, a list of yesterday's financial transactions, it should be recovered from the last backup that (we hope) you made. Suppose the corrupted file was named "a/BoA/accounts/receiv". To recover this file, take the diskette (we will assume 5 1/4" for this example) from last night's backup and insert it in your first floppy drive. Then type

```
# mount mfa /f
# cp /f/BoA/accounts/receiv /a/BoA/accounts
# umount mfa
# []
```

which mounts (attaches) the diskette in directory /f, copies the file "receiv" to the proper directory, and unmounts the diskette.

If you have several diskettes from yesterday's backup, or the file was backed up on a different day, you may have to repeat this process (mount, cp, umount) until you find the diskette with the file on it. If the file is not on the diskette, the message "No such file or directory" will be displayed on the line after the cp command. Unmount the diskette and repeat this sequence of commands with other backups until you are successful. May your conscience take note: **Careful labeling of your backups is always worthwhile.**

2.6.2. Victim 2: Possibly truncated

Fsck has two ways of telling how long a file is. First, there is a record for the file that says how many bytes are in it. Then there is the list of blocks claimed by the file, which implies a certain length in bytes, at 512 bytes per block. If these two figures disagree, fsck proceeds on the assumption that the size record is more reliable than the block list. Thus the blocks that go beyond the size record get chopped off, or truncated. Hence the message "possibly truncated". Once again, the person who owns the file is in the best position to decide if the file has been truncated. Let's look at our example message:

```
a/len/Util/Edit/YY: Possibly truncated
```

The file in question, "YY", is in a subdirectory of the user "len". You, as the system administrator, may decide to attempt to replace the file "YY" from backups. Or, you can talk to "len" and tell him that his file has been truncated. If "len" isn't at work yet, use the mail command to send him a note:

```
# mail len
Len - The file Util/Edit/YY was reported as Possibly
truncated this a.m. If you need to recover it from backups,
see me later.    -- Rik
.
# []
```

A note like this in the owner's mailbox is appropriate for almost any action you might take with someone else's files. The file's owner may have a more recent backup (made Saturday before the party got rolling), for example. If you replaced "YY" from the backup made on Friday, "len" would lose all the changes made on Saturday, and possibly not notice it. Always be considerate of other users. Communicate with them before copying over their files.

If Len wants "YY" restored from backups, follow the same sequence of instructions (mount, cp, umount) as in Victim 1.

2.6.3. Victim 3: Orphaned directory

In the "real" world, an orphan is a parentless child. In the world of Micronix, an orphaned directory has no name and doesn't know who its parent directory is. Normally, there are two ways a directory is connected: by its name appearing in another directory (the parent directory), and by a special entry in the directory itself, known as dot-dot (..). Orphaned directories are somehow missing both of these connections.

Because the name of the directory is not known, fsck constructs a name for it. The name is made by concatenating the letters "dir" with a number (the "inode number", but you don't need to know that). And, because the orphaned directory has no parent, it is given a "foster" parent, the **/lost+found** directory. Let's look at our example victim:

```
lost+found/dir342: Orphaned directory
```

You can see that the orphaned directory is indeed now a "child" of lost+found, and it has a name, dir342, made from the letters "dir" and a number. After changing to the directory lost+found, we'll use the **owner** command to see whose directory it is:

```
# cd /lost+found
# owner dir342
     bob   dir342
# mv dir342 /a/bob
# mail bob
Hi Bob - Look's like you're missing a directory: fsck found
a directory of your's loose in the system.  It has been
renamed /a/bob/dir342.  Use mv to get the name right again.
- Rik
.
# []
```

As the system administrator, we moved dir342 to Bob's home directory, /a/bob. When Bob reads his mail, he should use the dir command to examine this directory. Bob would type

```
% dir dir342
Exec       Func       Util
% []
```

to discover what's in this directory. Then he would use a mv (move) command to rename it, and coincidently replace it in its proper position in the file system:

```
% mv dir342 Prog
% []
```

For this victim, the solution was simple. Bob recognized the orphaned directory as his Prog subdirectory, and moved it back to where it belonged. The three subdirectories of Prog, (Exec, Func and Util), and all their files and subdirectories were unharmed by the experiance of having an orphaned ancestor.

2.6.4. Victim 4: Orphaned file

An orphaned file is a file without a name in any directory. In Micronix parlance, this file has no parent directory; thus, it is an orphan. Our victims list provides us with an example to play with:

```
lost+found/file1329: Orphaned file
```

Obviously, we need to discover the name of the orphaned file. A good place to start is to get the name of the file's owner by using

```
# ls -l file1329
   rwxrwxrwx   1 len              768 Jan  6 12:25 file1329
# []
```

We are still in the /lost+found directory after helping Bob. The **ls -l** command shows that the orphan's owner is Len, and that the file has read, write and execute permission, is 768 bytes long, and was last modified Jan 6, at 12:25. Now we have some clues about the mysterious file1329. The **file** command can tell us more:

```
# file file1329
file1329: Text file
# []
```

File1329 is a text file, so it should be relatively safe to "type" it out. This is a task for Len, since it is his file, and he should be able to recognize it. If Len recognizes his file, he should use mv to restore it to its old name. As it happens, Len recognized file1329 as a "makefile" for some networking software he was writing. To get file1329 back in its place, Len typed

```
% mv /lost+found/file1329 /a/len/Util/Net/makefile
% []
```

changing its name and moving it simultaneously. If file1329 had been unidentifiable, it could have been removed by typing

```
# rm file1329
# []
```

while still in the lost+found directory.

2.6.5. Victim 5: Directory's name changed

A directory with a name-change is less serious than an orphaned directory, and easier to repair. A directory's name changed because it either had an illegal name (the slash character, for example), or its name didn't appear in its parent directory. The name-changed directory is still in its place in the file system because it has maintained its link to the parent directory (the dot-dot, .., directory entry); the problem is that it has lost its name.

Fsck handles this problem by making up a name for the name-changed directory and entering it in the parent directory. The names used are constructed by adding a number to the letters "dir". Let's take a look at our victims file example:

```
a/dir261: Directory's name changed
```

You can see that the directory's name has been changed to "dir261". And, more important, the directory dir261 is not in the /lost+found directory, but is a subdirectory of /a. So, we know where dir261 belongs: it's still in place. All we need to do is discover its rightful name. The owner command will give us the information we need:

```
# owner dir261
     johnv  dir261
# []
```

This is the point where some familiarity with your file system will come in handy. Dir261 is owned by the user johnv, and is a subdirectory of /a. The /a directory is where users' home directories are created. It takes a small inductive leap to guess that dir261 is actually johnv's home directory. To change the name of dir261 to johnv, simply use

```
# mv /a/dir261 /a/johnv
# []
```

If you can't determine the true name of the changed-name directory, you should send some "mail" to the owner (or talk to him) about the problem.

```
# mail johnv
John - One of your directories has lost its name:  it is
temporarily named /a/dir261... Rik
.
# []
```

Because dir261 IS John's home directory, John will be unable to log-in to get his mail. What will happen is this:

```
Name: johnv
Password:
Last logged in on  ttyD, May 24  15:53
```

```
/a/johnv: No such file or directory
Name: []
```

John won't be able to log-in until you, as the root, change (mv) /a/dir261 to /a/johnv so that John can log-in.

2.6.6. Victim 6: File's data lost

This is a nasty one. File's data lost means exactly that there is a filename without any data to go with it. All that is left of the file is it's name, and, hopefully, some backups of it.

```
a/rik/ws/invent: File's data lost
```

Our sample file is called /a/rik/ws/invent. To restore it from a backup, we first find the diskette with the most recent backup on it, insert it in a floppy drive, and type (for 8" drives)

```
# mount fla /f
# cp /f/rik/ws/invent /a/rik/ws
# umount fla
# []
```

If you don't have a backup of a file whose data is lost, you simply inform Rik, so he can type

```
% rm /a/rik/ws/invent
% []
```

and remove the file's name, which is all that is left.

2.6.7. Victim 7: File's name changed

A file's name is changed because it contains an illegal character in it. The best example of an illegal character in a filename is the slash (/). If a filename has a slash in it, you won't be able to access it because the slash is the divider between directory names. Micronix would see the slash in your filename and assume a directory name preceeds the slash.

Fsck builds a new name to replace a defective one by adding a number to the word "file". (The number is the inode number, if you are interested). Let's examine our victim:

```
a/jayne/Euro/file1221: File's name changed
```

From the filename, we can see that file1221 is in a subdirectory of Jayne's home directory. The correct course of action here is to send Jayne some mail, or talk with her about file1221 having had its name changed. When Jayne remembers, or discovers, the file's name, for example, "clients", she can change it by typing

```
% mv Euro/file1221 Euro/clients
% []
```

from her home directory.

2.6.8. Disaster Victims

This wouldn't be a complete discussion of fsck.victims unless we pointed out to you that your list of victims won't be like our sample list. What is more likely is that you will either have one or two victims with corrupt data or changed names, or a veritable flood of orphaned files and directories. When you have several victims, the techniques outlined here will be of great help. However, if half your file system has been ravaged, other methods are in order.

An incident resulting in a large number of fsck.victims is likely related to a hardware malfunction. If this malfunction were to reoccur just after you finish rebuilding your file system, imagine how upset you'd be. So, instead of leaping into the fray, we suggest that you check out your hardware first.

The easiest thing that you can do is reconstruct your system from scratch, if you have been keeping good backups. The instructions for doing this are in the Installation part of this Guide. Start by formatting your hard disk, which will simultaneously check out your hardware and map out unusable blocks. If you have trouble formatting, you definitely have a hardware problem, and should call your Service Representative.

After formatting, follow the instructions for Installing Micronix on the Hard Disk, and Adding the Software. The **finstall** script can be used to add your backups to the system. To restore your backups, start with the oldest backup diskette, and continue until you have restored the most recent backup.

2.7. Other System Checks - dcheck, icheck and ncheck

Although fsck offers the simplest and most comprehensive check, and is the recommended program to use, three other programs - dcheck, icheck and ncheck- are also available to check and maintain the system. These programs are mentioned for completeness only, since fsck performs the same checks as these programs. Information on these programs is contained in the Reference Manual in the Programs section.

3. MAINTAINING FREE DISK SPACE

Keeping your hard disk from filling up is possibly the hardest task in maintaining your Micronix system. Creating files, copies of files, backup files, revisions of files, temporary files, archives, etc., is so easy to do, you will be surprised at how easy it is to fill up 10 or 20 megabytes of storage. You can use the df (disk free) command to monitor the amount of free space remaining in a file system.

The place to start in keeping some free space is with each user doing their part. Users need to remove temporary files or directories, and move long-term backups to floppy disks. It may be convenient to have every revision of a program or text around, but it sure will fill up a hard disk in a hurry. A Micronix system with a single user might go for a couple of months before running out of space. A system with several users might last only three weeks.

When users prove unable to control their use of disk space, it is up to the system administrator to provide some assistance. You can do this by using the du command to monitor each user's disk usage. You can send mail to the users that are hogging disk space, asking them to clean up their act. If this fails, you can try stronger methods, such as public humiliation (list the space hoggers in the message of the day).

You may actually need more disk space than your present hard disk provides. The way you can determine this is to calculate the amount of space used by the resident programs and their support files, for example, the data files in a database. The du command (explained in the next section) will calculate this for you. If the programs and files shared by all users take up most of the hard disk, there will be no space for user files. The solution is to move shared data files to a new hard disk, keeping the users on the original disk. Making an additional hard disk a part of the file system is explained in the section on the Routine Commands file, mounting commands.

3.1. Checking Disk Usage: du

The disk usage command (du) counts the number of blocks used by files in a directory, including all the files in subdirectories. Remember that a block is 512 bytes, or 1/2 k of disk space. So, if a directory uses 2000 blocks, it is taking up one megabyte of disk space.

The Micronix file system is organized so that the user directories are all subdirecties of /a. This makes it simple to locate each user's home directory and summarize the portion of the file system that they are using. We should point out that any file can grow until it has used all the free space in the system. There aren't any limitations put on programs or users that prevents them from overindulging in disk space use. Let's look at a file system that is shared by programmers and text writers and see how the du command works:

```
# du -s /a/* .
    1 /a/alliene
  403 /a/cpm
 1589 /a/david
 1003 /a/frank
    4 /a/howard
    1 /a/john
 7216 /a/len
   12 /a/marc
   35 /a/mike
    3 /a/network
   82 /a/norm
 1136 /a/rik
# []
```

A glance at this list uncovers the major disk users: David, Frank, Len and Rik. If disk space has become (or will soon become) an issue, a pleasant note can be mailed to these users requesting that they be more considerate, and use floppy disks for backups. You might guess that for this command to work, shared data space, i.e., databases, would have to be kept in a place other than a subdirectory of a user.

The du command compiles disk usage reports for the files or directories listed after the command and options. Du recognizes two options,

-s Summarize disk usage for names specified, or

-a All files and subdirectories are reported.

If neither option is used, a summary of disk usage by directory is produced.

Individual users can benefit by using the du command. For example, Len could discover where he is using all that disk space by typing "du -a /a/len", which would list out every file and directory connected through his home directory with the blocks used.

3.2. Disk Free Command

The disk free command (df) is simple, quick and easy to use. Compared to the trouble you will be in if you don't have any free blocks left to use in a file system, you should use df every day, if not more often.

The disk free command counts the number of free blocks remaining in a file system. This makes df the exact opposite of du, which counts the blocks in use. To use df, you simply specify the names of the file systems that you want to check. For example,

```
# df /dev/root
```

```
   8117 /dev/root
# []
```

informs you that there are 8117 free blocks remaining in the root file system. Keeping in mind that a block is 512 bytes long, 8117 blocks corresponds to 4.0585 megabytes remaining on root, a healthy amount.

How much free space is necessary? Well, that depends a lot on your daily use, and how many people are using the system. A nice rule of thumb would be to maintain 1 megabyte, that is 2000 blocks, of free space available on each hard disk file system. If you ever get down to only 100 blocks of free space remaining, you are in imminent danger of running out of free space. The solution is to clean up the file system by convincing users to remove unused files or copy old files to backups. The **find** command can be used to find old or unused (not recently accessed) files.

3.3. Finding Types of Files: find

The find command is the system administrator's, and user's, most useful tool for uncovering unused files. The find command uses information that is part of the description of files (the inode) to select files. For example, find can be used to print a list of the files that haven't been modified (written to) in 90 days, as in

```
# find /a/jan -mtime +90 -print
/a/jan/Pilot/stat.c
/a/jan/Pilot/global.c
/a/jan/Foreign/term.h
/a/jan/Foreign/bio.c
/a/jan/Foreign/cio.c
# []
```

which discloses that Jan has some files that haven't been modified in more than 90 days. Files such as this (unmodified for 90 days) are prime candidates for removal to a backup diskette.

Another use of the find command is to seek out and remove files in your own user directories. For example, some wordprocessors create files with the extension ".bak" every time they are used to edit a file. Eventually, this means that you will have two copies of every file you have ever edited. The find command can locate and remove these files, as in

```
% find . -name "*.bak" -exec "rm {};"
% []
```

which starts in your current directory, and searches it and all its subdirectories removing files with the .bak extension. As a system administrator, you could use this command to free enormous amounts of disk space in the entire file system. However, many

people will not take kindly to your removal of their files. Please save despotic measures (removing other people's files without their permission) for situations when reason has not prevailed.

Here's one more idea for using find. Whenever a program bombs, an image of the memory it is using, and a copy of the process status, is saved in in the current directory in a file named "core". Since not everyone appreciates just what a core file is, (it is used for debugging purposes), they may remain in the file system, unused, forever. You may want to search out "core" files that are over a week long by typing

```
# find /a -name core -atime +7 -print
/a/len/Edit/YY/core
# []
```

This time, we found only one "core" file over a week old. The next thing to do is send Len some mail asking him (politely) about the file.

We have only touched upon the power of the find command. If you want to learn more about find, check out the entry for find in the reference manual, under Programs.

4. USERS

Users aren't the people who use Micronix; users are the names that people log-in with. This may seem somewhat confusing, but "user names" are all that a poor computer knows about. A person logging in as the "root", for example, becomes the root for the computer, regardless of the person's true identity. The computer's only communications with you are the keys that you type on the keyboard, so typing "root" as your user name makes you the root.

Files and directories are owned by users. Thus, logging in as a particular user makes you the owner of that user's files. The "user" system is the basis for "security" on the Micronix system. And, even if you don't desire any security, having each person log-in as a unique user helps to organize the file system and provides more information about the files in it.

The information about user names is kept in the file called **/etc/passwd**. Each line in this file is called an account. An account describes between five and seven characteristics of a user name:

1	the user name
2	an encrypted password
3,4	user and group identification numbers
5	space for a descriptive comment
6	the home directory
7	the program to execute after log-in

Here is a typical account from the /etc/passwd file. Notice that colons (:) are used to separate characteristics.

```
sandy:5ui7i9W5pHdj:21:3:Sandy E. 555-1212:/a/sandy:/bin/sh
```

The **user name** is the same as the name entered during log-in. The encrypted password is a disguised form of the password the person must know to log-in as that user. All users should have passwords if you desire system security, but they are not mandatory. It is possible for some users to have passwords while others do not.

The **user and group identification numbers** are used in file descriptors (inodes) as a shortened form of the user or group name. Programs that include user names in their output, like owner and ls, use the /etc/passwd file to convert the user identification number to a user name.

After the identification numbers, a space is provided for a comment on the user name. This is often the real name of the user, and their phone number. Comments are optional.

When you skip an optional field, there should be two pairs of colons side-by-side with nothing in between them. A /etc/passwd entry with no comment entry would look like

```
sandy:5ui7i9W5pHdj:21:3::/a/sandy:/bin/sh
```

Users are placed in their **home directory** after logging in. The home directory is also established as part of the information about a user. For example, using the cd (change directory) command without an argument moves the user to their home directory.

The last characteristic of a user is the name of a **program to execute after log-in** is complete. Usually, this is "the shell", a program that interprets commands typed by the user. It is also possible to use other programs instead of the shell, for example, a word processing program. Having a log-in program other than the shell restricts that user to that program. The user restricted in this manner logs out automatically by quitting the program. More on this in the section on changing the accounts.

Micronix initially has only one user: the root, or superuser. Although it is possible to use Micronix with only the root user name, this is not recommended. The root user is only intended for use by the system administrator for backups and maintenance. An inexperienced root user can swiftly destroy the entire file system with a simple command (which we'll tell you about later). To prevent the accidental destruction of your carefully configured Micronix system, you and your associates should always use their own user names. The root user name should be reserved for system administration tasks.

How do you create, change, or remove users? Well, it's almost easier done than said. The account command does all the tricky work; all you need to supply is the user name. One thing we need to mention: only the root user can use the account program.

4.1. Adding New Users: account

Before using the account command, you should select a user name. User names can be up to eight letters long. Since this is the name used for receiving mail, it is best to pick a name that will be recognized by other people. Also, the name should be entirely in lower case letters. Let's start up the account program and add the user jayne.

```
# account

        A) add an account
        B) delete an account
        C) change an account
        D) short display
        E) long display
        F) finish
    <RUB> abort

Selection: a
User name: jayne
Account added.
```

As you can see, adding a new user required that you make two entries: you selected "a" (add an account), and you typed the name "jayne". That's all there is to it. What the account program does for you is make an entry in the /etc/passwd file for the new user, create a home directory attached to /a, change the ownership of the new directory (/a/jayne) from root to jayne and set the log-in shell to be the standard Micronix shell.

If you want to have some security on your system, you will want to assign a password to the new user. You can either use the passwd command, or select "c", change an account.

Immediately, after the line "Account added" appeared, the main menu (choices A to F) was redisplayed. You can choose to make more accounts, or change existing ones. When you are finished, select "f" to finish. The changes that you are making won't take effect until you type "f". If you interrupt the account program by typing DELETE or RUBOUT, none of the changes will be made.

4.2. Removing a User

Removing a user from the system is a two step process. The first step is to backup and remove all the user's files. The second step is to use the account program to remove the user.

You want to remove all the user's files from the system just to prevent wasted space. Whether you want to back the files up or not is your decision to make. The following commands will help you backup a user's home directory, along with any subdirectories. Let's assume that the user Jayne has moved on to greener pastures, and you want to backup her files in case she ever comes back. What you do is type

```
# find /a/jayne -name "*.bak" -exec "rm {};"
# td /a/jayne /dev/mfa
Insert new media and press RETURN to continue
or type the name of a new device containing a file system.
          ->
Function complete.
# []
```

which removes all the ".bak" files, and then copies all Jayne's files to a 5 1/4" floppy diskette with a file system on it. (If there isn't a file system on the floppy disk, td will tell you and offer to make one for you.) Now, you are safely backed up. You can remove Jayne's files by typing

```
# rm -r /a/jayne
Last chance before obliterating jayne.  OK? (Y/N)y
# []
```

The rm command with the -r option is very potent. This is the simple command that can destroy an entire file system if used carelessly (or maliciously). To prevent accidental damage to your file systems, don't log-in as root unless you need to, and only share the root password with people you trust.

Now, for the second step. Use the account program and select "b", for "delete an account". A list of the current users will be displayed, and you will be prompted for a "User name:". Simply type "jayne", select "f" to finish, and you're done.

4.3. Changing Accounts

There are four characteristics of accounts that you can change with the account command:

the user name,
the password,
the home directory, and
the shell, or log-in program.

You will use the change function to add passwords to previously unprotected users, or to change the default home directories and login shells that are created when you add accounts.

Changing the user name is the simplest change operation. The account program will request the current user name, then the new user name, and you're done. From now on, the new user name must be entered at log-in, and will appear in output as the name of the owner of files. The home directory is not changed.

Changing the password is almost as simple. Once again, you will need to supply the name of the account you wish to change. Then, you will be asked to provide a password. While you enter the password, the screen display (echo) will be turned off. Since you won't be able to see what you are typing, you will be asked to enter the password a second time to double-check what you think you typed. The password is then encrypted for entry in the /etc/passwd file. (The **passwd** command can also be used to enter or change passwords. Non-root users can use passwd to change their own passwords.)

The next characteristic that you can change is the home directory. Say, for example, you add a new hard disk mounted in the root directory as /b. You want to move the home directory of user "becca" from /a/becca to /b/becca. The first thing you'll need to do is create the directory on the b drive. (**account** automatically creates home directories on the a drive, so normally this particular step isn't needed.)

Assuming that the new hard disk has been formatted (**formatmw**), a file system put on it (**mkfs**), and it is **mounted**, use the "make directory" command (**mkdir**) to create becca's new home directory:

```
# mkdir /b/becca
# []
```

You're not done yet, though. You still need to use the **account** program to tell Micronix to always log becca into that newly-created directory. You also should "give" the ownership of the directory to the user by the **owner** command:

```
# owner -becca /b/becca
# []
```

If the directory originally created by the account program (/a/becca) isn't going to be used, copy the old files to the new directory and remove the old one in order to keep the file system neat.

```
# cptree /a/becca /b/becca
# rm -r /a/becca
```

4.3.1. Logging Directly Into a Program

Perhaps the most interesting account characteristic that you can change is the log-in program, or "shell". Normally, when a user logs in, a program called **/bin/sh** is executed. This is the Micronix shell, which puts the command prompt on the screen and

sees that the jobs you want done get done. Micronix users almost always access programs by way of the shell program.

The thing to understand is that the shell program is just another program; you could just as easily set up a user that automatically executes a word processor or demonstration program (or any program, for that matter) as soon as he logs in. When this user finishes with the program, he is logged out of Micronix. So in effect, you have created a login name that automatically performs a single function, without giving the user access to other functions or to the operating system.

Such users are called "restricted users". Remember, though, that "user" in terms of a login name doesn't have to correspond one-to-one with actual human beings. The superuser, who logs in as "root", might be any of ten different people who happen to know the root password. A restricted user may in fact be a typist who logs directly into and out of a word processing program, but it could also be a login name that does nothing but display some advertisement on the terminal.

Other uses of the restricted user feature are:

- Logging directly into the CP/M emulator instead of Micronix
- Creating login names that perform specific functions like **who** (report who is logged onto the system), **tty** (tell me which port I'm connected to), or **date** (what day is this, anyway?)

SIMPLE EXAMPLE

You want to set up the system so that you can walk up to any terminal, and without logging on yourself, find out who else is currently logged on.

Use **account** to create a user named "who" with no password and a startup shell of **/bin/who**. From now on, anytime you login as "who", you will get the report you wanted. It displays momentarily, then the login banner reappears.

BETTER EXAMPLE:

To lock a user into a wordprocessor named ws.com (a CP/M program) in the /cpm directory, you would change their shell, with the account command, from /bin/sh, to

/bin/upm A:/cpm B:./ B: A:ws

What you have done here is (1) execute the CP/M emulator; (2) then you assigned the A: and B: drives to /cpm and the user's home directory, respectively. (3) You switched the current drive over to the B: drive, where the user's text files would reside. (4) Finally, you executed **ws**, a CP/M wordprocessor on the A: drive (which now corresponds to directory /cpm). When the user

gives the "exit" command for the wordprocessor, he is logged out, and the login banner returns.

4.4. Keeping Track of Accounts, Timewise

Micronix has a built-in program called **last** that tells you how much time each user has spent logged on to the system. You can get this data in a number of levels of detail, including start and finish times of each individual session.

4.5. Account Maintenance Summary

You have seen that Micronix initially has only one user, named "root". You quickly grasped that root should create other "users", and learned that the **account** program was the way to do this. You can add users and remove them with **account**. You can specify a password, a home directory, and a login shell. Passwords are optional, but strongly advised. Home directories are created for new users on the a drive, but can be moved anywhere in the file system. The login shell is wide open for customization, but it defaults to the Micronix shell if you don't specify otherwise. You can examine the results of running the account program by looking at the **/etc/passwd** file. Finally, the **last** program will tell you who has spent how much time on the system.

5. ADDING TERMINALS

Terminals are a computer's ears and mouth. Computers "listen" to what you type on the keyboard, and respond via their video-screen "mouths". Back in the 70's, when UNIX was being developed, video-terminals, alias crt's, were rare and expensive; teletypewriters were used instead. Teletypewriters are noisy, mechanical monsters that use a gang of solenoids to pound out a computer's responses, and have a hard-to-use keyboard of cylindrical keys. Teletypewriters, also known as ttys (pronounced tee-tee-whys), are still used to send telegrams and cables. Anyone forced to work in a room full of ttys is slowly tortured to deaf by the noise.

Today, we are blessed by having crt's as terminals. Crt's (cathode ray tubes) work quietly, emiting only a high-pitched whine which soon becomes inaudible. They also don't require paper, ribbons, or solenoids to pound out messages. They have civilized typewriter-like keyboards and work as much as 175 times faster than the old ttys.

Terminals are connected to the Decision through serial ports. When transmitting through serial ports, letters are converted from their internal form of 8 bits wide to a stream of 10 bits, which includes start and stop bits. So, a terminal that operates at 9600 baud, that is, 9600 bits per second, can transmit and receive 960 characters per second. The old ttys can manage, at best, 600 characters per minute.

There are a number of variables in the way that these ports operate. Many computers require you to set switches inside them so that the computer and the terminals understand each other's communications. With Micronix, these switches have been replaced by programs and files that determine how to set up the ports. The primary file of concern to you is the **/etc/ttys** file, which was discussed at length back in the INSTALLATION section of the User's Guide.

In Micronix, the /etc/ttys file is used to establish the initial configuration for both serial and parallel ports. The /etc/ttys file also provides a connection between the ports and the software that will use the ports. For example, the serial ports can be used by either terminals or printers.

Initially, the /etc/ttys file is set-up so that the first two serial ports, ttyA and ttyB, will be connected to terminals, and the third serial port, ttyC, will be connected to a serial printer. Let's take a look at the lines of the /etc/ttys file that describe this set-up:

```
ttyA   9600   login
ttyB   9600   login
ttyC   1200   lpr   lst
```

The format of the ttys file is quite simple. Each line,

describing one port, begins with the name of the port. If you look in the /dev directory you will find the same names. After the name of the port comes the baud rate and the name of a program associated with the port. "login" means that this is a terminal port that users can log in on. "lpr" means this is a printer port that will be used by the lpr program. You could swap these settings around by editing the file, and swap your device connections on the Decision accordingly, and things would still work. But don't run off and do that right now.

There are 14 different baud rates that you can use with the Decision serial ports. They are

50	600
75	1200
110	1800
134.5	2400
150	4800
200	9600
300	19200

Changing baud rates or port connections is done by editing the /etc/ttys file, which can only be done by the root. You can configure a serial port for a terminal by adding the baud rate and the word "login" to the line for the port. For details on the format of the /etc/ttys file, you should review the pages for ttys in the Files part of the Reference Manual. Or you can let the recon program (for recon-figuration) handle the details for you.

The recon program makes changes to the /etc/ttys file through an interrogative process, similar to the account program. Changing the printer configuration is discussed in the section on Configuring Micronix for your Printer in the INSTALLATION section of the User's Guide. Here, we'll discuss using recon to change your terminal configuration.

One point we must make before going any further is that the console must always be connected to port ttyA and set to 9600 baud. During the loading process, Micronix doesn't use the /etc/ttys file; instead, Micronix assumes that there is a terminal connected to ttyA and set to 9600 baud. If you'd rather set the console to 19200 baud, you can do so after going multi-user, but you'll have to reset the console to 9600 baud every time you reset the system so that you can communicate with single-user Micronix. (Many terminals tend to lose characters if operated at 19200 baud, anyway).

When you type the **recon** command, you will be provided with 5 choices:

1 Help
2 Configure login terminals
3 Configure printers
4 Quit and ABANDON all changes
5 Quit and make the changes

You want to select "2" to make changes in terminal configuration. The recon program will tell you which terminals are currently configured for login, and ask if you wish to change this. If you respond "y" for yes, you will be required to list all the ports that you wish to designate as capable of logging users in. (Even if you don't have as many terminals as ports, setting up all the serial ports not used for printers as login-capable may prove invaluable, as will be discussed later in this section under Desperate Measures.)

To specify the ports to the recon program, it is sufficient to type just the last letter of the serial port name, for example, A for ttyA, or D for ttyD. And you will always need to include ttyA, for the console. If your Decision has the standard three serial ports, ttyA, ttyB and ttyC, and an additional three ports, ttyD, ttyE and ttyF, on a MULTI/O board, you would enter the letters

A B D E F

to select these five serial ports for login, reserving ttyC for a serial printer.

Next, you will need to enter a baud rate for each of these ports. You will also be reminded to specify 9600 baud for ttyA. You should select the highest speed possible for the rest of the ports connected to terminals, which will usually be 9600 baud. Remember that the baud rate for the terminal has to match what you say here.

When you have finished entering baud rates, the main menu (the 5 choices) will be redisplayed. If you are satisfied, type "5" to make the changes effective immediately. If you aren't satisfied, you can type "2" and redo things. And if you are hesitant, perhaps you'd do best by typing "4" and getting things straight first. The changes made by the recon program go into effect as soon as you type "5" to quit. The recon program will go and adjust baud rates to agree with the new /etc/ttys file. You can type **type /etc/ttys** to observe the changes made by recon.

If the changes you made disrupt your communications with Micronix, you might have to reset the system and run the recon program while in single-user mode. This is the worst that can happen, and isn't really very bad at all.

6. CUSTOMIZING THE ENVIRONMENT

The Micronix operating system makes it easy for you to alter the outward workings of Micronix in ways that make day-to-day operations more convenient for the system manager and other users. We call such changes "customizing the environment" because there are no definite prescriptions for specific changes. We will provide suggestions and examples, but your situation and experience will determine which features you customize. These are changes that anyone can make, with no knowledge of Micronix programming.

Now then. What features are open for adaptation to your needs?

- your command prompt,
- shorthand abbreviations of your most-used commands (aliases),
- the order in which to search directories for command files,
- routine commands to be automatically executed at login time,
- routine commands to be executed when the system "goes multi-user",
- the message of the day,
- the initial sign-on message,
- the login banner, and
- the directories and LST: device to use with the CP/M emulator, upm.

Changes are accomplished by using a text editor to modify certain Micronix files. The editor can be either the Micronix editor (**edit**) or a program such as WordStar.

The easiest customizing changes to explain, and understand, are the various message files. These files are read by Micronix at the appropriate times and displayed on the console or user's terminal. We'll talk about these first.

The other aspects that you can customize involve creating a new file with the text editor that contains nothing but a list of normal Micronix commands. The new file must have a certain name and be in a certain place so that Micronix can find it at the appropriate times. A typical example of this feature is to put an **fsck** command in a file called **/etc/rc**, so the file system automatically gets inspected every day when you go from single to multi-user mode. You could put **mount** commands into the same file to get any additional hard disks mounted without having to enter the commands manually.

The files involved in this customizing process are called:

- /etc/rc - the routine commands to be executed while going multi-user,
- .sh - commands for customizing the user's interface, the shell, and
- .upm - drive and LST: device designations for the CP/M emulator, upm.

We'll discuss each of these files in one of the sections that follows.

6.1. Message Files

The message files are ordinary ascii character files that are transmitted to a terminal in exactly the form that they appear in the file. Thus, by using an editor, you can create the very image that you want to appear on the screen.

NOTE

> If you use a CP/M-based editor like WordStar or NewWord, you will need to run the **clean** program on the file after you finish editing it. Repeat this cleaning process each time you edit the file. Example: after editing the message of the day with NewWord, at the Micronix prompt enter **clean /etc/motd**.

In the examples we will be using, we use the Micronix edit program. The edit program is a line-oriented editor, meaning that it works on a single line at a time. This makes it nice for the purposes of this manual, but you may prefer to use a screen oriented editor. Screen-oriented editors allow you to edit a whole screen at a time by moving the cursor to different positions. WordStar and NewWord are screen-oriented editors.

6.1.1. Message of the Day

The contents of the message of the day file appear when users log in. The same message is used for all users, making it an ideal way to convey information of general interest. Messages are entered by editing the "motd" file, for message of the day, in the "/etc" directory.

For example, to change the message to read "The hard disk is almost full again. Please remove old files", you would enter the following:

```
# cd /etc                    COMMENTS (don't type)
# edit motd
"motd", 1 lines
:%d                          deletes all lines
:a                           adds new lines
The hard disk is almost
full again.  Please remove
old files.
.                            until you type the dot in col. 1
:w                           writes it out to the disk
"motd", 3 lines
:q                           and quits editing!
# []
```

Now, every person who logs on to the system will see the message about the hard disk filling up. The message of the day file can be changed every day, as needed. Your next message could be a thank you to the people who cleaned up their act by copying old files to backups, and removing them from the hard disk. Remember that you can temporarily get rid of the message of the day by deleting the file /etc/motd (**rm /etc/motd**), or by changing its name (**mv /etc/motd /etc/oldmotd**).

6.1.2. Custom Banners

The banner appears on terminals while Micronix is waiting for a log-in name. You may wish to edit the "banner" file in the "/etc" directory to customize it for your environment. You could, for example, edit the banner to reflect the name of your business. Note that in this example you delete the old banner entirely. If you just want to make a few changes to it, DON'T delete it first.

```
# edit /etc/banner           COMMENTS (don't type)
"/etc/banner",  15 lines
:%d                          removes the old banner
:a                           ready to add lines

          WEST COAST SHIPPING AND STORAGE

          "You don't wait for our freight"


           Tracking and Invoicing System

.                              a dot in col. 1 to finish
:w                           w writes the file, and
"/etc/banner", 9 lines
:q                           q quits
# []
```

This editing session changes the banner to something more appropriate for the business described above than "Micronix" in 3D letters. Of course, we don't mind if you leave our banner up. If you don't want a banner at all, just **rm** or **mv** the file.

6.1.3. Sign-on Message

The sign-on message is displayed on the console whenever Micronix is reset. Thus, only the first user to log-on will see this message. The delivered sign-on message assumes that this user has logged in as "root", and issues reminders about the responsibilities and hazards of being the superuser.

You can change this message, which is in the file /etc/signon, to suit your particular environment. For example, you might include in the message instructions for running fsck, correcting the date and going multi-user. This way, instructions are always available to the person bringing up the system, and can be edited as necessary to reflect your individual practices.

6.2. Command Files

"Command files", in the parlance of UNIX, are known as scripts. Just think of a script used by actors and you get the idea. A script is a list of commands to be followed that can be interpreted by a program, usually the shell. As mentioned previously, a script is just a list of the same sort of commands you use day in and day out; using the script, however, saves you the time and effort of remembering to enter routine commands.

The scripts that we are interested in here are the ones that Micronix will "automatically" look for at certain points. The is executed when the first person to log in at the console types "exit", causing the system to "go multi-user". The .upm script is used by the upm program to designate disk drives for the CP/M emulator. The next three sections give examples of what you can do with these scripts.

6.2.1. Log-in Commands: The .sh Script

The .sh script is the user's principal tool for customizing his or her working environment. After a user has typed in their log-in name (and their password), a shell program (command interpreter is started for them. You may recall from our earlier discussion of "accounts" that a user has a home directory and a login shell. The home directory is usually **/a/username**, and the shell is normally a program called **/bin/sh**, which processes Micronix commands.

When the /bin/sh program starts up, it always looks in the user's home directory for a file named **.sh** . If the shell program finds the .sh file, it reads it and executes the commands that it finds there. Applications of the .sh file are given below. If .sh isn't there, the shell program just proceeds to display the Micronix prompt.

Turnkey Users

The .sh file may seem to be the ideal method for creating a "turnkey" user, that is, one that automatically executes a file. (If you want to create turnkey users, look up the **account** command description, earlier in this chapter, for instructions on Changing the Shell.) The difference between using .sh and account for turnkey users is that account creates restricted users, that is, they are logged out when they finish with the turnkey program. The .sh turnkey programs exit to the Micronix prompt without logging the user out.

If you only use .sh to start up turnkey programs, you are really misusing .sh, and overlooking most of its potential.

Aliases

With the .sh file, you can change the name of, or abbreviate, your favorite commands. This is called creating aliases. An alias has exactly the same meaning as an alias in real life: an assumed name. The aliases that you will use are there to replace long or often-used command lines with an alias named by you that is easy for you to remember. For example, typing

```
% alias ws "cd ~/ws; upm b:/backup ws"
% []
```

causes your shell to substitute the letters "ws" for the line "cd ~/ws; upm b:/backup ws", saving you 22 keystrokes. If you often work with CP/M formatted floppy disks, here are several aliases that you can use:

```
alias fard "far mfa"        (displays floppy directory)
alias farr "far mfa -r"     (copies to the floppy)
alias farx "far mfa -x"     (copies from the floppy)
```

The "far" command reads or writes floppy diskettes in CP/M format. (More information about far is contained in the section on COPYING FLOPPY DISKETTES, and in the Program entry for far.) If you have trouble keeping the r and x flags straight, you could create these aliases:

```
alias copyto "far mfa -r"
alias copyfrom "far mfa -x"
```

As an example, the following commands show the listing of the filenames on a diskette, and copying (extracting) a file from the diskette:

```
% fard
maint.bak
first.doc
install.txt
% copyfrom first.doc
% []
```

There are other aliases that are useful for dealing with floppy disks that have Micronix file systems on them. To use a Micronix disk, you must first mount it (see the section on COPYING MICRONIX DISKETTES, or the program pages for **mount**.) You can create aliases for mounting and unmounting diskettes, as in

```
alias mof "mount mfa /f;dir /f"
alias umof "cd; umount mfa"
```

to save time when using these commands. These examples demonstrate stringing sequential commands together with the semi-colon. "mof" connects the floppy file system to the /f directory and then displays the current directory there; "umof" takes you out of that directory (a precondition for unmounting) back to your home directory, and then unmounts the floppy.

By keeping all these aliases in your .sh file, they are ready to use as soon as you log-in. You can check which aliases you have in effect at any time by typing the command "alias" by itself.

Path

Another aspect of your environment that you might need to change is called the **path**. The path is the list of directories to be searched for the program that you have requested. The default path is: first, your current directory; next, the directory /bin; and finally the directory /usr/bin.

You would want to change your path if, for example, you have programs installed into a directory that is not part of the default path. Suppose you have added a directory with database programs in it, called /db. You would add this to your path by typing

```
path . /db /bin /usr/bin
```

making /db the second directory to be searched for commands. (The dot stands for your current directory.) **If you want this path to be in effect every time you log in, put the path command in your .sh file. Otherwise it stays in effect only until you log out.**

Another interesting, and fun, aspect of the environment to change is the **prompt**. The prompt that comes with your system is a percent sign (%), or a pound sign (#) for the superuser. You can change the prompt to make it more descriptive by substituting the user name for the percent sign, for example,

```
prompt "rik: "
```

changes the prompt to "rik: ". Leaving a space after the prompt makes your command line easier to read. If having your own name on the command lines isn't interesting enough, you can try something like

```
prompt "What is your command, master? "
```

for a real feeling of power, or

```
prompt "Like, how's your karma?"
```

if you want to be mellow. Adding the prompt line to your .sh file will make it automatic and permanent, until you remove the line from .sh.

Miscellaneous .sh Tricks

You can also add other commands to your .sh file. You can include the **mail** command, for example, and automatically check your "mailbox" while logging in. Another neat one is to include the command

```
echo There are `who | lines` users on the system.
```

which will report how many other users are logged in.

The thing to remember with most .sh commands is that they will take some time to execute EVERY time you log in. If you are sometimes impatient, you can interrupt these commands by typing rubout or delete. If you are always impatient, refrain from adding commands to your .sh file. The alias, path and prompt commands are built into the shell so they will be done quickly; thus, they can still be used by impatient people.

The .sh file belongs in your home directory. Use an editor program to create your own .sh file. (Remember to "clean" the file if you create it with a CP/M editor like NewWord). For starters, here is the .sh that we have been talking about ("rik" on the first line is the customized prompt):

```
rik: type .sh
alias fard "far mfa"
alias copyto "far mfa -r"
alias copyfrom "far mfa -x"
alias mof "mount mfa /f;cd /f"
alias umof "cd;umount mfa"
prompt "rik: "
path . /db /bin /usr/bin
echo There are ` who | lines ` users on the system.
rik: []
```

6.2.2. Routine Commands File: /etc/rc

The commands in the /etc/rc file are automatically executed as part of the process of going multi-user. After the root user has accomplished the tasks of checking the root file system and setting the date, the command "exit" is typed to make the system go multi-user. Since you can count on this point being reached reliably, it is a good place to perform routine tasks.

Typical tasks that belong in this file are removing temporary files from the previous day's activities, testing additional file systems, mounting those file systems, and waking up daemons. Let's look at the /etc/rc file that may have come with your Micronix system.

```
# type /etc/rc
echo Starting update daemon
echo Starting network daemons
netdaemon
update
# []
```

The echo commands are pretty straightforward: they display the message that follows them on the console. **netdaemon** is the command that wakes up the network daemon which manages networking. **update** awakens the update daemon. The update daemon is responsible for periodically executing the **sync** command, which synchronizes disk buffers in memory with the appopriate blocks on the disk. No multi-user system should be without an update daemon to watch out for it.

So, you see that you already have a routine command file with some commands in it. Now we'll tell you how to improve on it.

One of the things that you do when you use a computer is generate temporary files. Some programs, such as edit, create temporary files when they are used. Humans are also known to be

the creators of temporary files. Micronix has prepared for this by having a special directory appropriately named /tmp.

Making the /tmp directory truly temporary should mean that files are erased from /tmp on a regular basis. This is obviously a job for the rc file. Including the commands

```
echo Removing temporary files from /tmp.
era /tmp/*
```

sends a message to the console and removes any files in the /tmp directory. Now, files left in the /tmp directory are truly temporary, because they will be removed by the rc file.

Another routine task that you may be faced with is mounting additional file systems. Your Micronix system has the root file system "built-into" it, so it isn't mountable. But, if you buy another hard disk to use with your system, it will need to be mounted every time after you reset the system.

To mount the disk drive, you have to enter a **mount** command , along with the name of the drive and the place in the file system to mount it, and wait for the process to complete. For example, to mount a second 16 megabyte hard disk, the command is

```
mount m16b /b
```

You should put this command in the rc file so that you won't have to think about doing it every day. And, we hope that you remember that file systems should be checked before they are mounted. So, we'll bundle these two commands together, and add them to the rc file, as in

```
echo Checking the second file system.
fsck /dev/m16b
echo Mounting the second file system.
mount m16b /b
```

When you type "exit" from single-user mode, the commands in the rc file will be executed. After the echo command reports "Checking ...", the fsck command performs its thorough checks of the unmounted file system on the second hard disk. This will take about 5 or 10 minutes, but is well worth the time. Then, the second echo command notifies you that it is "Mounting ...", and the mount command adds the new file system to the directory /b. From this time until the system is reset, the second hard disk will comprise the /b directory.

OK. Let's add the commands we discussed to the rc file and see what we've got:

```
# type /etc/rc
echo Checking the second file system.
fsck /dev/m16b
echo Mounting the second file system.
mount m16b /b
echo Removing temporary files from /tmp.
era /tmp/*
echo Starting update daemon
echo Starting network daemons
netdaemon
update
```

6.2.3. Configuring upm With .upm

"upm" is Micronix's way of imitating CP/M, which is an operating system quite distinct from Micronix. It is there to enable you to take advantage of all the well-tested and inexpensive software available for CP/M systems. ".upm" is a file that you create to tell upm where to find and put things.

CP/M has a very different file system from Micronix. The location of a file under CP/M is determined by which disk drive it is on. Therefore a CP/M filename might be "A:frogeyes", meaning the file "frogeyes" is on drive A:.

Get this straight right now: **There is no inherent connection between CP/M drive names (like A:) and Micronix directories (like /a).** You make these connections with explicit commands. The details of using upm are discussed under "upm" in the Programs section of the Reference Manual.

The .upm file sets up drive designations for the CP/M emulator, upm. A .upm file is not a shell script. Rather, it is a set of instructions that are passed to upm. When upm begins to execute, it looks for a .upm file in either the home directory (like a .sh file), or in the current directory. A typical .upm file looks like

```
% type .upm
a:/cpm  b:./ lst:/dev/ttyC
% []
```

which sets the A: drive to the /cpm directory and the B: drive to the current directory. Any printer output to the LST: device is routed to the port ttyC, where presumably a serial printer is connected. The upm entry in the Reference manual provides a clearly written and detailed description of how upm works, and what you can do with your .upm file.

7. SECURITY

Whether you like it or not, your Micronix file system requires that you do something about security. You might only need to secure your files from being erased by the nine year old computer wizard on your home system. Or, the files you wish to protect may contain truly sensitive material, such as your payroll accounts, financial analysis of your next year's sales potential, or letters to your mistress.

Security in Micronix is focused on two issues: passwords and file access permission. Passwords are relatively easy to discuss and implement; file access permission is a more complex topic, and an ongoing process. This being the case, we'll talk about the easy one - passwords - first, and tackle access permissions next. Finally, we'll talk a little about file systems on floppy disks, and the threat they may pose to your system's security.

7.1. Passwords

Passwords are entered for each user with the **passwd** command or with the change account selection of the **account** program (explained toward the beginning of this division). The passwd program prompts the user for the information required, and is self explanatory. Whenever passwords are entered, the display to the terminal is turned off. So don't be surprised when the letters you type for your password are not displayed. This is done for security.

Passwords are kept in encrypted form in the file **/etc/passwd**. A user's "passwd" entry also contains several fields of information, which were collected when the user entry was created with the account program. This includes the user's name, a numerical user and group identification number, comments about the user, his home directory and command interpreter program. This informaion was explained previously in the section on Users. A typical /etc/passwd file will have entries that look like:

```
root:kse86bklsdf:0:0:super:/:/bin/sh
pam::10:1:Pam Smiles, sec.:/a/pam:/bin/sh
```

The colons separate fields of information. Passwords are the second field of information. Looking at the encrypted password for root only discloses the encrypted word. Thus, the passwd file may be read by anyone without disclosing anyone's unencrypted password. Notice that Pam is not using a password so her password field is empty (the two colons come together ::).

Because Pam isn't using a password, anyone can log-in as "pam" and become the owner of her files. Now, maybe the files owned by Pam aren't considered sensitive security-wise, but the issue extends beyond Pam's files. For example, just knowing

Pam's name would allow an intruder to log on to your system. And once logged in, the intruder has access to more than Pam's files. The first thing that a clever intruder does is look at the /etc/passwd file and get the names of other user's without passwords. The /etc/passwd file cannot be read-protected without disrupting other Micronix programs, thus the passwd file will always be accessible.

The intruder who has logged in as Pam has free access to files owned by Pam, to unprotected files owned by anyone else, and to files shared by other members of Pam's group. (See the next section or the Programs division entries for **chmod** and **owner** to get more information on groups and file ownership.)

If you feel like we're trying to encourage you to force all the people on your system to use passwords, you're right. Although it is possible for an individual user to provide protection for his or her files, your system is exposed to great danger by having even a single user without a password. Always add a password to new user accounts. The individual user can change the password if they so desire. And periodically check the password file for unprotected accounts. You can use the **grep** program to pick out accounts with blank password fields.

```
# grep "::[0-9]" /etc/passwd
pam::10:1:Pam Smiles, sec.:/a/pam:/bin/sh
# []
```

The grep command picks out the lines in the file that contain two colons (::) back to back followed by any number ([0-9]). Don't forget to quote the search pattern ("::[0-9]") or the shell will expand the pattern before it reaches grep.

An entire chapter could be written on ways of broaching file system security. Breaking a system's security has become something of a sport among hackers (computer programmers and lifeforms that prefer dealing with computers to dealing with people). Most of the ways of breaking into UNIX systems, which includes Micronix, rely on having access to a user account. Password protection denies this basic access to intruders. Hopefully, the people who have legitimate access to the system won't be your security risks by giving away their passwords. Be especially careful deciding with whom you will share the root password.

7.2. File Access Permission

Micronix creates files that are accessible to any user. The owner of a file, that is, the user who requested the creation of the file, may choose to limit access to the file. This is useful even in a solitary user environment because it allows the user to write-protect files to prevent accidental erasure. In more sensitive environments, changing the access permission of files must become a routine, as files are created without any protection.

Micronix divides the users that may access a file into three categories:

- owner - the owner/creator of the file,
- group - members of the owner's group, and
- other - all other users.

The owner and other categories are clear in themselves; groups are a special category that hasn't yet been mentioned. The group category allows the users on a system to share access with other group members. Membership in a group is determined by the information in the **/etc/group** file. We would like to refer you to the Files section of the Reference Manual for more information about establishing group membership, if you are interested. Our experience has shown us that few installations actually make use of groups.

For each of the three categories of users (owner, group and others), the same three types of access permissions are possible for files: read, write and execute permission. The **ls** command with -l (long listing option) displays the access permissions for files as part of the listing. As an example, we can create a file by redirecting the output of an echo command and look at the access permissions for it:

```
% echo Hello > test
% ls -l test
   rw-rw-rw-  1 rik            6 Jun 14 17:38 test
% []
```

The first nine characters in the listing for "test" are the access permissions. (The name of the file owner, rik, appears shortly after the access permissions). With only a little bit of deduction, you can see that r stands for read, w for write and - for permission denied. What's missing in this example is x for execute. Micronix assumes that redirected output will be a text file, not an executable one, so execute permission is denied here. You should also note that read and write permission is given for all three categories:

```
owner  group  other
   \     |     /
    rw-rw-rw-
```

This means that our test file is totally unprotected. Now, let's take a look one of the programs in the /bin directory and see how it is protected.

```
% ls -l /bin/fsck
  rwx--x--x   1 root          27833 Mar 31 11:34 /bin/fsck
% []
```

The fsck program is certainly one that you wouldn't want to erase by accident, or allow anybody to change. Looking at the ls listing for fsck, you can see that the file is owned by the root user, who has read, write and execute permission. All others have execute-only permission: reading and writing the fsck file is denied. So, anybody may execute the fsck program, but only the root can copy it (read permission) or change it (write permission).

Keeping in mind that files are created with all permissions granted to everyone, you need to know how to set the permissions to protect your files and directories. The **chmod** program can change the access permissions on both files and directories. The description of how to use the chmod program in the Reference Manual is quite good, so we won't repeat it here. Rather, we'll give some examples of how to protect your files and directories that will work until you feel like reading the reference entry.

The most common way that you will be setting your access permissions to protect a file is removing all permissions for others. Then only you, the owner, will be able to access the file. The way to do this is with the command

```
% chmod go-rwx test
% ls -l test
   rw-------  1 rik            6 Jun 14 17:38 test
% []
```

which denies (-) the group and others read, write and execute permissions on the file test. You will substitute your own filename(s) for "test" when you use chmod. Since you will need to use this chmod command to protect any file that you create, you might want to alias it to save yourself having to type it and remember it. For example,

```
% alias deny "chmod go-rwx"
% deny topsecret
% []
```

makes "deny" an alias for "chmod go-rwx", and on the second line, "denies" access to the file "topsecret" to the group and others. Add this alias to your .sh file to make it automatic. To add write-protection to the files you don't want to inadvertantly erase yourself, you can use the command

```
% chmod u-w topsecret
% []
```

which write-protects the topsecret file from your own misdoings.

Read, write and execute permissions work somewhat differently for directories. Read permission allows the listing of the directory with dir or ls. Write permission allows files to be written in a directory, or filenames changed. Neither read nor write permissions do anything that protects the files within

the directory, that is, a file that is in a write-protected directory may be written to, unless the file itself is write-protected. In Micronix, a directory is a special file containing filenames and links to files. Read or write protecting a directory controls the ability to read or write in the directory, not in the files listed by the directory.

Execute permission for directories means permission to search through a directory. Essentially, searching is a form of reading the directory performed by the operating system to access the link to a file. When execute/search permission is denied on a directory, no files or subdirectories listed in that directory may be read or written. This means that you can protect files within a directory and its subdirectories by removing execute permission for it. The deny alias we set up earlier will serve here to remove execute permissions (and incidentily read and write permissions) from a directory that we want to protect.

```
% mkdir secret
% ls -l secret
d--rwxrwxrwx   1 rik              32 Jun  15 11:04 secret
% deny secret
% ls -l secret
d--rwx------   1 rik              32 Jun  15 11:04 secret
% []
```

In this example, we created a directory named "secret" and displayed its initial access permissions: it's open to anyone. Then, we changed it using the deny alias (for chmod go-rwx) to remove permissions for everyone except the owner. Now, any file copied or created within secret, and any subdirectories, will be protected without having to use chmod on the individual files.

7.3. Protecting Diskettes

There are three system security aspects relating to diskettes: backups, the standalone Micronix diskette and mounting diskettes. The issues here a somewhat advanced; only an experienced UNIX or Micronix user would be a real threat in these cases. Access to the root user (presumably on a different Micronix system) is also necessary for most of these techniques. And, adding protection for these cases may involve a nuisance factor that far outweighs the benefits.

The most obvious of these threats involves backup diskettes. While you are backing up your hard disk, you are creating duplicate files on a file system on a floppy diskette. These files, as are all files in Micronix, are created with all access permissions. What this means is that anyone who has access to the backup diskettes can mount a diskette and examine any file on it. There are two things which you can do to prevent this: keep the diskettes locked in a safe place (not a bad idea in general), and use chmod to limit access to the files on the diskette. For example, to limit access to files on a backup diskette, you need to type

```
% mount mfa /f
% chmod go-rwx /f/*
% umount mfa
% []
```

The chmod command limits access to the owner, presumably the root, of the files on the diskette. As explained in the previous section, removing execute/search permission from a directory prohibits access to its files and subdirectories. This three step process would need to be repeated for any backup diskette containing sensitive files. You can make this process simpler for yourself by creating a file with these three commands in it, and executing the file by using the **source** command with the diskette already inserted.

The Standalone Micronix diskette forms the second diskette security hazard. Anyone who has access to a copy of the Standalone and can reset the system can become the superuser, and access any file on the hard disk file system. Once again, the solution is to keep the Standalone diskette locked in a safe place. Adding a password to the root user on the Standalone is not advised, because having the Standalone is protection against forgetting or losing the root password.

The third and final hazard posed by floppy diskettes will be outlined only. Essentially, a person with access to the root user on a different system can create a shell program on a floppy diskette that will enable him or her to have root privileges on any Micronix system that the diskette can be mounted on. This is getting rather paranoid, since it involves: a knowledgeable user with bad intentions who knows the root password on another Micronix system, and an ordinary password on your system.

If you feel that it is necessary to protect yourself from this, you can restrict the use of the mount command to the root user by using chmod (or the deny alias). Then, only persons who know the root password will be able to mount diskettes. (Anyone knowing the root password won't need to mount a special diskette to have unlimited access to the file system). Such draconian measures as this are generally reserved for banks and the National Security Agency because of the inconvenience involved in restricting mount to the superuser.

To summarize, most of the security problems involving diskettes can be eliminated simply by keeping the diskettes with sensitive material on them under lock and key. Since you wouldn't want to lose any of these diskettes anyway, locking them away is a practical and almost painless solution.

8. COPYING FLOPPY DISKETTES

You will be duplicating floppies to create redundant backups of the hard disk or if you have data you want to share with other users.

We'll give you step-by-step instructions for copying diskettes momentarily. "Source diskette" means the floppy you want to copy; "destination diskette" is the freshly formatted diskette that you'll copy to. The general procedure goes something like this:

1. Use the df command to make sure you have enough room on the hard disk to receive the contents of the source diskette.

2. Format the destination diskette with the fdj program.

3. If you want the destination diskette to be "mountable", that is, if you want it to possess a Micronix file system, give it one with **mkfs**.

4. Copy the source diskette into a temporary hard disk directory.

5. Copy the temporary hard disk directory to the destination diskette.

6. Remove the temporary hard disk files.

NOTE: If you are copying a floppy that was produced by dumping a single chunk of the hard disk, it would probably be easier for you to get duplicates by repeating the hard disk dump instead of using the procedure above. And, if you have two floppy drives, you won't need to use the hard disk for temporary storage because you can copy directly between the floppies.

8.1. Preparing Your Diskettes (Formatting)

New diskettes must be prepared before using them with your Decision. This preparation is called formatting, and serves three functions:

- dividing the diskette surface into sectors, like slicing up a pie,
- filling in the sectors with an "empty" pattern, (the byte E5), and
- marking the first sector of the diskette with a byte that labels the diskette with the type of format used.

When you buy floppy diskettes they have (most likely) been preformatted. This is all well and good because diskettes are tested during the formatting process. However, you want to

format diskettes specifically for use with the Decision for the best results. Also, if you are planning to recycle old diskettes, it is a good idea to reformat them, as formatting will erase everything on the diskette.

Formatting erases EVERYTHING that was previously on a diskette, so never format a diskette that contains files you want to keep.

The Micronix formatting program is called fdj, short for format DJDMA (DJDMA is your floppy disk controller board.) To get the ball rolling, you can type

```
% fdj
                    DJDMA Formatter

          5 inch soft sectored formats
                  A) Morrow single sided
                  B) Morrow double sided

          5 inch hard sectored formats
                  C) Single sided
                  D) Double sided
                  E) Morrow Micronix

          8 inch formats
                  F) CP/M standard single density
                  G) Morrow CP/M double density
                  H) Morrow Micronix

          Do it yourself
                  I) Other
```

which invokes the fdj program in interactive form. We'll show you how to use fdj's menu first, for 5 1/4 and 8 inch formats, then we'll explain fdj's command line options, for non-interactive use.

8.1.1. FDJ's Menu Selections: 5 1/4" Drives

If you have a 5 1/4 inch drive, you will be selecting one of the first five (A-E) formats. These five formats are divided into soft and hard sectored types. Micronix diskettes are hard sectored. Hard sectored means that there is a hole punched in the diskette material marking the beginning of each sector. There is also an extra hole that marks the first sector. Micro Decision diskettes are soft sectored. Soft sectored diskettes have only the single hole that marks the first sector.

You can check to see if a diskette is hard sectored or not by looking through the small hole in the diskette envelope (called the index hole) and rotating the disk material through the large hole in the middle and counting the holes. As soon as

you have seen several holes, you know you've got a hard sectored diskette. Soft-sectored diskettes have a single index hole in them. Please be aware that there are also 16-sectored diskettes available which will not work with your system (you use 10), but these are rare.

If you are planning to use both sides of the diskettes for maximum capacity, buy diskettes certified as double-sided.

The first two format selections are for creating diskettes compatible with the Morrow Micro Decision. The Micro Decision format creates 1024 byte sectors on soft sectored diskettes. Obviously, you will want to select A for compatibility with single sided Micro Decisions, and B for double sided.

The next three format selections are for hard sectored diskettes. Selections C and D create formats compatible with North Star Computers hard sectored formats. The North Star format is 35 tracks of 512 byte sectors. You must, of course, select either single or double sided depending on the capability of the other system you wish to exchange information with. All Micronix systems come equipped with double sided drives.

The E selection is the one for use with Micronix systems. E selects double-sided, 512 bytes sectors and 40 tracks per side. This gives you the maximum storage capacity possible for 5 1/4" diskettes.

After selecting a format for 5 1/4 inch diskettes, you will be asked to choose a drive number. As far as the fdj program is concerned, your drives are numbered from 0 to 3. In single drive systems, the drive number will always be zero. (Confusion factor: some programs number drives 0-3, while others use letters a-d. There is a direct correlation between the two. The thing that determines the drive number is a jumper or switch inside the disk drive.)

Selection:

A) drive 0
B) drive 1
C) drive 2
D) drive 3

Selection: Drive 0.

Insert a write enabled diskette in 5 1/4 inch drive 0
Press <RETURN> to format, anything else to quit

Back in the Installation sector of the Guide, we discussed write-protection in the Floppy Disk Tutorial section. In brief, the write enabled diskette that you will be inserting will have the notch uncovered for 5 1/4 inch diskettes. Any new diskette

that you buy will have the notch uncovered. When you have inserted the diskette, press the RETURN key to start formatting. After formatting is finished, you will asked (again) to insert a diskette and press return to format. Pressing any other key will return you to the shell.

If the diskette you are formatting has any bad sectors, we advise you to throw it away (or return it to the vendor). Bad diskettes are not worth the risk of losing valuable data. If you want to make the newly formatted diskettes capable of booting CP/M or loading Micronix, read several sections ahead on WRITING THE SYSTEM TRACKS. After the section on formatting 8 inch diskettes, we will show you how to create one line format commands.

8.1.2. FDJ's Menu Selections: 8" Drives

IBM 3740 format is the accepted standard for 8" diskettes. But there are many variations of the standard in use. The variations involve differences in the sector size, and minute differences in the patterns that separate sectors. IBM 3740 formats are soft sectored, meaning that the software must determine where the sector boundaries are and the identity of the sectors. This is where there are differences in some of the "standard" formats. If you are trying to make a diskette readable on other "standard" 8" disk systems, choose F for single density CP/M format with 128 byte sectors.

If you are formatting diskettes for use with Micronix or CP/M, you will be selecting G or H. Micronix uses a block size of 512 bytes and this is the sector size you choose by selecting H for diskettes used on Micronix. If you are creating diskettes that will be shared with a Morrow 8" CP/M system, rather than exclusively with Micronix, use the G selection for 1024 byte sectors instead. Using 1k sectors will work with Micronix, but will result in somewhat slower writing, slightly faster reading and about a 7% increase in storage capacity over 512 byte sectors.

There is a physical difference between single and double sided 8" diskettes. On single sided diskettes, the index hole, a small hole used in determining the beginning of the first sector, is almost directly opposite the oblong slot used for reading and writing. On double sided diskettes, the index hole has been punched more to the side (about one o'clock, looked at label side up). The disk drive will determine which type of diskette you have inserted, and fdj will behave accordingly.

You will usually want to select H for maximum efficiency with Micronix. Invoke the fdj command, select H and a drive number (drive 0 in single drive systems), and insert a write enabled 8" diskette. Write enabled 8" diskettes have the notch on the edge opposite the label covered. Press the **RETURN** key when you are ready to begin formatting.

After formatting is completed the program starts over. If you want to format more diskettes, insert another diskette and press return. Otherwise, type anything else to return to Micronix. If you wish to make a diskette capable of booting CP/M or loading Micronix, read the section called WRITING THE SYSTEM TRACKS.

8.1.3. FDJ's Other Selection

The last of fdj's format selections is called "Other", the I selection. Other actually includes all of the formats previously mentioned. When you select I, you will be able to choose all of the parameters used by the formatting program, instead of selecting a particular subset of them, for example, double sided Micro Decision.

The Other selection is for those who know which format parameters they wish to select. You need to know, for instance, not only the sector size you want, but also the number of sectors per track. And, if you choose an impossible combination of sector size and sectors per track, your formatting will fail. If none of the first eight selections will work for your particular application, you may be able to use the Other selection. Other is intended to give you the maximum control over formatting for unusual circumstances. For normal formatting, do yourself a favor and stick to the first eight selections.

8.1.4. Format Options: fdj One-liners

Now that you have learned to use fdj interactively, you probably have some idea about the formatting choices that you will generally be using. You can combine your choices and the fdj command into a single line. Then you can alias this line so you'll not have to think about it again.

If you ever forget the fdj options, there's no need to be embarassed or to consult your manual. You can just type fdj with an illegal option and it eill display the legal options for you.

```
% fdj -x
x: Missing numeric value
Valid argments are b, i, d, s, and t.
Here are meanings and legal values:

   b means bytes per sector.        (128, 256, 512, 1024)
   i means inches.                  (5, 8)
   d means drive number.            (0, 1, 2, 3)
   s means sides.                   (1, 2)
   t means tracks.                  (35, 40)

Example:   fdj -b512 -i5 -d0 -s2 -t40
% []
```

The example given by fdj can be translated into

```
-b512          512 byte sectors,
-i5            5 1/4 inch diskettes,
-d0            drive 0,
-s2            double sided and
-t40           40 tracks.
```

This would be most appropriate for formatting diskettes on a single drive system with a 5 1/4" drive. If you were using single sided diskettes, you would substitute -s1 for -s2. If you were formatting an eight inch diskette, you would change -i5 to -i8. This should give you some idea of what's happening. To make things as clear as glass, the following aliases can be added to your .sh file to automate your formatting procedures.

```
alias fmt5 "fdj -b512 -i5 -d0 -s2 -t40"    <- hard sectored
alias fmt8 "fdj -b512 -i8 -d0"             <- IBM 3740
alias fmt8sin "fdj -b128 -i8 -d0"          <- IBM single d.
alias fmtmd "fdj -b1024 -i5 -d0 -s2 -t40"  <- Micro D. soft
alias fmtns "fdj -b512 -i5 -d0 -s2 -t35"   <- North Star
```

When you are selecting an alias for formatting, make sure you select ones that are more than one or two letters long and easy to remember. You don't want to be formatting diskettes by accident.

8.2. Checking the Hard Disk's Free Space

Since you'll be copying the source diskette into a temporary hard disk directory, you better make sure you have enough room. This is very simple. You can use any hard disk in the system, but we'll assume you're using the root device (/).

Type:

```
% df /dev/m16a
```

df will return the number of available blocks to you. For single sided 5 1/4" diskettes, 500 blocks will be enough. Double-sided 5 1/4 floppies are covered by 1000 blocks, as are single-sided 8"

diskettes. Double-sided 8 inchers may need as many as 2000 blocks. These figures assume your source diskette is full; if it is only part full, the required number of blocks can be adjusted downward in proportion.

If you don't have enough space on the hard disk, you'll need to back up some of the least used files to floppies and remove them from the hard disk. Refer to the later section "Backing up the Hard Disk" for instructions.

8.3. Copying CP/M Diskettes

There is a simple four step procedure for copying diskettes with CP/M directories on them. (Look at the next section if you want to copy diskettes with Micronix file systems on them.)

The first step is to establish a place for temporarily storing the contents of the source diskette.

```
% mkdir /tmp/temp
% cd /tmp/temp
```

to create a new and empty directory and make it the current directory. Now, insert the diskette you would like to make a copy of and type

```
% far mfa -xv          for 5 1/4" drives, or

% far fla -xv          for 8" drives.
```

This command copies the entire floppy disk to the new directory that you just made.

The next step is to remove the diskette your are copying and insert a newly formatted diskette. Then type

```
% far mfa -rv          for 5 1/4" drives, or

% far fla -rv          for 8" drives
```

and the contents of your current directory will be copied to the floppy diskette. If you changed directories between the extraction (far xv) and replacing the files (far rv), this won't work properly.

The final step is to clean up the temporary directory by removing it. Do this by typing

```
% cd
% rm -r /tmp/temp
Remove all files? y
% []
```

You can put off this final step and repeat step three if you want

additional copies of the same diskette made.

8.4 Copying Micronix Diskettes

Micronix diskettes are ones that you (or someone) have created file systems on with the **mkfs** command. The standalone Micronix disk all have file systems on them. Any disks that were made using the **td** (tree dump) command also have file systems on them. To copy these disks, you temporarily attach them to the main (root) file system, copy them into a temporary directory and disconnect the disks. (If you have two disk drives, you mount the file systems on both diskettes and copy directly between them). Then a file system is made on the new (formatted) diskette, it is mounted (attached), copied to and unmounted (disconnected). This section explains these steps in detail. See the section **mount** in the Programs division of this manual for more discussion of what mounting is all about.

The root directory, /, has a subdirectory, named /f, intended for use in mounting floppy disks. If you are using this directory for something else, no problem. Just create another directory with the mkdir command and use it. For example, the command

```
% mkdir /flop
```

creates a new directory named /flop. If you go this route, be sure you substitute "/flop" for "/f" in the commands that follow.

If there is a diskette already mounted, don't remove it until it has been unmounted (see step 3). To mount the floppy diskette that you wish to copy, insert the diskette and use this command:

```
% mount mfa /f          for 5 1/4" drives, or

% mount fla /f          for 8" drives.
```

For the rest of this explanation, we will use the name "mfa" and not repeat the commands for those of you with 8" drives. Instead, we will rely on you with 8" drives to substitute "fla". Thanks.

The mount command adds the file system on the diskette to the file system on the hard disk. This is like grafting a branch onto a tree, except in this case the joint is invisible. A mounted file system is completely a part of the entire file system. If the diskette you are trying to mount does not have a file system on it, the mount command will issue a complaint

```
% mount mfa /f
/dev/mfa: Not a file system
% []
```

Perhaps you have a CP/M directory on this diskette? Look in the previous section for advice on copying CP/M diskettes.

Now for step 2. We can use the **cptree** command to make a duplicate of the file system on the diskette on the hard disk. You need to make a temporary directory to copy into on the hard disk. Cptree handles creating the necessary subdirectories, so just type

```
% mkdir /tmp/f
% cptree /f /tmp/f
```

and sit back and wait a couple of minutes. If you get a disk full error message, it's not going to be possible to copy the entire diskette at once unless you clean up your hard disk by removing unused files or copying them to backups. The Maintaining Free Disk Space section of this manual has a few tips on doing this. Otherwise, you can use the **cp** command and copy only a few files at a time, rather than the entire diskette.

Finished copying? Good, time for step 3.

In step 3, we disconnect the floppy diskette from the file system by unmounting it. The command for unmounting diskettes is quite simple:

```
% umount mfa
% []
```

There is a trick to it, though. Notice that there is no "n" after the first "u" of umount. If you type the command as

```
% unmount mfa
Command not found.
% []
```

the command "unmount" won't be found. Use **umount**.

Step 4 entails making a file system on a formatted diskette. Like the other commands, there is nothing complicated about this. Insert the formatted destination diskette and type

```
% mkfs /dev/mfa
```

Please remember to substitute "fla" for "mfa" if you are using an 8" drive. The mkfs command automatically determines the size of the disk you are using and writes a Micronix file system on it. The display after using the mkfs command on a standard Micronix 5 1/4" diskette (selection E of fdj) is:

```
% mkfs /dev/mfa
Device size: 760 blocks
File system size: 760 blocks

Function complete.
```

```
% []
```

The whole process takes about 20 seconds. The mkfs command will check the diskette that you are using before creating a file system on it to see if there is already a file system there. Making a file system obliterates any file system previously on the disk, so the command produces this warning if a file system is present:

```
/dev/mfa: Already contains a file system!

DESTROY the old file system ? (Y/N)
```

If you type "y" <RETURN>, a new file system is created on the diskette. Typing anything else will abort the mkfs command. Mkfs can not detect a CP/M file system on a disk.

Step 5 is mounting the newly created file system. This is exactly the same as step 1. With the diskette that contains the new file system still in the drive, type

```
% mount mfa /f
% []
```

and your new diskette is mounted.

In Step 6, we copy the contents of the directory we created by copying the original diskette. This is very much like step 2, only the arguments are reversed:

```
% cptree /tmp/f /f
% []
```

This may take as long as 7 or 8 minutes, depending on the the number of files. When the cptree command is finished, you have a complete copy of the original Micronix diskette resting in the drive. Before removing the diskette from the drive, use the umount command:

```
% umount mfa
% []
```

You can remove the diskette after unmounting it. Now, if you want additional copies of the same diskette, repeat steps 4 through 7. When you have all the copies you need, remove the temporary directory that was created on the hard disk:

```
% rm -r /tmp/f
Remove all files? y
% []
```

Don't forget this step! You don't want to leave any unused files cluttering up your hard disk. Develope the good habit of immediately erasing files that you don't need any more and you can put off the time when your hard disk becomes full.

Since there are many steps in the process of copying Micronix diskettes, the following is a summary of the commands used. The comments on the right side are just that: comments, not part of the commands, so don't type them.

8.4.1 Summary of Copying Micronix Diskettes

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                COPYING MICRONIX DISKETTES

    % mount mfa /f              insert diskette to copy
    % mkdir /tmp/f              make a temporary directory
    % cptree /f /tmp/f          copy it to the hard disk
    % umount mfa                and remove it; insert new diskette
    % mkfs /dev/mfa             make a file system on it
    Device size:  760 blocks
    File system size:  760 blocks

    Function complete
    % mount mfa                 attach the new diskette
    % cptree /tmp/f /f          copy to it from the hard disk
    % umount mfa                disconnect  and remove it
    % rm -r /tmp/f              erase the temporary files
    Remove all files? y
    % []                            Done.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

These steps copy all the files from one diskette to another via the hard disk. However, the "system tracks", traditionally reserved by CP/M, have not been copied. The next section, WRITING THE SYSTEM TRACKS, explains how to do this.

8.5 Writing The System Tracks (SYSGEN)

When you copy one full diskette to another, the first two tracks on the destination diskette will remain blank. These tracks are reserved for one of two things: the CP/M operating system, or a Micronix load program. You can leave these tracks blank, which would be the thing to do under normal circumstances. However, if you are copying your CP/M Cold Boot Loader diskette (a good idea), you'll need to use the SYSGEN program to copy the CP/M operating system to it.

The other use of the system tracks is for storing a Micronix load program. Typically, when you bring up your system from a cold start (reset), you first boot up CP/M. Then you give a CP/M command like "m16boot" that loads the essential part of Micronix into memory and transfers control to it.

The Micronix load program gives you a way around the CP/M step. You can take a formatted diskette and use the SYSGEN program to write the appropriate load program on the system tracks of that diskette. From then on, when you RESET with the load diskette in the floppy drive, the system goes directly into Micronix without anyone entering a boot command.

The steps for writing either brand of system tracks are presented below. You will need the CP/M Cold Boot Loader diskette to write the system tracks. All SYSGEN operations are performed under CP/M, not Micronix or upm.

8.5.1 Copying CP/M

As mentioned above, the primary practical use of this is when you are duplicating your Cold Boot Loader diskette. Computer users that have strictly CP/M systems will copy their system tracks all the time; but as a Micronix user with a hard disk, you won't have quite as many uses for the CP/M system tracks.

We're assuming you've duplicated the Cold Boot Loader according to the instructions for copying CP/M diskettes. All that's left to do, then, is to reset the system and bring up CP/M. At the A> prompt, respond **sysgen.**

Using the SYSGEN program is very simple. We would like to point out that when you type drive names you don't want to type a <RETURN> after them. SYSGEN expects a single letter, "A", and will interpret the <RETURN> as the next response. The following script shows you exactly how to copy CP/M from one disk to another:

```
A>sysgen
Morrow Designs Sysgen Version 4.4
Source drive name (Or return if in memory) A
Source on A, then type return <RETURN>
Function complete
Destination drive name (Or return to warm boot) A
Destination on A, then type return <RETURN>
Function complete
Destination drive name (Or return to warm boot) <RETURN>
A>[]
```

When SYSGEN asks for your "Source drive name", you respond by typing the letter "A". The next line "Source on A" is just asking you if you are ready. If the old Cold Boot Loader is still in the floppy drive, you are ready, so type <RETURN>. After you see the line "Function complete", remove the Cold Boot Loader from the drive, and insert the copy. Type "A" for "Destination drive name", and <RETURN> to start the copy. When you see "Function complete", the copy is finished. You can now use this diskette by typing <RETURN> "to warm boot", or reinsert the old CP/M diskette and type <RETURN>.

8.5.2 Copying a Micronix Loader

Having a Micronix loader on the system tracks simplifies loading Micronix: you can insert the diskette with the loader on it, press RESET, and loading Micronix commences. The "LOAD" program does the same job as the "BOOT" program that you may have been using.

You may want to add the appropriate loader to a disk that you use for backups, or some other Micronix disk. Or you can make a formatted disk that is blank except for the load program, labelled "LOAD".

You DON'T want to add a loader to a CP/M diskette that you SYSGENed with the CP/M operating system, because the loader will overwrite the CP/M operating system on the diskette. If you accidentally copy the loader over the only copy you have of the CP/M operating system, you will really be in trouble. Just to be safe, write-protect the Cold Boot Loader you are using right now, notch covered for 5 1/4" diskettes, uncovered for 8".

We will be using a disk file as the source for SYSGEN, instead of the Cold Boot Loader's system tracks. The file that you will use is one of the four LOAD files on the Cold Boot Loader. The four files are

name	loads Micronix from
M5LOAD	5 megabyte miniwinchester
M10LOAD	10 megabyte miniwinchester
M16LOAD	16 megabyte miniwinchester
DJLOAD	Standalone floppy

How do you know which file to use? Well, normally you will have Micronix installed on your hard disk, so you would select the "M LOAD" file that matches the size of your hard disk. The only exception to this is if you wanted to copy your Standalone Micronix floppy. You would first copy its files using Copying a Micronix Diskette procedure, and then SYSGEN the copy with the DJLOAD program.

To use a disk file with SYSGEN, you simply add the name of the file you want to use after the command name. The following example shows the loader for the 10 megabyte miniwinchester being added to a diskette:

```
A>sysgen m10load
Morrow Designs Sysgen Version 4.4
Destination drive name (Or return to warm boot) A
Destination on A, then type return <RETURN>
Function complete
Destination drive name (Or return to warm boot) <RETURN>
A>[]
```

Remove the Cold Boot Loader diskette when the line "Destination

drive name" appears and insert the diskette to be copied to. Then type "A", and <RETURN> on the next line. The line "Function complete" announces the successful completion of the operation. If you want to make additional copies of the same loader, remove the diskette and insert another one; type "A" and <RETURN>. When you are finished, remove the last diskette, insert CP/M and type <RETURN>.

The difficulties you might encounter are: the diskette you are trying to use is not formatted correctly, or the file you are trying to copy, for example m10load, is not on the CP/M diskette. The five files we mentioned previously should all be on the "Cold Boot Loader (CP/M)" diskette that was delivered with your system. You can use the CP/M "DIR" command to examine the directory of the diskette you are using.

Unformatted, or wrongly formatted diskettes, will cause the SYSGEN program to get lost and never complete. This is a non-recoverable error, meaning that you will need to reboot CP/M. The section called PREPARING DISKETTES (FORMATTING) will help you get your diskettes formatted correctly.

8.5.3 Backup of the Stand Alone Master

You definitely want to have more than the single copy Stand Alone Micronix diskette that came with your system. This is the diskette that is used to rebuild Micronix from backups if the hard disk has been corrupted, or if you are in need of Desparate Measures (see the chapter by that name). We even considered "forcing" you to make a copy, but decided to leave it up to you to do.

There are two steps in making a copy of the Stand Alone diskette. The first step is to copy the diskette using the instructions in the section called Copying Micronix Diskettes (8.4.1). The second step is to make the diskette loadable by using SYSGEN (under CP/M) and copying the floppy disk loader (SYSGEN DJLOAD) as explained in the previous section (8.5.2). Or, you can use an alternative procedure that works for soft sectored diskettes ONLY. Stand Alone diskettes produced after November '83 are soft-sectored. The procedure is to copy a disk image to a temporary file, copy the file to a formatted diskette, and erase the temporary file. Simply type,

```
% cp /dev/mfa2 /tmp/temp
% cp /tmp/temp /dev/mfa2
% era /tmp/temp
```

remembering to change diskettes between the first and second copy. This procedure will work for any soft sectored double-sided diskette. Once you have finished the copy by either method, put both the original and the duplicate Stand Alone Micronix diskettes away in secure locations (preferably two separate secure locations, to guard against fire or theft).

9. BACKING UP YOUR HARD DISK

Making regular backups of the hard disk is one of those onerous tasks that most people don't like to do. And, you can probably use your Micronix system most of the time without needing to resort to your backups. Of course, if fsck discovers that a prized file has been trashed, or if a well intentioned associate cleans up your directory, wiping out two months work, you will wish that you had been keeping good backups.

What are good backups? Well, ideally, you should be able to recover 100% of the most recent version of a file from a perfect backup. This is possible for files that are unchanging, for example, a configured version of your favorite word processor program. But for files that are in the process of being modified, or written for the first time, everything that has changed since the last backup is missing from that backup. The trick is to decide how often to perform backups. With a once a week backup procedure, you risk losing as much as a week's work. Then again, if you back up every half hour you will spend most of your time backing up instead of working.

We will outline two procedures for backing up: a short term procedure and a daily procedure. You should optimize these yourself by selecting the length of time between backups for the way that you use your system.

9.1 Short Term Backups

Short term backups are copies of individual files that may be created at any time by the individual responsible for the files. Since this is dependent on the individual moods and whims of users, it is not considered totally reliable. But, if a person has failed to make a backup during the day and loses his file, than he or she is responsible for it.

Basically, a short term backup is made by using the copy (cp) command. The user simply makes a copy of the file he or she wishes to backup in another directory. To restore a lost file, the user copies it back from the backup directory. Ideally, the backup directory is on a different disk device. For example, if you have a second hard disk that is mounted on the /b directory, you can make your temporary backups by coping to a directory appropriately named /b/backup. To copy a file named BACKUP from the current directory to the backup directory, you would type

```
% cp BACKUP /b/backup
```

Now, you have a copy of BACKUP in a place that will be safe even if your current directory is lost.

If you aren't so fortunate as to be able to afford two hard disks, you still can use short term backups. You can mount a floppy disk with a file system on it to use as the backup directory, and at the end of the day either put the disk away as

an additional backup or erase all the files on it and reuse it the next day. Or, create a backup directory on your root file system by typing

```
% mkdir /backup
```

and let people use this as their routine place for short term backups.

At the end of the day, before performing a system wide (permanent) backup of the hard disk, the files in the backup directories should be removed. This will prevent the backed up files from being backed up twice on the permanent backup, and keep the hard disk from filling up with redundant files, that is, copies of old temporary backups. You can remove all the files in the backup directory by typing

```
# rm /backup/*
# []
```

9.2 System-wide Hard Disk Backup

Of all the things that you will do with your system, backing up the hard disk is both one of the most important and most neglected tasks. Only businesses and universities with full time computer system personnel actually seem able to routinely back up their systems. All we can do is to urge you to follow the suggestions in this section which will provide you with some protection from the winds of fate, and the inevitable process defined by Murphy's law.

9.2.1 First Hard Disk Backup

When you received your Micronix system, it came with a complete set of backups for the operating system, programs and manual pages. These files are, by this time, all on your hard disk. You have also configured your system by adding a printer, passwords, user accounts, etc. What we'd like you to do is to backup the entire hard disk the first time. Then, you will have a set of diskettes with your entire CONFIGURED system on them. If you ever need to rebuild your root file system from scratch, you would use this first set of backup disks and not need to reconfigure. We will name these diskettes the CONFIGURED MASTERS.

This backup will take longer than any other backup. This is because you will be using the -i (incremental) option for the first time. The incremental option writes an entry in the file /etc/dtab with the exact time of day of the backup, and the directory where the backup started. In subsequent backups, this entry will be used to determine which files have changed since the last time you backed them up. The subsequent backups are known as **incremental** backups.

To perform your very first backup, you first need to prepare a pile of diskettes. For systems with 5 1/4" drives, prepare at least eight diskettes by formatting them. If you are using an eight inch drive, you will need four prepared double-sided diskettes. For information on formatting, read the previous section on Preparing Diskettes.

Then, label these prepared diskettes with the title

CONFIGURED MASTERS VOLUME # DATE:

filling in the date, and numbering the Volumes starting with 1.

Insert the Volume 1 diskette in the drive that you want to use. Then type,

```
# fp cril /
```

for a disk in 5 1/4" floppy drive 0, or

```
# fp crilf /dev/fla /
```

for the first 8" drive. The fp program should merrily copy the entire files system. When you run out of room on the floppy disk, fp will advise you to insert a new disk.

The fp program copies a portion of the file system that you wish to backup with the pathnames unchanged. For example, if you have a file named /a/rik/ws/.upm, the directories a, rik and ws will be saved on the backup diskette, and the file .upm copied. This means that the file .upm will have identical pathnames on the hard disk file system and fp backup volume.

9.2.2 Daily Hard Disk Backups

For every other backup after the first, you will be using the incremental feature of backup. The incremental option, -i, will cause fp to read the /etc/dtab (for date table) file to get the last time a backup was performed on the directory. This will result in only the files that have been modified or created since the last backup being included in this backup.

You can use the same diskette that you used for the first daily backup until it becomes full. Fp essentially copies files to the floppy disk, and won't destroy other copies. However, it will replace old copies of files with their more recent versions. If you wish to keep daily records of old versions around, you should use a separate diskette for every day. You will also be safer using a different diskette for every day's backup rather than filling up the same diskette. If you have used one diskette for a whole week's backups, and that diskette is lost, you could be in big trouble. Regardless, always label your backup diskettes with:

BACKUP DATE DISK #

If you use a diskette for several days' backups, write the dates of each day you used the diskette, so that you know what you've got. The instruction to use for a daily backup is

```
# fp cril /                    for 5 1/4" floppy 0, or

# fp crilf /dev/fla /          for 8" floppy drive 0
```

If you remember to do this every night before going home, you will be fairly safe. If you keep duplicate copies of the diskettes in different locations and you will be safer still.

After you have been performing daily backups for a month, you will have quite a stack of Backup diskettes. You can imagine what it will be like finding a file to restore out of this pile of diskettes. What you need to do at this point is to erase the /etc/dtab file (era /etc/dtab) and follow the same procedure as listed in First Backup to create new CONFIGURED MASTERS. The new set of CONFIGURED MASTERS will require more floppy diskettes than the first one because your file system has undoubtably grown, so have more prepared diskettes at hand.

After creating new CONFIGURED MASTERS, you can recycle your old daily Backup diskettes. You may also start recycling the CONFIGURED MASTERS diskettes after you have two complete sets of MASTERS. In other words, when you start to make your third set of CONFIGURED MASTERS, you can reuse the diskettes that you used for the first set. This way, you will always have at least one complete set of MASTERS on hand at all times.

9.3 Restoring Backed Up Files

There are two ways of going about this, depending on whether you are restoring individual files, or the entire file system. The next section discusses restoring the entire hard disk. To restore an individual file, locate the diskette with the desired file on it. You can use fp to list the Volume directory, as in

```
# fp
```

which displays the fp directory diskette in the first 5 1/4" drive. When you find the appropriate diskette, you use fp with the x, for extract, option to get it back. For example, to get the .upm file back in the /a/rik/ws directory (mentioned in an earlier example), type

```
# fp x a/rik/ws/.upm
x a/rik/ws/.upm
# []
```

and you're done.

9.4 Restoring Your Hard Disk

If some awful calamity befalls your hard disk and you must rebuild it from scratch, you will be using the instructions included in the INSTALLATION Chapter on Installing Micronix on The Hard Disk. (If it is your second hard disk that suffered the disaster, follow the instructions for Adding a Hard Disk to prepare the disk instead.) There are three steps to this procedure:

1. Formatting the hard disk,

2. Using the Stand Alone Micronix diskette to build a skeletal system on the hard disk, and

3. Booting up the skeletal hard disk Micronix and copying the rest of the software to it.

You will be following these instructions exactly as they are listed in the Installation Chapter until you reach the part on adding the software from the Volumes using **finstall** (during step 3). At this point, we'd like you to use the Volumes created in the earlier section on First Hard Disk Backup, named the CONFIGURED MASTERS, instead of the Volumes sent with your system. This will save you from having to reconfigure your system. If you have a more recent version of CONFIGURED MASTERS, use it.

The procedure you use is to insert the first Volume of your CONFIGURED MASTER diskettes which you created during the previous section, and type

```
# source finstall
```

and wait for the process to complete. Then, insert the second disk, type !s (shorthand for "source finstall"), and wait, and so on with all the Volumes.

Then, use the same command to incorporate all the daily backups that you have made since the last time you made CONFIGURED MASTERS, starting with the OLDEST backup first. If you accidently restore older backups after more recent ones, the most recent backups will be copied over by the older backup. Eventually, you will have added all the files that were backed up using the daily backup routines and your system should be as good as new.

10. DESPERATE MEASURES

If you are reading this chapter of the manual, you must either have a serious problem, or just be curious. What are desperate measures? They are techniques that allow you to recover from serious problems without requiring deep knowledge of Micronix. The problems that we will teach you how to deal with here are:

- o Freeing up a terminal that was captured by a runaway program,
- o Replacing a forgotten root password, and
- o Repairing the root file system from the Standalone Micronix diskette.

You might have come to this chapter looking for a translation of a message that appeared on your console. The explanations of console messages are contained in a later chapter of this division named Console Error Messages, of course. This chapter will bail you out of other difficulties, and does explain how to use **fsck** from the Stand Alone Micronix diskette.

10.1. Runaway Terminals

Ever see a runaway terminal? Runaway terminals mostly get away from the designer of a new piece of software. What happens is that the new software has a serious defect (a bug) that causes Micronix to ignore everything you type on the terminal to try and stop the program. The program just keeps running wildly along, doing its own thing. On a simpler system, this problem is solved by resetting the computer. With Micronix, you don't want to reset the system: you might damage the file system and/or disrupt other users who don't have any problems, but will if you reset the system. What you do instead is KILL the program.

Now, maybe you don't like killing things, but relax. In Micronix, you kill a program by sending a signal to Micronix that gracefully halts the runaway program. Killing a program, (actually better described as killing a process), stops execution of the program, closes all its open files, flushes all disk buffers and releases the memory and swap space used by the program. After killing the program, the runaway terminal will show the normal shell prompt and respond correctly.

Killing a program is a three step procedure:

1. Log on as yourself on a working terminal,
2. Identify the process number of the offending program,
3. Kill the process.

STEP ONE

The first step may be performed in two ways. If you have more than one terminal hooked up to your Decision, log in on one of the other terminals. If someone is currently logged in, you can use the su program (switch user) to temporarily log-in. For example, suppose your user name is john and you want to temporarily login on someone else's terminal. After asking the person politely, you would sit down and type

```
% su john
Password:
% []
```

Now, you are logged in at two different terminals. (You could also do this as superuser, but it's more complicated.)

If you don't have two terminals, we hope you were paying attention earlier (during the Adding Terminal chapter) and left unused ports configured for log-in. Micronix comes configured for two terminals: one on port ttyA, and a second on ttyB. If you have a single terminal system, your terminal is the console, and is connected to port ttyA. And is also the runaway terminal. So here's the trick: reach around the back of your Decision, remove the RS232 connector from the ttyA socket (in the lower right hand corner of the back), and plug the connector into ttyB, immediately to the left of ttyA. Now, type a RETURN and you should see

```
Name: john
Password:

Last logged in on ttyA at 11:37 Thu Jun 22, 1983

% []
```

As the example shows, you should log in as yourself, and you'll be ready for the next step. If our trick doesn't work, and you're certain that the RS232 cable is plugged in correctly, you must have changed the /etc/ttys file so that ttyB is not a login port. Maybe you made ttyC a login port? If nothing is plugged in there, try connecting your cable to it instead.

If you can't login, your only option is to reset the system and run fsck. You should also add the word "login" to the entry for port ttyB in /etc/ttys, or use the recon program, so you won't get stuck again.

STEP TWO

Once you are logged in on a functioning terminal as your self, you need to identify the offending process. To start with, let's call your runaway program "mustang". To get the process number of mustang, you use the process status program, ps, to display all current processes:

```
% ps a
  PID TTY          COMMAND
    2 ttyA          -sh
    4 ttyA          update
    5 ttyA          mustang
    6 ttyB          -sh
    7 ttyB          ps
% []
```

The process number of mustang, also known as PID, is readily apparent in the output of ps as being 5. This is what we needed from step two, so onward.

STEP THREE

The final step is to kill the program. This is simply done with the command

```
% kill 5
% []
```

Look over at the runaway terminal and see if the shell prompt, usually a percent sign (%), has appeared. If it hasn't, you might need to use **ps a** to see if you got the process number wrong. Once your runaway terminal is back, log off on the terminal you are temporarily using by typing **exit**. If you only have a single terminal, you need to switch the RS232 cable back to port ttyA. Otherwise, you're done.

10.2. Replacing the Root Password

This procedure is reserved for those instances when you have somehow forgotten the root password. (Perhaps some malicious person has changed it without telling you, the "official" system administrator.) You will need the root password to backup the entire file system, create or change user accounts, check the file system with fsck, etc. And you can only change the root password when you ARE the root. A real catch-22, eh?

Well, where there's a will, there's a way. You should have a copy of the Standalone Micronix floppy diskette safely locked away somewhere. (If you don't lock the standalone diskette away, anyone can use this procedure.) With Micronix properly brought down, load the system with the Standalone Micronix floppy in the floppy drive. (This will involve booting CP/M and using the DJBOOT program, unless you have SYSGENed DJLOAD onto your Standalone diskette).

Then, we want to edit the /etc/passwd file on the hard disk. This is a three step process, with fairly simple steps, so here goes...

STEP ONE

The first step is mounting the hard disk. This is performed with a single instruction. The only trick to it is knowing which hard disk to mount. There are four possible hard disk names which you could use with the mount command. Each name is related to the size and capacity of the first hard disk in your system, your root hard disk. Here are the four possible names:

m5a	5 megabyte capacity 5 1/4" hard disk
m10a	10 megabyte capacity 5 1/4" hard disk
m16a	16 megabyte capacity 5 1/4" hard disk
hda	any capacity 8" or 14" hard disk

Choose one of these four names for the mount and umount commands that follow. Do NOT use the name "root", because when you are using the Standalone Micronix, the root device is the floppy diskette, and you want to change the passwd file on the hard disk.

Okay, insert your choice on the command line that follows (where we used m10a, insert the chosen name instead, if different):

```
# mount m10a /b
# []
```

If this doesn't work, you've chosen the wrong name. Please try again.

STEP TWO

The mount command in step one added the hard disk to directory /b of your floppy file system. Now, we can edit the passwd file, and remove the encrypted password from the entry for the root user.

```
# edit /b/etc/passwd
"/b/etc/passwd" [reading] 14 lines
:/root/
root:1"ie>3rBnvHo:0:0:Super User:/:/bin/sh
:c
root::0:0:Super User:/:/bin/sh
.
:w
"/b/etc/passwd", 14 lines
:q
# []
```

The bold face type is, as always, the letters that you will type when you edit the passwd file on the hard disk. The first editor command, "/root/", locates the line with the root user entry. The "c" command stands for change this line. On the following line, you type a new root entry without a password: just enter the line exactly as it appears in our example. A period by itself

on the next line ends the change. Then, exit the editor by typing "w" <RETURN> and "q" <RETURN> and you're finished with step two.

STEP THREE

All that you need to do now is unmount the hard disk. Just type

```
# umount m10a
# down
# []
```

replacing "m10a" with the name of your hard disk (from step one). The next command, "down" prepares Micronix for being reset. Now, you can insert your normal boot floppy and bring up Micronix. After bringing up Micronix, log-in as "root" and assign a password. Please remember to lock the Standalone diskette in a safe place, so it will be there when you need it.

10.3. Repairing the Root Hard Disk

You run the fsck program on the hard disk every day if you have been following our recommendations. This section is for problems that fsck seems to be fixing over and over again on the root hard disk. For example, while fsck was checking the I-list, (second pass), it produced the messages

```
Inode 1,  13 Directory entries, Link count 12
Inode 491,   1 Directory entries, Link count 2
```

and during the "Hunting up filenames of casualties" produced the message

```
dir491:  Directory's name changed
```

These messages will occur over and over again in this particular case because the fsck program is trying to repair Inode 1. Inode 1 will always be in memory while you are running fsck from the hard disk. So when fsck finishes fixing Inode 1 on the disk, the copy of Inode 1 in memory will be writen over the repaired Inode 1 on the hard disk.

We are sorry if this explanation is a little bit obscure. But, there are really only two things you need to understand. One, problems that fsck repeatedly reports and fixes are being stored in memory while fsck repairs the disk copy. Two, the way to clear up the problem is to run fsck from the Standalone Micronix diskette.

In the previous section, we discussed how to find out the name of your root hard disk device (as part of Step One). If you don't know the name of your root hard disk device, please refer to that section, then return here.

To run fsck from the Standalone Micronix diskette, you must start when hard disk Micronix is down. Reset the system with the Standalone Micronix diskette in the floppy drive. (This is also explained in the previous section). Once you have Standalone Micronix running, type

```
# fsck /dev/m10a
Checking /dev/m10a
...
# []
```

replacing "m10a" with the name of your root hard disk, if it is different. When fsck finishes, you can bring down Standalone Micronix by typing

```
# down
# []
```

Insert your regular load diskette and turn the key to RESET to boot Micronix.

10.4. Conclusion

No doubt we haven't covered every possible set of dire straits that you'll come into over the years, but these have been shown to be the most common. If you do encounter other situations that call for desperate measures, and they aren't the fault of defective hardware, you should let us know about them. Address correspondence to the Documentation Department, Morrow Inc., at the address shown in the front of this manual.

11. ADDING A HARD DISK

You may find it necessary or desirable to add an additional hard disk to your Micronix system. Physically, adding a hard disk means changing the cabling to your current hard disk to include an extension of the control cable to the new disk. You will also need to connect the data cable to the new hard disk, remove the terminating resister from the hard disk in the Decision, and set a jumper on the new hard disk to make it the second (or third or fourth) hard disk. The details for connecting the hardware should be included with your new hard disk.

Once you have installed the new hard disk, you will need to format it. Please follow the instructions for formatting that are in the INSTALLATION section of this guide.

You next need to add a Micronix file system to the formatted hard disk. This is done with a single command, **mkfs**. For example, if you have added a 16 megabyte hard disk as the second hard disk in your system, you type

```
% mkfs /dev/m16b
```

to create a file system on it. The third 16 megabyte hard disk is called m16c, the fourth is named m16d. If you were adding a 5 or 10 megabyte hard disk instead, you would be using m5b or m10b for the second hard disk. You can only have one "a" drive, one "b" drive, etc., regardless of the drives' capacity.

The next step is to check the new file system. The fsck program is used with the same name as you used with the mkfs command.

```
% fsck /dev/m16b
```

The new hard disk can now be mounted on the root file system. For example,

```
% mount m16b /b
```

mounts the second hard disk (that happens to have a 16 megabyte capacity) on directory /b.

The best way to handle checking and mounting additional hard disks is to add these commands to the /ect/rc file. This is discussed back in the section on Customizing the Environment. Having these commands in the /etc/rc file makes them automatic each time the system goes multiuser.

12. CONSOLE ERROR MESSAGES

Micronix is designed so that system errors are reported to the console. The error messages will appear on the console terminal regardless of the current task being performed on the console. This may result in the error message appearing in the middle of the console screen during the editing of a file. If this happens, don't worry about the message being added to the file you are editing. Your real concern at this point is dealing with the problem that caused the error message.

Some of the error messages received are simply warnings; others indicate that a serious problem has occurred, and the system is bringing itself down. Most of the error messages that you are likely to see on your console are warnings about running out of space on a disk device, or disk read or write error reports. Other errors reflect trouble occurring in the operation of Micronix.

There are three classes of console error messages: file system warnings, operating system warnings and fatal error messages. The three sections that follow contain information about the interpretation of these messages and the responses that are needed to remedy the situation.

12.1. File System Warning Messages

File system warning messages are reports about the file system(s) that you are using. As we mentioned previously, all messages will appear on the console terminal for any mounted file system. Each message will tell you which disk the file system with the problem is resident on. For example,

```
No more space on disk hddma/8
```

means that you have no free space left on the first 16 megabyte hard disk in your system. The "No more space on disk" part of the message is clear enough. Translating "hddma/8" into the device name /dev/m16a is a little obscure at first.

When you get a file system warning message, the disks will be named according to the controller board used for the disk. There are three different controller boards used:

djdma	floppy disk controller
hdca	8" and 14" hard disk controller
hddma	5 1/4" hard disk controller

These names will make up the beginning of the disk name. The slash and number following the name refer to something known as the minor device number. The minor device number is used to select which type and number of drive. Complete information on the major and minor device numbers appears in the Device section of the binder, for those of you who are interested. This is also part of the online documentation, so you can find out what

hddma/4 means in absolute terms by typing "man hddma".

For the rest of us, the following tables provide the information needed to interpret which drive the message refers to, for systems with multiple drives using the same controller. If you only have one floppy disk drive, a disk designated as djdma/140 will always be the floppy disk drive, regardless of the number that follows the slash after the controller name. The same is true for hard disks, that is, if you have a single 10 megabyte hard disk, disk error messages that refer to hddma/4 means your only hard disk, of course.

There are three tables that you might need. The table for the HDCA controller is very simple, since the HDCA controller can detect the drive capacity, the number after the slash (hdca/1, for example), refers to which of four hard disks the message refers to.

HDCA

hda	hdb	hdc	hdd
0	1	2	3

The table for the hddma controller is a bit larger than the hdca table because different capacity 5 1/4" drives are indistinguishable to the controller, and must be specified by minor device numbers. The following table reproduces the minor device numbers that you will see as part of file system warning messages:

HDDMA TABLE

5 megabytes		10 megabytes		16 megabytes	
0	m5a	4	m10a	8	m16a
1	m5b	5	m10b	9	m16b
2	m5c	6	m10c	10	m16c
3	m5d	7	m10d	11	m16d

This table includes the device name following the minor device number (that is, minor number 4 represents /dev/m10a). By now, you should be a pro at deciphering things like hddma/9, which is the second 16 megabyte miniwinchester, /dev/m16b, right?

Now, let's look at the DJDMA table, which is the most difficult one to use. The numbers that will appear after the slash come from an 8 bit byte where all of the bits are used. This tends to hide the minor device number, which is what we are interested in. Of course, if you have a single floppy drive system, you don't need this table. Any error message that refers to djdma/"anything" means your only floppy drive has a problem.

DJDMA TABLE

drive #	8" drives	5 1/4" drives
0	000	100
1	001	101
2	010	110
3	011	111

This table represents the lowest three bits that will appear in the disk designation for the djdma in error messages. The lowest two bits stand for the drive number (notice how these bits are repeated in both columns), and the third bit stands for an 8" drive if it is zero, and a 5 1/4" drive if it's a 1. The number that will appear after the slash is, unfortuately, in decimal, so you will need to convert it to binary to make sense out of it.

Okay, so now we have a nice table for interpreting which floppy disk is being referred to. Let's try it out with an example. Suppose the message

No more space on disk djdma/140

appeared on the console. The decimal number 140 is 10001100 in binary, making the last three bits 100. If you look in the table, you will see that the number "100" appears in the column under 5 1/4", and in the row labeled "drive 0". So, the message means that the file system located in the 5 1/4" floppy drive 0 is full. If the message had refered to djdma/137, you should be able to translate this as 8" drive 1 by using the table. (Hint: if you have just two drives of the same type, an even numbered message means drive 0, and an odd numbered message drive 1.)

Let's proceed to understanding the file system warning messages and what to do about them.

There are currently five file system warning messages. The end of each of these messages will identify the disk where the error occurred, which we just explained how to translate. The messages are:

- No more space - There are no more free blocks available for writing to the disk. This is what happens if you don't keep your hard disk clean, or use df to check for free space.

  Response - You must remove some files to free up some disk space. If you know of some large files that you can delete, you're all set. More likely, however, you will have to go looking for something to get rid of. The critical thing here is that noone can write to the file system until you free up some space. If this problem occurs often, you need to be more disciplined about managing free disk space. The find command can help you discover unused or large files that can be transferred to backups and removed from the disk.

The files that were being written to disk may have been partially copied when the disk became full. Uncompleted transfers will complete if you are able to free up disk space without stopping any ongoing processes. You can, for example, login on another port (as outlined in Step 1 of Runaway Terminals, in Desparate Measures) to remove or backup and remove files.

- Out of range block number - A block address is discovered in a file (inode) or the free list that cannot possibly exist in the file system on disk. This is one of the things that fsck discovers and fixes. If the bad block number is in the free list, the entire free list will be scrapped.

  Response - Unmount the file system where the error occurred and run fsck on it. If the error occured in the root file system, you will need to kick the other users off the system, bring the system down, reboot Micronix and run fsck on the root device. Fsck will rebuild the free list if necessary.

- No more directory space - This is similar to the "No more space on disk" message. Micronix requested a free inode for creating a file, but there are no free inodes.

  Response - The response is the same as for "No more space": remove some files to free up some inodes. Inodes are the file descriptors used by Micronix. Please refer to the previous response for out of space.

- Bad free list - An illegal block number has been discovered in the free list. The free list will be scratched, and you may soon see a "No more space" message because the free list is empty.

  Response - Your response should be the same as for an "Out of range block": run fsck. Follow instructions in the earlier response for "Out of range block".

- Block #n io error - A hard error has occurred while trying to read or write this block number (#n). This is your basic disk error report.

  Response - Once again, run fsck, this time with the -t option. Follow the instructions listed in the response for "Out of range block."

You may have noticed in reading the responses that running the fsck program is the response of choice for everything except full disks. This is why we suggest that you use fsck as a beginning-of-the-work-day ritual.

And, when you use fsck, make sure that no one is using the file system being repaired. If your are fixing a mounted file system, unmount it first. If it is the root file system, you

must get people to log off, bring down the system, reboot and run fsck. If you use fsck while the file system is active, it will create problems rather than fix them.

12.2. Operating System Warning Messages

This second group of messages refers mainly to difficulties encountered by Micronix that are not specifically file system related, but involve instead ripples in the normally transparent operation of Micronix. Most of these messages report that the system has reached its operating capacity. A couple deal with memory problems, or non-file system disk problems. These messages and the necessary responses are explained in the text that follows.

- Process table is full! - During the boot up process, you may have noticed that your system has "20 processes". When the process table is full, you already are using all the processes that you can start, and no more processes can be started. This means that no program can be run until some other process completes. Built in commands, like dir, cd and era, will still work.

  Response - You must wait for some of the processes to complete. If you are currently logged in as root, you will also be able to kill processes when you are desperate. The long range solution is to reduce the system's workload.

- Bad inumber - An out of range inode number has been encountered. Micronix will discard the number, which was probably part of the free inode list or a system call to open the (non-existent) inode.

  Response - Once again, fsck is the best bet for clearing up the problem. It is possible that the bad inode number is from a directory entry pointing to empty space. Fsck will discover the identity of this file and report that "File's data lost". Recover the file from backups. If the bad inode number is from the free inode list, the problem may be corrected when Micronix discards the corrupt free inode list and rebuilds another (automatically).

- Memory allocation fault - This indicates a failure in the software or hardware that handles memory allocation.

  Response - No action is necessary, unless the problem is recurrent. In that case, contact your service representative, and tell him about the problem.

- Out of swap space, or Swap space full - This is similar to having a full process table. The space allocated on disk (distinct from the file system) is already full of waiting processes, and no more processes can be started.

  Response - Nothing can be done except to wait for some

process to finish and give up its space. The superuser has the option of trying to kill a process to make room.

- Swap error: process #i killed, or Swap error: process #i hung - These messages both mean that a read or write error occurred in the swap region of the disk. If the message says "process #i killed", the process that was copied to the bad area in the swap space was successfully killed. However, if the process is "hung", essential information was lost in the swap area for killing the process. In this case, your system will be have room for one less process.

  Response - If this is a recurrent problem, the solution is to backup the hard disk, reformat it, and rebuild it from backups. The procedure for rebuilding Micronix on a hard disk is contained in the Restoring the Hard Disk section. Reformatting may correct the problem with the disk outright, or will cause the badblock to be mapped out.

- Bad memory in the following 4K segments (in hex):
  (These segments will not be used by the system) - This message will only occur during the boot-up process. It means that a 4096 byte region in memory failed a memory test and will not be used by Micronix.

  Response - Get the memory repaired.

In cases where you may wish to use the su command to become the superuser and free up some process space, if there are no more processes available, or no swap space, the su command will fail because it requires the creation of a process. Only if someone is currently logged in as root will you be able to kill processes to free up space. Of courese, anyone may kill their own process if they are running in background at the time. But only the superuser may kill other persons' processes.

12.3. Fatal Error Messages

These messages are known as "Panic messages" in standard UNIX parlance. Micronix has removed the panic aspect by handling system shutdown in a graceful manner, giving you a chance to calmly reboot your system. In many cases, rebooting the system and using fsck will solve your problem. In some cases, the fatal error was brought about by a hardware fault that will have to be repaired before the system will work again with the hard disk. When the hard disk seems to be the problem, go back to the sections in the INSTALLATION part of this guide for checking out the hard disk from CP/M.

Fatal error messages will always have the form:

```
Something's wrong: [ type of trouble ]
To prevent damage, the system is going down.
Please reboot, check your file system, and try again.
If the problem recurs, please contact your dealer
```

The importance of running fsck after rebooting can not be overstressed. Something is definitely wrong, and the file system was very likely damaged by its sudden termination. Run fsck if you can reboot.

There are eight different type of trouble messages, most of which refer to problems with the hard disk with the root file system and the swap area on it. For most installations, this will be your only hard disk, or your first hard disk. The messages are:

```
can't get superblock
can't get root directory
can't mount root directory
can't open root device
can't open swap device
hddma hung
Timeout slots are full
```

The first five messages mean the the root file system (or the swap area) was not successfully accessed. If this occurs, and rebooting fails, you may try the Restoring the Hard Disks procedures for Installing Micronix on the Hard Disk after formatting.

The last type of problem is called "Not enough memory to run". Micronix requires an absolute minimum of 128K bytes to boot, and 192K bytes to run a program. This type of trouble will only occur during the boot process, and indicates that you have hardware problems with your memory (or that someone has borrowed your memory without telling you.) Find or repair your missing memories.

A. HARDWARE ADDENDUM FOR MICRONIX SYSTEMS

A.1 Removing the Decision Cover

1. DISCONNECT the Decision power cord from the AC power receptacle on the back panel.

2. Locate the four (6 x 32) screws on the base of the computer. Figure 11-1 shows the underside of the computer and the location of these screws.

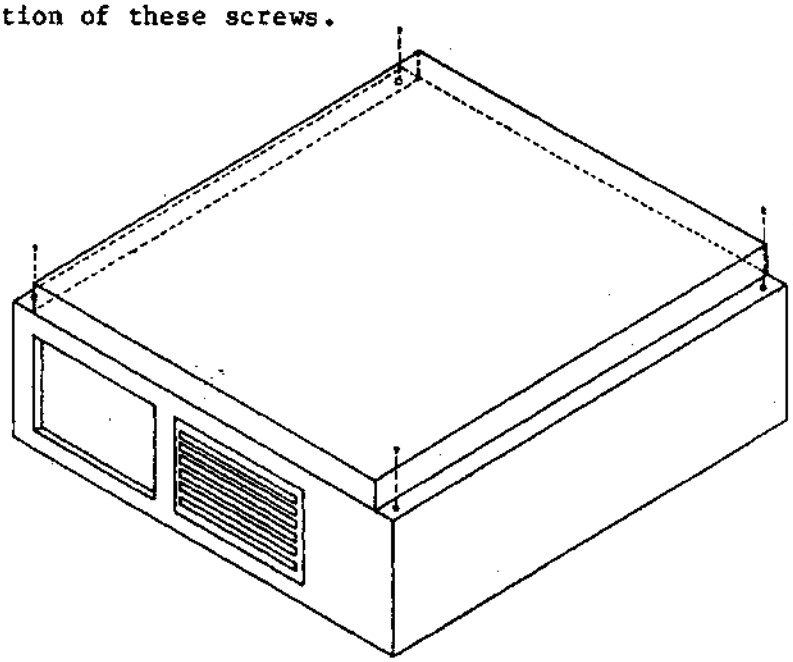


Figure A-1: Decision Cover Screws

3. Remove the screws and put them in a place where they are not likely to drop into the Decision.

4. Slide the cover toward the front of the computer, and then lift the cover off.

WARNING:

> The Decision contains hazardous voltages inside. Extreme care should be taken when installing cables and components within this system. Never open the cover unless the AC power cord is disconnected to prevent electrical shock.

A.2 Checking Internal Connections

The Decision is shipped from the factory with all internal connections in place, but sometimes connectors are jarred loose during shipping. If you are unable to successfully power on the Decision after installation, we recommend that you remove the cover, check the disk drive connectors, and make sure that the S-100 module boards are securely seated in the motherboard.

The disk drive connectors should be securely connected to the connectors provided on the drive controller boards. The next figure illustrates proper disk drive cabling.

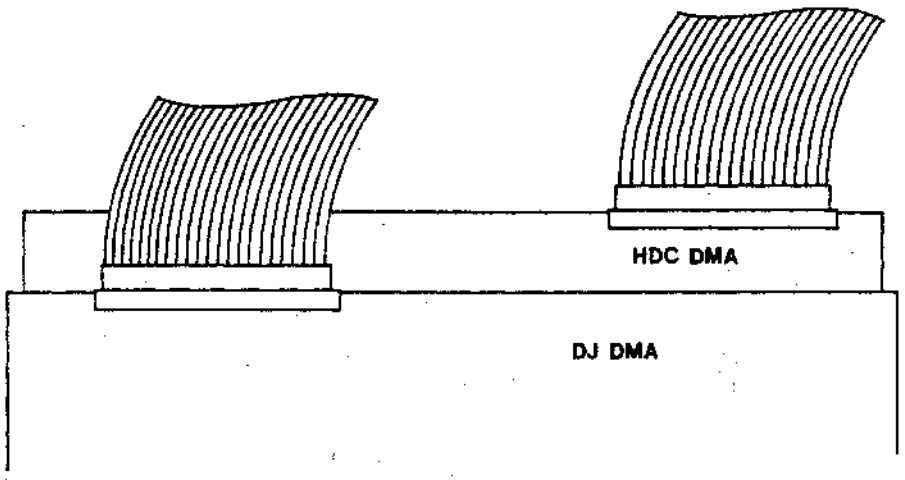


Figure A-2: Disk Drive Cable Connections

If the cable connection(s) does not appear as in the illustration, remove the cable CAREFULLY. Turn it over and reconnect it. Be careful not to bend any of the pins in the process. The cables will always extend over the back, that is, side without parts, of the disk controller boards.

If you are satisfied that all connections are made properly, but are still unable to get your system to operate correctly, contact your dealer.

A.3 Removing the S-100 Module Boards

The S-100 module boards (the printed circuit boards you see when you lift off the cover) are secured by a retaining bracket and four screws (see Figure A-1). To ensure that these boards are properly seated in the 14-slot motherboard, or if you find it necessary to remove one or some of the boards (to enable an automatic hard disk boot, for example), you should follow the procedure outlined below:

1. Make sure all power to the computer is OFF, and that the Decision is unplugged.

2. Remove the Decision cover. (See Section A.1)

3. Remove the four screws holding the retaining brackets and lift the brackets off.

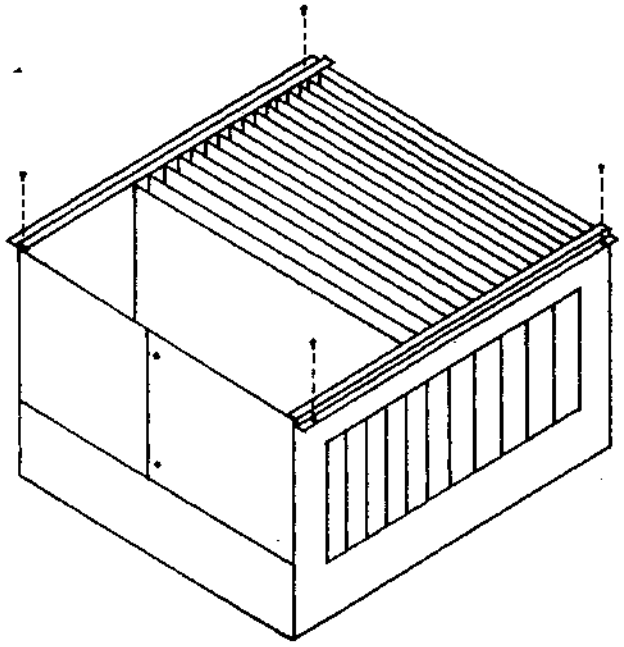


Figure A-3: Retaining Brackets for S-100 Module Boards

4. Grasp the top corners of the board and carefully pull the desired board out of the motherboard (gently but firmly).

5. To return the board to the card cage, slide the S-100 module board into any available slot. It need not be the slot from which you removed the board.

6. Reseat the board by gently rocking the board from side to side while firmly pressing down.

5. When you have finished the hardware reconfiguration, make sure that any disconnected cables have been properly reconnected (see Section A.2) and the boards are well seated in the motherboard.

8. Return the retaining brackets to their original location (see Figure A-3).

9. Return the Decision top cover.

A.4 Modifications for Automatic Hard Disk Boot

If you wish to enable the automatic hard disk boot of the Decision, you must change the settings of Switch 16D on the Decision MPZ80 CPU board. This board is illustrated in Figure A-4 ; the required switch settings are provided in Figure A-5 . Note that switches may be protected by plastic covers; these covers must be removed to change the switch settings, but should be put back on the switch after the changes have been made.

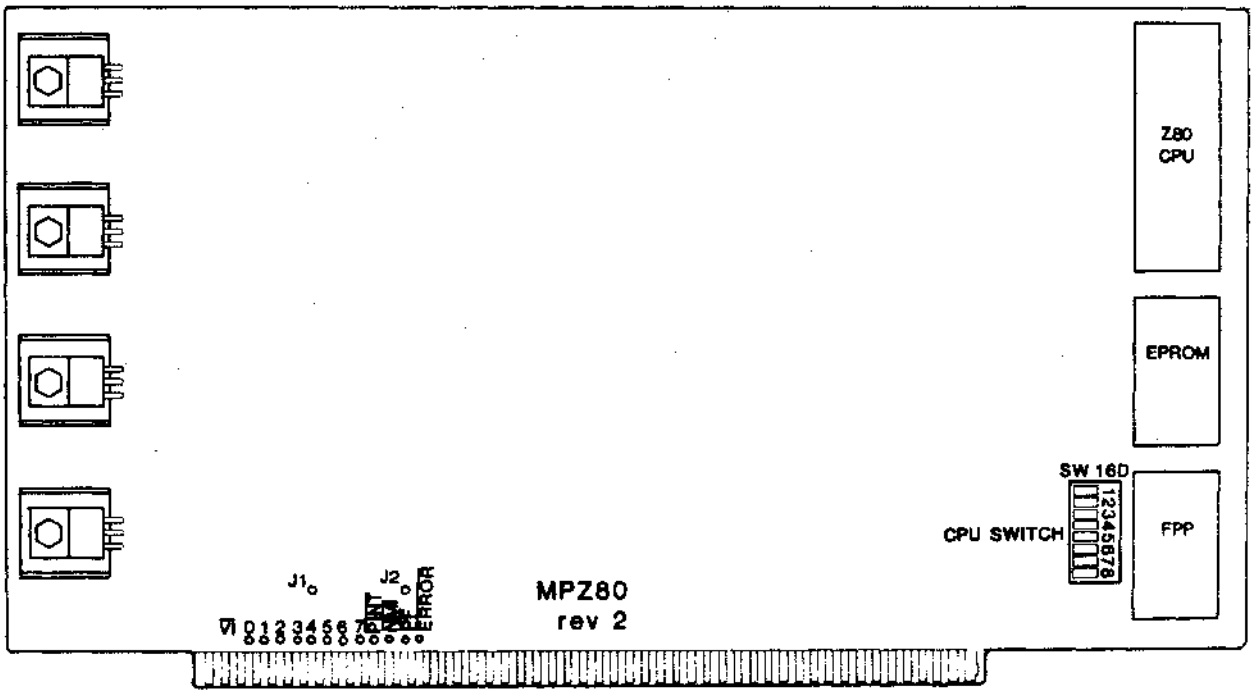
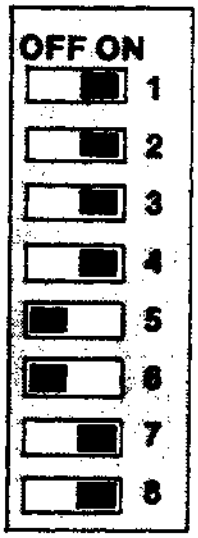


Figure A-4: Decision MPZ80 CPU Board

NOTE:

When first applying power to the computer, the hard disk performs a 4 second warm up/diagnostic. There will be a slight delay between resetting the system, and the sign-on message that appears. Subsequent resets will immediately display the sign-on message.



To enable an automatic hard disk boot, Paddle 4 must be ON and Paddle 5 must be OFF on the MPZ80 (see Figure A-5).

The DJDMA board must also be modified to enable an automatic hard disk boot. A shunt must be installed between jumpers A and B at J2 (near bottom of board).

Figure A-5: Automatic Hard Disk Boot

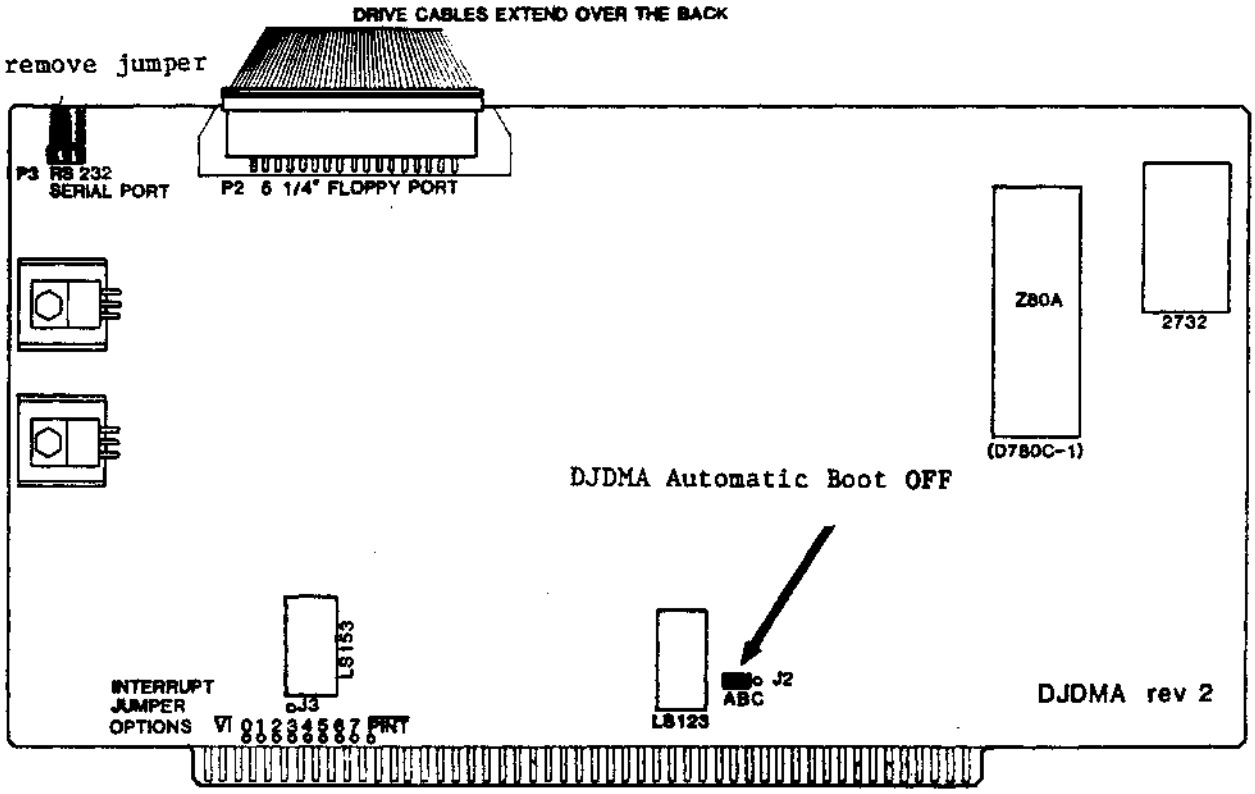


Figure A-6: DJDMA Board Configured for Automatic Hard Disk Boot

If you decide to return to an automatic floppy disk boot, the board configuration process is reversed. Switch 16D of the MPZ80 CPU board should be set as illustrated in Figure A-7 . Paddle 4 must be OFF and Paddle 5 must be ON.

The DJDMA controller board must also be modified; a shunt must be in place to jumper B to C. The shunt on P3, connecting pins 2 and 3 must be in place.

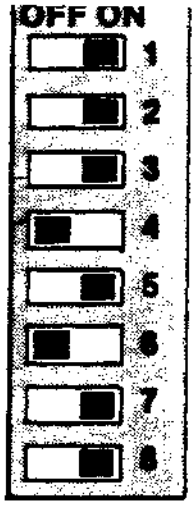


Figure A-7: Automatic Floppy Disk Boot

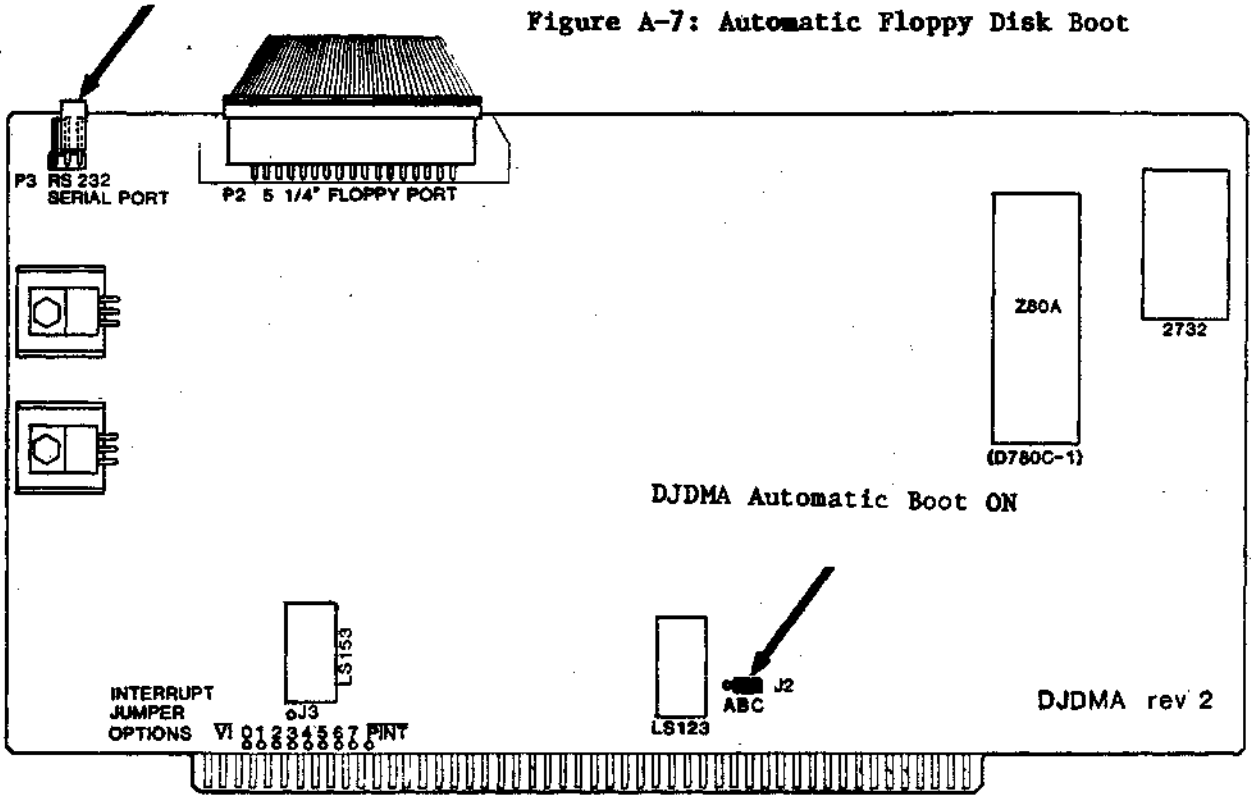


Figure A-8: DJDMA Board Configured for Automatic Floppy Disk Boot

Refer to the MPZ80 section of the Decision CPU Technical Manual for further details.

A.5 Installing Additional Disk Drives

A.5.1 General Information

Up to four drives of the same type (i.e. four floppy disk drives or four hard disk drives) may be connected to each of the drive controller boards installed in your Decision. This manual, however, will only discuss the addition of one 8 inch floppy disk drive. If you wish to install any other type of drive, refer to the "Add-On" documentation supplied by Morrow with your drive.

A.5.2 Connecting an 8 Inch Floppy Disk Drive

Eight inch floppy disk drives are connected externally, through the 50 pin connector just above the daisy printer connector on the rear panel of the computer. The cable required for this connection is available from Morrow. Specify Morrow part number 060-50782P when ordering.

1. Plug one end of the cable into the Decision rear panel 50 pin connector, just above a similar 50 pin connector for the daisy printer port. The ribbon cable should come out of the bottom of the connector, and the stripe on the cable should be on the left (looked at from the back of the Decision).

2. Plug the other end of the cable into the corresponding pin connector on the back panel of the disk drive. The ribbon cable should come out of the bottom of the connector, and the stripe on the cable will be on the left (looked at from behind the drive).

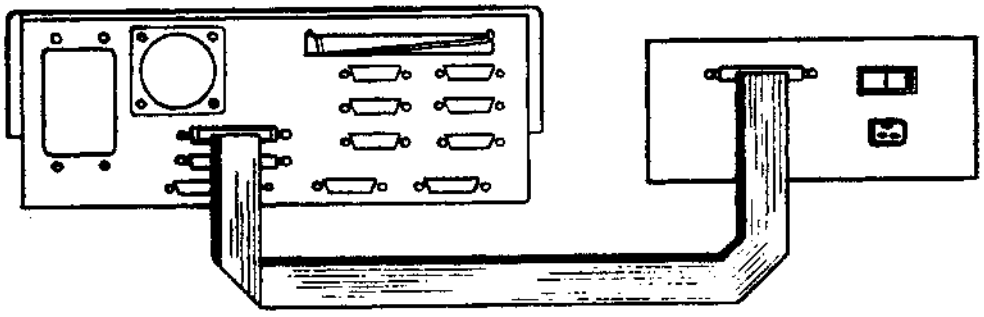


Figure A-9: Connecting an 8" Floppy Disk Drive

3. The 8" drive can now be accessed as /dev/fla.

4. If you have added a dual 8" floppy disk drive unit to your system, the left drive will be accessed as /dev/fla; the right drive will be referred to as /dev/flb from Micronix.

5. The last drive connected to the cable must be terminated. All drives purchased from Morrow will have termination on the last (or only) drive. If, after you add an 8" drive, you start getting write errors, there is a problem with termination on the last drive. The pins on the termination resister for WRITE GATE and WRITE DATA must be lifted. If you are unable to located these pins, call Customer Service for assistance.

Micronix Operating System
user's manual

tutorials

MORROW

Micronix Operating System

User's Manual

Tutorials

Table of Contents

1. INTRODUCTION

In this section you will be given a "hands on" introduction to the Micronix operating system. The tutorials in this section were designed to help you see exactly how the Micronix operating system works. Read through the following pages first, then begin the tutorials.

Five main areas are covered in this section:

1. Using the Computer. Entering commands, using special function keys and the type-ahead feature.

2. Entering the system. Logging in, establishing passwords, directories and files, logging out.

3. Using the system. Using mail and message features, and accessing files.

4. Writing and editing text. Using the Micronix text editor and format commands to create written text.

5. Programming. Using languages and programs included in the Micronix Operating System.

Begin your introduction to Micronix now by reading through the text and following the instructions given.

1.1. Some Basics

Let us begin with a few simple things to keep in mind while learning to use this system.

Micronix is geared toward lower case letters. This is especially important when entering commands. This does not pertain to text being entered, just remember that commands are entered into Micronix using lower case letters.

Using a computer terminal is similar to using an electric typewriter. At first glance, the computer terminal keyboard even looks like the typewriter keyboard. One common function in all electric typewriters is the RETURN key, pressed at the end of each line to return the carriage. This key is also used on terminal keyboards. Like lines typed on a typewriter, after each command, line of commands and lines of text entered on the computer, the RETURN key must be pressed to effectively terminate the line. This process is not always specified in the tutorials, just remember: Press RETURN at the end of each line entered.

One key found on computer terminals and not on typewriters, however,is the control (CTRL) key. Sometimes this key is

used to carry out a certain process. This process is performed much like a shift sequence to enter a capital letter - hold down the CTRL key, then press the indicated letter. As a form of shorthand, control sequences are written with a "^" and the letter key. An example of this is the ^X sequence used to erase a line.

If you happen to enter something incorrectly at the terminal, there are two ways you may correct it: 1) If it is a short (two or three character) entry, use the BACK SPACE key (if this key exists on your terminal) to back up over your mistake, then retype the entry. 2) If it is a long entry, use the ^X sequence mentioned above. This deletes the line and jumps the cursor to the next line so you may begin all over again. If your terminal doesn't have a BACK SPACE key, enter ^H. This also back spaces the cursor.

Micronix also has a type-ahead feature. This means you may type as fast as you want or enter several commands one after another without waiting for the system to catch up. This may look confusing on the screen, but the characters will be stored and interpreted in correct order.

1.2. Logging In

Before you can enter Micronix, your login name must be entered in the system. Once the system operator installs this, logging in is easy. If the initial header is not displayed on your terminal screen, try pressing the RETURN key. Type in your login name after the name prompt.

```
Name:
```

Press the RETURN key. Micronix responds with a message that resembles the following:

```
Welcome to Micronix.
```

This is an initial welcome message. After the first login, users are greeted with a message similar to the example below:

```
Last login on tty# on (date and time of last login)
```

1.3. Passwords

Passwords may be entered by using the password command. Passwords are used mainly in larger, multi-user systems. If your setup requires passwords, enter the mnemonic **passwd** after the shell prompt (%):

```
% passwd
```

The system returns with

```
Name:
```

Enter your established login name. The system returns with

```
New Password (or RETURN for no password):
```

Type in your password, but do not be surprised that it does not appear on the screen. Instead the system prompts

```
Once again to be sure:
```

Type in the password again exactly as you previously entered it. You may get a message warning you that your password is too short. A "secure" password is at least eight characters in length.

Once your password is entered, the shell prompt (%) is returned to the screen and you may resume work in the program. If you entered a password, you must enter it now every time you log in.

1.4. Logging Out

Pretend you are finished using the system for the day. To exit Micronix, type in

```
% exit
```

The system has logged you out and returns now with the login header.

Log in again. (If you created a password, you must enter it, too. If you lose your password, the system supervisor can help you enter a new password.)

1.5. Mail

Micronix has an inter-user mail service. Sometimes when you log in you may get the message

```
You have mail.
```

Mail provides a convenient means of conveying messages to other system users. If you receive a mail message, simply enter

```
% mail
```

Your letter appears on the screen. At the end of the letter, this message appears:

Do you wish to keep your mail?

Type yes or no. Kept mail is stored in a file called "mbox".

To send a letter to another system user, enter the following:

% mail (user's login name)

Type your letter. After the last line of the letter enter a ^D alone at the beginning of a line. This automatically sends the letter to the specified user. You may also use this feature to send yourself mail. This is a convenient way to remind yourself of appointments - just enter your name after the mail command.

In fact, why not try the mail feature now and write yourself a letter?

1.6. On-Screen Messages

It is also possible to send messages via the terminals. The format for sending messages looks like this:

% write sally

The designated user receives a message on his screen that says

% Message from tom tty# (time and date)

To establish a two-way communication path with Tom, Sally would enter

% .bd write tom

Otherwise, the message from Tom is only one way - to Sally. Messages are typed on the screen. Each time the RETURN button is pressed, the typed line is sent to the designated person. This process is terminated by entering ^D.

1.7. Checking Out the System

Take a look now at what is available on your system. One simple check is to find out who else is logged on. To do this, enter

% who

A display of system users is returned in the following format:

```
root    console   day      time   year
tom     tty2      jan 25 15:10:30 1980
```

Another easy check is for the date. Enter

```
% date
```

The system responds with the current date and time.

Next ask for a listing of your directory:

```
% ls
```

The response here will depend on whether any directories have been made. Next ask for your present working directory. (Very helpful when you want to find out where you are within the system.)

```
% pwd
```

This display is simply

```
/a/ (name of a user, like "tom")
```

Change directories to "root" (/), then list this directory

```
% cd /
% ls
```

You should get a list that looks something like

```
bin     etc     lib     install     usr
a       hdunix  cpm     finstall    hdinit
b       dev     include tmp         djunix
```

Move down another branch in the tree to display files in this directory. Change directories to "bin" and again display the files.

```
% cd bin
% ls
```

The result should be a long list of file names. This directory is where many often-used commands live. Feel free to explore them.

1.8. User Directories

Once you begin making files, you will need a place to store them. In Micronix (and all UNIX systems) files are stored in directories. It is suggested you name directories so they relate to the work you are doing. For example, you may want to store form letters in a directory called "letters". Or perhaps you may want to store notes for a manual in a directory called "prelim", which are then transferred to one called "book" when the manual is finished.

To make a directory called "letters" you would enter:

```
% cd
% mkdir letters
```

To remove a directory, you would enter

```
% rm directory/filename
% rm directory
```

The first command removed the file from the directory, the second command removed the directory. (It is not possible to remove a directory with files in it.)

This brief introduction should give you an idea of how Micronix works and help you to continue with the following tutorials.

Additional tutorials are also included in the User's Guide to the Unix System handbook included with your system. Read through this book and try some of them - you'll find that most of them work with this system!

2. CP/M PROGRAMS UNDER MICRONIX

Micronix offers a CP/M look-alike, **upm**, which provides a close functional replica of the environment that application programs written to run under CP/M expect. Most CP/M programs function unchanged under Micronix.

All of the built-in commands found under CP/M are available in **upm**, as are the transient commands and file reference characters (* and ?). The user will find that **upm** offers all the functions expected from CP/M - file transfer, "housekeeping" functions, program assembly and debugging and system utilities.

The user must keep in mind, however, that the intended purpose of **upm** is to bridge the gap between low-cost multi-user systems and the enormous body of special-purpose

software and application programs written for the CP/M system, thus offering true multi-user function and CP/M.

The organization of .bd upm differs from that of CP/M. Under Micronix, there are no records, extents, disk drives (in the CP/M sense), nor are there file control blocks (FCBs), disk parameter blocks (DPBs), allocation vectors, etc. When the upm program opens a CP/M file, it is really opening a Micronix file. Upm handles conversion back and forth between CP/M FCBs and Micronix file path names by maintaining a table of correspondences between "drives" and Micronix directories. (Keep in mind that CP/M is more drive-oriented than Micronix. Drive designations in upm are actually directed to files within Micronix. Typically, these files contain the "cpm" file and user directories, but may also be device files.) File I/O is monitored and the extent numbers are changed at the appropriate times by upm.

Upm also contains a "Bios jump table" bearing a striking resemblance to the original, and corresponding in function. In fact, once you begin running your application programs under upm, you should find yourself in a familiar environment. The operation of upm becomes invisible. The documentation of your application programs applies.

Upm does have some limitations: Programs attempting 8080 I/O instruction or interrupt manipulation (such as a program to format the hard disk) will not work, nor will those that "move" a conventional CP/M system, such as MOVCPM or SYSGEN. If a CP/M program does go awry, however, it will have only a local effect; the rest of the operating system will continue running.

Current free space in upm stands at 52,064 bytes. Your CP/M program must fit in this space.

It is also possible to execute Micronix commands within upm by temporarily escaping to the shell program. This is done by preceding Micronix commands with an exclamation point (!). For example

A>!df /dev/dja .sp would display the number of free blocks (1 block = 512 bytes) available on drive A (much like the CP/M STAT command).

2.1. Entering the "CP/M" Mode

To access the upm program, enter

% upm

which returns:

```
CP/M Version 2.2 emulator
For Morrow Designs Decision I under Micronix
With the following Drive designations

A: ->./
A>
```

Upm is in the current directory. You may now re-designate this "drive" and/or designate others.

To effectively use upm it is necessary to understand the relationship between upm's concept of drive letters and Micronix' directories.

Upm maintains the 16 standard CP/M drives (A: - P:). Since the most natural analogous structure in Micronix is the directory, upm maintains a table of CP/M-like "drive" designations and the names of the Micronix directories to which they correspond.

You may change the drive designation pointers with a command of the form

```
A:dirname
```

so that A is replaced by any of the "drives" (A: through P:) and "dirname" must contain at least one slant character (/):

```
A> b:./
```

The above example would point "drive B" at your current directory.

One means of entering the upm program is to address the "cpm" and user directories:

```
A> a:/cpm
```

returns

```
A>  ->/cpm
A>

A> b:./
A> c:/a/(user name)
```

returns

```
A: ->/cpm
B: ->./
C: ->/a/user name
```

The above display is automatic. It is also possible to get an automatic display of the Micronix-upm relationship by entering an equal (=) sign after the A> prompt. .sp To obtain a listing of files within these designations, enter the CP/M directory (DIR) command:

A> dir a:

(lists cpm)

A> dir b:

(lists ".")

A> dir c:

(lists /a/user name)

You may continue addressing drive designations for all 16 "drives" with as many directories as you wish.

It is possible to have two or more drive letters refer to the same Micronix directory. As an example

A:/cpm
B:/cpm
C:/cpm

are all valid drive designations.

2.2. Upm Modes

Upm also has two distinct modes, DIRECT and INTERACTIVE. If you intend to run only one CP/M application program, enter in the DIRECT mode. If you intend to run a series of CP/M programs, enter in the INTERACTIVE mode.

To enter the INTERACTIVE mode, type

% upm

alone on a line.

To enter the DIRECT mode, type

% upm program name

It is also possible to enter **upm**, designate the "drive" and call a program. These drive designation modifiers may appear as commands to the **upm** "command console processor" (CCP), or as a command line. In this example,

```
% upm a:/cpm b:./ b: a:ws
```

the user enters upm with "Drive A" corresponding to the Micronix "cpm" file and "Drive B" to directory ".", then directly enters the Wordstar program.

Exit from upm's DIRECT mode occurs immediately upon completion of the called application program. You may exit from INTERACTIVE mode by typing exit.

2.3. Creation of Text Files

Herein are some practical suggestions on the handling and transportation of text files between upm and Micronix. We will begin with a description of the differences in their formats.

2.3.1. CP/M-Micronix Differences

Micronix text files and files created under upm may appear the same, but there are differences:

1. At the end of each line of a Micronix file there is a single NEWLINE character; at the end of each line of a CP/M text file there is a RETURN character followed by a NEWLINE character.

2. A CP/M file's length must be an even multiple of 128.

3. A Micronix file's length may be any number of bytes (although both are constrained to maximum file sizes).

4. CP/M text files are padded with ^Z characters at the end to round them out to an even multiple of 128; Micronix text files are not.

There is a convenient utility for converting a text file back and forth between the two formats. To make a text file suitable for Micronix, type

```
% clean filename
```

To make a text file suitable for CP/M, type

```
% clean -u filename
```

The idea is to "clean" out the excess RETURN and ^Z characters at the end of each line, "-u" stands for "unclean".

2.3.2. Naming and Accessing Files in upm

A file that is created in the user's directory under Micronix appears under **upm**.

Files in the root or other Micronix directories may not be accessible to CP/M programs under **upm** because they have been "coded" by the **chmod** program (1), making them inaccessible to the user, or giving the user read-only privileges. Trying to call these files will result in an error message indicating either the file does not exist or space to write the file does not exist. Wordstar, for example, cannot conceive of the file differences in Micronix; these error messages may at first be disconcerting to a user who is secure in the fact that a file, or space for the file, .ul does exist on his system.

Files entered in uppercase letters are "invisible" to **upm**, and directories and special files are off limits. (It is possible, however, to create a new file in a CP/M program with the same name as a directory or special file. Be careful when naming your files!)

2.3.3 Printing CP/M Files

CP/M files may be printed by directing them to an output file addressed by the CP/M LST: device. This may be done upon entering the **upm** mode by following this example:

```
A>LST:output.fil
```

Executing a print command (the P File Command in Wordstar, for example) sends the output to the file addressed by LST:. This output file may then be directed to the **lpr** program for printing, following the standard Micronix format for printing files:

```
% lpr output.fil
```

The procedure for this is also described in the upm (1) program description in the first section of the Reference Manual.

2.4. Transferring Files

Suppose now you want to enter a CP/M program into Micronix that just arrived from one of the many companies that develop CP/M application programs. Copy over the contents of the new diskette using the **far** program.

To copy the contents of a CP/M diskette to the current Micronix directory, place the diskette in drive A and type

```
% far dja -xv
```

Far displays a list of files it is copying. Refer also to the far program in Section 1 of the reference manual for a further description of the flags and reports available.

Once your program exists in Micronix, simply address this file in the upm program. Unless this program is system-dependent, it should run flawlessly under the Micronix CP/M emulator. (For further information on upm see Section 1 of the Reference Manual.)

3. ADDRESSING DISK DEVICES: mount

During the initial software set up, you transferred several diskettes containing the Micronix system onto your hard disk. Once this process was complete, you had the beginnings of Micronix' basic file structure on your disk system. This structure begins with the "root" directory. Branching from root are directories and files, some that came with your system, some that you have created or will create, to form the Micronix "tree" structure.

The device upon which your system began and will continue to grow is the "root device". Micronix is structured such that the root device can never be detached from the file system, but the mount program can be used to extend the Micronix file system to include additional disk storage devices. Mount makes a temporary logical attachment or association between these devices and the Micronix file tree.

Section 1.3. of the Orientation portion of this binder described three basic types of Micronix files: ordinary files, directories and device files. Device files are the special files that mount uses to associate a directory with a specific disk device. These files reside in the "/dev" directory and are of two types: 1) character special and 2) block special. Character special files refer to devices that output characters, such as terminals and printers. Block special files refer to devices that read out large sections, or blocks of information, such as disk storage systems. Mount is used only with block device files.

When one of these files is "mounted," mount informs the system that the associated device contains a file system. The Micronix "tree" structure is temporarily extended during the time that the device is mounted. In order to preserve the integrity of the file system during this time, Micronix imposes certain "rules" which must be observed while using

mount:

- Mounting a device on the root directory temporarily "clears" all files that previously existed on root. To avoid any problems, do not mount a device on root (/).

- A device can only be mounted once. Trying to mount a device already mounted results in an error message. It is possible, though, to re-mount a device that has been unmounted.

- An error message is also displayed if a user tries to unmount a device while someone else is using it.

- A device may also be mounted with a read-only argument. The system is then prevented from writing to the device; files may only be read from the device.

- The opposite of the **mount** command is the **umount** command. Disks should be unmounted as soon as they have completed their function. Always unmount a floppy disk before removing the diskette.

3.1. Current Device Files

As currently distributed, Micronix has a group of standard files that correspond to devices sold with the system. These files may be displayed by listing the "/dev" directory:

```
% ls /dev
```

The display will be similar to the following:

```
hda  fla  pprB
hdb  flb  ttyA
hdc  mfa  ttyB
mwa  mfb  ttyC
mwa  mfb  ttyD
```

The "hd" files refer to the eight and 14 inch Discus series of hard disks (M10, M20 or M26). File "hda" corresponds to the first hard disk connected, "hdb" to the second, and so on. The disks are connected through HDCA controller boards.

The "mw" files refer to the mini-Winchester hard disks. Again, these units correspond to their logical connection through the HDCDMA controller boards.

The eight inch floppy disks are addressed by the files named "fl" and correspond to their connection through the Disk Jockey DMA controller boards.

The mini-floppy disks are addressed by the files named "mf" and a letter corresponding to their connection through the Disk Jockey DMA boards.

(The "ppr" files refer to daisy-wheel printers; the "tty" files to terminal, or teletype devices. These are of no concern to mount since they are character special files.)

Any data written to the device file is transmitted to the associated disk device. Similarly, reading from the device file causes data to be read off of the associated disk device.

3.2. Sample Mount Sequence

To demonstrate how a device is addressed with the **mount** program, let's run through a typical mount sequence. In this example, we will transfer a system file to a diskette in floppy device "f1a":

The **mkfs** command is entered first to make a file system:

```
# mkfs /dev/f1a
```

Next the device is "mounted" on a directory. Any directory (except root) may be used. In this example, we'll use a directory named "/f".

```
# mount /dev/f1a /f
```

To copy a file named "accounts" in the current directory to the floppy, enter the **cp** (copy) command as follows:

```
# cp accounts /f
```

To copy a file from the floppy to the current directory:

```
# cp /f accounts .
```

(Remember that "." is a name for the current directory.)

Similar to the copy command is **cptree**. If you had a need to copy out all the files in the current directory and sub-directories, the **cptree** command would be the most efficient means of transferring these files:

```
# cptree -v . /f
```

The last command entered is **umount** which unmounts the device:

```
umount /dev/fla
```

The floppy diskette may now be removed from device "fla".

A device may be mounted read-only by entering a "-r" option in the command line:

```
# mount /dev/fla/ /f -r
```

In this example, files may only be read from device "fla"; no files can be written to the device.

3.3. Device File Structure

Device files are unique in that they require no disk blocks, that is, they are "facades" that can be created and/or removed at will. To understand this more clearly, let's take a look at how these files are created.

The following commands were used to create the device files:

```
mknod ttyA c 1 1
mknod ttyB c 1 2
mknod ttyC c 1 3

mknod pprA c 1 0

mknod hda b 1 0
mknod hdb b 1 1

mknod fla b 2 0
mknod flb b 2 1

mknod mwa b 3 0
mknod mwb b 3 1

mknod mfa b 2 12
mknod mfb b 2 13

mknod fla b 2 8
mknod flb b 2 9
```

As you can see, **mknod** was entered with a "c" for the terminal (tty) and printer (ppr) files, indicating these were character special files.

The disk files were entered with a "b", indicating they were block special files. The numbers to the right of these commands specify the device the file will refer to. The left hand number specifies the type of controller device. Currently, 1 indicates the HDCA controller, 2 the DJDMA controller and 3 the HDCDMA controller. The number to the right indicates the drive the device will address. A 0

indicates drive A, 1 drive B, and so on.

Things get a bit trickier with the floppy drives, though. The 12 indicates drive A, 5 1/4 inch, alternating sectors. The 8 indicates drive A, 8 inch, alternating sectors. (These formats are listed in **djdma in Section 4 of the Reference Manual.**)

List out the "/dev" directory again, this time using the "-l" option. You should see a long list of information resembling the following examples:

```
crw--w--w-      1 root      1/2 Jan 14 17:34:54 1982  ttyB

crw--w--w-      1 root      1/0 Feb 10 13:10:43 1982  pprA

b---------      1 root      1/0 Feb 10 15:15:25 1982  hda

brw-rw-rw-      1 root      2/8 Mar 01 08:29:28 1982  f1a
```

Notice that a "c" precedes the first group of information for the "tty" and "ppr" files. This indicates that they are character special files.

The "b" preceding the "hd" and "fl" files indicate they are block special files. The numbers entered with the mknod command are reflected at the right in the number pairs preceding the date. Note that "hda" is connected to the HDCA controller and is addressed as drive A. "F1a" is connected to the DJDMA controller, is addressed as drive A, and is formatted for 8 inch alternating sectors.

Summary:

Mount is a means of making a temporary, logical association between Micronix and a disk storage device. "Mounting" a device means that device has been attached to a special block device file which addresses a specific disk storage device in the system.

The system imposes certain rules which must be adhered to when mounting a device.

Unless a "read-only" argument has been specified, it is possible to read from and write to mounted devices.

References

mount (1),(2), umount (1),(2), cp (1), cptree (1), djdma (4), mknod (1),

Micronix Operating System reference manual

programs

MORROW

CONTENTS OF REFERENCE SECTION 1: PROGRAMS

* The "account" program is described in greater detail under Users in the Maintenance and Administration section.

"fdj" is further discussed under Copying Diskettes in Maintenance and Administration.

There are more details about "recon" under Adding Terminals in Maintenance and Administration.

W Files labeled with a "W" are part of Whitesmith's C and Pascal. This is a separately-priced option, so these programs may not be included with your system. They are marked with the word "OPTION" n their heading lines.

NAME

account - manage user accounts

SYNTAX

account

DESCRIPTION

Account is an interactive, menu-driven program for listing, adding, deleting, and changing user accounts. Changes can be made to the account name, password, home directory, and shell.

SEE ALSO

newuser (1), chsh (1), passwd (1), group (1), owner (1)

NAME

anat - A-Natural assembler

SYNTAX

anat [flags] [files]

DESCRIPTION

Anat translates A-Natural narrative assembly language for the Intel 8080 to standard assembly language acceptable for either the ISIS-II asm80 or the Microsoft Macro-80 assembler. Since the output of the 8080 code generator, p2, is A-natural, anat is required to interface the C compiler to existing 8080 software development facilities.

The flags are:

-i Emit code in ISIS-II format. Identifiers have each "." changed to "@" and each "_" changed to "?". Default is Microsoft, in which case each "." is changed to "$" and each "_" is changed to ".".

-o Filename can be specified with the -o operand so that the output is written to the specified file and an error message is written to STDOUT. Default is STDOUT for output and STDERR for error message.

-s Emit an end statement. With a start operand, specifies a starting address. Default is no start address on the end statement.

If [files] are present, they are concatenated in order and used as the input file instead of the default STDIN.

If -o is absent, one or more files are present and the first filename begins with "8", as then behaves is if -o was specified using the first filename, except that the trailing "8" is changed to "m". Thus,

anat file.8

is the same as

anat -o file.m file.8

If -o is present, a name directive (title for Microsoft) is emitted using the output file name as the module name; any ".suffix" in the name is stripped off. If there is no -o, but [files] are present, the first input file name is used in the name directive. If neither is preset, no name directive is emitted.

Code generation begins in cseg, and reverts to cseg at the end before any literals are emitted. Any symbols not defined by the end of the input file are published via extrn statements at the end.

NOTES

The start string used with -s does not have "." and "_" mapped as identifiers.

SEE ALSO

as (1), ld (1), lib (2), cc (1), cp1 (1), cp2 (1)

NOTE: This program and documentation are products of Whitesmiths, Ltd., and are sold separately.

NAME

archive - archive maintainer

SYNTAX

archive {dptux} archname [files]

DESCRIPTION

Archive maintains a group of files combined into a single archive file. The archiver has good protection against data loss through disk overflow and works equally well with text and non-text files (for those systems on which there is a distinction).

Exactly one key character must be specified to indicate which action you wish archive to perform.

The meanings of the key characters are:

d Delete the named files from the archive file. If no names are given, the archive is not altered.

p Print the named files in the archive. Non-printing characters are filtered out.

t Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.

u Update the named files in the archive. If the named archive does not exist, it is created. If no names are given, all files in the archive are updated. If a name is given which is not in the archive, a new archive entry is created and its contents becomes the contents of the named file. Otherwise, the contents of archive's entry is replaced with the current version of the file.

x Extract the named files. The files in the archive are copied out to ordinary files of the same name. The former contents of the ordinary files are destroyed. If no names are given, all files in the archive are extracted. In neither case does x alter the archive.

EXAMPLE

To make an archive:

archive u newarch.a file1 file2 file3 file4

SEE ALSO

Brian Kernighan and P. J. Plauger Software Tools , Addison West Publishing Company, 1976.

NAME

as - A-natural assembler for 8080

SYNTAX

as -[o* x] [files]

DESCRIPTION

as assembles A-Natural narrative assembly language for the Intel 8080 to standard format relocatable object images. Since the output of the 8080 code generator is is A-Natural, as is required to produce relocatable images suitable for binding with link.

The flags are:

-o* write the output to the file *. Default is xeq. Under some circumstances, an input filename can take the place of this option, as explained below.

-x place in the object image only those symbols that are undefined or that are to be made globally known. Currently, this happens anyway; the flag is present for compatibility with other assemblers.

If [files] are present, they are concatenated in order and used as the input file instead of the default STDIN.

If -o is absent, and one or more files are present and the first filename ends in 's', then as behaves as if -o were specifed using the first filename, only with the trailing 's' changed to 'o'. Thus,

```
as file.s
```

is the same as

```
as -o file.o file.s
```

A relocatable object image consists of an eight word header followed by a text segment, a data segment, the symbol table, and relocation information. The header consists of the value 0x1499, the number of sytmbol table byes, the number of bytes of object code defined by the text segment, the number of bytes defined by the data segment, three zero words, and the data segment offset, which always equals the text segment size. All words in the object image are written less significant byte first. The text segment is relocated relative to location zero, the data segment is relocated relative to the end of the text segment.

Relocation information consists of two byte streams, one for the text segment and one for the data sement, each teminated by a

zero control byte. Control bytes in the ramge [1, 31] cause the many bytes in the corresponding segment to be skipped; bytes in the range [32, 63] skip 32 bytes plus 256 times the control byte minus 32, plus the number of bytes specified by the relocation byte following.

All other control bytes control relocation of the next word in the corresponding segment. The 1-weight bit and 2 weighted bit are both always zero for the 8080; the symbol code is the control byte right shifted two places.

A symbol code of 47 is replaced by a code obtained from the byte or bytes following in the relocation stream. If the next byte is less than 128, then the symbol code is its value plus 47; otherwise the code is that byte minus 128 times 256, plus 175 plus the value o the next relocation byte after that one.

A symbol code of zero calls for no further relocation; 1 means that a change in text bias must be added to the word; 2 means that a change in data bias must be added; 3 is not used. Other symbol codes call for the value of the symbol table entry indexed by the symbol code minus 4 to be added to the word.

Each symbol table entry consists of a value word, a flag byte, and a nine-byte name padded with trailing NULs. Meaningful flag values are 0 for undefined. 4 for defined, 5 for defined text relative, and 6 for defined data relative. To this is added 010 if the symbol is to be globally known.

SEE ALSO

link, cp2, obj

NAME

badspots - bad sector report

SYNOPSIS

badspots [-p] filesystem ...

DESCRIPTION

Badspots is a Micronix utility program which produces a display of bad blocks on a hard disk whose name is given. The program knows how to read the bad map formats of the two Morrow disk testing and formatting programs:

FORMATMW

and

FORMATHD

Badspots knows about exactly 6 kinds of disks.

5 inch:
M5
M10
M16

8 inch:
M10
M20
M26

The program only works properly on a freshly formatted or tested disk; that is to say, the Micronix file system is written over the the disk's bad sector map. Invalid bad sector maps are flagged in error and no list of block numbers is produced. In particular, either a valid block list or nothing is sent to the standard output, so it is always meaningful to use the block list badspots produces.

If there are no bad sectors in the disk's bad sector map, badspots produces no output.

The Default Display Mode: By default each bad sector is displayed as a Micronix block number.

The -p option causes the cylinder, head, and sector numbers to be printed for each bad sector.

EXAMPLE

```
fsck -b `badspots /dev/m16a` /dev/m16a
```

This is the way **badspots** is used in production at Morrow Designs. The results of **badspots are conveyed to fsck** in this example.

To simply view a list of badspots:

```
badspots hdb
```

The **badspots** program is clever enough to try adding "/dev/" to the filename used if it is unable to open the file named on the command line. To see that same **badspots** list expressed as cylinder, head, sector triplets:

```
badspots -p hdb
```

SEE ALSO
fsck (1), hdca (4), hddma (4)

NAME

boot - Micronix bootstrap procedures

SYNTAX

djboot, hdboot, m5boot, m10boot, m16boot

DESCRIPTION

The CP/M diskette that comes with Micronix includes the following bootstrap programs:

djboot Boot from the DJDMA controller with a 5 inch or an 8 inch drive.

hdboot Boot from the HDCA controller the 10, 20, or 26 megabyte drive

m5boot Boot from the HDDMA controller with the 5 megabyte drive

m10boot Boot from the HDDMA controller with the 10 megabyte drive

m16boot Boot from the HDDMA controller with the 16 megabyte drive

These are CP/M programs, ie, they are stored on the CP/M diskette as djboot.com, hdboot.com, etc, and are invoked from CP/M by typing the name without the .com extension.

In each case, they assume that the target disk contains a Micronix file system, and they search its root directory for files that could be Micronix kernels. If exactly one such file is found, it is loaded and executed. If more than one kernel is found, the names are listed and the operator is asked choose one. This allows backup copies or alternate versions to be kept.

CP/M's only role in this process is the invocation of the bootstrap program. Once the bootstrap begins executing, it is independent of CP/M, and CP/M itself is overwritten as Micronix is loaded.

CP/M can be eliminated entirely by placing the desired bootstrap program on the "system tracks" of a floppy disk. Then as long as that diskette is in drive A, Micronix will boot automatically from the appropriate hard disk every time the Decision's RESET button is hit. This should save the operator about 5 seconds per day. (CAUTION: if Micronix is already running, make sure you type sync before hitting RESET.)

The loaders that can be placed on a floppy's system tracks are stored as files on the Micronix CP/M diskette, and are called

hdload, m5load, m10load, and m16load, corresponding to the similarly-named bootstrap programs. See below for an installation example. The djboot program has already been placed on the system tracks of the Micronix stand-alone floppy, so if you put this in drive A and hit RESET, it will boot automatically.

EXAMPLES

COMMENTS	SCREEN DISPLAY
To boot the Micronix stand-alone floppy:	
Make sure the diskette is write-enabled	
Insert the diskette into drive A	
Hit RESET	
Machine responds	FFFF
Machine types :, you type b	:b
Loader takes over	DJDMA loader ...
To boot Micronix from the m5:	
Insert the Micronix CP/M diskette in drive A	
Hit RESET	
Machine responds	FFFF
Machine types :, you type b	:b
CP/M announces itself	Morrow Designs CP/M
Type m5boot	A> m5boot
Loader takes over	HDDMA loader ...
To prepare an auto-boot floppy for the m5:	
Insert the Micronix CP/M diskette in drive A	
Hit RESET	
Machine responds	FFFF
Machine types : you type b	:b
CP/M announces itself	Morrow Designs CP/M
Type formatdj. Follow the instructions to format a new diskette. 512 byte sectors are recommended.	A> formatdj
Type sysgen m5load	A> sysgen m5load
Type an A	Destination drive? A

```
Insert the formatted              Insert a write
  diskette in drive A              enabled diskette
  and hit RETURN.                  in drive A
The diskette is ready to use      Function complete...
```

To use the m5 auto-boot:

```
Insert the auto-boot
  diskette in drive A
Hit RESET
Machine responds                  FFFF
Machine types :, you type b       :b
Loader takes over                 HDDMA loader ...
```

BUGS

The bootstrap programs only search the first block of the root directory, so you should limit your root directory to 32 entries (which is a good idea anyway, to speed up rooted file-name searches), or else you should make sure that the Micronix kernel occurs as one of the first 32 entries.

NAME

cal - print a calendar

SYNTAX

cal [month] [year]

DESCRIPTION

The cal program is capable of printing full-year calendars or single-month calendars for any year or any month after year 0 (A.D).

Cal with no arguments prints a full-year calendar for the current year.

Cal with one numerical argument prints a calendar for that year.

Month names may be given a english names, abbreviations, or as numbers. If a month name is given, a single-month calendar is printed.

If both arguments are numeric, the first is taken to be the month.

EXAMPLES

cal 1932

Would print a full year calendar for the year 1932.

cal august 1955

To print a single month calendar for August, 1955.

cal

Would print this year's calendar.

cal november

For a single month calendar for November of this year.

NAME

cat - concatenate and print

SYNTAX

cat file ...

DESCRIPTION

Cat reads each file in the order given and writes it to the standard output. Thus,

cat file

prints the file and

cat file1 file2 > file3

concatenates the first two files and places the result in the third. If no files are given or if "-" is given as a file name, cat reads the standard input up to an end-of-file.

NOTES

Beware of

cat a b > a

and

cat a b > b

they destroy the input files before reading them!

SEE ALSO

clean (1)

NAME

cc - C and Pascal compiler

SYNTAX

cc [options] file ...

DESCRIPTION

Cc is the Whitesmiths' C compiler supervisor; it accepts several types of arguments:

Those ending with ".p" are assumed to be Pascal source files and are converted to ".c" files. Arguments ending with ".c" are taken to be C source files and are compiled with each relocatable object module left in the file named for the source with ".o" substituted for ".c".

In the same way, arguments whose names end with ".s" are taken to be A-natural source files and are compiled to produce an ".o" file.

The following options are meaningful to cc:

-c Suppress the loading phase of the compilation, and force object file(s) to be produced for each source file.

-S Produce A-natural output. A-natural is Whitesmith's narrative assembly language and is one of the intermediate products of a C compiler run. Normally the A-natural version exists only in a temporary file which "cc" removes. Each of the named ".c" files and ".p" files is compiled into a similarly named ".s" file.

```
cc -S prog.c
```

would produce a file "prog.s" containg the A-natural translation of the given C source program.

-i name Passed on to cpp, the C compiler preprocessor. Name is a prefix to be used on "# include <files>".

-o file The result of the compilation will be written on the named file. If no "-o" option is given, the output is written to "a.out"

-lX Library inclusion. The file "/lib/libX.a" will be included in the linking phase of the compilation. An example is included below.

-lS Use the standard I/O library. Note that this only affects the loading phase of the compilation. If you use the library, your programs should include the line:

```
# include <stdio.h>
```

-cpm Compile to run under CP/M.

-ws Use the Whitesmiths' C library.

-xN where N is a number. Arguments of this type are passed on to pass 2 of the compiler; see cp2.

-z Strip out control Z's from C source files produced with the aid of CP/M programs (for instance). The -z flag arranges for a precleaning of your source files. The "clean" utility is used to accomplish this. (See clean in section 1 of the Micronix reference manual). The source files are cleaned in place.

EXAMPLES

To compile echo.c:

```
cc echo.c -o echo -ws
```

To compile a program which uses the Standard I/O library.

```
cc prog.c -o prog -lS
```

To compile a program to be run under the CP/M operating environment:

```
cc prog.c -o prog.com -cpm
```

Or with the Standard I/O library and for CP/M:

```
cc prog.c -o prog.com -cpm -lS
```

If you want to see what's going on:

```
cc a.c -v
```

And the compiler passes and flags will be displayed as they are run.

If you have written you own library, put it in /lib and then:

```
cc prog.c -o prog -lnew
```

To create an object module

```
cc -c prog.c
```

(The -c stands for compile only (no link))

To construct an executable program from several previously produced object modules:

```
cc a.o b.o c.o d.o -o out -lS
```

To run a compile in the background (so you don't have to wait)

```
cc prog.c -o prog -lS &
```

SEE ALSO

Kernighan, B.W. and D.M. Ritchie, The C Programming Language, (Prentice-Hall, 1978).

Kernighan, B.W., Programming in C - A Tutorial

Ritchie, D.M., C Reference Manual, (Addison-Wesley, 1976).

cpp (1), cp1 (1), cp2 (1), as (1), ld (1), lib (1)

NAME

change - make global changes in a stream

SYNTAX

change from [to]

DESCRIPTION

Change reads the standard input up to an end-of-file. It writes the standard output exactly as it was read, except that all instances of the regular expression **from** are replaced with **to.** If no replacement pattern is specified, all instances of the regular expression are eliminated. The rules for regular expressions and replacement patterns are exactly as in **edit.**

EXAMPLE

To make a global replacement:

change lead gold <input >output

SEE ALSO

edit (1), find (1)

Brian Kernighan and P. J. Plauger Software Tools , Addison West Publishing Company, 1976.

NAME

chars - count characters in standard input or files.

SYNTAX

chars [file ...]

DESCRIPTION

The number of characters in each of the named files is printed on the standard output. If it would provide additional information, a total is then printed. If no file names are specified, or if "-" is given as a file name, the standard input is read up to an end-of-file and the number of characters read is printed.

Note: On some systems, (but not Micronix), the number of characters in a file is poorly defined. There are sometimes two kinds of files - text and binary. The standard input is assumed to be a text file while files specified by name are assumed to be binary files.

EXAMPLE

To count the characters in a number of files:

chars file1 file2 file3 file4

SEE ALSO

lines (1), words (1)

NAME

chmod - change mode of files (accessibility for reading, writing, & execution; set user/group number bit; set write lock bit)

SYNTAX

chmod mode file(s)

BRIEF DESCRIPTION

chmod performs three general functions as noted above: (A) setting up access restrictions to files, (B) setting/resetting the user and group number bits, which is typically used to get around access restrictions under special circumstances, and (C) setting/resetting the write lock bit, which prevents two processes from writing to the file simultaneously. The format of access mode commands is:

chmod [u g o a] [= - +] [r w x] filename(s)

The format of user/group bit commands is:

chmod [u g] [+ -] s filename(s)

Finally, the format of write lock commands is

chmod [+ -] l filename(s)

You should also refer to the entry on the ls program in this section of the manual, as it closely relates to the following explanation. Use the "-l" flag with the ls program to monitor the effects of chmod operations.

EXPLANATION A. File Access

Read, write, and execute access are controlled through the "mode" argument, which may be an octal number (e.g. 677) or a combination of symbols as described below. Before proceeding, you should have a basic understanding of the Micronix file protection system.

The file protection system allows or denies access to files according to two criteria, namely, WHO wants to get into the file and HOW he wants to use it. There are three categories of users and three types of access, so there are nine bits altogether which define the accessibility of the file. WHO is represented in chmod commands by these symbols:

a - for "all", in other words, the restrictions that you specify will apply to everyone in the system. As you will see, "a" is not one of the user categories

mentioned above, but rather the sum of all three.

u - for "user", in effect the owner or creator of the file. The owner of a file may be changed by way of the "owner" program. Note that only the superuser (login "root") may use the "owner" program.

g - for "group", which comprises users with the same group code as the owner. (See "group" in the Files section of this manual).

o - for "others", who is anyone else in the system not covered by u and g.

Now, as to HOW these various parties may wish to use the file in question, there are these three categories:

r - to read the file;

w - to write to the file;

x - to execute the file, in the case of a program or control string; or search, in the case of a directory.

There is a final set of symbols to consider, the operands +, -, and =. "+" will add a privelege that had earlier been restricted. "-" accomplishes the reverse. "=" enables only those HOW's listed after it, and disables those not listed. The general form that chmod expressions take is:

chmod WHO operand HOW filename(s)

When performing chmod, keep in mind that files are created without restrictions of any sort.

EXAMPLES A. File Access

To set up a new file so that only the owner has access (of all 3 types):
chmod go-rwx filename

To restore full priveleges to everyone in the system for a file that had previously been restricted:
chmod a+rwx filename

To understand the difference between the + and = operands, consider the following. If the group now has read and execute priveleges only, and you want to give them write ability as well:
chmod g+w filename will work, as will
chmod g=rwx filename. However, if you had entered **chmod g=w filename**, the read and execute priveleges would have been unintentionally revoked.

Multiple chmod operations for the same file may be strung together in one command, using commas but no spaces. Say you have a new (unrestricted) file that you want to set up so that you have full priveleges, the group has read only priveleges, and others cannot access it at all. This could be your command:
chmod g=r,o-rwx filename

EXPLANATION B. Set User/Group Number Bit(s)

As noted above, when you try to open a file (either directly or through a program you are executing) your user and group numbers are checked to see whether access is permitted. Now, suppose you execute a program owned by someone else, but to which you have been granted execute priveleges. However, within that program is a command to read a file to which only the programmer has access. You will not be granted access, and a "permission denied" message is passed back to your unhappy process.

The chmod function being described here can often overcome this sort of problem by replacing your user and/or group number with the numbers of the owner of the program. See the example that follows for further information.

EXAMPLE B. Set User/Group Number Bit(s)

Programmer Mary has written a program EXTRACT that reads year-to-date information from her company's payroll file (PAYROLL). It is decided that each system user should be allowed to look at his own payroll record. So Mary modifies the program accordingly, and enters **chmod go+x EXTRACT** to open up the previously restricted execute privelege to other users.

However, when John tries to run the program, all he gets is the message "PAYROLL: Permission denied." What has happened is that user John has tried to open the payroll file, which Mary owns and to which he has no access priveleges. The solution in this situation is to invoke chmod so that when EXTRACT goes to read PAYROLL, PAYROLL will think EXTRACT's owner (Mary) is running the program, regardless of who is actually running it.

The command in this case (which must be entered by Mary) is:
chmod u+s EXTRACT

The letter u means that Mary's user number will be attached to any file openings requested by EXTRACT. If the file PAYROLL was owned by someone other than Mary but to which she had group priveleges, the letter g would be used in place of or in addition to the u in the command above.

To reset the user/group number flags, repeat the command you

used to set them, but with a minus sign in place of the plus sign.

EXPLANATION C. Set Write Lock Bit

This function is quite simple and straightforward. Without the write lock bit set, Micronix allows anyone with write access to a file to update it whenever they please. However, some files can become corrupted if write accesses become interleaved. As an example, a data base system written for a single user (such as Personal Pearl) should have its critical data files write-locked. This is accomplished by setting the write lock bit. Enter:

chmod +l database

This sets the lock flag on the database. (Since the lock flag applies equally to user, group, and others, the WHO part of the operand may be left out here.) Using a minus sign in place of the plus in the above command will reset the bit, thus removing the lock.

SEE ALSO

group(5) ls(1) owner(1)

NAME

clean - clean CP/M format text files

SYNTAX

clean [-u] file ...

DESCRIPTION

Clean removes return characters from each of the named files and truncates the file upon encountering a CP/M end-of-file character (^Z).

The "cleaning" is done in place. To clean a file for general Micronix use, enter the **clean file** command.

The -u option reverses the process. It stands for "unclean". Use

clean -u file

to make a file suitable for CP/M programs to look at.

EXAMPLE

To import a CP/M text file, enter

far /dev/dja -x file; clean file

SEE ALSO

far (1)

NAME

cmp - compare two files

SYNTAX

cmp [-l] file1 file2

DESCRIPTION

Cmp compares two given files. Nothing is printed if they are identical. If a difference is encountered, cmp announces the byte number and line number at which the difference occurred.

If one file is an initial sub-sequence of the other, that fact is also noted. If the -l option is used, the byte number (decimal) and the differing bytes (octal) are printed for each difference.

SEE ALSO

diff (1), common (1)

NAME

comm - print lines in common

SYNTAX

comm [-123] file1 [file2]

DESCRIPTION

Comm prints lines unique to the first named file in column 1, lines unique to the second named file in column 2, and lines common to both files in column 3. Leading blank space is removed for printing purposes. The files must be sorted. If no second file name is given, the standard input is used for the second file.

Flags 1, 2, or 3 suppress printing of the corresponding columns. Thus **comm -12** prints only the lines common to the two files; **comm -13** prints only the lines that are unique to the second file; **comm -123** prints nothing.

EXAMPLES

To find spelling mistakes

```
translit A-Z a-z <input >t1
translit ^a-z "@40" <t1 >t2
translit "@12@40" "@12" <t2 >t3
sort <t3 >t4
comm -13 dictionary <t4 >mistakes
```

NOTES

Only works with sorted inputs.

NAME

common - print lines in common

SYNTAX

common [-wN] [-1] [-2] [-3] file1 [file2]

DESCRIPTION

Common prints lines in the first named file in column 1, lines in the second named file are printed in column 2, lines common to both files are printed in column 3. Leading blank space is removed for printing purposes. The files are assumed to be sorted. If no second file name is given, the standard input is used for the second file.

If -1, -2, or -3 is given as an argument, those columns specified are printed.

Horizontal positioning of the columns may be changed with the -w option. (Default is 80 columns wide.)

-wN - where N is a number - sets the width to N columns wide.

EXAMPLES

Find spelling mistakes

```
translit A-Z a-z <input >t1
translit ^a-z "@40" <t1 >t2
translit "@12@40" "@12" <t2 >t3
sort <t3 >t4
common -2 dictionary <t4 >mistakes
```

NOTES

Only works with sorted inputs.

SEE ALSO

Brian Kernighan and P. J. Plauger Software Tools , Addison West Publishing Company, 1976.

NAME

compare - compare two files for equality

SYNTAX

compare file1 file2

DESCRIPTION

Compare reads the two named files, line by line, and announces any differing lines, including the line number.

EXAMPLE

To find the differing line in two nearly identical files:

compare version1 version2

NOTES

The algorithm is very poor if lines have been added or deleted to one of the files. Compare will then report that all of the following lines differ when in fact only a small change may have been made between the two files.

SEE ALSO

diff (1) for a more sophisticated text comparator.

Brian Kernighan and P. J. Plauger Software Tools , Addison West Publishing Company, 1976.

NAME

compress - compresses files

SYNTAX

compress [input [output]]

DESCRIPTION

The first named file is compressed and written onto the second named file. If no file names are given, the standard input and output are used. If only one file name is given, the named file is used for input and its compression is written to the standard output. If the second named file does not exist, it is created.

Compress searches for opportunities to abbreviate long runs of identical characters in an input stream. It encodes these into a form that the **expand** program can later be used to undo. The remainder of the file is merely copied through. Most text files are probably not worth compressing unless they contain long runs of spaces. Data files, such as executable modules, benefit somewhat from compression. A typical compression value is 7%.

EXAMPLE

To compress a file (for disk file space saving):

compress input compressed

NOTES

Due to I/O idiosyncracies on some systems, the standard input and output cannot be used. The automatic addition of carriage returns, etc. in the input and output streams is unacceptable for this program. The construction

compress input output

should work regardless of the local environment.

SEE ALSO

expand (1)

Brian Kernighan and P. J. Plauger Software Tools , Addison West Publishing Company, 1976.

NAME

concat - concatenate named files onto standard output

SYNTAX

concat file1 file2 ...

DESCRIPTION

Concat reads each of the files named in its argument list and produces output which is the concatenation (in the order given) of the input files. If no file names are given, no output is produced.

EXAMPLE

To read a file:

concat file

SEE ALSO

cat (1)

Brian Kernighan and P. J. Plauger Software Tools , Addison West Publishing Company, 1976.

NAME

copy - copy input characters to output

SYNTAX

copy

DESCRIPTION

Copy simply copies its standard input to its standard output unchanged, up to an end of file.

EXAMPLE

To copy one file to another:

```
copy <source >destination
```

NOTES

On some systems, (but not Micronix), due to I/O conventions non-text files would be mangled if copied with the above example. In this instance, use the cp program.

SEE ALSO

cp (1), concat (1)

Brian Kernighan and P. J. Plauger Software Tools , Addison West Publishing Company, 1976.

NAME

cp - copy files

SYNTAX

cp source destination

or

cp source ... directory

DESCRIPTION

The first named file is copied to the second.

If the last named file is a directory, each of the preceding named files are copied "into the directory". That is, "file" is copied to "dir/file".

Cp refuses to copy a directory, and refuses to copy a file to itself.

Cp will also copy files over the Micronix network. Source or destination names can be arbitrary network names. A network name looks like machine_name%path_name. Examples:

```
cp document sales%/a/joyce
cp admin%/ken/payroll /tmp/checks
```

Cp, as a command under Micronix, is analogous (but not identical) to PIP under CP/M. It is the most common and efficient way to copy files.

SEE ALSO

mv (1), cptree (1), network (4)

NAME

cpl - parse C programs

SYNTAX

cpl <flags> <file>

DESCRIPTION

Cpl is the parsing pass of the C compiler. It accepts a sequential file of lexemes from the preprocessor, cpp, and writes a sequential file of flow graphs and parse trees, suitable for input to a machine-dependent code generator, cp2. The operation of cpl is largely independent of any target machine. The flag options are:

-a Compile code for machines with separate address and data registers.

-b# Enforce storage boundaries according to #, which is reduced modulo 4.

A bound of 0 leaves no holes in structures or auto allocations.

A bound of 1 (default) requires short, intermediate and longer data to begin on an even bound.

A bound of 2 is the same as 1 except that 4 to 8 bytes of data are forced to a multiple of a four-byte boundary.

A bound of 3 is the same as 2, except that eight bytes of data (doubles) are forced to a multiple of an eight-byte boundary.

-c Ignore case distinctions in testing external identifiers for equality, and map all names to lower case on output. By default, case distinctions do matter.

-e Do not force loading of external references that are declared, but never defined or used in an expression. Default loads all declared external references.

-l Take integers and pointers to be four bytes long. Default is 2 bytes.

-m Treat each structure/union as a separate name space, and require x.m to mean that x is a structure and m is one of its members.

-n# Ignore characters after the first # in testing external identifiers for equality. Default is 7; maximum is 8.

-o file Write the output to the specified file and write error messages to STDOUT. Default is STDOUT for output and STDERR for error messages.

-r# Assign no more than # variables to registers at any one time, where # is reduced modulo 4. Default is three register variables.

-u Take "string" as an array of unsigned characters, not an array of characters.

If <file> is present, it is used as the input file instead of the default, STDIN. On many systems, the -o option and <file> are mandatory because STDIN and STDOUT are interpreted as text files, and hence become corrupted.

EXAMPLE

Cpl is usually sandwiched between cpp and cp2, as in:

```
cpp -x -o templ file.c
cpl -o temp2 templ
cp2 -o file.mac temp2
```

SEE ALSO

cpp (1)
C Compiler User's Manual for language definition.

NOTE: This program and documentation are products of Whitesmiths, Ltd.

NAME

cp2 - generate code for 8080 C programs

SYNTAX

cp2 -[flags] <file>

DESCRIPTION

Cp2 is the code generating pass of the C compiler. It accepts a sequential file of flow graphs and parse trees from cp1 and writes a sequential file of A-Natural statements, suitable for input to the A-Natural assembler, as.

As much as possible, the compiler generates free-standing code; but for those operations which cannot be done compactly, it generates inline calls to a set of machine-dependent library routines. The 8080 runtime library is documented in Section IV of the C Reference Manual.

The flags are:

-p Emit profiler calls on entry to each function. No profiler is currently provided for the 8080.

-o * Write the output file * and write the error messages to STDOUT. Default is STDOUT for output and STDERR for error messages.

-x# Map the three virtual sections, for Functions (04), Literals (02), and Variables (01), to the physical sections Code (bit is one) and Data (bit is zero). thus, "-x4" is for separate I/D space, "-x6" is for ROM/RAM code, and "-x7" is for compiling tables into ROM. Default is 6.

If <file> is present, it is used as the input file instead of the default STDIN. On many systems, <file> is mandatory because STDIN is interpreted as a text file, and hence becomes corrupted.

Files output from cp1 for use with 8080 code generator should be generated with: "-b0" to eliminate holes in structures and auto allocation, and "-n8" for full length external names. A boundary of "-b1" (default) is also acceptable if compact storage is less important than compatibility with data structures on other machines. Note that use of the cp1 flags "-b2", "-b3", or "-1" will produce bizarre behavior and incorrect code in cp2, often with no meaningful diagnostics.

Wherever possible, labels in the emitted code each contain a comment which gives the source line from which the code immediately following obtains.

EXAMPLE

cp2 usually follows cpp and cp1 as follows:

```
cpp -x -o temp1 file.c
cp1 -b0 -n8 -o temp2.80 temp1
cp2 -o file.s temp2
```

SEE ALSO

as (1), cp1 (1), cc (1), cpp (1)

NOTE: This program and documentation are products of Whitesmiths, Ltd.

NAME

cpp - preprocess defines and includes

SYNTAX

cpp <flags > <files>

DESCRIPTION

Pp is the preprocessor used by the C compiler, to perform **#define**, **#include**, and other functions signaled by a # before actual compilation begins. It can be used to advantage, however, with most language processors. The flag options are:

-c Do not strip out /* **comments** */ nor continue lines that end with \.

-dname Define name; if the definition string (=def) is included, it reads it in before the input; if "=def" is omitted, the definition is taken as "1". The "name" and "=def" must be in the same argument - no blanks are permitted unless the argument is quoted. Up to ten definitions may be entered in this fashion.

-i Change the prefix used with **#include <file name>** from the default ("") to the string prefix. Multiple prefixes to be tried in any order may be specified, separated by the "|" character.

-o <File> can be specified with flag. Writes the output to the specified file and writes error messages to STDOUT. Default is STDOUT for output and STDERR for error messages. On many systems, the -o option is mandatory with -x because STDOUT is interpreted as a text file, and hence becomes corrupted.

-p? Change the preprocessor control character from # to ?.

-s? Change the secondary preprocessor control character from @ to ?.

-x Put out lexemes for input to the C compiler cp1, not lines of text.

-6 Put out extra newlines and/or SOH ('\') codes to keep source line numbers correct for UNIX/V6 compiler or ptc.

Cpp processes the named file, or STDIN if none is named, in the order specified, writing the resultant text to STDOUT. Preprocessor actions are described in detail in the Preprocessor section of the C Compiler User's Manual. The presence of a secondary preprocessor control character permits two levels of parameterization. For instance, the invocation

```
cpp -c -p@
```

will expand define and ifdef conditionals, leaving all # commands and comments intact. Invoking **cpp** with no arguments would expand both @ and # commands. The flag, **-s#**, effectively disables the secondary control character.

EXAMPLE

The standard for writing C programs is

```
   /* name of program
    */
#include <std.h>

# define MAXN       100

COUNT things[MAXN];
etc.
```

The use of uppercase identifiers is not required by **cpp**, but is strongly recommended to distinguish parameters from normal program identifiers and keywords.

NOTES

Unbalanced quotes (' or ") may not occur in a line, even in

NAME

cptree - copy Micronix file hierarchies

SYNTAX

cptree [-v] [source destination]

DESCRIPTION

The contents of the source directory and all its descendants are copied into the destination directory.

The destination directory must exist, but **cptree** will create any needed sub-directories.

If either the source or destination names are not given, **cptree** will ask for them.

If -v is in the command line, cptree will print each operation in the tree copy as it is performed.

It is permissible to copy a tree into a sub-tree of itself. Cptree will avoid getting into a wormhole.

SEE ALSO

mv (1), cp (1), rm (1)

NAME

crypt - file encryption

SYNTAX

crypt key

DESCRIPTION

Crypt reads from its standard input and writes to its standard output, encrypting as it goes, according to the key. Crypt is its own inverse; to read an encrypted file, all you need to know is the key word.

Note: this is not a very sophisticated encryption scheme (NSA would have it overnight).

You can increase the complexity of the encryption by using multiple passes of crypt, each with a different keyword.

EXAMPLES

To encrypt a file:

```
crypt bandersnatch < file > encrypted
```

To read the encrypted file

```
crypt bandersnatch < encrypted
```

Encrypt a file with three keywords:

```
crypt honeybee < file | crypt Beaumont | crypt oscar > encrypted
```

NAME

cu - call up

SYNTAX

cu device [-s speed]

DESCRIPTION

Cu may be used to connect to other machines through your Micronix machine.

When the cu command is given, one process is set up to handle sending characters to the remote system, one is set up to receive them. Much of this is not obvious to the user.

When cu is ready it prints a "~". Thereafter, whatever you type will be immediately sent to the remote port, and whatever is received from it is sent to your terminal. So, typically, you press a single key on your terminal, Micronix takes it and re-transmits it to the remote machine, which will usually send it back, at which point Micronix receives it and sends it to your terminal and you see the character on your screen!

There is definitely an upper limit to the baud rate you can use for the remote port - 1200 baud is the highest recommended. If the remote machine is another Micronix machine, additional commands apply. You may copy a file from local to remote machine via:

```
~%put file
```

Or from remote to local via:

```
~%take file
```

Note only text files may be copied reliably. This need not be a drawback, however, because any file can be converted into a text file, transferred, and then converted back again.

Both the sending and receiving processes recognize

```
~>file

~>
```

as starting and ending delimiters for file diversion, so that in order to copy a file from a foreign system to the local one, all you have to do is get the foreign system to generate:

```
~>filename
Text of file ...
more file ...
~>
```

That is, a beginning marker, the text of the file, and then an ending marker.

To get out of cu type:

~.

EXAMPLE

To set up communication over a modem, get your hardware set up and then type:

```
cu /dev/ttyC -s 300
```

In this example the modem would be plugged into Micronix port C (the third one).

Typically, one must reverse the transmit and receive lines in the cable running from the computer to the modem. One reverses the wires running to pins 2 and 3 of the connector at one end or the other.

NAME

cxr - C cross reference listing generator

SYNTAX

cxr [-exclusions ...] [program.c ...]

DESCRIPTION

Cxr generates a C cross reference listing for each of the named files. If an argument begins with a minus, the rest of the argument is taken as the name of file containing symbols to be excluded from the cross referencing. The symbols found therein are added to the list of already excluded symbols. The exclusion files are free form. The symbols may appear most anywhere, as long as they are separated by non-alphanumeric characters. Note that the output of **cxr** is in order of appearance. For a useful listing, run **cxr**'s output through **sort**. If no cross reference input files are given, the standard input is used. For each symbol, the following is printed:

<symbol> <file> <function> <line> <context>

where

<symbol>

is simply the name of the symbol.

<file>

is the name of the file in which the current instance of the symbol occurs. <file> is used when the standard input is being used.

<function>

is the name of the function in which the current instance of the symbol appears. If the symbol is global, no function name is printed.

<line>

is the number of the file.

<context>

is the text of the line in which the current instance of the symbol appears. Leading white space is removed from the line before printing.

All of the C keywords and the std.h typedefs are automatically excluded. This includes the preprocessor keywords. Comments, string constants, character constants, and numerical constants

are excluded.

NOTES

If you make your structure references or function declarations into onions of parentheses, cxr may gaffe.

EXAMPLE

```
cxr file1.c file2.c file3.c file4.c >cross

sort <cross >sorted
```

Under Micronix:

```
cxr *.c | sort > output &
```

SEE ALSO

sort (1), kwic (1)

NAME

dar - access MSDOS format floppy diskettes

SYNTAX

dar device [flags] [files]

DESCRIPTION

Dar provides a simple interface to MSDOS floppy diskettes. It may be used for backing up files on diskettes, and recovering them as necessary. "Device" is assumed to be the name of a Micronix special file (such as mfa) corresponding to a floppy drive that contains a MSDOS diskette. Note that mfa stands for mini floppy A. Dar can copy files in either direction, and can also be used to read the directory of a MSDOS floppy. When the -v flag is used, it reports the amount of space still available on the diskette.

Names such as /dev/mfa may be abbreviated as mfa. You can check the names of the storage devices by entering

dir /dev (return)

Currently only single sided 5.25" IBM PC format diskettes are recognized.

The flags are:

-c **Create.** Start a new dos diskette from scratch. Remove all files, clean out the file allocation table. This option must be used to copy fils to a freshly formatted diskette. It may be used in conjunction with any of the other arguments, and if so, it is performed first.

-d **Delete.** Delete the named files from the floppy diskette. If no file names are given, no action is taken.

-p **Print.** Each named file is sent to the standard output (most likely your terminal).

-r **Replace.** Each named file is copied to the floppy. If no file names are given, all the files in the current directory are copied. If a file of the same name already exists on the floppy, it is silently overwritten.

-t **Table.** Print a table of contents of the floppy diskette. If any files are named, only those named and present are listed.

-x **Extract.** Each named file is copied from the diskette to the current Micronix directory. If no file names are given, the entire contents of the diskette is extracted.

-v **Verbose.** Be verbose about it. Each operation is printed along with the affected file name. Files deleted are preceded by a "d", those replaced are preceded by an "r", those extracted are preceded by "x". The percentage of the diskette that is occupied, along with the number of bytes still available, is reported when dar is finished.

At most, one of the flags - d, t, r, x - may be present. If none are present, -t is assumed. -v may be used in combination with any of these as in "-xv".

The file list may contain wild card matching patterns to be applied to the set of file names in the MSDOS diskette's directory. Refer to the section "Argument Expansion" in the sh program writeup in this manual.

EXAMPLES

You receive a MSDOS diskette and want to copy the files off. To copy its entire contents to Micronix, insert the diskette into floppy drive A and type:

```
dar mfa -xv
```

To copy a file from a diskette, changing its name in the process:

```
dar mfa -p filename > newname
```

To copy only the ".doc" files over from a MSDOS mini-floppy:

```
dar mfa -xv "*.doc"
```

To copy a file from diskette into a file that is elsewhere than your current directory:

```
dar mfb -p "name" > /dir/...
```

(path from root to desired location; you could also specify a new name)

The following discussion is intended primarily for users that are having trouble getting dar to do what they want it to.

One aspect of dar command syntax somewhat complicates the issue of using it, namely, the use of quotation marks. An example should help clear this up. Let's say your current Micronix directory contains two files, **a.com** and **a.bak**. Your MSDOS floppy contains a file **a.pil** that you wish to copy into your current directory. Since this is the only file on the floppy that begins with "a", you lazily enter **dar mfa -xv a***. Next you are wondering what's wrong when you are faced with the message "a.com not found on diskette / a.bak not found on diskette". Of course they weren't found. You didn't even want Micronix to be looking for them.

What has happened is this: Your command interpreter (shell) thought you meant to search the current directory for a match to the wildcard argument before going out to the floppy for the copy. It indeed found two matches (a.com & a.bak), so Micronix went gleefully out to the floppy to copy them. In vain, unfortunately.

The solution here is to use quotation marks around your filename(s). Had you entered **dar mfa -xv "a*"**, the shell would have left the command alone, and dar would have gone directly out to the floppy to find anything that began with an "a". Success.

Now, just to set the record straight, there is a further complication that may help you to understand how the shell operates. Suppose the file on your floppy is called **b.pil** instead of **a.pil**. If you enter **dar mfa -xv b.*** (no quotes), the file will in fact be copied, contrary to the previous discussion. Furthermore, everything on the floppy diskette will be copied, much to your chagrin. This is because your shell took b.* to your current Micronix directory, looked to see if a file of that name was really there, and finding no such file, it passed the command to Micronix as " **dar mfa -xv** ". As noted above in the description of the x flag, if there is no file specified, the whole diskette gets copied.

In summary, you <u>should</u> enclose the argument in quotes whenever you want to copy specific files from the floppy to Micronix. The only exception to this is a situation where you have certain files already established on Micronix and you want to extract only those floppy files that match the existing Micronix filenames.

SEE ALSO

far - to access CP/M diskettes.

NAME

date - print or set the date and time of day

SYNTAX

date [year] [month] [date] [hour:[minute:[second]]]

DESCRIPTION

Only the super-user may set the time. If no argument is given, the current date and time are printed. Otherwise, all of the arguments are taken together with the present date and time to produce a new date and time. The systems' idea of the time is set.

[year], [date], [hour], [minute], and [second] are each given as decimal numbers.

[month] is the name of a month, in English; case is ignored. The first three letters of the name of the month must be given.

If any specification is omitted, the current value of that specification is used in its stead. A common error is to attempt to set the day of the week. This is not permissable in the command line.

EXAMPLES

date 25 Dec 1983

date 14:45

NOTES

The clock is subject to hardware-induced problems that may cause the time to be off occasionally.

NAME

dc - desk calculator

SYNTAX

dc

DESCRIPTION

Dc is a stack-oriented desk calculator allowing integer arithmetic and conversion. Its operation is much like a Reverse Polish Notation calculator. It reads the standard input and writes the standard output. Each token is processed as soon as it is read; ? is the only error diagnostic. The following is a list of the tokens, their meanings, and the action dc will take.

<number>

A number. The number is pushed on the stack, (i.e., remembered). A number must begin with a digit or a minus sign followed by a digit. For example, to input ten in base sixteen, type 0a.

'<character>

An apostrophe followed by a character is converted to that character's ASCII value and pushed onto the stack.

+, -, *, /, %, &, |, ^, <, >

These are the binary operators. The top two values on the stack are combined. Division by zero is not permitted. If a binary operation is attempted with fewer than two numbers in the stack, an error will result.

The following are the operators and their meanings.

+	add
-	subtract
*	multiply
/	divide
%	modulo
&	bitwise and
|	bitwise or
^	bitwise exclusive or
<	bitwise shift left
>	bitwise shift right

~ The top value on the stack is bitwise inverted. It is an error for the stack to be empty.

i Set the input radix. The input radix is set to the value on the top of the stack. Values less than 2 are not accepted. Values larger than 36 are liable to cause garbage. All further input is assumed to be in the new radix.

o Set the output radix, as above.

p Print the value on the top of the stack.

c Clear. The stack is emptied.

EXAMPLES

What is 243 in hex?

```
 bd dc
16 o 243 p
```

What is a capital N in octal?

```
dc
8 o 'N p
```

NAME

dcheck - file system directory consistency check

SYNTAX

dcheck [-f] file system ....

DESCRIPTION

Dcheck reads the directories in a file system and compares the link-counts in each i-node with the number of directory entries by which it is referenced.

ERRORS

When a file turns up for which the link-count and the number of directory entries disagree, the relevant facts are reported. A dangerous situation occurs when there are more entries than links; if entries are removed, the link-count drops to 0 and the remaining entries point to thin air - they should be removed.

When there are more links than entries, some disk space may be wasted but the situation is not dangerous. If the -f option is given, dcheck will try to ratify the situation.

NOTES

Extraneous diagnostics may be produced if **dcheck** is applied to an active file system.

SEE ALSO

fsck (1), ncheck (1), icheck (1)

NAME

ddt - a program for examination and manipulation of non-text files

SYNTAX

ddt [file]

DESCRIPTION

Ddt can be a handy tool for programmers and others who need to manipulate arbitrary (i.e. non-ASCII text) files.

It is functionally analogous to CP/M's DDT, but the Digital Research version works with memory, while this one works on files.

Changes made to a file under ddt are immediately effective. There is no intermediate buffer as with some editors.

Addresses (mentioned below) are given in hexadecimal, (up to six digits long). If the first character of an address is an ' (apostrophe), the value of that number will be that of the ASCII character following.

The following are valid addresses:

```
0       6       b       B       'y
c3b0    78ba4   'A      FFFF    '/
```

Address list may be comma-separated or space-separated.

The following are valid address lists:

```
0,ffff   0,5,7   0,3000,'a       'a,'b,'c
```

No command takes more than three addresses. Extra addresses are ignored.

Ddt always remembers the last file address accessed. In most cases, if you give no address on the next command, this old remembered address is used.

Commands:

d - display

Takes 0, 1, or 2 arguments. Displays the contents of the file in hex bytes on the left and ASCII characters on the right. If no address follows d, the next 128 bytes, starting at the given location, are displayed.

Finally, if two addresses are given, that part of the file which lies between the two addresses is displayed.

Examples:

```
d 0, ffff

d

d 55

d a000
```

f - fill (requires three arguments)

f A, B, C

The portion of the file which lies between addresses A and B (inclusive) is overwritten with a flood of the character C.

If C is large, then the low order byte of C is used.

Example:

```
f 0, 3000, e5
```

writes E5s from 0 to 3000 hex.

r filename (Read)

The file on which ddt is operating is changed to filename if possible.

Examples

```
r new
r /usrsrc/cmd/pbx.c
r /etc/dtab
r a/b
```

s (address) Substitute

Takes 0 or 1 arguments. if no argument is given, the currently remembered file address is used. The **substitute** command allows you to examine and optionallly change the file on a byte-by-byte basis. After entering the **substitute** command you will be in "substitute" mode until you exit it by entering a single period on a line. If while in the substitute mode you press RETURN, the byte you were looking at will remain unchanged and you will be stepped to the next byte. If you enter a byte value and then push RETURN, the byte will be changed to the byte entered.

/ (address)
// (search)

Ddt has the ability to search for patterns in a file. When the "/" command is given, you will be asked to enter a series of bytes to search for. If an address follows the "/", the search commences at the given address in the file. Without an address, the search commences at the current file address.

The "//" command is a shorthand to search again for the previously given pattern.

NAME

detab - converts tabs into an equivalent number of spaces

SYNTAX

detab

DESCRIPTION

Detab acts as a filter, copying the standard input to the standard output unchanged with the exception that tabs are expanded to an equivalent number of spaces. **Detab** assumes tabs are every eight positions at columns 9, 17, 25, etc.

EXAMPLE

To prepare a file for printing on a device that does not understand tabs:

```
detab <file >output
```

SEE ALSO

entab (1)

NAME

df - disk free space

SYNTAX

df device ...

DESCRIPTION

Df finds and prints the number of free blocks on each named device, each of which is assumed to contain a file system.

This command is somewhat analogous to the CP/M STAT command.

EXAMPLE

df /dev/root

SEE ALSO

dcheck (1), icheck (1), ncheck (1), fsck (1)

NAME

diff - find differences in lines of text files

SYNTAX

diff [-e] file1 file2

DESCRIPTION

Diff is a file comparator intended for use with text files. It attempts to find a minimal set of differences between lines of the given files.

Differences are reported by lines of the form

N a R,S

N,M d R

N,M c R,S

Where N and M are line numbers in file1; R and S are line numbers in file2.

Following each line of this type is a list of the affected lines in file1 marked by "<" and a list of the affected lines in file2 marked by ">".

If the -e flag is given, the output takes the format of a command script for **edit** to change the first file into the second.

ERRORS

There are limitations on the size of files. An "Out of memory" error message is printed if the problem is too large.

Spurious blanks lines are sometimes generated in output produced under the -e option.

EXAMPLE

```
diff apple pear

4 a 7, 9
> A text line
> another text line
> a third text line
```

means that one of the changes required is to add lines after line number 4 in the first file.

After the addition is done, the lines will become lines 7 thru 9 in the second file.

NAME

down - take the system down

SYNTAX

down

DESCRIPTION

The **down** command is the orderly way to bring Micronix down.

Only the super user may use this command and it only functions when Micronix is in the multi-user mode.

SEE ALSO

User's Reference Manual, Installation and Operation

NAME

du - occupied disk space, in blocks (512 bytes per block)

SYNTAX

du [-a] [-s] [name ...]

DESCRIPTION

Du is a very useful memory management tool that reports how much disk space is currently being held by the various files in the system. The flags determine the level of detail given in the reports. (-a = all files; -s = grand total; neither = directories) Du is complementary to the df function, which reports free disk space available per device.

There are several combinations of flags (a, s, or neither) and arguments (file or directory names), each of which produces a slightly different breadth of coverage and level of detail. The examples in the following section address the practical uses of du.

First, if you specify a particular filename, the file must be in your current directory and you have to use one or the other flag (it makes no difference which one you choose in this case). Obviously, du will report only the size of this one file, in the format

filename

Next, if you specify a directory name, it too must be contained in the current directory, or else you can give the path leading to it from the root directory (e.g. du -a /usr/bin). Du will begin at this directory and proceed down the tree through all the files and subdirectories contained within it.

If you don't give a file name or a directory, du assumes that you mean the current directory.

Options:

-a All. This flag gives you the most detail, with a block count for each file. There will also be subtotals for directories embedded in the list, so don't be surprised if the bottom line doesn't look like the sum of all the numbers above it.

-s Summary. Only a grand total is displayed.

Using neither flag produces an intermediate level of detail, where only directory subtotals and a grand total are displayed.

EXAMPLES

To find out just how much space your whole system is using, simply enter:

du -s /

To find out which directory has that enormous file in it:

du / (to begin at the root directory, in this case)

To get the name of the offending file itself:

du -a directoryname

To get an overview of the top layer of directories in the file system:

du -s /*

NOTES

If there are several links to the same file, the space used by that file is counted only once. Space used by directories themselves and by pointer blocks is not counted, as these are considered overhead and in any event do not amount to more than a few percent. Du doesn't give special consideration to files with large vacant areas within them; the empty space is counted as if it were filled.

SEE ALSO

df (1), ls (1)

NAME

edit - text editor

SYNTAX

edit [file]

DESCRIPTION

Edit is a line-oriented text editor which operates on a temporary copy of the file it is editing so that changes made have no effect on the file until a w(rite) command is issued. The copy of the text being edited resides in a temporary file called the buffer.

Commands to edit have a simple and regular structure: zero or more addresses followed by a single command character, possibly followed by parameters to the command. Missing addresses are supplied by default. Certain commands allow the addition of text to the buffer (i - insert, c - change, a - append, etc., see below). Input governed by these commands is merely collected; no additional commands are recognized in this mode.

To get back to normal command mode, type a period (.) alone at the beginning of a line.

Edit supports a limited form of regular expression notation. A regular expression is used in commands and addresses to refer to patterns in the text. Following are the specifications for regular expressions:

1. Any character except a special character matches itself. These are the regular expression delimiters: { }, [ ], ?, and sometimes %, *, and $.

2. A ? matches any character (except newline).

3. A @ followed by any character except (or) matches that character.

4. A string enclosed in [and] (or [^ and]) matches any character in, or not in, the set consisting of the characters of the string (example: [abc] matches a, b, or c). If a substring of the form a-b with a and b letters or digits within the brackets, it is understood to mean the set of characters spanned by the two limits. (Examples: [0-9] matches any digit; [a-zA-Z] matches any upper or lower letters.)

5. A regular expression of forms 1-4 followed by * matches a contiguous run of 0 or more occurrences of the preceding regular expression. (Examples: * matches any number of spaces (including zero); ?* matches the

entire line (any number of occurrences of any character).

6. A regular expression enclosed in { and } matches what the regular expression matches. [It also has side effects - see the s - (substitute) command below.]

7. The concatenation of regular expressions matches the concatenation of what is matched.

8. A % as the first character of a regular expression restricts matches to the left end of the line. A $ as the last character similarly limits matches to the right end of the line.

9. A regular expression matches the longest among the left-most of possible matches.

10. An empty regular expression stands for a copy of the last regular expression encountered.

To understand addressing in edit, you need to know that at any time, edit remembers a current line. In general, the current line is set to the last line affected by the previous line. The current line is initially set to the last line of the file. Addresses are constructed as follows:

A (.) addresses the current line.

A $ addresses the last line.

A decimal number (n) addresses the nth line of the buffer.

A regular expression enclosed in / addresses the line found by searching forward from the current line for the first line, which is matched by the regular expression. If necessary, the search wraps around to the beginning of the buffer.

A regular expression enclosed in \ addresses the line as above, except that search is backwards through the file.

Two addresses separated by a + or - addresses the line in the buffer which is the sum (or difference) of the two addresses. (Example: 2+1 addresses the third line of the buffer.)

If an address begins with + or -, addition or subtraction is taken with respect to the current line (i.e, -5 is equivalent to .-5).

An address consisting entirely of minus signs or entirely of

plus signs is taken to mean the line forward or behind the current line by as many lines as plus or minus signs. (Example: "--" addresses the line before last.)

Commands may require one, two or no addresses. Commands which require no addresses regard the presence of an address as an error. Commands which accept addresses assume default addresses if too few addresses are given.

Addresses are separated by a ,. The second address of a two address sequence must correspond to the following line. (Example: 3,2 is an illegal address list.) A % is shorthand for 1,$.

In the following list of edit commands, the default addresses are shown in parentheses. The parentheses are not part of the address, but are used to show that the given addresses are the defaults. Most commands may be followed by a "p" meaning that the line is to be printed after the operation is performed. With many commands this happens automatically.

(.) a
<text>
(.)

The given text (which may be excessively long and/or consist of many lines) is a(ppended) after the present line and before the next line if there is one. 0 is legal as an address for this command; the text is placed at the beginning of the buffer.

(.,.) c
<text>
(.)

The c(hange) command first deletes the addressed lines, then accepts text which replaces these lines.

(.,.) d
<text>
(.)

The addressed lines are d(eleted) from the buffer. (Example: 1,$d completely erases the buffer.)

e[!] [filename]

The e(nter) command causes the entire contents of the buffer to be deleted, then the named file to be read in. If you are finished editing one file and wish to switch to another, use **enter**. If the file you are presently working on has not been written, a diagnostic is printed. This is intended to reduce the possibility of accidental loss of work. The optional exclamation point overrides this safety feature.

f [filename]

The remembered f(ilename) is printed. If a filename is given, the remembered filename is changed to it.

(1,$) g/regular expression/command

G(lobal). First, every line in the buffer matching the regular expression is marked. Then for each marked line, the command is executed with (.) initially set to that line.

(.) i <text> (.)

The given text is i(nserted) just before the addressed line. This command differs from **append** (above) only in the placement of the text.

(.,.+1) j

The j(oin) command joins the addressed lines into a single line. In any case, an additional space is inserted between the text of the formerly separate lines.

(.,.) l

The addressed lines are l(isted) in an unambiguous way. Control characters are displayed as a caret ("^") followed by the upper case letter corresponding to the control character. Dirty characters (with the sign bit set) are displayed as three-digit octal preceded by a back slashes ("\").

(.,.) m a

The m(ove) command repositions the addressed lines after the line referred to by the address "a".

n[!]

Edit the n(ext) file, if any, in the argument list. This is an easy way to edit a group of files without having to remember all their names yourself. If the exclamation point is given, the command is forced. (Example: edit a b c d e; ... n ... n ...)

(.,.) p[l][=]

The addressed lines are p(rinted) in the specified format. The "l" means to print in the manner described in the "l" command (above). The = means to number the lines. Note that the printing format remains in effect until it is changed again.

q[!]

Use the q(uit) command to terminate an editing session. No automatic write of the file is done. A diagnostic is printed if the buffer has been changed since the last write command. This feature offers some degree of protection against accidental loss of work. The optional exclamation point immediately following the "q" overrides this safety feature, (i.e., quit impetuously).

(.)r [filename]

The r(ead) command reads in the named file after the addressed line. If no filename is given, the remembered filename is used. The filename is remembered if there is as yet no remembered filename. Address "0" is legal for this command and causes the named file to be read in at the beginning of the buffer.

(.,.)s/regular expression/replacement/[g]

In the s(ubstitute) command, each addressed line is searched for strings matching the regular expression. An optional "g" (global) may follow the substitute command. In the presence of a "g" all occurrences of matched strings are replaced by the given replacement text. In the absence of a "g" only the first matched string in each line is replaced. It is an error for the substitution to fail on all addressed lines.

An & appearing in the replacement text is replaced by the string matching the regular expression. (Example: s/toves/slithy &/ would change "toves" into "slithy toves".) @n where it is a digit (1 through 9) is replaced by the nth regular subexpression enclosed in } and {. When nest parenthesized regular sub-expressions are present, n is determined by counting occurrences of { starting from the left.

These regular sub-expressions tagged by number can be extremely useful in automating otherwise impossible editing tasks. (Example: s/{[A-Z][a-z]*}, {[A-Z][a-z]*}/@2,@1/ would convert a list of people's names from: "lastname, firstname" to "firstname lastname".)

Lines may be split by substituting newline characters into them. @n in the replacement pattern is understood to mean newline.

If no replacement pattern is given, the previous replacement pattern is used, (i.e., "s/pattern").

If neither pattern is given, then the previous substitute command is re-executed, (i.e., "s"). This feature saves a lot of time.

(.,.)t a

The t(ranscribe) command works exactly like the **move** command, except that a copy of the addressed lines is placed after the line corresponding to address "a".

u

Undo previous command; only works if file is still in buffer.

(1,$) w[!] [filename]

The addressed lines are w(ritten) onto the named file. If no file name is given, the remembered name, if any, is used. If the named file exists and is not the remembered file, a diagnostic is printed. This is intended to prevent accidental overwriting of valuable files. The optional exclamation point immediately following the write command character suppresses this safety feature.

(1,$) x/regular expression/command

Exactly like the global command (above) except that all lines not matching the regular expression are affected.

(.+1) z [N]

Print the next n lines (default 22). The value n is sticky, in that it becomes the default value for future scroll commands. This command is quite useful for marching your way through text.

($) =[l]

Lines are printed with line number. If [l] is given, then the line is additionally printed in the style described in the "l" command (above). Lines are printed out with numbers until the "=" sign is entered again.

< file

The named file is read as further input for edit. The command may contain further < commands.

(.+1,.+1) <newline>

An address alone on a line causes the text of the addressed line to be printed. A blank line alone is equivalent to and is useful for stepping through text.

NOTES

Sending a ^C character to edit under CP/M (Digital Research) will cause your work to be ungracefully lost (but this is not a problem under Micronix).

SEE ALSO

Kernighan, Brian W. and P.J. Plauger, Software Tools, (Addison-Wesley, 1976).

NAME

entab - replaces blanks by tabs

SYNTAX

entab

DESCRIPTION

Entab acts as a filter, copying the standard input to the standard output unchanged with the exception that groups of blanks are replaced by a tab where possible. **Entab** assumes tabs are every eight positions, at columns 9, 17, 25, etc. **Entab** is useful for speeding up printing and communication over slow paths, or to effect file compression. **Detab** is its inverse.

EXAMPLE

To compress a file in which there are many spaces:

```
entab <file >output
```

SEE ALSO

detab (1)

Brian Kernighan and P. J. Plauger Software Tools , Addison West Publishing Company, 1976.

NAME

expand - uncompress files

SYNTAX

expand [input [output]]

DESCRIPTION

Expand reverses the effects of **compress. Expand** is not guaranteed to do anything sensible with input which is not the result of a **compress** run. The first named file is expanded and written into the second named file. If no file names are given, standard input and output are used. If only one file name is given, the named file is used for input, and its expansion is written to the standard output.

EXAMPLE

A (Micronix) do-nothing pipeline:

... | compress | expand | ...

SEE ALSO

compress (1)

Brian Kernighan and P. J. Plauger

Software Tools

, Addison West Publishing Company, 1976.

NAME

far - floppy archiver

SYNTAX

far device [flags] [files]

DESCRIPTION

Far provides a simple interface to CP/M floppy diskettes. It is the primary means of incorporating new CP/M software into the system. It may also be used for backing up files on diskettes, and recovering them as necessary. "Device" is assumed to be the name of a Micronix special file (such as /dev/fla) corresponding to a floppy drive that contains a CP/M diskette. Far can copy files in either direction, and can also be used to read the directory of a CP/M floppy. When the -v flag is used, it reports the amount of space still available on the diskette.

Names such as /dev/fla may be abbreviated as fla. You can check the names of the storage devices by entering

dir /dev (return)

Note the different names for 5.25" floppy drives: mfa and mfa2. Use mfa to copy to or from diskettes that are formatted as single-sided. Use mfa2 for double-sided diskettes. Both names will direct the far command to the same physical device. Using the wrong name by mistake won't hurt anything; it'll just let you know that you should use the other one instead.

The flags are:

-c **Create.** Causes the previous contents of the diskette to be thrown away. The entire directory is cleared prior to other operations. This won't have any effect on the actual diskette unless the -r or -d flag is specified. This flag also has the effect of disabling the check for a CP/M diskette. Normally, far refuses to write on a diskette which doesn't look like a CP/M diskette. The -c flag makes any diskette look like a CP/M diskette.

-d **Delete.** Delete the named files from the floppy diskette. If no file names are given, no action is taken.

-p **Print.** Each named file is sent to the standard output (most likely your terminal).

-r **Replace.** Each named file is copied to the floppy. If no file names are given, no action is taken. If a file of the same name already exists on the floppy, it is silently overwritten.

-t **Table.** Print a table of contents of the floppy diskette. If any files are named, only those named and present are listed. The contents of the diskette are not changed.

-uN **User number.** The default user number is zero. The letter u followed immediately by a decimal number will cause any of far's actions to happen with respect to this user number. With -t, only files with the same user number show up. With -r, files are created on the diskette with the specified user number. With -x, only files with the same user number are extracted. Most files on most diskettes are "user zero" and you don't usually have to worry about the -u option flag. The -u flag may be embedded in the flag argument as in "-xul4v".

-x **Extract.** Each named file is copied from the diskette to the current Micronix directory. If no file names are given, the entire contents of the diskette are extracted.

-v **Verbose.** Be verbose about it. Each operation is printed along with the affected file name. Files deleted are preceded by a "d", those replaced are preceded by an "r", those extracted are preceded by "x". The percentage of the diskette that is occupied, along with the number of bytes still available, is reported when far is finished.

At most, one of the flags - d, t, r, x - may be present. If none are present, -t is assumed. -v may be used in combination with any of these.

The file list may contain wild card matching patterns to be applied to the set of file names in the CP/M diskette's directory. Refer to the section "Argument Expansion" in the sh program writeup in this manual.

EXAMPLES

You receive a new CP/M software package in the mail on diskette and want to run it under Micronix. To copy its entire contents to Micronix, insert the diskette into

floppy drive A and type:

far fla -xv

To copy a file from a diskette, changing its name in the process:

far fla -p filename > newname

To copy only the ".doc" files over from a CP/M mini-floppy:

far mfa -xv "*.doc"

To view the files of user 3 on a CP/M mini-floppy:

far mfa -u3

To copy them off:

far mfa -u3x

To copy a file from diskette into a file that is elsewhere than your current directory:

far flb -x "name" > /dir/...
(path from root to desired location; you could also specify a new name)

The following discussion is intended primarily for users that are having trouble getting far to do what they want it to.

One aspect of far command syntax somewhat complicates the issue of using it, namely, the use of quotation marks. An example should help clear this up. Let's say your current Micronix directory contains two files, **a.com** and **a.bak**. Your CP/M floppy contains a file **a.pil** that you wish to copy into your current directory. Since this is the only file on the floppy that begins with "a", you lazily enter **far mfa -xv a*.** Next you are wondering what's wrong when you are faced with the message "a.com not found on diskette / a.bak not found on diskette". Of course they weren't found. You didn't even want Micronix to be looking for them.

What has happened is this: Your command interpreter (shell) thought you meant to search the current directory for a match to the wildcard argument before going out to the floppy for the copy. It indeed found two matches (a.com & a.bak), so Micronix went gleefully out to the floppy to copy them. In vain, unfortunately.

The solution here is to use quotation marks around your filename(s). Had you entered **far mfa -xv "a*"** , the shell would have left the command alone, and far would have gone directly out to the floppy to find anything that began with an "a". Success.

Now, just to set the record straight, there is a further complication that may help you to understand how the shell operates. Suppose the file on your floppy is called b.pil instead of a.pil. If you enter **far mfa -xv b.pil** (no quotes), the file will in fact be copied, contrary to the previous discussion. Furthermore, everything on the floppy diskette will be copied, much to your chagrin. This is because your shell took b.pil to your current Micronix directory, looked to see if a file of that name was really there, and finding no such file, it passed the command to Micronix as " **far mfa -xv** ". As noted above in the description of the x flag, if there is no file specified, the whole diskette gets copied.

In summary, you should enclose the argument in quotes whenever you want to copy specific files from the floppy to Micronix. The only exception to this is a situation where you have certain files already established on Micronix and you want to extract only those floppy files that match the existing Micronix filenames.

FORMATS

The far program can read files on CP/M diskettes with the following formats:

NorthStar single and double sided;

Morrow single and double sided hard sectored;

Morrow MicroDecision, single and double sided soft sectored;

Osbourne single sided;

IBM PC single sided (CP/M86 only, see dar for PC-DOS)

XEROX 820 single sided;

IBM 3740 single density 8";

Various 8" formats (256, 512, 1024 bytes per sector, double density).

The DJ/DMA attempts to determine the diskette format, and far will attempt to translate the directory for the appropriate file system parameters (i.e., number of sectors, block size, etc.)

NAME

field - pick selected fields from each line of the input

SYNTAX

field [-t separator] field-numbers <input >output

DESCRIPTION

Field reads its input line by line. It breaks each line up into "fields". Then the fields requested by the "field-number" arguments are printed out in the requested order. By default, input fields are separated by whitespace (spaces and tabs), while output fields are separated by tabs. The field separators can be changed (both at once) with the -t option. The first field is numbered 0. If a line begins with a separator, then field 0 is taken to be empty.

EXAMPLES

field 3 0 2 2 <x

reads whitespace separated fields from each line of the file x and writes out the 3rd, 0th, 2nd, and 2nd fields, in that order. For instance, if x contained

```
Greg Smith      $2.50   4/9/83
     Jones      $3.00   7/4/83
```

then the above example would print out

```
4/9/83  Greg    $2.50   $2.50
7/4/83          $3.00   $3.00
```

field -t: 5 4 </etc/passwd >temp

reads the password file and writes a list of home directories and user names to temp. For instance, if /etc/passwd contained

```
root:xxxxxxxx:0:0:Super:/:sh
sally::10:1:Sally Jones:/a/sally:upm
```

then the resulting temp file would contain

```
/:Super
/a/sally:Sally Jones
```

ls -l | field 8 1

will print a directory listing giving only file names and permissions.

BUGS AND CAVEATS

Input line length is limited to 512 bytes. If an input line

begins with a field separator, then the 0th field is empty.

SEE ALSO

paste (1)

NAME

file

SYNTAX

file file1 ...

DESCRIPTION

File attempts to guess the type of each named file. Currently, the following types of files are recognized:

A-natural source file

Block special file

C source file

CP/M text file

Character special file

Data

Directory

Empty

Executable

Pascal source file

Ratfor source file

Relocatable module

Software Tools archive

Text file

Text processor input

Whitesmiths library

NAME

find - find files

SYNTAX

find pathname-list expression

DESCRIPTION

Find searches each of the given pathnames for files matching the given expression. If the pathname of a directory is given, all files in that directory and all of its subdirectories (recursively) are searched.

The expression above is built from constructions like the following:

-name filename

True if the filename argument matched the current file name. Normal shell argument syntax may be used if escaped (watch out for '[', '?', '*', '^').

Examples: '-name a.out' would match all files named "a.out". '-name "*.o" ' would match all files in the tree whose name ends in ".o"

-perm onum

Where onum is an octal number. True if the file's permission flags match the given number.

Example: ' -perm 777 ' would match all files whose permissions are "wide open".

-type c

True if the type of the file c, where c is b, c, d or f for block special, character special directory or play file.

Example: ' -type f '.

-links

True if the file has n links.

Example: '-links 1' would consider only files with exactly one link.

-user name

True if the file belongs to the user uname (login name or numeric user ID).

Example: ' -user root ' would screen all files except those owned by root.

-group name
True if the file belongs to group gname (group name or numeric group ID).

Example: '-group sales' would consider only files in group "sales".

-size n
True if the file is n blocks long (512 bytes per block).

-inum n
True if the file has inode number n.

-atime n
True if the file has been accessed in n days.

-mtime n
True if the file has been modified in n days.

-exec command
Always true. If the command is complex it must be enclosed in quotes ("). If the sequence {} occurs in the command, the current filename will be substituted for it.

-ok command
Like -exec except that the generated command is written on the the standard output, then the standard input is read and the command executed only upon the response 'y'. In other words, you are asked if it's ok to execute the command. Expect to see something like: < rm /a/smith/books/3.1 > ok ?

-print
Print the file name. Always true, causes the current pathname to be printed. (Note that 'find' does its work silently and if you don't tell it otherwise, nothing will be printed.)

-newer file
True if the current file has been modified more recently than the argument file.

If two of the above constructions appear one after the other, then the effect is a logical AND of the two. That is, a primitive will be processed only if all of the previous primitives were successful.

EXAMPLES

To remove all files named a.out that have not been accessed in a week

```
find / -name a.out -atime +7 -exec "rm {}"
```

To simply list the names of all the file in a hierarchy:

```
find . -print
```

FILES

/etc/passwd - to look up user names.

/etc/group - to look up group names.

SEE ALSO

sh (1), filsys (5)

NAME

form - format text

SYNTAX

form [-t] [-o output] [file ...]

DESCRIPTION

Form is a general purpose text formatter with text filling, underline, indenting, centering, and other useful capabilities. This document was prepared using **form.**

The standard input is read and formatted according to instructions embedded in the input file; the result is written to the standard output.

If a list of files names is given, they are used as the input. If "-o" followed by a file name appears on the command line, then the output is written to the named file.

If "-t" appears on the command line, then the output is made more suitable for a glass tty, i.e. long sequences of blank lines are avoided.

Input consists of text lines interspersed with command lines destined to be printed.

If the first character of a line is a period (.), **form** attempts to interpret the line as a command; in any case, the text of such lines will not appear in the output. If a command is not recognized, it is ignored. Only the first 2 characters of the command name are significant, the rest are ignored. So, for example, you may use .break or .br with the same effect.

The following commands are implemented:

.ad

Turn on right adjust mode. Causes the right edge of the text to be squared up. Spaces are embedded between words in the text if necessary to achieve this. Right adjust mode is on by default. It can be cancelled by a .na command (See below). Right adjust mode will remain in effect forever or until a .na command is encountered. This paragraph is done in right adjust mode.

.bd n

Cause the next (or number of given) line(s) of text to be double-striken for bold face appearance. (Default is 1.) With the mode commands that take a line count argument (.bd, .ce, and .ul) If you want to embolden, center, or underline a large

number of lines and don't want to bother with counting them, the best way to do it is:

```
.bd 1000

text
text
text

.bd 0
```

This paragraph is done in bold face. Printer must have back spacing ability, otherwise overstrike (Section 1) must be used.

Example:

```
form file | overstrike | lpr &
```

.bp n

Begin page number n. Jump to the top of the next page. Set the page number to n (default is sequential order beginning with 1).

.br

Cause break. The next input line will begin at the left margin. Many commands cause a break. Beginning a line with white space will cause a break.

.ce n

Center the next (or number of given) input line(s).
Each line here is centered.
Default is next line only.

.fi

Set filled text mode. All text lines will be filled out with blanks if necessary so that the right margin is justified. This paragraph is filled. Fill mode is the initial default condition.

.fo TEXT

Set the footer. The specified text will be printed at the bottom of each page. If the symbol # appears in the text, the page number will be substituted for it.

.he TEXT

Set the header. The specified text will be printed at the top of each following page. If the symbol # appears in the text, the page number will be printed in its stead.

.in n

Set the indent. The indent is set to the nth column. The command ".in" is interpreted to mean no indent. Default is 0.

.ls n

Set the line spacing to n lines. The command ".ls 2" is interpreted to mean double space. The line spacing is set to 1 (default) initially.

.na

No adjust. Turns off right adjust mode. The right edge of the text will be ragged. This mode gives a more human look as the average typist does not produce copy with a squared up right edge. Adjust mode will remain off forever or until an .ad command is encountered. This paragraph is done with right adjust mode turned off.

.nf

No fill.
Turns off fill mode.
With fill mode in effect, words are collected
until no more will fit on a line, then a new line is started.
In No Fill mode, each line is preserved in the output.
This paragraph was done in No Fill mode.

.pl n

Page length. The page length is set to n lines long. If no number (n) is specified, the page length is set to 66 lines (default).

.rm n

Right margin. The right margin is set at column n. If no number (n) is specified, it is set at column 60 (default).

.sp n

Space down n lines. If no number (n) is given, spaces 1 line (default).

.ta n

Set the tab. Value (n) determines amount of tabbed spaces. Entered with ^I sequence or TAB key. When used with the (.ti) command, it can be used to produce "hanging" paragraphs.

```
.ta 10
.in 10
.ti -10
Melons<TAB>Watermelons, among the nation's
best selling fruit can weigh up to umpteen
pounds.
```

would produce something like:

Melons Watermelons, among the nation's best selling fruit can weigh up to umpteen pounds.

.ti n

Temporary indent. The indent is set to a value (n) for the next line only. This is quite useful for numbering paragraphs - a negative value produces a "hanging" paragraph.

.ul n

Cause the next (or number of given) input line(s) to be underlined. The default is 1. This paragraph is underlined. Printer must have double-strike capability. If not, see the overstrike program (Section 1) for instructions on using this program.

Example:

form text | overstrike > /dev/ttyH

SEE ALSO

edit (1), overstrike (1), diablo (1), lpr (1)

NAME

fp - floppy backup service

SYNOPSIS fp [rxt][vfiqs] [device] [filenames ...]

DESCRIPTION

fp allows you to backup files on floppies under the Micronix system.

In this document the word "volume" will be used to refer to a single instance of storage medium, for example, diskette, tape reel, tape cassette, or removable hard disk.

The word "device" will refer to the disk or tape drive which manipulates the volume. For example floppy drive, tape drive, hard disk drive.

The word "set" will mean a series of volumes produced by a single fp run that you get when you make a copy of a large file structure. Theoretically, a set could run to 30 volumes or so in an imaginable case with the current Micronix system.

FEATURES

* ability to split files across volumes for backup of large files.

* minimal seeking on the device to reduce running time.

* incremental dumps possible using the central "dtab" dump table.

FLAGS (choose exactly one)

r write files to the device.

x read files from the device.

t display files on the device.

More flags (optional)

v verbose - The v flag causes a note to be printed as each file is processed. The v flag is always on (unless the s flag is used to suppress it). With the t flag, v causes extra detail about the files on the fp volumes to be given. The v flag is mainly here for backward compatability.

i incremental - Normally, fp processes all the files it encounters. With the i and r flags in effect, only files which have been changed since the dumpdate are acted upon. The i flag forces fp to make an entry with

the current date in the file /etc/dtab. (Dtab stands for dump table.) If this file is lost, so is the memory of previous dump dates. One entry in the dtab file is produced for each file named on the command line. If no files are named on the command line, the entry in "dtab" will be for the current directory. If no dtab file exists, fp will attempt to create one. This will work subject to the permissions of the /etc directory. See the examples for an actual use of the i flag.

c create - This flag only works with the r flag. Create means to erase any files that may already be on the volume. Obviously, you only want to use this with new volumes, or volumes you are recycling.

l label set - This flag causes fp to ask for a name to label this set of volumes with. The label may be up to 250 characters long.

e formatting program - If fp encounters an unformatted disk during a backup operation, it will automatically invoke the fdj program, which will query the operator for information and format the currently inserted diskette. This may save you during a long backup. If you want to change the name of the formatting program invoked, use the e option, followed by the name of the program to be used.

f device file name - Specifying f allows the choice of a different device name. The default device name is /dev/fp. /dev/fp is normally linked to /dev/mfa which is the first 5 1/4 inch floppy drive connected to a Morrow DJDMA running under Micronix. If your system uses 8 inch drives, you should relink /dev/fp. HINT - use the following commands for an 8" drive system:

```
cd /dev; era fp
ln fla fp
```

p print - Operates similarly to extract, but the files are sent to the standard output instead of a file. This is theoretically useful to manually peruse copied images of text files.

q quick copy - Normally all files are checked for readability immediately after being copied to a volume. Specifying the q flag bypasses this safty feature.

s silent operation - Normally, in the spirit of micros, fp announces each file transaction. Specifying the s flag suppresses much of this printing. You are still notified of error conditions.

GENERAL RULES

If the name of a directory is given to **fp**, it takes it to mean that dirctory and all its descendants.

If no file names are given, the implication is to do all files. For the t flag, this means all the files stored on the current volume. For r, this means the current directory and all its descendents. For the x flag, all files in the set of volumes are extracted.

If you name your files with absolute path names, that is, starting with the / of root, any file that you extract will be restored to its former position in the file system. If you backup files starting in the current directory, or use filenames that don't begin with a /, you can restore files to any directory in the file system.

It's possible for a volume to be full yet contain few or no files. If a read-back error occurs, the remaining space on the volume, after the block with the error, will be effectively marked out.

Links are preserved across **fp** operations. If you copy a file to which there are several links, only one copy of the file's data will be made.

For each file on the volume the t flag, for **table**, prints the length in bytes, the modification time of the file at the time it was copied, the file's name the way it was when backed up, and possibly a notation concering which part of the file represented.

```
4872 Nov 14 7:30 myfile [part 2]
```

In the above example, this part of the file is 4872 bytes long, the file had been last changed on Nov. 14th at 7:30 in the morning. The file is named myfile, a relative filename and only part 2 of the file is represented here. Part one fo the file is found on the previous volume of the set.

The most you can get on a current technology floppy is 800 blocks. 2400 blocks will fit on an 8 inch floppy. **fp** automatically sets its directory size to one eightieth the size of the volume, with a minimum of 20 blocks. In the case of soft sectored 1K 5 inch floppies, the volume size is 800 blocks, the directory size is 20 blocks. A maximum of 160 entries can be put on the volume. There are 8 directory entries per directory block. Sometimes this is wasteful (for instance when the average file size is small).

When splitting files, **fp** does not waste space. It writes right up to the end of the volume.

EXAMPLES

To produce the first full dump of your file system, have many volumes formatted and:

```
cd / ; fp cril
```

This creates a new volume (or set of volumes), backs up everything starting with the root directory (/), makes an entry in /etc/dtab, and requests a label for the volumes.

To produce an incremental (the next day for instance):

```
cd / ; fp cril
```

Using another formatted diskette, this backs up only the files that have been modified since the last dump of the root.

To print a label for a volume:

```
fp | lpr
```

To produce a backup of your directory alone:

```
cd ; fp cril
```

To completely restore your home directory from a dump, install volume 1 of the set in the device and:

```
cd ; fp x
```

To copy to an 8 inch floppy under Micronix:

```
fp crf /dev/fla
```

To copy a single (possibly large) file:

```
fp r filename
```

SEE ALSO

fdj(1), for details on the formatting program, and dtab(5).

NAME

fsck - file system check and repair

SYNTAX

fsck [-t] [-n] [-b] filesystem

DESCRIPTION

Fsck is a complete Micronix file system maintenance package. System crashes may cause the Micronix file systems (as well as standard UNIX version 6 file systems) to become internally inconsistent in potentially dangerous ways to your data. It is also possible for the physical device on which a file system resides to develop unreadable spots. The **fsck** program rectifies these inconsistencies and maintains the file system.

NOTE! Do not run **fsck** on an active file system. As a rule of thumb, you should run fsck on the root device while in single-user mode. Devices other than the root device should be checked before mounting.

If information in the file system is changing while **fsck** is taking its "snapshot" of the state of the file system, it will get the wrong idea and will try and fix things that were not wrong in the first place. By typing

fsck filesystem

where **filesystem** is the name of the device on which the file system in question lives, you can bring your file system completely up to par.

The device is referred to by its name as it appears in the /dev directory (e.g. /dev/fla).

If no specification is made, **fsck** searches for and attempts to fix all file system problems.

Options:

-t Test all of the blocks on the file system to see that each is readable. Systems on large disks may take some time to complete.

-n No changes will be made. **Fsck** will only look and give you its professional opinion on what it would do under the circumstances.

-b Take a list of block numbers that follows, or the name of a file containing a list of block numbers, and add these blocks to the /badblocks directory to remove them from the file system. Blocks are listed separated by white space

(blanks, tabs and newlines).

One way to produce a list of bad blocks is to use the badspots program. The badspots program reads the bad spot map on 5 and 8 inch hard disks, and converts the physical disk addresses found there to Micronix block numbers, and sends them to the standard output. Badspots is used with fsck on a newly formatted 10 megabyte hard disk to remove bad spots from the new file system with this command

```
fsck -b `badspots m10b` /dev/m10b
```

Fsck can fix:

- Two files claiming the same block. The block is copied and a copy is given to each file.
- File containing a bad block. An attempt is made to copy the bad block; in any case, a new block is allocated to fill the spot. The bad block is assigned to a file in the directory "badblocks".
- Duplicates in the free list.
- Allocated blocks in the free list.
- Blocks missing from the free list.
- Bad block in the free chain. The free list is rebuilt from scratch.
- File for which there is no directory entry (an orphan). An entry is made in the "lost+found" directory.
- Missing directory self link ".".
- Missing directory parental link "..".
- Disconnected directory. Appropriate links are made to directory names in "lost+found," if necessary.
- Imbalanced directory entry - link counts. Resolved in favor of the directory entries, i.e., the link count is changed.

Fsck cannot fix:

- Bad blocks in the i-list or super block.
- Ridiculous numbers in the super block.

SEE ALSO

filesystem (5), icheck(1), badspots(1), dcheck(1), df (1), ncheck (1)

NAME

grep - find instances of a regular expression in a stream

SYNTAX

grep [-c] [-i] [-l] [-n] [-v] pattern [filename...]

DESCRIPTION

Lines in each of the named files matching the regular expression "pattern" are written on the standard output. The rules for regular expressions are as in edit.

The following flags are meaningful in the command line:

-c Only a count of matching lines is printed for each file.

-i The case of letters is ignored in comparisons.

-l The names of files containing matching lines are listed once, one on a line.

-n Each matching line is preceded by its line number in its respective file.

-v All lines not matching the pattern are printed.

EXAMPLE

To search for occurrences of a string in a file:

grep string file

SEE ALSO

edit (1), change (1)

NAME

group - examine or change the group of a list of files

SYNTAX

group [-name] file ...

DESCRIPTION

Associated with each Micronix file is a group number. The **group** number is used to determine group access permission. If you are running a single user, informal, or very low user-population system, the whole concept of groups may be irrelevant to you.

The **group** command can be used to view the group names of files. The relationship of group names to group numbers is entirely established by the file "/etc/group" [see **group** (5)].

Group prints the group name for each file given. If the file's group number has no name, the group number is printed in decimal instead.

If the first argument to **group** is a group name preceded by a hyphen, **group** attempts to change the group of each of the named files to the given name.

EXAMPLES

group file1 file2 file3

to discover which group each of these files belongs to.

group file1 file2 file3

to discover which group each of these file belongs to.

group -sales file1 file2 file3

to "give" each of the files to the group named "sales."

SEE ALSO

group (5)

FILES

/etc/group - group name <-> group number relationships

NAME

help - help processor

SYNTAX

help [N] topic ...

DESCRIPTION

Help prints documentation (if it exists) for each named topic. Complete on-line documentation is resident on the Micronix system.

Help is the same program as **man**, execpt that when it is called as "help" it looks first in /usr/help for a file with the same name as the given topic. Then, like **man**, it searches /usr/man/man0, ..., /usr/man/man9 in order. These are the directories containing the corresponding sections of the Micronix Reference Manual:

/usr/man/man1	Programs
/usr/man/man2	System Entries
/usr/man/man3	Libraries and subroutines
/usr/man/man4	Devices
/usr/man/man5	File formats
/usr/man/man6	Other programs (such as CP/M programs)

The "N" option directs help to the corresponding section of the manual.

EXAMPLE

The command line

help help

prints this page.

Entering

help 2 stty

prints the stty.2 file instead of stty.1.

SEE ALSO

man (1)

- translate object file to ASCII formats

SYNTAX

hex -[flags] <ifile>

DESCRIPTION

Hex translates executable images produced by **link** to Intel standard hex format or to Motorola S-record object file format. The executable image is read from <ifile>, if specified, otherwise from the file xeq.

The flags are:

-h Do not start record for Intel hex files.

-m* Insert the string *, instead of <ifile>, into the Motorola S0 record generated when -s is given.

-r## Interpret the input file as "raw" binary and not as object file. Output is produced as described below, in either format, except that the starting address of the module is specified by the long integer ##.

-s Produce S-record rather than the default hex format.

+# Start output with the #th byte. # should be between 0 and one less than the value specified by the -#flag. -4 +3 produces bytes 3, 7, 11, 1 5, ...; -2 +0 produces all even bytes; -1 +1 outputs all odd bytes; 0 is the default which outputs all bytes.

-# Output every #th byte. -2 outputs every other byte, -4 every fourth; 1 is the default which outputs all bytes.

Output is written to STDOUT.

Hex Files

A file in Intel hex format consists of the following records, in the order listed:

1) A "$" alone on a line to indicate the end of the (non-existent) symbol table. If -h is specified, this line is omitted.

2) Data records for the text segment, if any. These represent 32 image bytes per line, possibly terminated by a shorter line.

3) Data records for the data segment, if any, in the same format as the text segment records.

4) An end record specifying the start address.

Data records each begin with a ":" and consist of pairs of hexadecimal digits, each pair representing the numerical value of a byte. The last pair is a checksum such that the numerical sum of all the bytes represented on the line, modulo 256, is zero. the bytes on a data record line are:

a) The number of image bytes on the line.

b) Two bytes for the load offset of the first image byte. The offset is written more significant byte first so that it reads correctly as a four-digit hexadecimal number.

c) A zero byte "00".

d) The image bytes in increasing order of address.

e) The checksum byte.

An end record also begins with a ":" and is written as digit pairs with a trailing checksum. Its format is:

a) A zero byte "00".

b) Two bytes for the start address, in the same format as the load offset in a data record.

c) A one byte "01".

d) The checksum byte.

S-Record Files

A file in Motorola S-record format is a series of records each containing the following fields:

<S field><count><addr><data bytes><checksum>

All information is represented as pairs of hexadecimal digits, each pair representing the numerical value of a byte.

<S filed> determines the interpretation of the remainder of the line; valid S fields are "S0", "S1", S2", "S8", "S9". <count> indicates the number of bytes represented in the rest of the line, so the total number of characters in the line is <count> * 2 + 4.

<addr> indicates the byte address of the first data byte in the data

field. S0 records have two zero bytes as their address field; S1 and S2 records have <addr> fields of two and three bytes in length, respectively; S9 and S8 records have <addr> fields of two and three bytes in length, respectively, and contain no data bytes. <addr> is represented most significant byte first.

The S0 record contains the name of the input file, formatted as data bytes. If input was from xeq, XEQ is used as the name. S1 and S2 records represent text or data segment bytes to be loaded. They normally contain 32 image bytes, output in increasing order of address; the last record of each segment may be shorter. The text segment is output first, followed by the data segment. S9 records contain only a two-byte start address in their <addr> field; S8 records contain a three-byte address. The start address of an object file is the start of the text section.

<checksum> is a single byte value such that the numerical sum of all the bytes represented on the line (except the S field), taken modulo 256, is 255 (0xFF).

RETURNS

Hex returns success if no error messages are printed, that is, if all records make sense and all reads and writes succeed; otherwise it reports failure.

EXAMPLE

The file hello.c, consisting of :

```
char *p {"hello world"};
```

when compiled produces the following Intel hex file:

```
% hex hello.o
$
:0E000002006856C6C6F20776F726C640094
:000000010
```

SEE ALSO

link (1), obj (1), cc (1), as (1), cp1 (1), cp2 (1), cpp (1)

NOTE: This program and documentation are products of Whitesmiths, Ltd.,

NAME

icheck - file system consistency check

SYNTAX

icheck [-s] [-b numbers] filesystem

c

DESCRIPTION

Icheck assumes "filesystem" is the name of a special file containing a Micronix file system.

Icheck checks the inodes (block pointers) and the free list to see that no block is referenced more than once.

It also prints statistics about the file system.

-s causes the free list to be reconstructed from scratch.

-b followed by a list of block numbers will cause the requested block numbers to be called out when they are encountered.

Icheck can take several minutes on a large file system.

EXAMPLE

To check the root file system:

icheck /dev/root

NOTES

Be sure that the file system in question is not active.

SEE ALSO

ncheck (1), dcheck (1)

NAME

include - file inclusion

SYNTAX

include

DESCRIPTION

The standard input is copied to the standard output with the exception of each line beginning with:

include filename

which is replaced by the contents of the named file. Includes may be nested; that is, an included file may invoke another file for inclusion. If a named file is not openable, an error message is printed and the program terminates abruptly.

EXAMPLE

As a compiler preprocessor:

include <program.z >program.tml

SEE ALSO

macro (1)
Brian Kernighan and P. J. Plauger
Software Tools
Addison West Publishing Company, 1976.

NAME

init

SYNTAX

/etc/init

DESCRIPTION

Init creates an automatic process for each terminal on a which a user may log in.

Init first brings the system up in single-user mode, with only the console active.

When the single user shell exits, **init** runs the shell script "/etc/rc" if it exists, then reads the file "/etc/ttys", and branches several times to create a process for each terminal specified in the file.

When one of init's child processes terminates, a new process is generated to take its place. Normally, this takes place automatically.

If **init** receives signal number 1, it re-reads the ttys file and resets any tty modes that are specified in that file. The update process checks the ttys file once every 30 seconds, and signals **init** if it has changed. You can also force a re-configuration by typing the command

```
# kill -1 1
```

which sends signal number 1 to process 1.

SEE ALSO

update (1), signal (2), ttys (5)

NAME

kwic - prepares key word in context index

SYNTAX

kwic

DESCRIPTION

Kwic is a special purpose program useful in the preparation of context indexes. It produces one output line for each word of input suitably rotated so that the word in question appears at the beginning of the line. End of line is marked by addition of a dollar sign ($).

EXAMPLE

In standard usage:

```
kwic <text | sort | unrot >index
```

SEE ALSO

unrot (1), sort(1) Brian Kernighan and P. J. Plauger Software Tools Addison West Publishing Company, 1976.

NAME

last - login accounting

SYNTAX

last [-s] [user ...] [tty ...]

DESCRIPTION

Last interprets the user login accounting file (/usr/adm/wtmp). It can be used to gain a perspective of the login history of the system.

Last displays the starting and ending times of each user session as well as the duration of that session. The display is given in reverse chronological order, most recent first.

If the -s option is specified, only a summary is given. The summary shows total login time for users or ttys without breaking it down into single sessions.

If a user name or list of user names is given, then the display is restricted to only those users specified.

If a tty name or list of tty names is given, then only login sessions on the specified ttys will be displayed.

If a mixed list is given then the display will include sessions matching any of the criteria.

It's possible to enable or disable the historical record keeping feature. Record keeping is turned on or off based on the presence or absence of the wtmp file (/usr/adm/wtmp). See the examples below.

If a login session is abnormally terminated (say by a power failure), last assumes that the session continued until the following boot and reports it as ending at "boot" instead of at a specific time.

EXAMPLES

Print login history:

```
last
```

Print usage summary:

```
last -s
```

Find out who has been using your tty:

```
last ttyB
```

Find out how much time oscar has logged:

```
last -s oscar
```

Probe the behaviour of a few selected users:

```
last bill jean harold
```

Start record keeping afresh, discarding old records:

```
type /dev/null > /usr/adm/wtmp
```

Disable record keeping and discard records:

```
era /usr/adm/wtmp
```

Save old records and then start over:

```
cp /usr/adm/wtmp /usr/adm/old
type /dev/null > /usr/adm/wtmp
```

FILES

/usr/adm/wtmp - the file containing the historical records.

NAME

lib - maintains Whitesmith's format libraries

SYNTAX

lib lfile [flags] [files]

DESCRIPTION

Lib performs all functions necessary to build and maintain object module libraries. It can also be used to collect arbitrary files into one bucket. **Lfile** is the name of an existing library file or, in the case of replace (-r), the name of the library to be created.

The flags are:

-d Delete the zero or more named files from the library.

-r Replace the zero or more named files in an existing library or, if no library **lfile** exists, create a library with the named files in the specified order.

Named files that are not present in the library are appended to the end in the order specified.

-t Lists files in the library in the order they occur. If any files are named, only those named and present are listed.

-x Extract any named files that are present in the library into files of the same name. If no files are named, all files are extracted.

-v Be verbose about it. Files retained unmodified are each listed with a preceding "c". Those deleted are preceded by a "d", those replaced are preceded by an "r" and those appended to the end are preceded by an "a". For the -t option, each file is listed with its length in bytes.

-6 Create a file with UNIX v. 6 format, if none exists. Effective only with -r.

-7 Create a file with UNIX v. 7 format, if none exists. Effective only with -r.

At most, one of the flags [drtx] may be present. If none are present, -t is assumed. Similarly, at most one of the flags [67] should be present. If none are present, Whitesmith's library file format is assumed when a new library is created by -r.

The Whitesmith's library format consists of a two-byte header having the value 0177565, written less significant byte first, followed by zero or more entries.

Each entry consists of a fourteen-byte name, followed by a two-byte unsigned file length, also stored less significant byte first, followed by the file contents proper. If a name begins with a null byte, it is taken as the end of the library file.

Note that this differs in several small ways from UNIX v. 6 format, which has a header of 0177555, an eight-byte name, six bytes of miscellaneous UNIX specific file attributes, and a two-byte file length. Moreover, a file whose length is odd is followed by a null padding byte in the UNIX format, but no padding is used in standard library format.

UNIX v.7 format is characterized by a header of 0177545, a fourteen-byte name, six bytes of UNIX specific file attributes, and a four-byte length. Odd length files are also padded to even.

RETURNS

Lib returns success if no problems are encountered, else failure. In most failures, an error message is printed to STDERR and the library file is not modified. Output from the -t flag, and verbose remarks, are written to STDERR.

EXAMPLE

To build a library and check its contents:

```
lib clib -r one.o two.o three.o

lib clb -tv
```

NOTES

If all files are deleted from a library, a vestigial file remains. Modifying UNIX v.6 or UNIX v.7 format files causes all attributes of all file entries to be zeroed.

It does not check for large files (over 65534 bytes).

NOTE: This program and documentation are products of Whitesmiths, Ltd., and are sold separately with the C compiler.

NAME

lines - count lines in standard input or files

SYNTAX

lines [file ...]

DESCRIPTION

The number of lines in each of the named files is printed along with the file name. If it would provide additional information, a total is printed at the end. If no file names are specified, or if "-" is given as a file name, the standard input is read up to and end-of-file and the number of lines is printed.

EXAMPLE

To count the lines in a number of files:

lines file1 file2 file3 file4

SEE ALSO

chars (1), words (1)

NAME

link - combine object files

SYNTAX

link -[flags] <files>

DESCRIPTION

Link combines relocatable object files in standard format for any target machine, selectively loading from libraries of such files made with **lib**, to create an executable image for stand-alone execution, or for input to other binary reformatters.

The flags are:

-a Make all relocation items and all symbols absolute. Used to prerelocate code that will be linked elsewhere, and is not to be relocated.

-c Suppress code output (.text and .data), and make all symbols absolute. Used to make a module that only defines symbol values for specifying addresses in shared libraries, etc.

-db## Set data bias to the long integer ##. Default is end of text section, rounded up to required storage boundary for the target machine.

-dr# Round data bias up to ensure that there are at least # low-order binary zeros in its value. Ignored if -db## is specified.

-d Do not define bss symbols, and do not complain about undefined symbols. Used for partial links, i.e., if the output module is to be input to **link**.

-eb* If the symbol * is referenced, make it equal to the first unused location past the bss area.

ed* If the symbol * is referenced, make it equal to the first unused location past the initialized data area.

et* If the symbol * is referenced, make it equal to the first unused location past the text area.

-h Suppress headers in output file. This should be specified only for stand-alone modules such as bootstraps.

-l* Append library name to the end of the list of files to be linked, where the library name is formed by appending * to "/lib/lib". Up to ten such names may be specified as flags; they are appended in the order specified.

-o* Write output module to file *. Default is xeq.

-r Suppress relocation bits. This should not be specified if the output module is to be input to **link**.

-tb## Set text bias to the long integer ##. Default is location zero.

-t Suppress symbol table. This should not be specified if the output module is to be input to **link**.

-u* Enter the symbol * into the symbol table as an undefined public reference, usually to force loading of selected modules from a library.

-x* Specify placement in the output module of the input .text and 1-weighted bit, .data. If either bit is set, the corresponding sections are put in the text segment output; otherwise they go in the data segment. The default value used, predictably, is 2.

The bss section is always assumed to follow the data section in both the input and output files. It is perfectly permissible for text and data sections to overlap, as far as **link** is concerned; the target machine may or may not make sense of this situation (as with separate instruction and data spaces).

The specified <files> are linked in order; if a file is in library format, it is searched once from start to end. Only those library modules are included which define public symbols for which there are currently outstanding unsatisfied references. Hence, libraries must be carefully ordered, or rescanned, to ensure that all references are resolved. By special dispensation, flags of the form "-l*" may be interspersed among <files>. These call for the corresponding libraries to be searched at the points specified in the list of files. No space may occur after the "-l", in this usage.

File Format

A relocatable object image consists of a header followed by a text segment, a data segment, the symbol table, and relocation information.

The header consists of an identification byte 0x99, a configuration byte, a short unsigned int containing the number of symbol table bytes, and six unsigned ints giving: the number of bytes of object code defined by the text segment, the number of bytes needed by the bss segment, the number of bytes needed for stack plus heap, the text segment offset. Byte order and size of all ints in the header are determined by the configuration byte.

The configuration byte contains all information needed to fully represent the header and remaining information in the file. Its value val defines the following fields: ((val & 07) << 1) + 1 is the number of characters in the symbol table name field, so that values [0, 8) provide for odd lengths in the range [1, 15]. If (val & 010) then ints are four bytes; otherwise they are two bytes. If (val * 020) then ints are represented least significant byte first, otherwise most significant byte first; order is assumed purely ascending or purely descending. (val & 0140) >> 5 is the strongest for bounds that are multiples of 0, 2, 4, 8 bytes. If (val & 0200) no relocation information is present in this file.

Relocation information consists of two successive byte streams, one for the text segment and one for the data segment, each terminated by a zero control byte. Control bytes in the range [1, 31] cause that many bytes in the corresponding segment to be skipped; bytes in the range [32, 63] skip 32 bytes, plus 256 times the control byte minus 32, plus the number of bytes specified by the relocation byte following.

All other control bytes control relocation of the next short or long int in the corresponding segment. If the 1-weighted bit is set in such a control byte, then a change in load bias must be subtracted from the int. The 2-weighted bit is set if a long int is being relocated instead of a short int. The value of the control byte right-shifted two places, minus 16, constitutes a "symbol code".

A symbol code of 47 is replaced by a code obtained from the byte or bytes following in the relocation stream. If the next byte is less than 128, the symbol code is its value plus 47. Otherwise, the code is that byte minus 128, times 256, plus 175 plus the value of the next relocation byte after that one.

A symbol code of zero calls for no further relocation; 1 means that a change in the text bias must be added; 3 means that a change in bss bias must be added. Other symbol codes call for the value symbol table entry indexed by the symbol code minus 4 to be added to the item.

Each symbol table entry consists of a value int, a flag byte, and a name padded with trailing NULLs. Meaningful flag values are o for undefined, 4 for defined absolute, 5 for defined text relative, 6 for defined data relative, and 7 for defined bss relative. To this is added 010 if the symbol is to be globally known. If a symbol still undefined after linking has a non-zero value, **link** assigns the symbol a unique area, in the bss segment, whose length is specified by the value, and considers the symbol defined. This occurs only if -d has not been given.

RETURNS

Link returns success if no error messages are printed to STDOUT, that is, if no undefined symbols remain and if all reads and writes succeed; otherwise, it returns failure.

EXAMPLE

To load the C program echo.o with separate I/D spaces for Micronix (UNIX/V6):

```
% link -lc.11 -rt -db0 /lib/Crts.o echo.o; taout
```

or with read-only text section:

```
% link -lc.11 -rt -dr13 /lib/Crts.o echo.o; taout
```

And to load the 8080 version of echo under CP/M:

A:link -hrt -tb0x0100 a:chdr.o echo.o a:clib.a a:mlib.a

SEE ALSO

hex (1), lib (1), lord (1), obj (1), cc (1), as (1), cp1 (1), cp2 (1), cpp (1)

NOTE: This program and documentation are products of Whitesmiths, Ltd.,

NAME

ln - make links

SYNTAX

ln name1 name2

DESCRIPTION

A link is made between two files (name1 and name2). Therefore, changes made to file name1 affect file name2 and vice-versa. A file may have up to 255 links.

ERRORS

Linked files must be on the same file system, otherwise, a "Cross-device link" error message is displayed.

SEE ALSO

rm (1), cp (1), mv (1)

NAME

login - sign on

SYNTAX

login

DESCRIPTION

Login is executed automatically by **init**. It requests a user name (and password if appropriate), performs the required bookkeeping, executes the shell and begins the user's Micronix session.

FILES

/etc/utmp	accounting
/usr/adm/wtmp	accounting
/etc/banner	The sign on message
/etc/motd	message-of-the-day
/etc/passwd	password file

SEE ALSO

init (1)

NAME

lord - order libraries

SYNTAX

lord -[c* d* i r* s]

DESCRIPTION

Lord reads in a list of module names, with associated interdepencies, from STDIN, and outputs to STDOUT a topologically sorted list of module names such that, if at all possible, no module depends on an earlier module in the list. Each module is introduced by a line containing its name followed by a colon. Subsequent lines are interpreted as either:

defs - things defined by the module,

refs - things referred to by the module, or

other stuff.

Refs and defs have the syntax given by one or more formats entered as flags on the command line. Each character of the format must match the corresponding character at the beginning of an input line; a ? will match any character except newline. If all characters of the format match, the rest of the input line is taken as a ref or def name. Thus, the format flag "-d0x????D" would identify as valid def any line beginning with "0x", four arbitrary characters and a "D", so that the input line "0x3ff0D_inbuf" would be taken as a def named "_inbuf".

The flags are:

-c* prepend the string * to the output stream. Implies -s. Each module name is output preceded by a space; the output stream is terminated with a newline. Hence, **lord** can be used to build a command line.

-d* use the string * as a fromat for defs.

-i ignore other stuff. Default is to complain about any line not recognizable as a def or ref.

-r* use the string * as a format for refs.

-s suppress output of defs and refs; output only module names in order.

Up to ten formats may be input for defs, and up to ten for refs.

If no -d flags are given, **lord** uses the default def formats: "0x????????B", "0x????????D", "0x????????T", 0x????B", "0x????D", "0x????????U" and "0x????U". These are compatible with the default output of rel (obj).

If there are any circular dependencies among the modules, **lord** writes "not completely sorted" to STDERR and outputs a partially-ordered list. In general, rearrangements are made only when necessary, so an ordered set of modules should pass through **lord** unchanged.

RETURNS

Lord returns success if no error messages are printed, otherwise failure.

EXAMPLE

To create a library of ordered object modules under Idris:

```
% rel *.0 | lord -c"lib libx.a -c" | sh
```

To order a set of objects using Micronix nm:

```
% nm *.o > nmlist
% lord < nmlist -c"ar r libx.a" | \
-d"??????T" -d"??????D" -d"??????B" -r"??????U" | sh
```

SEE ALSO

lib (1), obj (1)

NOTE: This program and documentation are products of Whitesmiths, Ltd.

NAME

lpr - line printer spooler

SYNTAX

lpr [-igpqrs] file ...

SUMMARY

(A less technical description follows.)

Lpr causes each of the named files to be queued for printing. If no file names are given, the standard input (generally your terminal) is used.

Options:

-i Print the file "in place" without copying.

-p Pause before printing, send a message to the terminal, and delay until the command "lpr -g" is entered.

-g Go. Starts the printer daemon up again after a pause.

-q Display the printer queue. The owner and size of each queued file is displayed on the standard output (usually your screen).

-r Remove all files in the queue owned by you.

-s Stop the daemon. The queued files are not removed.

Alternate printers are accessed by names such as "dpr", "xpr", etc. See below for details.

DESCRIPTION

In a single user system, the basic idea of printing is very simple: you have only one printer and a machine which is able to run only one person's programs at a time.

In a multi-user system, things may not be so simple: there may be more than one printer, and two users may wish to use the same printer at the same time. The lpr program addresses these difficulties.

Here's how it works in principle. (This indented section is optional material, not needed by the ordinary user.)

Say that several users all wish to print files. All of them type "lpr filename" at once. Each one uses the same or a different file name as is his or her preference.

Each of the several lpr programs creates a uniquely named

file in the "spooling directory" and copies the files which the users wish to print into these files.

Each of the several lpr programs attempts to initiate a printer "daemon". Because of locking mechanisms, only one succeeds. (A daemon is a process, just like the ones run from terminals, except that it exists independently of any terminal. It performs its tasks (see below) without human intervention.

All of the lpr programs finish loading their files into the spooling directory. Meanwhile, the daemon is still alive and running.

It scans the spooling directory and observes that it has several files waiting in line to be printed.

It determines which one is next in line and starts printing that one.

Now on to the practical description.
Lpr options:

-i Print "in place". Normally files to be printed are copied into the spooler's directory. With the the -i option specified, only the name of the file will be remembered and it will be read and sent to the printer at print time. It is therefore important that the file not be removed before the print finishes. Using lpr this way saves time if you have an enormous file to print, but does not work if the material to be printed is not a file (see the example with "print" below).

-p Pause before printing. When it comes your file's turn at the printer, the daemon will send a message to your terminal and delay until the command "lpr -g" is entered. This is intended to allow the insertion of special forms or the like.

-g Go. Starts the printer daemon up again after a pause.

-q Display the printer queue. The owner and size of each queued file displayed on the standard output (usually your terminal).

-r Remove all your files in the queue. If the file being removed is also being printed, the printing will be prematurely curtailed.

-s Stop the daemon. A fatal signal is sent to the daemon. The queued files are not removed, so they will be started over again next time an lpr command is executed. If you want to remove the files, use lpr -r.

EXAMPLES

The simplest way to print a file is:

```
lpr filename
```

This does not perform such services as skipping over creases in the paper, numbering the pages, etc. (Often, you do not want these services, since they were performed by another program such as "form" or Wordstar.) If you do want these services, type

```
print file | lpr
```

This sends the output of the print program (which names, dates, ans paginates) to the spooler.

If you are using the "form" program (see **form** in this manual), you can format and print at the same time by typing

```
form file | lpr
```

(If you simply type "form file", the output goes to your screen.)

If you want to print directly from your screen, without going through the formalities of creating a named file and running it through lpr simply type

```
lpr (ret)
```

Then type your text. When you are finished, type

```
(ret)
(control-D)
```

Be aware that when you do this you will not have special effects (like bold print) and you will not have a disk file of the text.

To do a "form feed", type:

```
lpr (ret)
(ret)
(control-D)
```

This causes a file to be queued which contains a single line-feed character. Lpr generates a printer form-feed after each file it prints.

ALTERNATE PRINTERS

The name by which the spooler is invoked is significant. When invoked as "lpr", it uses the directory

/usr/spool/lpr

for spooling, and it decides which device to write to by looking in the configuration file "/etc/ttys" for a line containing the word "lpr". For instance, if the following line occurs in /etc/ttys

```
ttyC  lpr  1200
```

then the lpr program will write to port ttyC at 1200 baud. (See ports (4) if you don't know what ttyC means.)

Suppose that you want to support a second printer. Say that it is connected to ttyD, runs at 9600 baud, and requires hardware handshaking. Think of a name for it such as "dpr" and put the following line in the ttys file:

```
ttyD  dpr  9600   shake
```

(The order of the words is not significant.) Finally, make a link to lpr named "dpr" by typing

```
cd /bin
ln lpr dpr
```

Now when you type the command "dpr file", the spooler will write to ttyD at 9600 baud, and it will use

/usr/spool/dpr

for spooling. (It will create this directory automatically, so that you don't have to do it in advance.)

SEE ALSO

print (1), form (1), cables (4), ports (4), printers (4), ttys (5)

FILES

/usr/spool/lpr - spooling directory
/usr/spool/lpr/pid - contains the daemon's process ID
/etc/ttys - To find out where to send the output

NAME

ls

SYNTAX

ls [-adfgilrstu] name ...

DESCRIPTION

Ls lists all the named non-directories first, then the contents of each named directory. If no names are specified, the current directory is listed.

There are three basic printing formats: line, multi-column, and long.

Line format is one file to a line.

In multi-column format, ls produces a rectangular array of file names sorted down the page and compacted as much as possible within an 80 column width.

In long format, files are listed one to a line with a lot of information about each file packed into the line (see -l below).

Long format is explicitly forced by the -l or -g option (see below). Otherwise, ls chooses line or multi-cloumn format automatically, depending on whether its standard output is a file or a terminal.

Options:

-a All. Normally, files names beginning with '.' are not listed. If -a is specified, they are listed.

-d Directory. Normally, if a directory name is given, its contents is listed. If -d is specified, then only the name is printed (along with any other attributes called for by other options).

-f File order. Disable all sorting and print the files in the order they actually appear in the directory.

-g Group. For each file, print the name of the owner's group, instead of the owner's login name. The latter is the default for long listings. (This option forces a long listing, whether or not -l has been specified.)

-i Print the inode number of each file.

-l **Long format.** Print mode, link count, owner, size, time of last modification, and name.

-r **Reverse.** Files are sorted in reverse order.

-s **Sizes are given in blocks** (of 512 bytes each) instead of characters.

-t **Time sort.** Sort files by modification time (most recent first) instead of alphabetically.

-u **Use the access time instead of the** modification time for -t and -l.

Long printing format is laid out as follows:

Each line refers to one file. The fields printed are:

mode links owner size time name

Mode is the access permissions of the file symbolically displayed.

Links is the number of links to the file.

Owner is the login name of the user who owns the file.

Size is the size of the file in bytes (or blocks with the -s option)

Time is the last modification time (or last access time) of the file.

The mode takes the form:

-[bcd]rwxrwxrwx

The - stands for plain file or permission not granted; the

b stands for block special file
c stands for character special file
d stands for directory

Files may also have the following characters interspersed within these descriptions:

l stands for write locked file
s stands for set user or group ID on execution
r stands for read permission
w stands for write permission
x stands for execute permission

If a letter is missing in the above representation, it indicates that permission for that mode of access is not permitted.

EXAMPLES

drwxr-xr-x

The d means this is a directory.

The rwx means the owner of the directory has all permissions.

The r-x means those users in the directory's group have only read and search permission on the directory. Likewise for others (those users who are neither the owner of the directory nor in the same group).

-rl-r--r--

- means this is a plain file.

rl- means the file is readable by the owner and write locked.

r-- means the file is readable by those in the group.

the second r-- means the file is readable by anyone else on the system also.

FILES

/etc/passwd - to get names for user ID numbers
/etc/group - to get names for group ID numbers

SEE ALSO

chmod (1) - to change the mode of a file

owner (1) - to change the owner of a file

group (1) - to change the group of a file

passwd (5) - the structure of the password file

group (5) - the structure of the group file

NAME

macro - macro expansion

SYNTAX

macro

DESCRIPTION

The standard input is copied to the standard output, with macros expanded along the way. After encountering a line of the form:

```
define(token,string)
```

the token will be replaced by the string everywhere it appears. A token is a contiguous string of alphanumerics delimited by non-alphanumerics. Macro exansions are applied recursively [i.e. define(a,b) define(b,c) is equivalent to define(a,c)]. Macros may be defined with arguments. If the sequence $n where n is a digit occurs in the defining string, $n will be replaced by the nth argument. $1 is the first argument. In an instance where a macro has fewer than n arguments, $n is replaced by the null string, (i.e., simply ignored). An example:

```
define(add, $1 + $2 + $3)
add(firstarg, secondarg, thirdarg)
```

would produce:

```
firstarg + secondarg + thirdarg
```

EXAMPLE

As a language preprocessor stage:

```
macro <program >tml
```

SEE ALSO

include (1), ratfor (1)

NAME

mail - send and receive mail

SYNTAX

mail [-[yn]] [user]

DESCRIPTION

Micronix has an inter-user mail system. The mail program sends letters to the appropriate system user. Letters are sent to another user by entering:

```
% mail user-name
```

The user-name may be a "local" name or a "network" name. Local names are simply login names (ie, the name by which a user logs in). Network names look like machine-name%login-name.

To enter the letter, just begin typing. Enter as many lines as necessary. Letters are terminated by entering a single period, or a ^D at the beginning of the last line.

The next time the user logs in, he will be greeted with the message "You have mail".

To read his mail, he simply enters

```
% mail
```

The letter is printed on-screen. The user is then asked if he wants to keep his mail.

The appropriate reply is "yes" to keep it, or "no" to delete it. Kept mail is stored in the "mbox" file in the user's home directory.

EXAMPLE

```
% mail sales%ron
Lunch Wednesday at 12:30?
        -Jill
.
%
```

SEE ALSO

network (4)

NAME

make - automated program construction

SYNTAX

make object ...

DESCRIPTION

The problem of keeping the assembly instructions straight for large programs composed of many modules is reduced to the one-time job of writing a complete description file.

The make program reads the file makefile in the current directory. Makefile consists of a number of lines describing interdependencies among program modules and the formulas necessary to contruct these modules.

Each named target module is then contructed.

Make then "walks" the tree of dependencies, updating target files as necessary.

Lines in the makefile take the form:

```
product : ingredients ; formula
```

Where "product" is the name of a target file, "ingredients" is a space-separated list of files upon which "product" depends.

NAME

man - manual

SYNTAX

man [N] program name ...

DESCRIPTION

Man prints documentation (if it exists) for each named program. Complete on-line documentation is resident on the Micronix system.

The "N" option directs man to a numbered or particular section of the files.

Typing

```
ls /bin
```

displays the system programs. Typing

```
ls /usr/man
```

displays the sections that contain the documentation for the Reference Manual.

EXAMPLE

The command line

```
man man
```

prints this page.

Entering

```
man 2 stty
```

prints the stty.2 file instead of stty.1.

NAME

mkdir - make directory

SYNTAX

mkdir newdir ...

DESCRIPTION

Mkdir creates the specified directory or directories. Entries for "." and ".." are created automatically.

ERRORS

The given directory names must not yet exist. **Mkdir** requires write permission on the parent directory.

If a multiple element path name is given, **mkdir** creates each of the directories in the path leading to the desired directory.

It is not possible to remove a directory from the system that contains files; files must first be removed using the rm command.

EXAMPLES

mkdir /a/sally mkdir /usr/spool/mail /usr/spool/lpr

Each directory created is accessible to any user. It is advisable to organize your work by keeping related files together in their own, separate directories (i.e., mkdir programs, mv *.c programs, etc.).

SEE ALSO

rm (1)

NAME

mkfs - make an empty file system

SYNTAX

mkfs device [size]
mkfs device [-exclude]

DESCRIPTION

Mkfs builds an empty file system on the named device. This command is normally used to initialize blank or outdated diskettes for use under Micronix. It also initializes hard disks when they are brand new or if for some other reason you wish to rebuild them from scratch.

If no size is specified, mount determines the device size automatically. A numerical argument without a leading "-" is taken as the desired size (in 512-byte blocks). A numerical argument with a leading "-" is taken as a number of blocks to be excluded from the file system at the end of the device (say for swap space). In this case, the actual device size is also determined automatically.

The ilist size is set to

$$s/43 + s/1000,$$

where s is the file system size determined above. The smallest practical file system is about 50 blocks.

Warning! Extreme care should be taken when using this command: It obliterates any former file system on the device. If, for example, you entered

mkfs /dev/root

you would have to rebuild your entire file system scratch. Needless to say, aim carefully!

ERRORS

"filename: Not a block special file" This message indicates that you have specified a /dev file that does not transfer data in blocks. In practice, any disk drive will be block-special; other peripherals (printers, terminals, tape drives) are typically character-special.

EXAMPLES

```
mkfs /dev/dja
mkfs /dev/hdb -2048
mkfs /dev/hda 18448
```

NAME

mknod - create special file

SYNTAX

mknod name type major minor

DESCRIPTION

Type is either b or c for block special or character special. Major and minor are the major and minor device numbers, respectively. A special file is created (mode 777) with the given name. This call is restricted to the super-user. Conventional use requires that all special devices be kept in the one directory, "/dev".

NOTES

Warning: it is dangerous to have to separate nodes with the same type and major/minor number. If you want multiple names, use ln.

SEE ALSO

ln (1), rm (1), mkdir (1), mode (1)

NAME

chmod - changes mode of files

SYNTAX

chmod mode file ...

DESCRIPTION

The mode of each named file is changed according to mode, which is an octal number constructed from the OR of the following modes:

4000	set user ID on execution
2000	set group ID on execution
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0070	read, write, execute by group
0007	read, write, execute by others

EXAMPLE

Mode 777 file (accessible by all)

Mode 700 file (inaccessible)

Mode 400 file (read only by owner)

SEE ALSO

ls (1), chown (1), chmod(1)

NAME

more - file perusal filter for fast terminals

SYNTAX

more [file ...]

DESCRIPTION

More allows you to scroll through the named files on a video terminal, one screenfull at a time. At the bottom of the screen, it prints "--More--" and waits. Press carriage return to continue, delete to quit. If no file names are given, the standard input is used.

EXAMPLES

To view a couple of files one screenfull at a time:

```
more file1 file2
```

To view some documentation without having to use START/STOP control keys:

```
man upm | more
```

NAME

mount - mount a file system

SYNTAX

mount [device] [directory] [-r]

DESCRIPTION

Mount is used to extend the Micronix file system to include additional storage devices. If you think of the system of files as a large tree, mounting is akin to hanging an additional ornament off a particular bough.

"Device" refers to the device name as it appears in the /dev directory. The device name can be entered as (for example) "/dev/fla", or you can abbreviate it to simply "fla". "Directory" is the position in the existing file system that you want the additional file system to begin. The optional -r flag means to mount the device as read-only.

Mount with no argument displays the current list of mounted file systems. It's advisable to just type "mount" to take a look before proceeding. That way you'll have a better idea of what's going on.

When you mount a device, you must specify a certain directory that already exists. Use **mkdir** to establish a new directory, if necessary.

Typically, you would enter something like

mount fla /b

This will graft the file system of floppy drive A into the root directory as directory "b". From then on Micronix can access any of the files on the floppy through the path /b/...

It's somewhat traditional to mount devices on directories in the root directory. "/f" or "/b" or "/c" or "/h", for example. This is not compulsory. It is done perhaps because it simplifies the concept of the file system.

If you mount a device on a directory that already has entries in it, it will appear that the existing entries are thrown away. Never fear; they are still there, and will reappear when the device is unmounted (**umount**). This situation would occur in the example above if directory /b already had files in it from some other device.

When the system is initially brought up, no devices are mounted (except the root device, which technically isn't mounted, since it is the core to which everything else is ultimately attached. Leave this one to to the semanticians.) However, you can stick

mount commands into the rc file, so that whatever devices you want are automatically mounted whenever you go from single- to multi-user mode.

A device may be mounted at most once. If you mount and mount again with the same device, the second instance will fail, yielding the message "File or device busy".

Mount refuses to mount a device which does not appear to contain a file system. A quick check is made to see that the device in question is reasonable. If you thought the device did contain a file system, but mount disagreed, you may run "fsck", which sometimes has the power to turn a non-file-system into a file-system and in any event will give it a good try. With new diskettes or a hard disk file system that is corrupted beyond repair, use mkfs to create a fresh, EMPTY file system.

A file system may be mounted "read/write" or "read only". The system is guaranteed to change the contents of file systems mounted "read/write". File systems mounted "read only" are guaranteed to be unchanged.

You may request read only status with the "-r" flag. Sometimes read only status is foisted upon you, for instance, if you mount a write-protected floppy diskette.

EXAMPLES

Mounting a device and addressing a file on it:

Suppose you have a floppy diskette with a directory "letters" containing a file "memo5". The floppy drive is currently unmounted and you want to edit memo5.

To mount the device (assume it is the first 8" floppy, named fla, and you already have a directory named f in the root directory):

mount fla /f

To access the file for Micronix editing:

edit /f/letters/memo5

Observe that this path name crosses a device boundary, from the root device (probably an integrated hard disk) to the 8" floppy. It's entirely possible to mount a second floppy drive to the first floppy's file system, say, alongside memo5 in this example. You could mount it as "g", for instance (**mount flb /f/letters/g**) and then address it as /f/letters/g/... This path thus crosses two device boundaries. This can go on as long as you have devices and controllers, but why? It's better to just mount them all to the root.

To prepare a mini-floppy for use under Micronix:

formatdj -i5 -t40 -s2 -b512 -d0

(5 inch, 40 track, 2 sides, 512 bytes, drive 0)

mkfs /dev/mfa

(make file system on mini-floppy A)

mount mfa /f

(mount the new file system)

mkdir /f/newdir

(make a new directory on the diskette)

cp * /f/newdir

(copy some files over)

umount mfa

(dismount the mini-floppy)

NOTES

Regarding 5.25" floppy disk drives, note that there are two ways of addressing these devices in mount commands. Since they are configured for double-sided use, you would normally refer to them as "mfa2", "mfb2", and so on. The exception to this rule is the case where a diskette was formatted for single-sided use. When using such a diskette, refer to the device without the "2", for example, as simply mfa.

WHEN TO UNMOUNT? you may ask. When using either size of floppy drive, **always unmount (umount command) before removing a floppy diskette from the drive.** Therefore, if you need to switch diskettes, first unmount the device, then change diskettes, and finally remount.

In the unusual case of changing hard disk drives (that is, actually unplugging one and plugging in another), unmount the first before disconnecting it.

To those of you who recoil at the use of such a verbal construction as "unmount", we apologize. But if we asked you to dismount a device, you'd just have silly images of spurs jangling while you drawl "Whoa, there, old floppy."

SEE ALSO

umount (1) - to reverse the effects of **mount**.
fsck (1) - to repair file systems.
mkdir (1) - to create a new directory.
mkfs (1) - to create a file system.
rc (5) - to automate selected mounts.

NAME

msgs - bulletin board and junk mail service

SYNTAX

msgs -fhq [[-] number]

DESCRIPTION

The msgs program provides a system wide bulletin board. Any user may post messages on the board and all users may view the messages on the board. Using the msgs program is much like reading the classified section of a newspaper.

The last message read by each user is automatically remembered. That is to say each user sees each message once and the system holds your place.

When the command "msgs" is given an interactive mode is entered.

You are shown the heading of the first message you haven't read yet.

You are advised of the number of lines remaining in the file.

You are asked if you'd like to see more.

This looks like:

```
Message 23:
From harold Mon 23 Aug 11:30:22
New database program installed!!
(17 more lines) More? (Y/N/Q)
```

Possible responses:

y Yes, I'd like to see the rest of this message.

n No, I'm not interested in this message, skip me to the next message.

q Quit. Get out of the program right now.

(number)
Jump to the message whose number is given, or the next highest numbered message.

\- Back up one message for a replay.

s filename
Append this message to the end of the named file. Create

the file if it doesn't already exist. If a minus (-) follows the 's', the previous message will be saved.

If you press the <RUB-OUT> or <DELETE> key in the middle of the printing of a long message, the remaining portion of the message will be skipped.

Command line flags:

-f Causes msgs not to say "No new messages".

-h Headings only. The heading of a message is arbitrarily the first two non blank lines.

-q Queries whether there are messages. Prints "There are new messages" if there are. This option is useful in startup files.

To post a message, just mail it to "msgs". Messages are automatically removed after 2 weeks, unless they are set to file mode 444, in which case they will be permanent. See chmod (1).

EXAMPLES

To catch up on local events:

```
msgs
```

To post a message:

```
mail msgs

There wil be a company party at 4 p.m. on April 6th.
All are invited to attend.
Guest speakers will include Horton Hornblower.

.
```

To post a message from a file:

```
mail msgs < file &
```

To view the complete list of message headings:

```
msgs -h 1
```

To notify you if there are new messages, add this line to your

".login" file in your home directory:

```
msgs -fq
```

FILES

/usr/spool/msgs ~/.msgs

NAME

mv - move or rename files

SYNTAX

mv file1 file2

mv file ... directory

DESCRIPTION

Mv moves (changes the name of) file1 to file2.

If the last argument is a directory, each of the named files is moved into the directory retaining their original names.

This command is similar to the RENname command in CP/M.

SEE ALSO

NAME

ncheck - generate names from i-numbers

SYNTAX

ncheck [-i numbers] filesystem

DESCRIPTION

Ncheck generates a pathname vs. i-number list for the named file system.

If -i is specified along with a list of numbers, pathnames are reported only for the i-numbers specified.

EXAMPLES

```
ncheck /dev/root
```

to generate a list of all the file names and their i-numbers for the root file system.

```
ncheck -i 26 87 204 /dev/root
```

to look up path names for a few selected i-numbers.

SEE ALSO

fsck (1), icheck (1), dcheck (1)

NAME

obj - examine object files

SYNTAX

obj -[flags] <files>

DESCRIPTION

Obj permits inspection of relocatable object files in standard form for any target machine. Such files may have been output by an assembler, combined by link, or archived by lib. Obj can be used either to check their size and configuration, or to output information from their symbol tables.

The flags are:

-d Output all defined symbols in each file, one per line. Each line contains the value of the symbol, a code indicating to what the value is relative, and the symbol name. values are output as the number of digits needed to represent an integer on the target machine. Relocation codes are: "T" for test relative, "D" for data relative, "B: for bss relative, "A" for absolute, or "?" for anything obj does not recognize. Lowercase letters are used for local symbols, uppercase for globals.

-g Print global symbols only.

-i Print all global symbols with the interval in bytes between successive symbols shown in each value field. Implies the flags -[d u v].

-o Output symbol values in octal. Default is hexadecimal.

-s Display the sizes, in decimal, of the text segment, the data segment, the bss segment, and the space reserved for the runtime stack plus heap, followed by the sum of all the sizes.

-t List type information for this file. For each object file the data file is: the size of an integer on the target machine, the byte order it observes, the most restrictive storage bound it enforces, and the maximum number of characters it permits in an external name. If the file is library, the type of library is output and the information above is output for each module in the library.

-u List all undefined symbols in each file. If -d is also specified, each undefined symbol is listed with the code "U". The value of each symbol, if non-zero, is the space to be reserved for it at load time if it is not explicitly defined.

-v Sort by value; implies the -d flag above. Symbols of equal value are sorted alphabetically.

If no flags are given, the default is -[d u]; that is, all symbols are listed, sorted in alphabetical order on symbol name. If more than one of the flags -[d s t y] is selected, then type information is output first, followed by segment sizes, followed by the symbol list specified with -d or -u.

<files> specifies zero or more files, which must be in relocatable format, or standard library format, or in UNIX/V6 library format, or UNIX/V7 library format. If more than one file, or a library, is specified, the name of ech separate file or module precedes any information output for it, each name followed by a colon and a newline; if -s is given, a line of totals is also output. If no <files> are specified, or if "-" is encountered on the command line, xeq is used.

RETURNS

Obj returns success if no diagnostics are produced, that is, if all reads are successful and all file formats are valid.

EXAMPLE

To obtain a list of all symbols in a module:

```
% obj alloc.o
0x00000074T_alloc
0x00000000U_exit
0x000001feT_free
0x000000beT_nalloc
0x00000000U_sbreak
0x00000000U_write
```

SEE ALSO

lib (1), link (1), lord (1), cc (1), as (1), cp1 (1), cp2 (1), cpp (1)

NOTE: This program and documentation are products of Whitesmiths, Ltd., and are sold separately.

NAME

od, hd - octal or hex dump

SYNTAX

od [-bcdox] file

DESCRIPTION

Od dumps file in one or more formats. If no file name or "-" is given, the standard input is used. The following flags are understood:

-b Interpret bytes in octal.

-c Interpret bytes in ASCII.

-d Interpret words in decimal.

-h Hex words

-o Interpret words in octal (default).

-x Interpret bytes in hexadecimal.

EXAMPLE

od -bo file

NAME

overstrike - converts back spaces to multiple lines

SYNTAX

overstrike

DESCRIPTION

Overstrike copies the standard input to the standard output. When it encounters a back space, it replaces it by a "return" and an appropriate number of spaces, thus eliminating backward motion in the middle of a line.

EXAMPLE

To prepare a file for printing on a device which does not understand back spaces:

```
overstrike <file >output
```

NAME

owner - file ownership

SYNTAX

owner [-user] file ...

DESCRIPTION

Every Micronix file has associated with it a user I.D. number - the "owner" of the file.

The **owner** utility program allows you to examine or change this file owner number. NOTE: Only the super-user may change the owner of a file.

There is usually a name for each user I.D. number. These names are kept in the system-wide file "/etc/passwd."

In order to discover who owns a file, type:

owner filename

The program will look up the owner of the file and attempt to express it as a user name. If this is not possible, it will list the owner as a numerical user I.D.

To change the owner of a file, type

owner -newowner filename

newowner may be a user name or a user I.D. number (names are recommended).

The owner program works equally well with list of files:

owner file1 file2 file3 file4 file5

reports on the ownership of each of the named files.

EXAMPLES

```
owner -root a.out
```

makes file "a.out" belong to user "root".

```
owner a.out
```

reports who owns the file "a.out".

FILES

/etc/passwd - user I.D. <-> name relationships

NAME

passwd - enter (or change) login password

SYNTAX

passwd

DESCRIPTION

Passwd is used to change the login password of the given user.

When a user's account is first created, it has no password, that is, no password will be demanded of the user at login time.

This command may be used to set or change your password. The super user may set or change anyone's password, but, not even the super user may know what a user's password is. It may, however, be changed to something known.

This program is fairly self-explanatory and prompts you for the name of the user whose password you want to change and the password itself. When you type the password, the "echo" is temporarily turned off, so that the password is unseen. Only the encryption of the password is kept around. If two or more users set their password to the same thing, the encryptions will not be the same.

NOTES

It is recommended, if the intention is password security, that passwords be long, complicated, fast to type, and frequently changed.

SEE ALSO

Tutorial in User's Manual

NAME

paste - print files side by side

SYNTAX

paste [-t separator] file1 file2 ...

DESCRIPTION

Paste reads a line from each file in the order given and pastes the lines together into single line, which is written to the standard output. A tab separator is used unless another character is specified. A file name of - refers to the standard input. No attempt is made to line up the columns vertically.

There is no limit on the input line length.

EXAMPLES

Suppose that file1 contains

```
Sally
Samantha
Holly
Agatha
```

and file2 contains

```
Mike
Mordred
Garth
```

paste -t+ file1 file2

would result in the following output

```
Sally+Mike
Samantha+Mordred
Holly+Garth
Agatha
```

ls | paste - - -

would give a two-column listing of the current directory (alphabetized from left to right, not top to bottom).

BUGS AND CAVEATS

No attempt is made to align the columns vertically, except to put tabs between the columns.

SEE ALSO

field (1)

NAME

pilot - an author language

DESCRIPTION

This program gets its name from and is loosely based upon that set of programs written for the CP/M system under the generic name PILOT, itself based upon some complex IBM software.

The present incarnation has adopted many of the ideas, but is not totally faithful to the original PILOT standard.

Pilot is an author language in the sense that it should be possible for "authors," i.e. non-programmers, to successfully and fairly rapidly get something working in the pilot language.

Pilot is a non-structured language, but structured language enthusiasts may be assured that "rp" (a rational pilot) is available as a preprocessor. (See rp (1)).

In order to run something in pilot, one issues the command:

pilot filename

where the given filename refers to a "pilot script". Pilot will simply read the script and do whatever is requested.

To write a "pilot script" one needs only a way to get text into a file. Most text editors will suffice.

Pilot Operation

In general, pilot reads its input file, one line at a time, performing the described actions before proceeding. The general layout of a pilot command is:

***label command conditional : word1 word2 word3 ...**

***label** is a tag by which one may refer to this line elsewhere in the script. It is optional, and in practice, most commands will not have labels. A label may appear on a line by itself.

command is a single letter command, discussed in detail below.

conditional is either the letter Y or N. A conditional specifier is optional. If one appears, the command will only do something if the Y or N matches the current state of pilot's condition flag.

words are optional and specific to individual commands.

Instruction Summary

A - accept
C - call
D - define
E - exit
J - jump
M - match
N - new file
R - return
S - save
T - type
U - unix
W - wait

Instruction Descriptions

A - accept - a: [variable]

One line of input is accepted from the standard input. If a variable name is given, a copy of the accepted data will be stored in the variable.

Examples

```
a:
a: $save
```

C - call - c: *label

Jump to the given label, but remember where you were. If a return instruction (r:) is encountered in the program flow after the label, pilot will jump back to the instruction following the call.

Example

```
c: *output
e:
*output
t: Hello world!
r:
```

1) Call instruction jump down to "*output".

2) Type instruction - type "Hello world!"

3) Return - go back to just after the last call

4) exit - all done!

D - define - d: $move data

The string "data" is evaluated if necessary and then a copy of the resulting string of characters is stored in the string variable "name". Note that "$" denotes string variables.

Example

```
d: $x The cheese
t: $x
```

1) Definition $x is set to "The cheese"

2) Type - "The cheese" will be typed.

E - exit - e:

This is a simple instruction. Pilot quits when it encounters an exit instruction. Anything to the right of the colon is ignored.

Example

```
e:
```

Stop the whole works right now.

J - jump - j: *label

Go to some other part of the current pilot script file and continue executing instructions from that point on. If no label name is given, pilot prints "Missing label" and stops. If there is no such label to be found in the current pilot script file, "name: Label not found" is printed and pilot stops.

Example

```
*top
t:Hello
j:*top
```

This would print "Hello" over and over again forever.

A more useful example:

```
a:
m:yes
jn: *default
t: you typed yes
e:
* default
t: you didn't type yes
e:
```

1) Accept a line from the controlling terminal

2) Match it against "Yes"

3) Jump to *default if the previous match failed.

4) type - "you typed yes" otherwise

M - match - m: pattern,pattern,pattern, ...

Compare the most recent input against the given list of patterns and set pilot's condition flag based on the outcome. The following instructions may base their behavior on the outcome of these pattern comparisons. Two results are possible:

1) The input matched one of the patterns (success).

2) The input matched none of them (failure).

Comparison of each pattern is by a marching window scan.

Case is ignored, that is, a capital Q is considered to be the same as a lower case q.

Example

```
m:a
m:one,two,three
```

The first will succeed whenever the most recent terminal input contained the letter "a" anywhere.

The second will succeed if the most recent input contained any of the three given character sequences which form the words one, two, three.

N - new file - n: filename

Fairly straightforward: Pilot has been marching along executing instructions in its pilot script file. When it encounters the line

n: newfile

pilot will have done with the current file and start executing instructions with the first line in the file named "newfile". (There must, of course exist such a file, and it must be readable.) Pilot stops if it can't read the given file.

R - return - r:

Discussed in C - call (above). A return instruction causes a jump to the line following the most recently executed call instruction. If there are no calls for which there have been no returns, pilot stops. It returns out of pilot to the calling

program, as it were. Anything to the right of the colon is ignored.

Example

```
r:
```

S - save - s: $variable

The current value of the input terminal input register is saved in the string variable whose name is given. If no name is given, no action is taken.

Example

```
s: $phial
```

The most recently accepted input is saved in "phial".

T - type - t: text

A simple command, really. That which follows the colon is simply sent verbatim to the standard output, (usually the terminal). If the last character of the line is a left slant (\), then no "newline" character is sent at the end of the line.

Example

```
t: A) First selection
t: B) Second selection
t:
t: Your choice ==>\
a:
```

The four lines above are typed.

U - unix - u: command

The command to the right of the colon is executed just as if it had been typed as a command to the shell. This command gives you access to the rest of the system. This instruction can take much longer than any of the others just to get set up. If the sub-process created by a U instruction is interrupted or otherwise abnormally terminated, pilot will stop.

Example

```
u: ls
u: cat $file | form | lpr &
u: pilot zzz
```

Each command is executed. Note that it is possible for pilot to continue before the command finishes (ex. 2) or for there to coexist several layers of pilot programs (ex. 3).

W - wait - w: [$variable] [number]

Wait is just like "accept" above only it "times out" after a certain period. The default period is six seconds. One can select a different period by placing a numeric value to the right of the colon as one of the arguments. If the specified time interval expires before the RETURN key has been pressed, then the data returned is the single word "TIMEOUT" which can later be tested.

Example

```
w:
w: $data
w: 9
w: $fast 2
```

1) Accept data for a six second period.

2) Accept data; wait for six seconds; store it in "data"

3) Accept data; wait for nine seconds.

4) Accept data; wait for two seconds; store it in "fast"

SEE ALSO

rp - rational pilot

NAME

print - prints files

SYNTAX

print file1 file2 ...

print [-N] [-lN] [-wN]

DESCRIPTION

Print produces output suitable for redirection to a printer. It also produces a heading at the top of each page, including the file name and the page number of the file.

The following flags may also be used:

-N Causes column N to be printed.

-lN Print N lines per page.

-wN Print page N columns wide.

EXAMPLE

To print a file:

print file

or to print with flag values:

print -4 -l188 -w132 file

SEE ALSO

lpr (1)

NAME

ps - process status

SYNTAX

ps [alx]

DESCRIPTION

Ps prints information about processes currently running on the system.

Any user is allowed to know the command name other users are running. The actual data being processed is not a matter of public record, however.

Ps can be very useful in tracking down errant processes and terminating them.

Options:

a All. Print information about all processes associated with a terminal.

l Long format. Print detailed information on each process.

x Print information about all processes not associated with a terminal.

The following is a description of each possible column in the output from ps.

STATE represents the current state of the program. It is the first column in the long format report, with the appropriate flags arranged in these positions: AlAwLdSwLkSy. The meaning of the flags is as follows.

Al Alive. The process has not yet terminated. It may be awake (running without waiting for input) or asleep (waiting for input from, say, a terminal or disk drive).

Aw Awake. The process is not waiting for a slow event and may be run immediately. The absence of this flag with an alive process indicates that it is asleep.

Ld Loaded. The process has been loaded into memory.

Lk Locked. The image of this process is currently locked in main memory, that is, it can't be swapped out to disk to make room for other processes.

Sw Swapped. The image of this process currently resides on the swap device. Micronix decides what to swap on the basis of how often a process is called.

Sy System. This process has made a system call and is waiting

for the results.

UID User ID number. Number identifying the user running the process.

PID Process ID. The identifying number of the process.

PPID Parent's process ID. The identifying number of the parent process.

CPU The CPU priority of the process.

PRI The run-time priority of the process.

NICE The user settable priority of the process.

WCHAN The address of the event on which this process is sleeping.

TTY The name of the controlling tty for this process.

COMMAND The name of the command. (Usually a close approximation of what was entered from the keyboard.)

EXAMPLE

To kill your runaway process from another terminal:

1) Log in.

2) Type "ps".

3) Locate the process of interest in the resulting printout.

4) Type "kill process-ID-number"; ex: "kill 7"

Another route to the same destination is to log into superuser mode (assuming you have such priveleges) using the su command. Superuser mode allows you to kill any process in the system, unlike the above sequence, which only allows you to kill your own processes. Use the a flag in this case to see all processes.

SEE ALSO

sh(1), signal(2)

NAME

ptc - Pascal to C translator

SYNTAX

ptc -[flags] <file>

DESCRIPTION

Ptc is a program that accepts as input lines of Pascal text and produces as output a corresponding C program which is acceptable to the Whitesmiths, Ltd. C compiler. If <file> is present, it is taken as the Pascal program to translate; otherwise input is taken from STDIN.

The flags are:

-c Pass comments through to the C program.

-f Set the precision for reals to single precision (float). Default is double.

-k Permit pointer types to be defined using type identifiers from outer blocks. Default is ISO standard, i.e., the type pointed to must be defined in the same type declaration as the pointer type definition.

-m# Make # the number of bits in MAXINT excluding the sign bit, e.g., MAXINT becomes 32767 for -m15, 1 for -m, etc. Default for MAXINT is 32766 [sic]. Acceptable values for # are in the range [0, 32).

-o* Write the C program to the file * and diagnostics to STDOUT. Default is STDOUT for the C program and STDERR for diagnostics.

-r Turn off runtime array bounds checks.

-s# Make # the number of bits in the maximum allowable set size, i.e., the size of all sets whose base type is integer becomes the specified power of two. Acceptable values are in the range [0, 32). Default is 8 (256 elements).

The CP/M operating system implementation on the Intel 8080 and Zilog Z80 restricts the acceptable value for # to the range [0, 16]; for maximum portability, this restriction should be honored.

Identifiers are mapped to uppercase to keep from conflicting with those declared as reserved words in C. Moreover, structure declarations may be produced that contain conflicting field declarations; and declarations are present for library functions that may not be needed. All of these pecadillos are forgiven by the use of appropriate C compiler options.

RETURNS

Ptc returns success if it produces no diagnostics.

SEE ALSO

cc (1), as (1), cp1 (1), cp2 (1), cpp (1)

NOTE: This program and documentation are products of Whitesmiths, Ltd.,

NAME

pwd - present working directory

SYNTAX

pwd

DESCRIPTION

Pwd finds and displays the full path name of the working directory under which it is executed.

NAME

ratfor - ratfor-to-FORTRAN translator

SYNTAX

ratfor

DESCRIPTION

The **Ratfor** preprocessor for FORTRAN enables you to write in a the free-form structured language, Ratfor, then compile your programs with your existing FORTRAN compiler. This implementation incorporates **include** and **macro** capabilities. **Ratfor** reads from the standard input and writes to the standard output. It may be used as a filter.

Lines of the form:

include file

are replaced by the contents of the named file. **Includes** may be nested. Note that some implementations include the file "file.rat". In this implementation, the file must be named explicitly.

Macro definitions such as:

define(token,string)

or

define(token,...$1...$2...$3...)

may appear. The left parenthesis must follow the word define immediately. In the first case, **string** is merely substituted for **token** everywhere it appears. In the second, occurrences of the form:

token(arga,argb,argc)

are replaced by the defining string, but with the values substituted for the arguments. The construction $n, where n is a decimal digit, signifies the nth argument. A given macro may have at most nine arguments. Macro definitions may be recursive.

Any valid FORTRAN statement is a valid **ratfor** statement. A statement or group of statements may be enclosed in curly brackets "{" and "}". The result is treated as a single statement. In addition to standard FORTRAN, the following are understood:

if (condition) statement;

The statement is executed if the condition is true.

if (condition) statement; else statement;

If the condition is true, the first statement is executed, otherwise the second statement is executed. An **else** goes with the last un-elsed **if**.

while (condition) statement;

The statement is executed repeatedly as long as the condition remains true (possibly zero times as the condition is always tested before the statement is executed.) The condition may not be left out. For an infinite loop, see **repeat** below.

for (statement; condition; statement) statement;

First, the first statement is executed. Then repeatedly the condition is tested, and if true, the last statement is executed, then the second. If the condition is false, the loop is broken and control proceeds to the next statement following the **for**. More clearly: the first statement is the initializer of the loop, the second is the incrementation, and the last is the body of the loop.

repeat statement;

The statement is executed repeatedly.

repeat statement; until (condition);

The statement is executed repeatedly until the condition is true. The condition is always tested after the statement is executed, which is to say, the statement will always be executed at least once.

do limits; statement;

A FORTRAN **do** loop is set up with the specified limits and the statement as the body of the loop. You are relieved of the burden of dealing with the **continue** statement and its statement number.

number statement;

Statements may be numbered. Numbers may comprise at most five digits. You need not worry about where on the line the number appears.

break;

One level of **while**, **for**, **repeat**, or **do** loop is broken out of.

next;

Transfers control to the looping mechanism of the innermost **while**, **for**, **repeat**, or **do** in which it occurs.

The following abbreviations may be used:

>	.gt.	greater than
>=	.ge.	greater than or equal
<	.lt.	less than
<=	.le.	less than or equal
==	.eq.	equal
!=	.ne.	not equal
^=	.ne.	not equal
!	.not.	logical negation
^	.not.	logical negation
&	.and.	logical and
|	.or.	logical or

EXAMPLE

To compile a Ratfor program using Microsoft FORTRAN:

```
ratfor <program.rat >program.for
f80 program.rel,tty:=program.for/n
l80 program.rel,forlib.rel,program.com/n,/e
```

NOTES

Do statements generate a spurious, although harmless, extra **continue** in the absence of a **break** statement.

ALSO READ

Kernighan, B.W. and P.J. Plauger, Software Tools, (Addison-Wesley, 1976).

NAME

rm - remove files (directories)

SYNTAX

rm [-r] file ...

DESCRIPTION

Each of the named files is removed, if possible. It is also used also to remove directories; files must be removed first, however.

The -r option removes entire hierarchies of files. This means all files, subdirectories, directories, etc. will be permanently erased, which means that this option is quite dangerous! When run interactively, this program asks if it may proceed with the requests, however, offering some amount of protection, but still, take care when using this command.

ERRORS

It is not possible to remove non-empty directories.

NAME

rp - rational pilot to pilot translator

SYNTAX

rp <rational_pilot_file >pilot_file

DESCRIPTION

The rp preprocessor for pilot enables you to write programs in the free-form structured language "rational pilot", then automatically translate them into standard pilot for interpretation by the "pilot" command. This implementation incorporates include and macro capabilities. Rp reads from the standard input and writes to the standard output.

Lines of the form:

include file

are replaced by the contents of the named file. Includes may be nested.

Macro definitions such as:

define(token, string)

or

define(token, ...$1...$2...$3...)

may appear. The left parenthesis must follow the word define immediately. In the first case, string is merely substituted for token everywhere it appears. In the second, occurrences of the form:

token(arga, argb, argc)

are replaced by the defining string, but with the values substituted for the arguments. The construction $n, where n is a decimal digit, signifies the nth argument. A given macro may have at most nine arguments. Macro definitions may be recursive.

Any valid pilot statement is a valid rp statement. A statement or group of statements may be enclosed in curly brackets "{" and "}". The result is treated as a single statement. In addition to standard pilot, the following are understood:

if (condition) statement;

The statement is executed if the condition is true. The condition may be either a match query or a define query (see pilot (I)).

if (condition) statement; else statement;

If the condition is true, the first statement is executed, otherwise the second statement is executed. For example

```
t: Give me your answer true.
a:
if (m: y)
        {
        t: Oh, Joy!
        t: ...
        }
else if (m: n)
        t: Oh, Woe!
else
        t: Eh?
```

An **else** goes with the last un-elsed **if**.

while (condition) statement;

The statement is executed repeatedly as long as the condition remains true (possibly zero times as the condition is always tested before the statement is executed.) The condition may not be left out. For an infinite loop, see **repeat** below.

for (statement; condition; statement) statement;

First, the first statement is executed. Then repeatedly the condition is tested, and if true, the last statement is executed, then the second. If the condition is false, the loop is broken and control proceeds to the next statement following the **for**. More clearly: the first statement is the initializer of the loop, the second is the incrementation, and the last is the body of the loop.

repeat statement;

The statement is executed repeatedly.

repeat statement; until (condition);

The statement is executed repeatedly until the condition is true. The condition is always tested after the statement is executed, which is to say, the statement will always be executed at least once.

break;

One level of **while**, **for**, or **repeat**, loop is broken out of.

next;

Transfers control to the looping mechanism of the innermost **while**, **for**, or **repeat**, in which it occurs.

ALSO READ

Kernighan, B.W. and P.J. Plauger, Software Tools, (Addison-Wesley, 1976).

NAME

sh

SYNTAX

sh [-v] [-c command] [filename]

DESCRIPTION

The **shell** establishes an interface between the user and the Micronix operating system. It is a general purpose command interpreter and it should be noted at this point that the **shell** is just another process to Micronix; it is by no means the final word in command interpreters. The user may wish to write his/her own command interpreter. That is to say, the shell is not part of the operating system, but merely serves to provide convenient access to the system's facilities.

The Micronix command interpreter, or **shell**, provides convenient access to the Micronix system facilities. It is intended to be simple, even transparent, to use.

The command line.

If the -v (verbose) option is given on the command line, the shell will "echo" each of its commands before executing them.

If the -c option appears, the shell will take the next argument literally as its only input, and then stop.

If a file name appears in the argument list, the contents of that file will be used as the input for the shell.

In normal operation, the **shell** enters interactive mode and successively reads, interprets, and executes commands, taking its input from the standard input (usually the user's terminal).

When it receives a line of input, the shell first breaks it into "words". Normally a word is a cluster of symbols surrounded by white space. Certain symbols have special meaning to the **shell**, i.e. characters in the set < > & | ` " ( ).

When encountered, they are treated as separated entities, regardless of the presence or absence of surrounding white space.

Shell Features:

Command Execution.

The first word of a command has special meaning for the **shell.**

It expects this word to be the "command name". When it encounters a word which it does not recognize, the **shell** assumes it refers to a program, which it then attempts to find and execute.

Path Searching.

If it fails to find the command, it prints an error message. The **shell** searches for commands in a series of pre-specified directories called the "search path". The default search path is ".", "/bin", "/usr/bin". The **shell's** current idea of the search path may be examined or changed by the built-in "path" command discussed below. If the first word of a command contains a / (slash) character, then the normal path searching mechanism is disabled and the shell attempts to execute the exact name specified.

Directory Hashing.

The **shell** internalizes the contents of the directories in the search path, so that in order to locate a command, it does not need to refer to the file system. This function is intended to be transparent. If, however, a program is added or removed to one of the directories in the search path in the middle of a session, (normally a very rare occurrence) the shell's internal tables will disagree. In this case it may be necessary to enter the command's name twice, (causing the **shell** to remake its internal tables). There is a significant speed payoff to this way of doing things.

I/O Redirection

There is the concept of standard input, standard output, and standard error.

An accepted way to fashion programs is to have them read from the standard input, write to the standard output, and send error or other transient messages to the standard error.

By default, each process is set up so that its standard input is read from the controlling terminal and standard output and error are written to the terminal. The **shell** has provision to change this. You may arrange to have input and/or output read from/written to files or other processes instead. In the following example, the command is read from the file.

```
command < file
```

Input Redirection.

Arrangements may also be made so that the command's standard input becomes the file:

command > file

command >> file

command >& file

command >>& file

Output Redirection.

In the first form, (>), the command's standard output is sent to the named file, destroying its previous contents. The second form, (>>), appends the command's output to the end of the named file. In the third form, (>&), both the standard output and the standard error are sent to the named file. The last form is the same as the third, except the option is appended to the end of the named file.
An expression of the form "(a > b) >& c" can be used to split the output from the command a, standard output to file b, standard error to file c.

Pipes.

Commands may be piped together, that is, the standard output of one process can be "connected" to the standard input of another. The processes are run concurrently. A "|" (vertical bar) is the symbol denoting a pipe. The following command line

print file | lpr,

would send the output of the print command to the input of the lpr command.

The **shell** has its own syntax for a linear array of pipes, limited in length only by the maximum available number of processes. As an example:

cat file find word1 | find word2 | words

comm1 | comm2 | comm3 | comm5 | ... etc.

Asynchronous Commands.

It is possible to run a command asynchronously, or in the background. Normally, a command is run and the **shell** waits for it to finish before prompting for additional input. When a

command is run asynchronously, however, the shell does not wait, but returns immediately, while the command continues to execute parallel to the shell. With this facility, a single user on a single terminal can be doing several things at once. A pipeline followed by an ampersand on the command line causes all of the processes in the pipeline to be run asynchronously. When an asynchronous command completes or is abnormally terminated, the shell gives notification. The process ID is given when the asynchronous process is dispatched.

Example:

```
ls -l&
```

Argument Expansion.

All words in the command line are subject to expansion. Words are expanded with the following characters:

```
* ? [...] [^...]
```

If a word contains any of these constructions, it is evaluated as a pattern to be matched against the set of extant file names. That is, the given word will be replaced by the list of all file names it matches.

Following are the rules for pattern matching.

`*` This character is "wild card" of sorts, matching any number of characters, and any character, including zero.

`?` Matches any single character.

`[ ... ]` Matches any single character listed within the brackets.

`[^...]` Matches any single character NOT within the brackets.

`x` Ordinary characters match themselves.

Matching patterns may be composed of arbitrary arrangements of the above constructs.

If a string of characters is enclosed in quotation marks ("), all argument expansion is disabled.

If a string of characters is enclosed in grave accents (`), that string is interrupted as a command, executed and replaced by the output of the command and broken into words.

Within braces ([]) there may appear expressions of the form X-Y, meaning that all of the characters between X and Y are inclusive.

Example:

```
[A-Za-z]
```

would match any single upper or lower case letter.

Grouping of Commands

Commands may be grouped together arbitrarily by using parenthesis. When enclosed in parentheses, any arbitrary command or series of commands is treated as a simple command.

Example:

```
(cd a; comm1) | (cd b; comm2)
```

would run the first part of the pipeline in directory "a" and the second in directory "b".

Parenthetical expressions may be nested. There are limits on the maximum number of processes allowed, however.

To run a series of commands in the background, for example;

```
(comm1; comm2; comm3)&
```

Sequential Execution Operator

More than one command may be put on a line by separating the command with semicolons (;)

Example

```
command1; command2
```

The commands are executed sequentially in the order given (left to right).

Built-In Commands

There are a number of commands internal to the **shell:**

alias

alias name1 name2

alias name3 "a complex command with embedded spaces"

Aliasing provides the facility to rename or develop abbreviations for commands. The first form prints the current list of aliases in effect. The others introduce a new alias. After these alias commands have been typed to the shell, giving the command "name1" will result in issuing the command name2. That is to say the right hand word or group of words is substituted for the left. Use unalias (see below) to turn off an alias.

How to use **alias:**

If you find you frequently type a long and cumbersome command, you should consider setting up an alias for it.

Some often-used commands are:

```
far /dev/fla -xv
mount /dev/dja /f
umount /dev/dja
```

To set up **alias**, edit the .sh file in your home directory (the directory in which you find yourself when you first log in). Add lines like the following:

```
alias x "far /dev/dja -xv"
alias m "mount /dev/dja /f"
alias u "umount /dev/dja"
```

Next time you log in, all of these "aliases" will be automatically set in the shell. You may type just the one letter abbreviation for the long command.

You may view the alias table by typing "alias". An "alias" is eliminated with **unalias.**

Note that this feature may also be used to smooth over trivial name differences found in different systems. If you are used to typing something else for one of the commands, make yourself an "alias."

chdir (cd)

chdir name

These change the current directory. Micronix maintains the concept of a current directory. The first form changes the current directory to the home directory (described below). The second form changes to the named directory. The command, cd is synonymous to **chdir**, and is easier to type.

dir

dir name

dir names

The "dir" is short for "directory". The first form list the files in the current directory. The second lists the files in the directory, "name". The third prints the list of names. All forms print their display in an easily read, alphabetized and columnar form.

echo

echo args

The first command toggles the **shell's** command echoing feature. The second prints the arguments.

home

home name

These commands examine or change the home directory. The first form prints the **shell's** currently remembered home directory name.

prompt word

This command changes the **shell's** prompt. The prompt is changed to the given word. The default prompt is % or #; the # prompt indicates super-user.

source file

The **shell** takes its next input from the named file instead of the standard input. Input is again taken from standard input upon reaching the end of the file. A "source" may be nested, that is, a named file may contain another source command.

sync

This command calls the **sync** system call (mass storage synchronization, see Section 2). It insures that all information destined for mass storage devices is actually written out. **Sync** should always be called before bringing the system down to insure the integrity of the file system(s).

type file(s)

The contents of each of the named files are printed verbatim on the standard output.

unalias name

The given name is removed from the alias table.

wait

This command causes a wait until all all asynchronous processes (if any) have completed. Wait may be prematurely interrupted by pressing the RUB-OUT or DELETE key.

NAME

sort - sort text files

SYNTAX

sort [-bdfnrutx] [+column] [-o output] [input ...]

DESCRIPTION

All of the named input files are sorted together.

If no input file names are given, then the standard input is used instead.

The result is written to the standard output unless an output file name is given (with the -o option).

The default sorting key is the entire line.

Ordering is lexicographic, by bytes, in machine collating sequence.

The following optional flags may be used to influence the sorting.

+N	Sort by the Nth column over from the left. Default is column 0.
-b	Ignore leading blanks (spaces and tabs) in comparisons.
-d	Dictionary order, only letters, digits and blanks are significant in comparisons.
-f	Ignore case in comparisons.
-i	Ignore non-ascii characters.
-n	Sort by integral number. Fields are ordered by numerical value.
-o file	Send the output to the named file "sort file -o file" will work.
-r	Reverse. Sort backwards.
-tx	Tab character is set to the character x.

'-' options listed above may be combined.

That is

```
sort -b -d -f -i
```

is the same as

```
sort -bdfi
```

Sort can be used on arbitrarily large files. It is limited only by the amount of temporary file space it can consume.

EXAMPLES

A simple example, to see what a file looks like once sorted:

```
sort file
```

To sort by the second column:

```
sort +1 file
```

To sort a list of numbers:

```
sort -n file
```

To sort a file in place:

```
sort file -o file
```

To sort the password file by numerical user id

```
sort +2 -t: -n /etc/passwd
```

SEE ALSO

unique (1)

NAME

split - split a file into pieces

SYNTAX

split [-n] [input] [proto]

DESCRIPTION

Split reads in the given file and writes it out again in n-line pieces.

The input file remains unchanged. It is only read, not written.

If no file names are given or - is given as the input file name the standard input is read.

If no prototype file name is given, the prototype file name "x" is used. The prototype file name is used to generate a whole series of automatically generated serialized file names. The serialized file names are produced from the prototype file name by adding a two letter suffix. The first two letter suffix used is "aa". The second is "ab", etc.

If no numerical argument preceded by a minus sign appears the file will be split into 1000 line pieces.

An argument such as -78 requests the individual files each containing 78 lines be produced.

Split produces meaningful results only if used on a text file. Line counting will be incorrect if applied to files having lines longer than 512 characters.

Split is useful if, for example, you have a file far too large for your text editor to process. One could then theoretically split the file, edit the pieces, and then reconstruct the file. An example of this appears below.

EXAMPLES

Consider the command:

```
split file
```

If file is a 7000 line file, the seven files:

```
xaa xab xac xad xae xae xaf
```

will be produced, each having 1000 lines.

The command:

```
split -57 znorg OUT
```

would split the file "znorg" into possibly many small files of 57 lines each named:

```
OUTaa OUTab OUTac OUTad OUTae OUTaf
```

etc. This example also demonstrates the significance of case for Micronix file names.

Split can be used in a pipeline:

```
man edit | split -100
```

This command would cause the formatted documentation for the "edit" command to be written into a series of small files.

No input or source file name was given so the standard input is read (through a pipe). Pipes are discussed under the documentation for "sh" in section I of the Micronix reference manual.

No prototype file name is given here, so "x" is assumed. The files produces would be named:

```
xaa xab xac etc.
```

One way to edit an outrageous file:

```
split -500 file

edit x*

cat x* > file
```

The first command in this example splits the file into 500 line pieces. The second line summons the editor to work on the pieces. The third line restores the orignal file. (Make sure you have no other files whose names begin with x in your directory for this example.)

NAME

stty - set typewriter (terminal) options

SYNTAX

stty [option ...]

DESCRIPTION

Stty sets certain I/O options on the current output typewriter. With no arguments, it reports the current settings of the options which differ from the default settings.

The following are options:

raw Raw mode input (no erase, kill, interrupt, quit, EOT, parity bit, passed).

-raw Negate raw mode.

cooked Same as -raw; default is cooked mode.

-nl Allow carriage return for new-line, and output CR-LF for carriage return or new-line; default is -nl mode.

nl Accept only new-line to end lines.

echo Echo back every character typed; echo is asserted by default.

-echo Do not echo characters.

lcase Map upper case to lower case.

-lcase Do not map case; default.

-tabs Replace tabs by spaces when printing; default.

tabs Preserve tabs.

erase Set erase character to "c"; default erase or backspace character is "H" (^H).

kill c Set kill character to "c"; default kill or line delete character is "X" (^X).

shake Do hardware handshaking using the clear-to-send (RS232 CTS) pin.

-shake No handshaking; hardware handshaking is off be default.

Baud Rates: 50 75 110 134 150 200 300 600
1200 1800 2400 4800 9600 19200

Set typewriter baud rate to number given, if possible.

Micronix presently has no built-in delays.

SEE ALSO

stty (2)

NAME

su - set user identification

SYNTAX

su [name]

DESCRIPTION

Su allows you to change your effective user ID in midstream, so to speak.

The most common use of this program is to temporarily become a super-user in order to install a program, change a system-wide file, kill off an errant process or other administrative task.

If no name is given, "root" is the default. This puts you in super-user mode. If "root" has a password, non super-users will receive a password prompt.

Another use for su is to grab someone else's terminal without the delays of logging them out, logging yourself in, and so on. Any time enter su with a password-protected name, you will receive a password prompt.

EXAMPLE

su - to enter super-user mode.

NOTE: Since the super-user has powers that are beyond the reach of normal users, "root" should definitely be password-protected. Use super-user with caution and only when necessary. System defenses that prevent you from making catastrophic mistakes are suppressed in super-user mode.

SEE ALSO

passwd (1)

NAME

sum - compute check sums

SYNTAX

sum [-r] [files]

DESCRIPTION

Sum computes and prints a check sum and block count for each named file. (Blocks are 512 bytes long.) If no file names are given, it computes a check sum for the standard input.

If the -r option is used, sum recursively descends the file hierarchy, summing each file as it goes. In recursive mode, special files and directories are not summed.

As an example, check sums can be used to verify the integrity of a file over the phone. Sums are in the range 0 to 65,535. If two files have identical check sums, it is quite likely that they are identical.

NAME

tail - prints the last few lines of a file

SYNTAX

tail [-N] [file]

DESCRIPTION

Tail prints the last n lines (default = 10) of the named file. If no file is given, the last n lines before the first end-of-file encountered on the standard input are printed.
br

BUGS

Tail reads the entire file in order to obtain the last few lines. If n is too large, tail will fail since it stores the text of lines in memory.

NAME

td - tree dump (file backup)

SYNTAX

td [-a] [-dN] [-hN] [-i] [-u]

DESCRIPTION

Td is your primary tool for routine backing up of files. You can select the whole file system (not the normal procedure) or certain parts of it. You can also do **incremental dumps**, which means copy only those files that have been changed since the last td. **Incremental dumps are standard procedure in proper system management.** Micronix maintains a "dump table" that keeps track of what was dumped when.

Files dumped with td are usually placed on floppy diskettes, but you can dump to hard disks just as well.

It doesn't matter if your dump won't all fit on one diskette; td allows you to change diskettes on the fly. Each disk is filled to capacity, then the program asks for operator service. In this sense it does need a bit of babysitting.

Before beginning a major dump, you should have a stack of formatted diskettes (see program fdj). These diskettes must be pre-formatted but they need not contain file systems. Td will notice that a diskette has no file system and ask you if it should make one. It will then call the **mkfs** program for you.

Unlike some other back-up systems, this program's back-ups are in file system form, meaning it is easy to look through a dump with the mount/dir, ncheck, or ls -R commands.

Td is a hybrid, in that some options must be entered on the command line while others are handled interactively.

Command Line Options:

-a All. Dump all files. Without this flag, only files under 1000 blocks long are dumped.

-dN Day. Dump only files newer than N days old. Avoid using this flag along with the incremental option, because the results depend on when the last incremental dump was performed, which gets confusing. Use the d and h options only when you mean "Copy all of the files changed since this many days/hours ago, regardless of whether they were dumped before or since then."

-hN Hour. Dump only files newer than N hours old. Note: The -d and -h options are additive.

-i Incremental. This flag, when used without the "u" option (see below), copies the files that have been changed since the last incremental dump WITHOUT updating the dump table. Therefore you could perform another incremental dump later and even if the files remain unchanged in the meantime, they'll still be included in the dump.

NOTE: If you want this incremental-without-update to take effect, respond F (for full) at the interactive "full or incremental" prompt.

-u Update. Update the central dump table at the conclusion of a successful dump. Be aware that using -iu has the same effect as using neither option, and responding I (for incremental) at the "full or incremental" prompt.

Interactive Options

Source Directory: Respond the pathname of the directory you want the dump to start at. The dump will follow downward through all of the subdirectories until all appropriate files have been copied.

Destination Device: Enter the pathname (/dev/...) of the floppy or hard disk to which you are dumping. For example, enter /dev/f1a for an 8" floppy.

"Do you want a full or an incremental dump (F/I)?" Respond accordingly and press RETURN. If the directory you specified as the Source above isn't in the dump table (having never been dumped previously), td defaults to a full dump.

"Insert new media and press RETURN to continue or type the name of a new device." Insert a formatted diskette into the drive you designated as the Destination Device. You can use diskettes that already have some files on them, assuming they are in Micronix format.

Note that if you are dumping a file that has the same pathname as one on the diskette, the dumped one replaces the existing one. You can use this fact to your advantage if you employ one diskette over and over as the recipient of regular incremental dumps. It will then always contain the latest'n'greatest versions of the files in the source directory. The danger inherent in this approach is that if anything damages that diskette, you won't have yesterday's or the day before's to restore from.

The "type the name of a new device" option allows you to switch destination devices without getting out of the td program. Thus you could fill up a floppy on /dev/fla and then move on to /dev/flb without getting up to swap diskettes.

EXAMPLES

The normal case: incremental backup with update of dump table. Dumping directory /a/jane to 5 1/4" floppy drive A -

```
td
Source directory: /a/jane
Destination device: /dev/mfa
...full or incremental (F/I)? I
```

Incremental backup of /b/joe/ws WITHOUT update of dump table on 8" floppy drive A -

```
td -i
Source directory: /b/joe/ws
Destination device: /dev/fla
...full or incremental (F/I)? F
```

Backup of all the files created or modified within the past 8 hours on hard disk m16b -

```
td -h8
Source directory: /
Destination device: /dev/m16b
...full or incremental (F/I)? F
```

TO EXAMINE THE CONTENTS of a 5 1/4" backup diskette: Insert the diskette into the drive, and type

```
mount /dev/mfa /f
cd /f
dir
(more cd's and dir's, if desired)
cd /
umount /dev/mfa
```

TO RESTORE A SINGLE FILE (e.g. /usr/bin/adventure) from an 8" backup diskette: Insert the diskette, and type

```
mount /dev/fla /f
cp /f/usr/bin/adventure /usr/bin/adventure
umount /dev/fla
```

TO RESTORE AN ENTIRE DIRECTORY (say b/bill) from a 5 1/4" backup diskette: Insert the diskette, and type

```
mount /dev/mfa /f
cptree /f/b/bill /b/bill
umount /dev/fla
```

FILES

/etc/dtab - the central dump table

SEE ALSO

cp (1), cptree (1), fdj (1), mount (1), ncheck (1), mkfs (1)

NAME

tee - a pipe fitting

SYNTAX

tee [-a] [-i] [file ...]

DESCRIPTION

Tee reads the standard input and writes it unchanged to the standard output, making copies in each of the named files.

Options

-a **Append.** Data is appended to the end of each named file, rather than being over-written.

-i **Ignore.** Interrupt signals are ignored.

EXAMPLE

```
edit | tee record
```

will keep a record of the editor's responses to commands.

NAME

touch - updtate the modification time for a file

SYNTAX

touch file ...

DESCRIPTION

Each of the named files has it's modification time set to the current time. This is done by reading a character from a file and then writing it back. No file's data is changed.

EXAMPLE

touch file

NAME

translit - transliterates characters

SYNTAX

translit from [to]

DESCRIPTION

Translit can be used to perform a variety of useful character mapping functions. **Translit** modifies its input on a character-by-character basis. **Translit** changes each character in its **from** set to the corresponding one in its **to** set. If there is no **to** set, the effect is to remove each character found in the **from** set. If the **to** set has fewer characters than the **from** set, the last character in the **to** set is repeated as many times as needed to fill out the to set.

Notation of the form x-y may be used as shorthand set notation to mean all the characters lexically between x and y, where x and y are letters of the same case or both digits.

If ^ is the first character of the **from** specification, it is understood to mean all characters not in the following specification.

Notation of the form @377 (an "at" sign followed by up to three digits of octal) may be used to specify characters by ASCII value.

If the **from** set contains fewer characters than the **to** set, or if the ^ is used as above, the last character of the to set is sufficiently replicated to fill the **from** set to the size of the **to** set. Furthermore, contiguous characters which are elements of the **to** set for which there is no explicit corresponding character in the to set are collapsed to a single occurrence of the last character in the specification for the **to** set.

EXAMPLES

To replace all occurrences of x by y:

```
translit x y <input >output
```

To change x to y and y to x:

```
translit xy yx <input >output
```

To delete all occurrences of A:

```
translit A <input >output
```

To map all lower case into upper case:

```
translit a-z A-Z <input >output
```

To put one white space separated symbol per line:

```
translit @40@11@12 @12 <in >out
```

SEE ALSO

change (1), edit (1)

NOTE

Translit is analogous to the UNIX tr program.

NAME

tree - draw a picture of the file system

SYNTAX

tree [starting_directory] [-a]

DESCRIPTION

Tree draws a picture of the file system tree structure starting from the root directory. Optionally, a different starting directory can be specified. Only directory names are listed, unless the -a option is given, in which case all files are listed.

Directories on which the user does not have search permission are shown as empty.

The picture is drawn on the standard output, so it normally appears on your screen, but it can also be sent to a printer (see below).

EXAMPLES

To see the entire accessible file system:

```
tree
```

To see Mary's directory hierarchy:

```
tree /a/mary
```

To see all files in Mary's hierarchy and print the result:

```
tree /a/mary -a | lpr
```

SEE ALSO

du (1), find (1)

NAME

tty - get terminal name

SYNTAX

tty

DESCRIPTION

Tty is a simple program that attempts to look up the full path name and number of the current "tty" device (teletypewriter or terminal). Normally, this is the name of the Micronix (terminal) port you logged in on.

To ascertain your device number, enter the tty command. A typical response is

```
/dev/ttyB
```

meaning that you are on "ttyB".

If no associated file name can be found, the message "not a tty" is printed instead.

FILES

/dev - Directory in which to search for file names.

NAME

umount - dismount a file system

SYNTAX

umount special

DESCRIPTION

The named special file is dismounted.

The special file must have been previously mounted with the mount command.

It is not permitted to dismount a file system with files still open. Diagnostic: "Device busy".

Sometimes it is necessary to use the "change directory" command before dismounting a device. As an example:

```
cd /; umount /dev/dja
```

EXAMPLE

```
umount /dev/dja
```

to dismount a floppy.

SEE ALSO

mount(1) - to mount file systems

NAME

unique - eliminate adjacent duplicate lines in an I/O stream

SYNTAX

unique [-n]

DESCRIPTION

Unique reads the standard input and writes the standard output. All but one of each set of adjacent duplicate lines are eliminated. Otherwise, the text is copied through unchanged. If the -n option is given, a count of the number of duplicates of each line is printed along with the text of the line.

EXAMPLE

To eliminate all duplicate lines in a file:

```
sort <file >temp
unique <temp >output
```

NOTES

In order to assure adjacency, the file in question must be presorted.

SEE ALSO

sort (1)

NAME

unrot - reformats lines produced by **kwic**

SYNTAX

unrot

DESCRIPTION

This is a special purpose program for use with **kwic** - the key word in context index generator. Output is assumed to be destined for printing on an 80 character wide device. **Unrot** reads the standard input and writes the standard output. A dollar sign ($) is assumed to be the folding marker put in by **kwic**. A blank column running down the middle of the page will be produced. To the right of the middle column will be keywords with following context. To the left will be context preceding the keyword in the original line.

EXAMPLE

To create a keyword in context index:

```
kwic <file >temp1
sort <temp1 >temp2
unrot <temp2 >index
```

SEE ALSO

kwic (1), sort (1)

NAME

update

SYNTAX

update

DESCRIPTION

Update periodically updates the system's disks from the in-memory cache. It calls **sync** (2) once every 30 seconds. Update puts itself in the background and runs silently.

Update also checks the Micronix configuration file /etc/ttys every 30 seconds. If this has changed, it sends a reconfiguration signal to the init process. This will take care of reseting baud rates, etc., as specified in the ttys file.

The usual way to invoke update is to put the command

```
update
```

in the /etc/rc file (see rc in section 5). This will start the update process every time the system goes multi-user. The advantages of running update are (1) less chance of data loss during a crash, and (2) automatic reconfiguration when the ttys file is changed. The disadvantage is that update uses memory space, and may cause swapping every 30 seconds if there is not enough memory for all active tasks. This can be cured by adding more memory, or not running update. This is not as serious as it sounds, since both sh and upm call sync at the end of each command, and reconfiguration can always be forced by tying the command

```
# kill -1 1
```

SEE ALSO

init (1), ttys (5)

NAME

upm - CP/M 2.2 for Micronix

SYNTAX

upm [upm arguments] [cpm program] [program arguments]

DESCRIPTION

Upm duplicates as closely as possible Digital Research's CP/M 2.2 operating system. It is essentially a rewrite of CP/M using Micronix system calls. It allows CP/M tools (such as "WordStar") and Micronix tools (such as "edit") to be used interchangably on the same files. Moreover, since Micronix runs on a Z-80, upm need not emulate an 8080. All CPM programs run at full speed. (The name "upm" was coined from the words UNIX and CP/M.)

This is a long document (about 20 pages). Its sections, in order of appearance, are:

Overview

Upm provides a close functional replica of the environment which applications programs written to run under CP/M expect. Most CP/M programs (such as Micropro's WordStar word processing program) will run unchanged under upm.

The intended purpose of upm is to bridge the gap between multi-user UNIX and the large amount of special-purpose software available for the CP/M operating system — giving true multi-user function for CPM software, and making up for the lack of some of these applications running directly under UNIX.

There is no significant difference between the way application programs are run under upm and the way they are run under CP/M. All documentation for these programs is valid. (However, some CPM documentation is oriented toward floppy disks. It may contain such phrases as "insert the working diskette into drive A". With Micronix, just as with a hard-disk CPM system, the user must realize that no insertion is needed.)

Upm is intended to be a means whereby applications software written for CP/M may be run unchanged directly under Micronix. Under Micronix there are no records, no extents, no disk drives in the CP/M sense, no file control blocks (FCBs), disk parameter blocks (DPBs), allocation vectors, etc., but upm does a good job of pulling the wool over the eyes of most CP/M programs.

Some CP/M programs do not work under upm, however, because of system dependence. For example, programs containing the 8080 instructions "in", "out", or "hlt" will not work. The instant one of these instructions is encountered, the operating system will be notified and the offending program will be terminated. In a multi-user system such as Micronix, all access to the hardware must be very tightly controlled and coordinated. If the above instructions were allowed, user programs could usurp the operating system's control over its resources. If a CP/M program goes awry it will have only a local effect - the rest of the operating system will continue running. Corrective action, if required, can be taken from another terminal on the system. (See the section on "exiting upm" below.)

Getting Started

Upm runs on the Micronix operating system, whose file structure is quite different from that of CP/M. The first thing you'll need to do to run your CP/M package is to copy it over to Micronix using the far (Floppy ARchiver) command. (See far in Section 1 of the Micronix Reference Manual.) Be sure you are in the directory that you want the programs to reside in. Insert your CP/M floppy diskette into floppy drive A and type:

```
% far fla -xv
```

for 8" diskettes, or

```
% far mfa -xv
```

for 5 1/4" diskettes. (mfa2 for double-sided soft-sectored diskettes)

The far command copies all of the files on the CP/M diskette over to the Micronix file system and places them in your current directory.

Now, type

```
% upm
```

and you're off and running! There's still a lot more to learn, however, so as soon as the novelty wears off, read on.

Running Modes

Upm has two modes: DIRECT and INTERACTIVE. If you intend to run only one CP/M application program, use the DIRECT mode.

If you intend to run a series of CP/M programs, use the INTERACTIVE mode.

To enter INTERACTIVE mode, type

```
% upm
```

(You can also log in directly to upm. When used this way, upm is called the "cpm shell", since it looks as though you have simply logged onto a hard-disk CP/M system. See **account** in Section 1 of the Micronix Reference Manual.)

To enter DIRECT mode, simply type the program name after the shell prompt, just as though it were a Micronix program. For example, you can enter Wordstar directly from the Micronix shell by typing

```
% ws
```

The shell will set up the emulator for you. (Just how it is set up is detailed below in the section on "auto entry").

You can also get into the direct mode by typing

```
% upm program_name
```

By itself, this has no advantage over the more direct approach. But it allows you to include other arguments to **upm** on the command line. This is discussed in more detail below.

Drive-Directory Correspondences

CP/M uses a "flat" directory structure, where all the files on a disk are globbed together in a single huge directory. You can do the same thing under Micronix if you want, but sooner or later you will want to take advantage of the Micronix hierarchical directory structure to better organize your work. So to use **upm** most effectively, you should understand the relationship between CP/M drive letters and Micronix directories.

CP/M organizes its information by drive letter (A: - P:). Since the corresponding structure in Micronix is the directory, **upm** maintains a table of CP/M drive letters and the names of the Micronix directories to which they correspond.

The special command "=" displays the drive-directory relationships currently in effect.

For example,

```
A> =

A: -> /cpm
B: -> ./
```

tells you that the A drive corresponds to the directory /cpm, and the B drive corresponds to the directory "./" (the current directory). (If these directory names seem mysterious, type the command "help pathnames".)

You may change the drive designation pointers with a command of the form:

```
B:dirname
```

wherein B may be replaced by any of the letters A through P, and dirname (directory name) must contain at least one slant character ("/"). Upm recognizes a directory name by the presence of at least one "/". If the directory name doesn't already contain a slant, you can always add one by putting a ./ at the beginning. As an example:

If you want to set drive B to the subdirectory called "letters", type

```
A> B:./letters
```

Remember that "./letters" means "start in the current directory (.) and look in the subdirectory (/) called letters." (If you had typed B:letters, it would have looked like a request to execute a program called letters from drive B.)

Entering upm

Perhaps it would help to understand this idea of drive-directory designations if we walk through some typical examples of entering upm.

Typing

```
% upm
```

returns

```
Morrow Designs upm
N Bytes free

A: -> ./

A>
```

Now drive A corresponds to the current directory (./). You may now re-designate this "drive" and/or designate others.

One common setup is to address the directory /cpm (where many of the CP/M application programs live) as drive A, and the user's current directory as drive B.

```
A> a:/cpm
A> b:./
```

returns

```
A: -> /cpm
B: -> ./
```

The above display is automatic. It is also possible to request a display of the Micronix-upm relationship by entering an equal (=) sign after the A> prompt.

Customizing the environment

Drive designation modifiers may be used as commands to upm (in INTERACTIVE mode), on the command line, or in a upm "startup" file called ".upm".

Upm can be customized to come up in any configuration you choose. There are three places upm takes its configuration information from (in order of appearance):

1) In the .upm file in the current directory or the user's home directory.

2) The command line (entered to the right of the word "upm").

3) Interactive commands (only in INTERACTIVE mode, of course).

An example of drive-directory designators on the command line follows:

```
% upm a:/cpm b:./ b: a:ws
```

This means: run upm, set drive A to the Micronix directory "/cpm", set drive B to the directory "./" (your current directory), select drive B as the "current drive", then run ws from the A drive.

The order of events from the time upm is called is as follows:

1) An attempt is made to read the .upm startup file in the current directory. If this fails, an attempt is made to read this file in the user's home directory. The .upm startup file may contain drive designators, drive select commands, a LST: redirection command, a CP/M program name

and arguments. The CP/M program may be given as a Micronix file name OR a CP/M program name. The commands found in the .upm file are executed in the order they appear.

2) The command line is scanned for drive designators, drive select commands, a LST: redirection command and a CP/M program name and its arguments (if any). The commands found on the command line are executed in the order they appear. (Drive selection occurs first; program execution second). **If both a startup file (i.e., .upm) and command line arguments are present, the arguments take precedence. If two .upm files have been set up, the one in the current directory takes precedence over the one in the home directory.**

   So, for example, if a .upm file sets drive B: to one directory and a command line argument sets it to another, the command line will win. This lets you selectively override your standard settings.

3) A choice is made. If no CP/M program name was specified on the command line or in the startup file, upm goes into INTERACTIVE mode and gives a CP/M prompt (e.g. "A>"). If a program was specified, that program is immediately run.

The features described above may be used to customize upm, and can be used to advantage in conjunction with shell startup files. Some examples follow.

A possible .upm file might contain:

```
A:/cpm
B:./
LST:/dev/ttyC,1200
B:
```

Which means:

Make CP/M drive A refer to Micronix directory "/cpm".

Make CP/M drive B refer to Micronix directory "./" (your current directory).

Arrange to have printed output sent to ttyC at 1200 baud.

Select drive B.

With the preceeding .upm file in existence, you need only type "upm", and the .upm file will be scanned and everything will be set up as described.

This spares you the hassle of repeatedly typing in drive designation commands each time you enter upm.

It's also possible to specify more information on the command line. Consider:

```
% upm a:/cpm b:./ lst:output b: a:ws document
```

This command tells the system to:

1) Start upm.
2) Set drive A to /cpm
3) Set drive B to ./
4) Arrange to have printed output written onto the file "output" on the currently logged drive.
5) Select drive B
6) Run WordStar
7) Pass the argument "document" to WordStar

Admittedly, it would be a bit cumbersome to type this entire command every time. Instead, you could use the **alias** command to make a "shorthand" command line.

Simply create a shell startup file named ".sh" in your home directory containing (amoung other things):

```
alias ws "upm a:/cpm b:./ lst:output b: a:ws"
```

The shell (command interpreter) will read this file each time you log in.

When you want to run WordStar, type:

```
% ws document
```

and the above command is run for you.

Auto Entry

As mentioned above, it is often enough to type the name of a CPM program after the shell prompt. Here is what happens when you type

```
% prog
```

1. The shell looks through each directory in its "search path" (see below) for a command called "prog". If found, it is run.

2. If not found, the shell appends ".com" to the name, getting "prog.com", and looks for that file.

3. If prog.com is found, the shell sets up upm as follows: drive A is set to the directory in which the file was found. Drive B is set to the user's current directory.

Drive B is selected as the current drive. Finally, the command is executed from the A drive.

So for example, suppose that "ws.com" is in the directory /cpm. If you type

```
% ws document
```

then the shell silently expands this into

```
% upm a:/cpm b:./ b: a:ws document
```

and runs it.

If you get the response "command not found", then the file ws.com was not in the shell's search path, that is, ws.com was not in your current directory or in any of the directories where the shell ordinarily looks for commands. To remedy this, you could give the command

```
% path . /bin /usr/bin /cpm
```

This tells the shell to look first in "." (your current directory), then in "/bin", then in "/usr/bin", and finally in "/cpm". Better still, put this command in your .sh file, so that you don't have tp retype it each time you log in.

The default path, ie, the path you get if you don't say otherwise, is ". /bin /usr/bin". You can change this to anything you want with the path command.

Note: The shell's automatic upm set-up does not work for all CP/M programs. For instance, the Logicalc program lc.com insists that it be on the "currently logged drive", (whereas WordStar can be on either the currently logged drive or the A drive). So the best way to run logicalc is to put the following alias in your .sh file:

```
alias lc "upm a:/cpm b:./ lc"
```

Then logicalc will run happily as long as you remember that your spreadsheet files are on your B drive.

Terminal Emulation

upm has the capability of emulating different terminals. A terminal emulator converts the control characters that your terminal and a terminal-configured piece of software use to communicate "special effects". These special effects are things like moving the cursor around, clearing the screen or highlighting parts of the screen. By converting control characters you can make several terminals all look the same for software. This may allow you to use the same configured version

of software for all your terminals.

Terminal emulation may also be of use to you if you have an unusual terminal that is not listed in software installation programs. You would simply add your terminal specifications to the Micronix file /etc/terminals, and upm will emulate one of the more common types of terminals for you. Then, you install software for the terminal that upm is emulating.

There are three different terminals that upm can emulate: an ADM-3A, a Soroc IQ-120, and an ADM-31. These three terminals represent increasingly sophisticated levels of terminal "intelligence". The ADM-3A, the least intelligent, can clear the screen, and move the cursor. The Soroc IQ-120 represents the next level. The Soroc can do everything the ADM-3A can do, plus it can display characters in high or low intensity and has erase to end of screen and end of line functions.

The most intelligent terminals have all the capabilities previously mentioned, plus delete and insert line and character functions. The ADM-31 has all of these capabilities and represents the "highest" level of terminal intelligence.

The use of levels categorizes similar terminals in a small number of groups. This allows the terminal emulator to translate many different terminals' control codes into three subsets, requiring only (at most) three different configured versions of software. Your terminal may have many bells and whistles that aren't included in the highest level definition, but software writers rarely make use of unusual features when creating software for use with general purpose terminals.

The recon program will display the terminals listed in the /etc/terminals file, and tell you which terminal upm will emulate for your terminal selection. Install your software for the terminal emulated. If your terminal name is not listed by recon, you can add it by following the directions given in the entry in the Reference Manual for Files, /etc/terminals, in the back part of this binder.

Terminal emulation is activated by adding the terminal type to a line in the /etc/ttys file. upm notices that the ttys file has been changed, and searches for the line corresponding to the port that the terminal running upm is attached to. If this line has a terminal name on it, .bd upm then searches the file /etc/terminals for the matching terminal. If there is no terminal name on the line in the ttys file, or a matching terminal in the terminals file, upm turns off terminal emulation.

Terminal names are added to the /etc/ttys file by the recon program. The recon program displays the names of terminals in the /etc/terminals file, asks for the name of terminal connected to each login port, and adds the name to the line for that port.

If you examine the ttys file after using recon, you would see that terminal names have been added after the word "term=", as in

```
ttyA 9600 login term=mor20
```

In this case, the terminal name is mor20. If you prefer not to use the recon program to add terminal names, you can add the names yourself with an editor. Be careful not to leave a space between "term=" and the terminal name. The terminal name must be one of the abbreviated mnemonic names listed in the /etc/terminals file.

Initially, the /etc/terminals file has only about twenty different terminal names in it, so you may not find your terminal in the file. The terminals file is purposely short in an attempt to reduce the time spent scanning through the file looking for terminal names and capabilities.

The process of checking the /etc/terminals and /etc/ttys files to see if changes have been made takes upm a noticeable amount of time. In fact, the longer the /etc/terminals file is, the longer it will take upm to search through it. To reduce the time added to include terminal emulation, a special file is created for each login port when upm is executed. If upm finds this file and neither /etc/ttys or /etc/terminals has been modified more recently than this file, then upm uses the information in this cache file for terminal emulation. This file is in the /usr/upm directory, and is named upmttyX, where X is a letter representing the port, for example, A for the console (upmttyA).

If you do not have a /usr/upm directory, upm will never be able to create the special file upmttyX, and will take 10 seconds or so to start working every time. Once a upmttyX file has been created, upm will start working much quicker. If your file system lacks this directory, you can make one by typing

```
# mkdir /usr/upm
# []
```

upm will change the upmttyX file if it discovers that either of the files /etc/ttys or /etc/terminals has been changed since the upmttyX file was created. This means that you don't need to signal a change in terminal type to upm because upm will always check for it.

Printers

Or how to select the best way to talk to your printer.

There have to be half a dozen ways one could set up a printer with upm.

First, consider your environment: Are you going to be running

single user all the time? Are many users going to be using one printer? Will you be printing software listings or text?

Next consider your requirements: Are you going to use special forms that will be changing? Will you run many small files or a few large ones? Do you want to allow simultaneous printer access or lock out later users, returning the message "Busy"? Are you going to print many copies of a single large file?

Let's try to sort all of this out.

Most all CP/M applications programs send their printer output to the LST: device.

Upm can be taught to take those characters sent to the LST: device and do any of a number of things with them. If you don't tell it otherwise (the default condition), characters go into the LST: device and appear on the controlling terminal (on your screen). This is normally not very useful or beautiful.

The easiest way to send these characters to a printer is to:

1. Attach the printer to one of the I/O ports on the back of the Decision, say to ttyC, and then,
2. Put the word "lst" on the ttyC line of the Micronix configuration file /etc/ttys. (Also, put the baud rate on this line.)

All this is explained in detail in the documents called "printers" and "ttys" (type the commands "help printers" and "help ttys").

Once you have done this, anyone who uses upm on your system will automatically use the same printer. BUT, this doesn't police simultaneous access to the printer. If users are located within shouting distance, or if it is very unlikely that two people will ever be printing simultaneously, then this is enough. Otherwise, read on.

You can arrange to have characters from the LST: device sent to a file and saved up for later access. This can be useful if you want to print something again and again, or if you want to print later or on another machine. Entering

```
A>LST:file
```

is the command to accomplish this. Characters sent to the LST: device are written to the file named after LST:. "File" may be any arbitrary Micronix file name. It need not contain a slant. (Be careful though, as any previous contents of this file will be lost.) If the file did not previously exist, it will be created.

Once you have saved your output in a file, you can send it to the printer with the "lpr" program by typing the command

```
A>!lpr file
```

The exclaimation point is the way to run a Micronix command from the CP/M environment. This will send files to the printer in first-come-first-served order, making sure that different users do not interfere with each other. There is more information on this below.

You can send characters direct to a "character special file" (or in other words to some other port).

```
A>LST:/dev/ttyC
```

channels characters coming from the LST: device to "ttyC" into which might be plugged a printer. If you get the message "File or device busy", that's the case. You'll be able to access the tty as soon as whoever has it right now releases it. If you get the message "Permission denied", you should get your local super-user (possibly yourself) to execute the commmand:

```
# chmod a+w /dev/ttyC
```

to enable writing to the device. Ttys used like this should not be listed in the /etc/ttys file as "login" ports. You need to choose which ports you will use to log in on and which ports you will use as printers. In addition, you can specifiy the baud rate of your printer. (Default is whatever the device was set to before.) It is always a good idea to specify your baud rate. Also be sure your hardware and software baud rates agree:

```
A>LST:/dev/ttyD,1200
```

would send characters to ttyD at a rate of 1200 baud.

Printer Handshaking

Micronix supports Clear-to-Send (CTS) and X-on, X-off handshaking. If it asks (as WordStar does), you should tell your CP/M program to do no handshaking, as Micronix will be handling that end of things. X-on, X-off handshaking is always there and you don't have to specify it. If you don't use it, it doesn't get in the way.

CTS handshaking, however, needs to be carefully set up. Note that the Morrow Designs Wunderbuss I/O (found in Decision I table top models) has CTS coming out on pin-8 of the 25-pin connectors. This is not standard. The CTS pin is normally ignored. If you specify the CTS type of handshaking described below, no data will be transmitted until the signal on pin-8 on the Wunderbuss IO (pin-5 on the Mult/IO) is TRUE. (See the help files on "printers", "setup", and "cables" for details.)

The following command line

```
A>LST:/dev/ttyB,9600,cts
```

would send characters through the CP/M LST device to ttyB at 9600 baud with CTS handshaking.

Policing the Printer

If you want only one person to be able to access your printer at a time, type:

```
# chmod u+l /dev/ttyC
```

(The # is a super-user shell prompt. You should log in as "root" (the super-user) to execute this command.) This will set the single write lock bit, after which only one user will be able to access the printer (ttyC) at a time. Other users will get the message "File or device busy." With the lock bit set, once a user gains access to the printer, he will hog it until he either A) exits from upm, or B) types the command

```
A> LST:
```

to release the printer (by re-assigning it to "nothing").

If you want many users to be able to use the printer at once (and do your own policing), type:

```
# chmod u-l /dev/ttyC
```

to clear the lock bit. In this state, the users will have to decide amongst themselves when to print. You'll also probably have to move the paper to "Top of form" manually after printing. If two users try to print at once, the characters will be intermingled (very ugly).

If you prefer to have software make the use decisions for you (set up a spooling printer), use the lpr program. It is recommended that you prepare your printed copy by formatting it, sending it to an output file (as described above), then type the command

```
A> !lpr file
```

Lpr is good at handling large printing jobs done on the same type of paper with printers which never fail, etc. Lpr works in a deferred bulk fashion. If you want immediacy, use the direct tty method outlined above. See lpr in section 1 for more details.)

The system-wide LST device.

It is possible to designate a particular device on the system to

be the recipient of all output from all instances of upm which are sending characters to their respective LST devices. (Note that if a user has specified a LST device in his own .upm file, it will override the system-wide LST device.)

This is accomplished by making an entry in the /etc/ttys file. The /etc/ttys file is scanned by upm for a line containg the word "lst". If such a line is found, then upm arranges that LST characters will be sent to the device named by the first word on that line.

We desperately need an example of this simple concept. Suppose your "/etc/ttys" file contains:

```
ttyA login 9600
ttyB login 9600
ttyC lst   1200
```

Any upm program run on the system will send its LST output to ttyC as specified.

The Micronix /etc/ttys file as shipped has such an entry.

Select Errors

It is not possible to get a CP/M SELECT ERROR under upm.

If you select a drive which has as yet no corresponding directory, upm inquires with the line

```
Select a directory for drive D:
```

for instance.

You should respond by typing the name of a Micronix directory (/a/tom/letters, for example). Upm will continue badgering you until you do.

This feature can be used to change directories right in the middle of a program (WordStar as a good example):

> Change the logged drive to drive E: (previously undefined). Upm will ask you for a directory name. Respond with the name of the directory you want to reference. Drive E: will now correspond to that new directory (and you didn't have to get out of WordStar do to it).

File Names

Upm intervenes in all file transactions originated by the CP/M program. When the CP/M program thinks it is opening a CP/M file, it is really opening a Micronix file. Upm handles conversion

back and forth between CP/M FCB's and Micronix file path names. It maintains a table of correspondences between CP/M drives and Micronix directories. For example, a CP/M program might read from a file named

B:TEST.DOC

Here "B" indicates CP/M drive B and "TEST.DOC" is the CP/M file name. Suppose for a moment that upm is currently remembering the correspondence

B: -> /a/hubert

then the above CP/M file name would be translated to:

/a/hubert/test.doc

We see here that "B:" got translated to "/a/hubert/" and TEST.DOC got translated to "test.doc".

File Names

CP/M file names are all upper case. Upm converts these to all lower case Micronix file names as seen in the above example.

(If a "/" appears in a CP/M file name, it will be converted to a "|" because "/" has special meaning in Micronix file names.)

It should be noted that some files may be inaccessible from upm because they have "CP/M impossible names". This is no great stumbling block, however, as one can simply rename the Micronix file(s). (See mv in Section 1 of the Micronix Reference Manual.)

For file names to be accessible from upm the rules are:

All lower case letters.

Maximum eight character filename.

Maximum three character extension (optional).

Names without an extension may be at most eight characters long. and should have no final ".".

No question marks (?) or asterisks (*) may be used as a character in the name.

Examples of valid upm accessible file names:

a.doc example.txt ppp

z.f prog.c peter

BIOS Jump Table

Upm also contains a "BIOS jump table" bearing a striking resemblance to the original, and corresponding in function. The two BIOS calls, "read sector" and "write sector," are considered illegal as they have no corresponding function in the Micronix environment. All the other BIOS calls are allowed, however, some are silently ignored. (See the CP/M 2.2 Operating System Manual for more information on the CP/M BIOS.) The BIOS calls having to do with character I/O are fully implemented. If you've never heard of a BIOS jump table, ignore this section.

Built-In Commands

Here is a summary of the available built-in commands under upm. Some of these are also discussed at length elsewhere in this document.

Note that character case is insignificant under CP/M but matters to Micronix. In the following table, CP/M-ish things are given in upper case; Micronix-ish things in lower case.

= Displays drive-directory correspondences.

!command Run a Micronix command. Any valid shell command may appear after the exclamation point.

ERA Erases specified files.

DIR Displays file names in the directory.

exit Leave upm. This is the only way to get out of upm INTERACTIVE mode (short of killing the process from another terminal).

REN Renames a specified file. Exactly as in standard CP/M. REN new = old.

SAVE Save memory contents in a file.
UNIMPLEMENTED.
If you want to patch files, see the Micronix utility ddt (in Section 1 of the Micronix Reference Manual).

TYPE Types the contents of a file.

USER Move to another area within the same directory.
UNIMPLEMENTED.
All files are considered to be "USER 0" for the purposes of CP/M programs. The far (1) utility can be used to access files with any user number on a CP/M floppy.

A: Select a drive.

A:/directory
Set a new drive-directory correspondence.

LST:file
Send LST: output to a file Creat the file if it didn't exist.

LST:tty
Send LST: directly to another tty. (X-on X-off handshaking will be used by default.)

LST:tty,speed
Send LST: directly to a tty and set the speed. (Again X-on, X-off handshaking used by default.)

LST:file,speed,cts
Send LST: output to a file and set the speed and CTS handshaking. Output is inhibited until the CTS line (pin-8 on the Wunderbuss or pin-5 on the Mult/IO) is driven **TRUE**.

LST:|program
Send (pipe) the LST: output to another program. Note that if this command is entered from the shell, the command line must be in quotation marks or else the shell will interpret the pipe incorrectly; i.e: entering

```
upm lst:|lpr
```

will be parsed into two commands by the shell. You can convey the intended meaning by enclosing the upm LST redirection argument in quotes, thusly:

```
upm "lst:|lpr"
```

LST: Disconnect LST:. LST: reverts to the default (characters appear on your screen).

Control characters

The control characters discussed below are effective only when the "standard" CP/M console I/O calls are used. They will not be understood if the CP/M BIOS jumps or CP/M direct I/O calls are used to generate the output or request input.

^P - printer toggle.

Normally console output is sent only to the console. If you type ^P, console output will be sent to both the LST device and the console. If you type ^P again, console output is once again sent only to the console.

^S - suspend console output.

If, while console output is occurring, a ^S is received, it is interpreted as an instruction to temporarily suspend further output. When in suspended mode, any character will restart the output, except ^C which will cause upm to exit immediately.

^C - abort.

A ^C as the first character of an input line will cause an exit from upm.

^R - retype current input line.

Submit files.

Standard CP/M submit file format is implemented. IMPORTANT: Submit file operation works only in upm's INTERACTIVE mode. From the user's point of view, operation under upm is identical to operation under CPM.

Here is the way submit files work internally, under both CPM and upm. Please note that that this is OPTIONAL TECHNICAL material, needed only by programers with special requirements. We repeat it here only because it is very hard to find this in the CPM documentation.

The submit file is named "A:$$$.SUB"

Upm checks for the presence of a submit file just before issuing its prompt ("X>"). A submit file consists of a series of records. Each record represents one CP/M command. Each record is 128 bytes in length. The records are in reverse order within the submit file. Last record first, first record last, etc.

The first byte of each record gives the length of the command line found in the following 127 bytes.

The following 127 bytes contain the CP/M command exactly as one would enter it from the console.

Standard CP/M reduces the length of its submit file by one record upon executing each command.

Upm doesn't actually change the size of the file, but keeps track of where it is.

Text Files

Herein are some practical suggestions on the handling of text files and the successful transportation thereof between CP/M and Micronix. Also included is a description of the differences in their formats.

CP/M text files and Micronix text files are identical, except for the following:

1. At the end of each line of a Micronix text file there is a single NEWLINE character; at the end of each line of a CP/M text file there is a RETURN character followed by a NEWLINE character.

2. A CP/M file's length must be an even multiple of 128 bytes. A Micronix file's length may be any number of bytes (both are constrained to maximum file sizes). CP/M text files are padded with ^Z characters at the end to round them out to an even multiple of 128. Micronix text files are not.

There is a convenient utility for converting a text file back and forth between the two formats. (See **clean** in Section 1 of the Micronix Reference Manual.) To make a text file suitable for Micronix, enter:

```
% clean filename
```

To make a text file suitable for CP/M or **upm**, enter:

```
% clean -u filename
```

The name **clean** comes from the idea of cleaning out the excess RETURN characters and ^Z characters at the end; "-u" stands for "unclean".

Upm and the Background

Non-interactive CP/M programs may be run in the background. The asm assembler is an example of a non-interactive program. Running something in the background means you don't have to wait for it to finish, but are able to do something else on the system while waiting. See also the section on background processes in "sh" in section 1 of the Micronix reference manual. It is also possible to run a non-interactive CP/M program in the background at reduced priority so that the subjective response time is not as adversely affected.

```
% asm program &
```

Would cause the named program's assembly proccess to be initiated, but you would immediately regain control of the console. Note that this command must be issued from the shell.

```
% nice 5 asm program &
```

Would cause the same action as above but at a reduced priority level. (The default "nice" is 0.)

Trace Mode

Upm incorporates a trace mode for use in tracking down trouble, or to analyze CP/M system call usage (or for any other reason which comes up).

To run upm in "trace mode", type:

```
% upm -v
```

(The -v stands for verbose.) Each CP/M call made by the CP/M program running under emulation will be displayed on your screen as a single character.

Below is a table of the CP/M calls and the associated characters.

CPM call name	CPM call number	Upm trace letter	Description
RESET	0	A	system reset
CONIN	1	B	console input
CONOUT	3	D	console output
READER	3	D	reader input
PUNCH	4	E	punch output
LIST	5	F	list output
DIRECTIO	6	G	direct I/O
GETIO	7	H	get I/O byte
SETIO	8	I	set I/O byte
PRINTSTRING	9	J	print string
READBUF	10	K	read console buffer
CONSTAT	11	L	get console status
VERSION	12	M	return version number
DISKRESET	13	N	reset disk system
SELECTDISK	14	O	select disk
OPEN	15	P	open file
CLOSE	16	Q	close file
FIRST	17	R	search for first
NEXT	18	S	search for next
DELETE	19	T	delete file
READSEQ	20	U	read sequentially
WRITESEQ	21	V	write sequentially
MAKE	22	W	make file
RENAME	23	X	rename file
LOGIN	24	Y	return login vector
CURDISK	25	Z	return current disk
DMA	26	[	setD MA address
GETADDR	27	\	get addr (alloc)
PROTECT	28	]	write protect disk
GETRO	29	^	get read/only vector
SETFILE	30	_	set file attributes
PARMS	31	`	get addr (disk parms)

USERCODE	32	a	set / get user code
READRANDOM	33	b	read randomly
WRITERANDOM	34	c	write randomly
SIZE	35	d	compute file size
SETRANDOM	36	e	set random record

Exiting upm

You may leave INTERACTIVE mode by typing "exit" as a command.

Exit from upm's DIRECT mode occurs immediately upon completion of the called application program.

The appearance is that the CP/M program was run directly under Micronix and then returned directly to Micronix.

If a program "bombs, freezes up, locks up, crashes," etc., what can you do? (Sometimes you can restore control by hitting DELETE or RUBOUT, but if you are running under upm, that will not work.) One feature of Micronix is that a single crashed program cannot bring the whole system down. The system manager can "kill" the abberant program from another terminal. Here is what the system manager should do:

Remember that each terminal is called tty-something, such as ttyA, ttyB, etc. Let's say that the crashed program is running on ttyB. Log in as "root" on another terminal, and type the command "ps a". This will give the Process Status of tasks running on the system. The "a" means "tell me about All the tasks, not just the ones belonging by this terminal". A list of tasks will appear, including those belonging to ttyB. Note the numbers of all the tasks associated with ttyB. You want to kill all of them (not just the ones called "upm"). Say that tasks number 4, 6, and 7 are listed as belonging to ttyB. You should type

```
# kill 4 6 7
```

The # is the super-user shell prompt, not something you type. Unless you have logged in as the super-user (who is usually called "root") you won't be able to kill someone else's programs. This will terminate the offending program, and will also log the user out. The user can then log back in and proceed.

BUGS

Here is a collation of all the sort of things which are <u>not</u> likely to work under upm:

8080 Instructions:

hlt, in, out

Hardware specific programs:

Hard disk formatters.
Floppy disk formatters.
Modem programs.

CP/M specific programs:

MOVCPM, SYSGEN

CP/M specific calls (the CP/M call 27 returns nonsense, because there is no allocation bit map, but this is often harmless.)

SEE ALSO

Upm tutorial in User Reference Manual

CP/M 2.2 Operating System Manual
by Digital Research, (C) 1976, 1977, 1978

clean (1), far (1), ls (1), chmod (1), ps (1), mv (1)

CREDITS

upm is a product of the Morrow Designs software group.

NAME

wall - write to all users

SYNTAX

wall

DESCRIPTION

Wall reads the standard input up to an end of file, then sends this to the terminal of each currently logged in user. You can enter

```
wall [return]
message...
more message...
^D [control-D]
```

to broadcast a message with your terminal as the standard input. Alternatively, you can broadcast the contents of a particular file by entering

```
wall < filename
```

Its normal use is to inform users of immediate system status. Only the super-user may use wall.

EXAMPLE

```
wall
System going down in 3 minutes -- Please prepare
^D
```

After which, each user would see:

```
Broadcast message ...
```

followed by what appears above.

SEE ALSO

su (1) for entering superuser mode.

NAME

who

SYNTAX

who

DESCRIPTION

Who prints a list of the currently logged-in users of Micronix.

FILES

/etc/utmp (current user information)

NAME

words - count words in standard input or files

SYNTAX

words

DESCRIPTION

The number of words in each of the named files is printed alongside the file name. If no file names are specified, or if "-" is given as a file name, the standard input is read up to an end-of-file and the number of words encountered is printed. A word is defined as a contiguous string of non-blanks separated by blanks characters. A blank is a space, tab, or newline character.

EXAMPLE

To count the words in a number of files:

```
words file1 file2 file3
```

SEE ALSO

chars (1), lines (1)

Micronix Operating System reference manual

system entries

MORROW

CONTENTS OF REFERENCE SECTION 2: SYSTEM ENTRIES

INTRODUCTION TO SYSTEM CALLS

This section of the manual lists all Micronix system calls. The syntax for these calls is in most cases identical with that for Unix Version 6 system calls, differing only in the assembly language register names. For most purposes, Section II of the Unix V6 Programmer's Manual can be used as a reference in place of this one.

Call syntax for both C and assembly language is given. Familiarity with one of these languages is necessary to understand this section.

Many system calls take a file name. This can be an arbitrary path name, of unlimited length, but it must be terminated by a 0-byte, and the calling process must have search permission on all directories in the path.

An error during a system call never causes an automatic error message, but an error number is always available (see below), and the program can request a standard error message if it wants [see perror (3)].

C System Calls

The type of declarations given for system call arguments are not meant to be copied literally; they are simply indications of how the system will treat the arguments. A string, for example, is always acceptable in place of a (char *) argument. When no declaration is given, an int is expected.

From C, an error is indicated by an otherwise impossible return value. This is usually -1; the individual sections specify the details. The error number is always left in the external variable, errno. This variable is not cleared on successful calls, so that it remains available across message writes, etc.

Assembly Language System Calls

Syntax is given for A-Natural (Whitesmith's) assembly language. Translation to Intel or Zilog mneumonics is straightforward. For example, the **chmod** system call in A-Natural is

```
sys; chmod; name; mode
```

In Intel assembly language, this looks like

```
db sys
db chmod
dw name
dw mode
```

The "sys" byte that begins each system call is a synonym for "restart 1". The names of the calls themselves are synonyms for 1-byte quantities that can be found in the individual sections, or by inspecting or including the files /include/an.h and /include/intel.h. Succeeding in-line arguments, if any, are always 2-byte quantities. A "name" argument is always a pointer to a null-terminated string.

Registers are preserved across system calls, except for the error flag and error number (see below), and for any return values (see the individual sections). For the sake of portability, however, it is unwise to depend on this.

From assembly language, an error is always indicated by turning on the carry bit of the condition flags. Thus a system call can be followed by a jc (jump on carry) to handle an error return, or a jnc (jump on no carry) to handle an error-free return. The error number is returned in register hl.

Error Numbers

Following is a list of error numbers, their names in the system source code, the standard error message available thru **perror**, and a short explanation.

0 (not used)

1 EPERM **Not super user** The current user was attempting something reserved for the super-user.

2 ENOENT **No such file or directory**
The last component of a path name does not exist.

3 ESRCH **No such process**
The process whose number was given to **signal** does not exist, or is already dead.

4 EINTR **Interrupted system call**
A slow system call (such as read or write on a tty, but never on a disk) was interrupted by a caught signal.

5 EIO I/O error
A physical I/O error occurred during a read or write.

6 ENXIO No such device or address
An I/O call was made to a non-existent device, or beyond the limits of the device.

7 E2BIG Arg list too long
A call to exec was made with an argument list longer than 512 bytes.

8 ENOEXEC Exec format error
An attempt was made to execute an inappropriate file (such as a directory or special file), even though the user had execute permission on the file.

9 EBADF Bad file number
A file descriptor refers to no open file, or a read or write request is made to a file not opened for that activity.

10 ECHILD No children
There were no child processes to wait for.

11 EAGAIN No more processes
Fork found the system process table temporarily full.

12 ENOMEM Not enough core
During an exec or break, the program asked for more core than the system could supply. This is not a temporary condition: the maximum program size is a system parameter.

13 EACCES Permission denied
The user attempted a file access forbidden by the file's permission bits.

14 ESYS System error
The system has detected an "impossible" error, caused by corrupted disk, memory, or a system bug, but not yet severe enough to warrant a panic stop. This should be reported.

15 ENOTBLK Block device required
A plain file or a directory was given where a block device was required, (in mount, for example).

16 EBUSY Device busy
An attempt was made to mount a device that was already mounted, to unmount a device on which there was an open file or a current directory, or to open a one-user-only device that is already busy.

17 EEXIST File exists
The new file name passed to **link** or **mknod** is already taken.

18 EXDEV Cross-device link
A link to a file on another device was attempted.

19 ENODEV No such device
An attempt was made to open a non-existent device.

20 ENOTDIR Not a directory
A non-directory was specified where a directory was required, as an intermediate component in a path name, for example, or as an argument to **chdir**

21 EISDIR Is a directory
An attempt was made to write on a directory.

22 EINVAL Invalid argument
An invalid argument was given to a system call: currently, unmounting a non-mounted device, mentioning an unknown signal in **signal**, or invoking a non-existing system call. Ordinarily, the latter case generates a fatal signal; the error is returned only if the signal is caught or ignored.

23 ENFILE File table overflow
The system's table of open files is temporarily full.

24 EMFILE Too many open files
Only 16 files can be open per process

25 ENOTTY Not a typewriter
The file mentioned to **stty** or **gtty** is not a typewriter or other character-device.

26 ETXTBSY Text file busy
Not currently used.

27 EFBIG File too big
The user attempted to extend a file beyond the maximum size of 2^24 bytes.

28 ENOSPC No space left on device
The disk's free space has been exhausted.

29 ESPIPE Illegal seek
A seek was issued to a pipe.

30 EROFS Read-only file system
An attempt was made to write to a device mounted read-only.

31 EMLINK Too many links
The program attempted to make a 256th link to a file.

32 EPIPE Broken pipe
The program wrote on a pipe that had no reader. Ordinarily this generates a fatal signal; the error is returned if the signal is caught or ignored.

NAME

access - test file access permissions

C

```
access(name, mode)
char *name;
int mode;
```

FUNCTION

Permission to access the named file, in the specified mode, is tested. The test is based on the real user and group IDs, rather than the effective IDs, so that a set-user-id program may test the permissions of its invoker. Mode is the sum of any of the following:

```
4  read
2  write
1  execute
```

RETURNS

Access returns a 0 if permission is granted, -1 if not.

ASSEMBLER

```
(access = 33)
sys; access; name; mode
```

RETURNS

Access returns with the error bit clear if the permissions are granted, or with the error bit set if they are not.

NAME

alarm - set the alarm clock

C

alarm(seconds)

FUNCTION

This call sets the "alarm clock", which will send an **alarm** signal to the calling process after the given number of seconds (see **signal (2)**). Meanwhile, the process continues to run.

An argument of 0 turns the alarm clock off.

See also pause (2) and sleep(2).

RETURNS

If an alarm was previously set but not yet triggered, **alarm** returns the number of seconds remaining until the old alarm. Otherwise it returns 0.

ASSEMBLER

```
alarm = 27)
(seconds in hl)
sys; alarm
```

RETURNS

The time remaining on any old alarm is returned in hl. There is no error return.

NAME

break, brk, sbrk - change core allocation

C

```
char * brk(addr)
char * sbrk(addr)
```

FUNCTION

Brk sets the system's idea of the lowest memory location not used by the program (called the "break") to addr (which may be rounded up to suit memory management hardware). The old break is returned.

Locations greater than or equal to the break and less than the stack pointer are not in the address space of the program. These locations may be overwritten by the system, and may cause memory faults if accessed. The system will refuse to set the break above the stack pointer, but a program is free to move its stack pointer below the break.

Sbrk adds incr more bytes to the program's data space and returns a pointer to the start of the new area.

When a program begins execution via **exec** (2), its break is set to the top of its program and data storage areas. Thus, only programs that must dynamically allocate data space need to use these calls.

RETURNS

The old break is returned on success. A -1 is returned if the request would require more memory than the system limit, or if it would move the break across the stack pointer.

ASSEMBLER

```
(break = 17)
sys; break; addr
```

RETURNS

Only success or failure is returned, by clearing or setting the carry flag, respectively.

NAME

chdir - change working directory

C

```
chdir(dirname)
char *dirname;
```

FUNCTION

The working directory of the current process is changed to the given directory. The user must have search (execute) permission on the directory.

RETURNS

A 0 is returned on success, a -1 on failure.

ASSEMBLER

```
(chdir = 12)
sys; chdir; dirname
```

RETURNS

The carry flag is cleared on success, or set on failure.

NAME

chmod - change mode of file

C

```
chmod(name, mode)
char *name;
```

FUNCTION

The mode of the file is set as indicated. Only the owner of a file, or the super-user, may change the mode. Modes are constructed by ORing together some combination of the following (octal) values:

4000	Set user id on execution
2000	Set group id on execution
1000	Currently ignored
0400	Read by owner
0200	Write by owner
0100	Execute (or search directory) by owner
0070	Read, write, execute (search) by group
0007	Read, write, execute (search) by others

When a file with the set-user-id flag is executed, the effective user ID of the process is set to the owner of the file. (In this way, for instance, a spelling program could use a secret dictionary.)

For security reasons, whenever a file is opened for writing, its set-id flags are cleared.

RETURNS

A 0 indicates success, a -1 indicates failure.

ASSEMBLER

```
(chmod = 15)
sys; chmod; name; mode
```

RETURNS

The carry flag is cleared on success, or set on failure.

NAME

chown - change owner and group of a file

C

```
chown(name, owner)
char *name;
```

FUNCTION

The owner of the file is changed to the low byte of "owner", and the group is changed to the high byte. Only the super-user may execute this call.

RETURNS

A 0 indicates success, a -1 indicates failure.

ASSEMBLER

```
(chown = 16)
sys; chown; name; owner
```

RETURNS

The carry flag is cleared on success, or set on failure.

NAME

close - close a file

C

close(descriptor)

FUNCTION

Given a file descriptor previously returned by **open**, **creat**, or **pipe**, **close** closes the associated file. A close of all files is automatic on **exit**, but since processes are limited to 16 simultaneously open files, close () may be necessary for programs that deal with many files.

RETURNS

A 0 indicates success, a -1 indicates failure.

ASSEMBLER

(close = 6)
(file descriptor in hl)
sys; close

RETURNS

The carry flag is cleared on success, or set on failure.

NAME

creat - create a new file (or truncate an old one)

C

```
creat(name, mode)
char *name;
```

FUNCTION

If the file does not exist, and if the parent directory is writable, it is created with the given mode. If the file does exist and is writable, it is truncated to 0 length, and its mode and owner remain unchanged. In either case, the file is opened for writing only, and a file descriptor is returned.

See chmod (2) for the construction of modes. Note that the file is opened for writing, even if the given mode does not allow writing. This can be used as a locking mechanism, in that another program will not be able to create the same file.

Creat is subject to the limit of 16 open files per process.

RETURNS

Creat() returns a file descriptor (a small, positive number) on success, or a -1 on failure.

ASSEMBLER

```
(creat = 8)
sys; creat; name; mode
(file descriptor in hl)
```

RETURNS

The carry flag is cleared on success, or set on failure. If successful, a file descriptor is returned in hl.

NAME

csw - read the console switches

C

getcsw()

FUNCTION

The setting of the console switches is returned.

RETURNS

The setting of the console switches is returned.

ASSEMBLER

(csw = 38)
sys; csw
(switch setting in hl)

RETURNS

The switch setting is returned in hl.

NAME

dup - duplicate a file descriptor

C

```
dup(descriptor)
```

FUNCTION

Dup takes a file descriptor previously returned by **open**, **creat**, or **pipe** and allocates a new descriptor synonymous with the original. Subsequent reads or writes with the new descriptor will have exactly the same effect as the same call with the old descriptor.

Since the algorithm that allocates file descriptors returns the lowest available value, combinations of **dup** and **close** can be used to move file descriptors in a general way. This is used mostly for manipulating the standard input (file descriptor 0) and the standard output (file descriptor 1).

Dup is subject to the limit of 16 active file descriptors per process.

RETURNS

Dup() returns a file descriptor (positive) if successful, or a -1 if not.

ASSEMBLER

```
(dup = 41)
(old descriptor in hl)
sys; dup
(new descriptor in hl)
```

RETURNS

On success, the carry flag is cleared and the new descriptor is returned in register hl. On failure, the carry flag is set.

NAME

exec - execute a program

C

```
execv(name, argv)
char *name;
char *argv[];

execl(name, arg0, arg1, ..., argn, 0)
char *name, *arg0, *arg1, ..., *argn;
```

FUNCTION

Exec overlays the calling core image with the named file, then transfers to the beginning of the new core image. There can be no return from a successful exec: the calling core image is lost.

Exec does not create a new process - the same process continues with the new core image. Previously opened files remain open (so standard input and output are preserved), and ignored signals remain ignored. Caught signals, however, are reset to their default behavior.

Each process has "real" user and group IDs and "effective" user and group IDs. The real IDs identify the user; the effective IDs determine the access privileges. If a file does not have "set-user-id" or "set-group-id" mode [see **chmod**, (2)], then **exec** sets all IDs, real and effective, to the individual executing the file. If the file does have either mode, **exec** sets the corresponding effective ID to the owner of the executed file. The real IDs remain unchanged. This allows a user to write a program that takes advantage of his own access privileges (rather than its invoker's privileges).

In order to be executed, a file must have one of the execute permission bits set, even for the super-user. The system expects one of two file formats. If the first byte is hex 99, then the first 16 bytes are taken to be a header with the following structure

```
struct header
        {
        char     ident,          /* hex 99 */
                 conf;           /* not used by exec */
        unsigned tablsize,       /* not used by exec */
                 textsize,       /* bytes in text segment */
                 datasize,       /* bytes in data segment */
                 bss_size,       /* bytes in bss segment */
                 heapsize,       /* minimum stack + heap */
                 textoff,        /* text segment offset */
                 dataoff;        /* data segment offset */
        };
```

(This header structure is produced by Whitesmith's compilers.) The rest of the file is assumed to contain textsize bytes of text, followed by datasize bytes of data. Text is loaded at address textoff, data is loaded at address dataoff, and bss space is allocated following the data. The break is set at the highest location in the text or data + bss segments.

If the first byte of the file is not hex 99, then the file is taken to be pure object code origined at address 256 (100 hex). The brake is set to the file size.

In any case, the program must meet the maximum-size restriction (currently 65024 bytes), and it must keep a "halt" instruction (the system-call trap) at address 8. (This is supplied by the system, unless the text offset is <= 8.)

C offers two different interfaces to **exec. Execl** is useful when a known file is being executed with known arguments. The arguments are all string pointers. Any number of arguments may be given, but the last must be a 0.

When the number of arguments is not known in advance, **execv** is handy. Pointers to the argument strings are collected into a list, a null pointer is appended to mark the end of the list, and **execv** is called with the address of the list.

Currently, the total number of bytes in the argument strings (including the terminating nulls) is limited to 512.

It is conventional to repeat the name of the file being executed as the first argument, so that programs can use the name with which they are invoked.

When a C program is executed, it begins as follows:

```
main(argc, argv)
        int argc;
        char **argv;
```

where **argc** is the argument count, and **argv** is a list of pointers to the argument strings themselves. Conventionally, **argc** is at least 1 and **argv[0]** is the program name. As delivered by the system, argv[argc] == -1, so that **argv** cannot be used directly in another **execv** until argv[argc] is set to 0.

RETURNS

Any return is an error return. In that case, the calling image is not lost and can continue. Possible errors include: the file cannot be found, is not an ordinary file, is not executable, is too big, or the argument list is too long.

ASSEMBLER

(exec = 11)
sys; exec; name; argv

Argv is the address of a list of string pointers. The last pointer must be 0. When the file starts execution, core is set up as follows:

```
(top of user core)
argn:   string\0
        ...
arg0:   string\0
        -1
        argn
        ...
        arg0
sp ->   argc
```

RETURNS

If exec returns at all, the carry flag is set to indicate an error.

NAME

exit - terminate this process

C

```
exit(status)
```

FUNCTION

Exit closes all open files, terminates the calling process, and notifies the parent process (if it is executing a wait). The low byte of status is available to the parent (via wait).

RETURNS

This call can never return.

ASSEMBLER

```
(exit = 1)
(status in hl)
sys; exit
```

RETURNS

None.

NAME

fork - create a new process

C

fork()

FUNCTION

Fork is the only way to create a new process. The calling process splits into a "parent" and a "child". The child's core image is a copy of the parent's, open files are shared, and signals remain unchanged. Fork() returns a zero to the child process, while it returns a non-zero number to the parent. This is the process ID of the child, and is used by wait (2) and kill (2).

RETURNS

Fork() returns a zero to the child, and the non-zero process ID of the child to the parent. A return of -1 (not just negative) indicates a temporary lack of process space.

ASSEMBLER

```
(fork = 2)
sys; fork
(child return)
(+ 3 bytes)
(parent return) (child id in hl)
```

RETURNS

The child process returns to the location immediately following the fork. The parent skips three bytes before returning, and receives the child ID in hl. If a new process cannot be created, the carry flag is set on return to the parent (and the child return never happens).

NAME

fstat - get the status of an open file

C

```
fstat(descriptor, buf)
struct stat *buf;
```

FUNCTION

Fstat is identical to **stat**, except that it operates on open files (via the file descriptor) rather than on files given by name. This is often used to examine the status of the standard input and output, whose names are usually unknown. Buf is the address of a 36 byte buffer, into which the following information is placed:

```
struct stat
      {
char  minor,          /* minor device */
      major;          /* major device */
int   inumber,        /* inode number */
      flags;          /* see below */
char  nlinks,         /* number of links */
      uid,            /* user id of owner */
      gid,            /* group id of owner */
      size0;          /* high byte of size */
int   size1,          /* low word of size */
      sddr[8];        /* block numbers */
long  actime,         /* time of last access */
      modtime;        /* " last modification */
      };
```

The flags are as follows (values in octal):

```
 100000               inode is allocated
 060000               2-bit file type
       000000         plain file
       040000         directory
       020000         character-special file
       060000         block-special file
 010000               large file
 004000               set user-id on execution
 002000               set group-id on execution
 001000               currently ignored
 000400               read (owner)
 000200               write (owner)
 000100               execute or search directory (owner)
 000070               read, write, execute (group)
 000007               read, write, execute (others)
```

RETURNS

A 0 indicates success, a -1 indicates failure (bad file descriptor).

ASSEMBLER

```
(fstat = 28)
(file descriptor in hl)
sys; fstat; buf
```

RETURNS

The carry flag is cleared on success, or set on failure. On success, the status is placed in the 36-byte area pointed at by buf.

NAME

getpid - get process ID

C

```
getpid()
```

FUNCTION

Getpid returns the process ID of the current process. This is the same ID that is used by the fork, kill, and wait system calls.

RETURNS

Getpid returns the process ID.

ASSEMBLER

```
(getpid = 20)
sys; getpid
(id in hl)
```

RETURNS

Getpid returns the process ID in hl.

NAME

getuid - get user IDs

C

```
getuid()
```

FUNCTION

Getuid returns the real user IDs of the current process. The real group ID is in the high byte of the returned word. The real user ID is in the low byte. (The real IDs identify the individual, while the effective IDs determine his current access privileges. See the **exec** and **chmod** system calls for a discussion.)

RETURNS

Getuid returns the real user IDs:
high byte = group ID, low byte = user ID.

ASSEMBLER

```
(getuid = 24)
sys; getuid
(IDs in hl)
```

RETURNS

Getuid returns the real process IDs:
high byte = group ID, low byte = user ID.

NAME

gtty - get typewriter status

C

```
gtty(descriptor, vec)
struct
      {
      char    ispeed,          /* input speed */
              ospeed,          /* output speed */
              erase,           /* erase character */
              kill;            /* kill character */
      int     mode;            /* see below */
      }
      *vec;
```

FUNCTION

Gtty gets the status of the terminal associated with the file descriptor, and writes the status into the 6-byte structure pointed at by vec. (This structure is the same as that passed by **stty**.) Actually, this call may be made to any character device, but devices that do not like it may return an error.

Ispeed and ospeed are one of:

0	1200 baud
1	50 baud
2	75 baud
3	110 baud
4	134.5 baud
5	150 baud
6	200 baud
7	300 baud
8	600 baud
9	1200 baud
10	1800 baud
11	2400 baud
12	4800 baud
13	9600 baud
14	19200 baud
15	1200 baud

The next two characters specify the erase and kill characters, respectively. (The defaults are ^H and ^X).

The mode specifies what services are performed by the system on input and output. Currently, these are:

```
0200  Use the RS-232 clear-to-send line
0040  Raw input
0020  Cr -> lf mapping
0010  Echo input
0002  Expand tabs
```

In raw mode, all characters are passed immediately to the program without waiting for a full line to be typed, there is no erase or kill processing, and there is no recognition of any special control characters. In cooked mode, the following control characters are recognized:

```
^D  end-of-file (when typed alone on a line)
^\  send a quit signal to this tty's processes
DEL send an interrupt signal to same
^B  send a background signal to same
ESC freeze the output from this tty
^S  same as ESC
```

Note that even while the tty is in raw mode, the other modes still have an effect.

In cr->lf mode, input crs are turned into lfs, and output crs or lfs are turned into cr-lfs.

In echo mode, input is echoed immediately. If the mode is also raw, it is echoed exactly as typed; otherwise, the special control characters are not echoed, the kill character is echoed as itself plus a newline, and the erase character is echoed as backspace-space-backspace.

In expand-tabs mode, tabs are output as the number of spaces needed to bring the cursor to the next 0-mod-8 column.

RETURNS

A 0 indicates success, a -1 indicates failure. On success, the status is placed in the 6-byte structure pointed at by vec.

ASSEMBLER

```
(gtty = 32)
(file descriptor in hl)
sys; gtty; vec
```

RETURNS

The carry flag is cleared on success, or set on failure. On success, the status is placed in the 6-byte area pointed at by vec.

NAME

indir - indirect system call

ASSEMBLER

```
(indir = 0)
sys; indir; syscall
```

FUNCTION

The system call at the location syscall is executed. Execution resumes after the **indir** call.

The main purpose of **indir** is to allow programs to construct system calls in their data segments, avoiding modifications to their code.

The system call at syscall must begin with the "sys" byte. An **indir** executed indirectly is a no-op.

RETURNS

The returns depend on the indirectly executed call.

NAME

kill - signal a process

C

kill(pid, sig)

FUNCTION

Kill sends the signal, sig, to the process with the given ID. The usual effect is to kill the process - see signal (2) for a discussion and a list of signals.

The sending and receiving processes must have the same effective user IDs, or the sender must be the super-user.

If the given process ID is 0, then the signal is sent to all other processes with the same controlling tty.

A process can never kill itself.

RETURNS

A 0 indicates success, a -1 indicates failure.

ASSEMBLER

(kill = 37)
(process ID in hl)
sys; kill; sig

RETURNS

The carry flag is cleared on success, or set on failure.

NAME

link - link to a file

C

```
link(old, new)
char *old, *new;
```

FUNCTION

A link to "old" is created, with the name "new". Either name may be an arbitrary pathname.

"New" must not already exist, its directory must be writable, and it must be on the same device as "old". "Old" must not be a directory (unless the user is the super-user), and must not have more than 254 links.

RETURNS

A 0 indicates success, a -1 indicates failure.

ASSEMBLER

```
(link = 9)
sys; link; old; new
```

RETURNS

The carry flag is cleared on success, or set on failure.

NAME

mknod - make a directory or special file

C

```
mknod(name, mode, addr) char *name;
```

FUNCTION

Mknod creates a new file. Unlike **creat**, it may be used to create directories and special files; it does not truncate or open files.

The mode of the new file (including the file type bits) is taken from the mode argument, and the first address is taken from addr. For directories, this address should be 0, while for special files it should be the device number.

This call is restricted to the super-user.

RETURNS

A 0 indicates success, a -1 indicates failure.

ASSEMBLER

```
(mknod = 14)
sys; mknod; name; mode; addr
```

RETURNS

The carry flag is cleared on success, or set on failure.

NAME

mount - mount a file system

C

```
mount(device, on, ronly)
char *device, *on;
```

FUNCTION

Mount informs the system that the given block device contains a file system. Subsequent references to the file "on" will refer to the root directory of the new file system. The old contents of "on" are inaccessible until the device is unmounted.

If ronly is non-zero, the system will not allow writing on the device. If the device is physically write protected, it should still be mounted read-only to prevent the system from trying to update access times.

This call is restricted to the super-user. There is a limit to the number of devices that can be mounted concurrently.

RETURNS

A 0 indicates success, a -1 indicates failure.

ASSEMBLER

```
(mount = 21)
sys; mount; device; on; ronly
```

RETURNS

The carry flag is cleared on success, or set on failure.

NAME

nice - set process priority

C

nice(arg)

FUNCTION

The "nice" of a process is the opposite of its intuitive "priority" -- the "nicer" it is, the less cpu time it hogs. This systen entry sets the nice of the calling process to the given argument. Nice values range from -128 to 127; the normal value is 0. Only the super-user can set a negative nice (== high priority). Long running background programs should be run at a positive nice as a favor to other users.

A process' nice is passed to its children via the fork system call.

RETURNS

A 0 indicates success, a -1 indicates failure.

ASSEMBLER

(nice = 34)
(nice value in h1)
sys; nice;

RETURNS

The carry flag is cleared on success, or set on failure.

NAME

open - open a file for reading or writing

C

```
open(name, mode)
char *name;
```

FUNCTION

The named file is opened for

reading	(mode 0),
writing	(mode 1), or
both	(mode 2).

The returned file descriptor should be saved for subsequent calls to read, write, and close.

There is a limit of 16 open files per process.

RETURNS

A non-negative return indicates success, a -1 indicates failure. On success, a file descriptor (small non-negative number) is returned.

ASSEMBLER

```
(open = 5)
sys; open; name; mode
(file descriptor in hl)
```

RETURNS

The carry flag is cleared on success, or set on failure. On success, a file descriptor is returned in hl.

NAME

pipe - create an inter-process channel

C

```
pipe(descriptor)
int descriptor[2];
(read descriptor = descriptor[0])
(write descriptor = descriptor[1])
```

FUNCTION

Pipe returns two file descriptors that can be used to communicate between processes created by subsequent fork calls. When the pipe is written using descriptor[1], up to 4096 bytes of data will be buffered before the writing process is suspended. A read using descriptor[0] will pick up the data.

Read calls on an empty pipe with no writers will return an end-of-file (i.e., 0 bytes read). Write calls under similar conditions will generate a signal.

RETURNS

A 0 indicates success, a -1 indicates failure.

ASSEMBLER

```
(pipe = 42)
sys; pipe
(read descriptor in hl)
(write descriptor in de)
```

RETURNS

The carry flag is cleared on success, or set on failure.

NAME

read - read a file

C

```
read(descriptor, buffer, nbytes)
char buffer[];
```

FUNCTION

A file descriptor is a word returned from a successful **open**, **creat**, **dup**, or **pipe** call. Buffer is a memory location where at most nbytes of data will be placed. The number of bytes actually read is returned. This may well be less than nbytes; a read on a terminal, for example, will return at most one line. If the value 0 is returned, then the file has been exhausted.

RETURNS

A - 1 indicates an error. A return of 0 indicates end-of-file. Any other value indicates a successful read of that many bytes.

ASSEMBLER

```
(read = 3)
(file descriptor in hl)
sys; read; buffer; nbytes
(number of bytes read in hl)
```

RETURNS

The carry flag is cleared on success, or set on failure.

NAME

seek - move read/write pointer

C

```
seek(descriptor, offset, from)
```

FUNCTION

Each open file has an associated read/write pointer. Seek moves this pointer as follows:

- If from is 0, the pointer is set to offset.
- If from is 1, the pointer is set to its current location plus offset.
- If from is 2, the pointer is set to the size of the file plus offset.
- If from is 3, 4, or 5, the meaning is the same as for 0, 1, or 2, except that offset is multiplied by 512.
- If from is 0 or 3, offset is treated as signed. Otherwise it is unsigned.

IN SUMMARY:

Byte seeks	Block seeks	Motion is relative to
0	3	Beginning of file (unsigned)
1	4	Current location (signed)
2	5	End of file (signed)

Seeks are not allowed on pipes, but are allowed on character devices, although most such devices ignore them. Seeking far past the end of a file and writing will create a "hole" in the file that occupies no space. Reading the hole will allocate zero-filled space.

RETURNS

A 0 indicates success, a -1 indicates failure.

NAME

setuid - set process user IDs

C

```
setuid(uid)
```

FUNCTION

This call sets the real group and user IDs and the effective group and user IDs. The group IDs are set to the high byte of the argument, and the user IDs are set to the low byte. This call can be used to set the effective IDs to the real IDs. Only the super-user is permitted to change the real IDs.

RETURNS

A 0 indicates success, a -1 indicates failure.

ASSEMBLER

```
(setuid = 23)
(uid in hl)
sys; setuid;
```

RETURNS

The carry flag is cleared on success, or set on failure.

NAME

signal - set disposition of signals

C

```
signal(sig, func)
int (*func)();
```

FUNCTION

A signal is a means of notifying a process of some external event. A signal can be sent from the system (in response to an attempt to execute an illegal instruction, for example), from a terminal (to abort a process), or at the request of another process (via the kill system call). Normally, a signal causes termination of the receiving process, but this call allows the process either to ignore it or to "catch" it via an interrupt to a specified location. There are 15 signals:

1	-- (hangup)
2	interrupt (caused by typing DEL)
3	quit (caused by typing control \) *
4	illegal instruction *
5	-- (trace trap)
6	background (caused by typing control b)
7	record available at terminal **
8	-- (floating point exception)
9	kill (cannot be caught or ignored)
10	-- (bus error)
11	-- (memory fault)
12	bad argument to a system call *
13	write to a pipe with no one to read it
14	-- (alarm clock)
15	terminate (catchable kill)

* causes a core dump unless caught or ignored

** not reset when caught

Signals marked "--" are not currently sent by the system. (The Unix V6 descriptions are given).

A "func" of 0 tells the system to institute the default action for the given signal. For all signals but 6 and 7, this means process termination on receipt of the signal. For signal 6, the default action is to put the process into the "background" (all further signals, except 9, will be ignored, and reads on a terminal will return eof). For signal 7, the default action is to ignore the signal.

If "func" is 1, the signal will be ignored. If "func" is any value greater than 1, then it is taken as an address, and receipt of the signal will cause an interrupt to that address. Except as noted, a signal is reset to 0 after being caught. So in general, the catching routine must issue another signal call if it wants to continue catching the signal.

If a signal is caught during a slow system call (read or write to a terminal, sleep , or wait), the call will terminate prematurely and return an error (number EINTR - see the Introduction).

After a fork, the child inherits all signal dispositions. Exec passes on all default and ignore dispositions, but resets all caught signals to default action.

RETURNS

The return value is the old signal disposition. A 0 indicates success, a -1 indicates failure (signal number out of range).

ASSEMBLER

(signal = 48)
sys; signal; sig; func;
(old value in hl)

RETURNS

The carry flag is cleared on success, or set on failure.

NAME

sleep - stop execution for an interval

C

```
sleep(seconds)
```

FUNCTION

The calling process is suspended for at least the given number of seconds.

RETURNS

The call returns after the given number of seconds. A 0 indicates success, a -1 indicates failure. The only cause of failure is an early return caused by a caught signal.

ASSEMBLER

```
sleep = 35)
(seconds in hl)
sys; sleep
```

RETURNS

The carry flag is cleared on success, or set on failure. The only cause of failure is an early return caused by a caught signal.

NAME

stat - get the status of a named file

C

```
stat(name, buf)
char *name;
struct stat *buf;
```

FUNCTION

Stat is identical to **fstat**, except that it operates on named files rather than open-file descriptors. Buf is the address of a 36 byte buffer, into which the following information is placed:

```
struct stat
      {
char  minor,          /* minor device */
      major;          /* major device */
int   inumber,        /* inode number */
      flags;          /* see below */
char  nlinks,         /* number of links */
      uid,            /* user id of owner */
      gid,            /* group id of owner */
      size0;          /* high byte of size */
int   size1,          /* low word of size */
      addr[8];        /* block numbers */
long  actime,         /* time of last access */
      modtime;        /* " last modification */
      };
```

The flags are as follows (values in octal):

100000	inode is allocated
060000	2-bit file type
000000	plain file
040000	directory
020000	character-special file
060000	block-special file
010000	large file
004000	set user-id on execution
002000	set group-id on execution
001000	currently ignored
000400	read (owner)
000200	write (owner)
000100	execute or search directory (owner)
000070	read, write, execute (group)
000007	read, write, execute (others)

It is not necessary to have read permission on the file, but all directories leading to the file must be searchable.

RETURNS

A 0 indicates success, a -1 indicates failure.

ASSEMBLER

```
(stat = 18)
stat; name; buf
```

RETURNS

The carry flag is cleared on success, or set on failure. On success, the status is placed in the 36-byte area pointed at by buf.

NAME

stime - set date and time

C

```
stime(time)
long * time;
```

FUNCTION

Stime sets the system's idea of the date and time. The argument is the number of seconds since 0:00 GMT, January 1, 1970. Only the super-user may make this call.

RETURNS

A 0 indicates success, a -1 indicates failure (not super-user).

ASSEMBLER

```
(stime = 25)
(time in hl-de: hl = high word, de = low word)
sys; stime
```

RETURNS

The carry flag is cleared on success, or set on failure.

NAME

stty - set typewriter status

C

```
stty(descriptor, vec)
struct
      {
      char    ispeed,          /* input speed */
              ospeed,          /* output speed */
              erase,           /* erase character */
              kill;            /* kill character */
      int     mode;            /* see below */
      }
      *vec;
```

FUNCTION

Stty sets the status of the terminal associated with the file descriptor. The status is taken from a 6-byte structure pointed at by vec. This structure is the same as that obtained by gtty. Actually, this call may be made to any character device, but devices that do not like it may return an error.

Ispeed and ospeed are one of:

0	1200 baud
1	50 baud
2	75 baud
3	110 baud
4	134.5 baud
5	150 baud
6	200 baud
7	300 baud
8	600 baud
9	1200 baud
10	1800 baud
11	2400 baud
12	4800 baud
13	9600 baud
14	19200 baud
15	1200 baud

The next two characters specify the erase and kill characters, respectively.

The mode specifies what services are performed by the system on input and output. Currently, these are:

```
0200  Use the RS-232 clear-to-send line
0040  Raw input
0020  Cr -> lf mapping
0010  Echo input
0002  Expand tabs
```

In raw mode, all characters are passed immediately to the program without waiting for a full line to be typed, there is no erase or kill processing, and there is no recognition of any special control characters. In cooked mode, the following control characters are recognized:

```
^D  end-of-file (when typed alone on a line)
^\  send a quit signal to this tty's processes
DEL send an interrupt signal to same
^B  send a background signal to same
ESC freeze the output from this tty
^S  same as ESC
```

Note that even while the tty is in raw mode, the other modes still have an effect.

In cr->lf mode, input crs are turned into lfs, and output crs or lfs are turned into cr-lfs.

In echo mode, input is echoed immediately. If the mode is also raw, it is echoed exactly as typed; otherwise, the special control characters are not echoed, the kill character is echoed as itself plus a newline, and the erase character is echoed as backspace-space-backspace.

In expand-tabs mode, tabs are output as the number of spaces needed to bring the cursor to the next 0-mod-8 column.

RETURNS

A 0 indicates success, a -1 indicates failure (file descriptor does not refer to a character device).

ASSEMBLER

```
(stty = 31)
(file descriptor in hl)
sys; stty; vec
```

RETURNS

The carry flag is cleared on success, or set on failure.

NAME

sync - update the disks

C

```
sync()
```

FUNCTION

Sync causes all information in core memory that should be on disk to be written out. This includes modified super-blocks, modified inodes, and delayed block I/O.

RETURNS

No information is returned. Sync always succeeds.

ASSEMBLER

```
(sync = 36)
sys; sync
```

RETURNS

No error indication is returned.

NAME

time - get date and time

C

```
time(tp)
long *tp;
```

FUNCTION

Time fills the long value pointed to by the argument with the number of seconds since 0:00 GMT January 1 1970.

RETURNS

The user-supplied long integer is filled in. There are no error conditions.

ASSEMBLER

```
(time = 13)
sys; time
(time in hl-de)
```

RETURNS

The time is returned with the high order word in hl and the low order word in de. There are no error conditions.

NAME

umount - dismount a file system

C

```
umount(device)
char *device;
```

FUNCTION

Umount tells the system that the given special file should no longer be treated as a file system. The file on which the device was mounted reverts to its ordinary interpretation.

Umount will return an error if there are still any active files on the mounted system.

This call is restricted to the super-user.

RETURNS

A 0 indicates success, a -1 indicates failure.

ASSEMBLER

```
(umount = 22)
sys; umount; device
```

RETURNS

The carry flag is cleared on success, or set on failure.

NAME

unlink - remove a directory entry

C

```
unlink(name)
char *name;
```

FUNCTION

Unlink removes the indicated entry from its directory. If this was the last link to the file, the file is removed and its space is freed. If the file was open in any process, this removal is delayed until the file is closed, even though its last directory entry has disappeared.

In order to unlink a file, a user must have write permission on its directory. Write permission is not required on the file itself. Only the super-user can unlink a directory.

RETURNS

A 0 indicates success, a -1 indicates failure.

ASSEMBLER

```
(unlink = 10)
sys; unlink; name
```

RETURNS

The carry flag is cleared on success, or set on failure.

NAME

wait - wait for a process to terminate

C

```
wait(pstat)
int *pstat;
```

FUNCTION

This call waits for the termination of any of the caller's children. If any child has died since the last wait, return is immediate. If there are no children, an error is returned. If there are several children, several wait calls are necessary to learn of all the deaths.

Wait returns the process ID of the terminated child, and fills in the user-supplied integer with the termination status. In the case of a normal termination via **exit** , the low byte of the status is 0, and the high byte is is the low byte of the child's exit argument. In the case of a signal-caused termination, the low byte of the status is the signal number, and the high byte is the child's l register.

If a parent process terminates without waiting for its children, the initialization process (process id = 1) inherits the children.

RETURNS

A positive number indicates success, a -1 indicates failure (no children). On success, the pointed-to status is filled in with the child's termination status, and the child's process ID is returned.

ASSEMBLER

```
(wait = 7)
sys; wait
(process id in hl)
(status in de)
```

RETURNS

The carry flag is cleared on success, or set on failure (no children). On success, the child's process ID is returned in hl, and its termination status is returned in de.

NAME

write - write to a file

C

```
write(descriptor, buf, nbytes)
char *buf;
```

FUNCTION

This call writes nbytes from the indicated buffer to the given open file. The number of bytes actually written is returned. Unlike the **read** call, this number should be the same as requested; otherwise, an error is indicated.

RETURNS

A -1 indicates an error (bad descriptor or physical IO error). Otherwise, the value nbytes is returned.

ASSEMBLER

```
(write = 4)
(descriptor in hl)
sys; write; buf; nbytes
```

RETURNS

The carry flag is cleared on success, or set on failure. The number of characters actually written is returned in hl.

Micronix Operating System reference manual

subroutines

MORROW

CONTENTS OF REFERENCE SECTION 3: SUBROUTINES

The Subroutines are part of the libraries included with Whitesmith's C and Pascal. If you haven't purchased C, you won't have any of these subroutines.

Subroutines with "(3s)" in their page headings are contained in the standard I/O library (/lib/libS.a).

There is more information about the libraries contained in the file /lib/readme.

SUBROUTINE(S)

alloc, free, realloc, calloc - main memory allocator

C

```
char *alloc(size)
unsigned size;

free(ptr)
char *ptr;

char *realloc(ptr, size)
char *ptr;
unsigned size;

char *calloc(nelem, elsize)
unsigned nelem, elsize;
```

FUNCTIONS

The routines **alloc** and **free** provide an easy-to-use memory allocation package. The **alloc** routine returns a pointer to a block of at least **size** bytes long, which begins on a word boundary.

The argument to **free** is a pointer to a block allocated previously by **alloc**. This block is available for further allocation; its contents are left undisturbed.

Alloc allocates the first large reach of free space it finds in its circular search from the last allocated or freed block, uniting neighboring free blocks as it searches. If no additional space is free, **alloc** calls **sbrk** (see section 2) for more system memory.

Block sizes pointed to by **ptr** are changed by **realloc** to **size** bytes. It may also return a pointer to a block, if the block is removed. The values of the new and old block sizes determine the changes in block contents; the contents remain unchanged up to the lesser value of the old and new sizes.

Alloc contains a valuable search function which may be used to do storage compaction. **Realloc** works if **ptr** points to a block that has been freed since the last call of **alloc**, realloc or calloc. Compounding sequences of **free**, **alloc** and realloc thereby augments **alloc**'s search strategy for storage compaction.

Allocation of space for an array of **nelem** elements of **elsize** size is done by **calloc**. Each of these routines returns a pointer to a space aligned for any type of storage.

ERRORS

Note that chaos results if space assigned by **alloc** is overrun, or if a random number is given to **free**.

A NULL pointer (0) is returned by **alloc**, **realloc** and **calloc** if no memory is available, or if a corrupt arena is detected that was caused by storage outside the bounds of a block. If user wishes to recompile the **alloc** routine to rigidly check the arena during every transaction, he/she is referred to the source code.

NOTES

The block pointed to by ptr may be destroyed if **realloc** returns a 0.

SUBROUTINE(S)

ctime, localtime, gmtime, asctime, timezone - convert time and date to ASCII

C

```
char *ctime(clock)
long *clock

#include <time.h>

struct tm *localtime(clock)
long *clock;

struct tm *gmtime(clock)
long *clock;

char *asctime(tm)
struct tm *tm;

char *timezone(zone,dst)
```

FUNCTION

Ctime converts a time pointed to by **clock** [as returned by **time** (2)] into ASCII. It returns a pointer to a 26-character string in the following form:

Wed Mar 10 12:12:59 1982\n\0

All fields have constant width.

Localtime and **gmtime** return pointers to structures containing the broken-down time.

Localtime corrects for the time zone and daylight savings time, if necessary.

Gmtime converts to Greenwich Mean Time - the time that Micronix uses.

Asctime converts broken-down time to ASCII and returns a 26-character string.

The **include** file structure declaration follows:

```
struct tm { /* see ctime(3) */
        int     tm_sec;
        int     tm_min;
        int     tm_hour
        int     tm_mday;
        int     tm_mon;
        int     tm_year;
        int     tm_wday;
        int     tm_yday;
        int     tm_isdst;
};
```

The time is given using the following elements:

- 24 hour clock
- day of month, 1-31 days per month
- month of year, numbered 0-11
- day of the week, numbered 0-6
- date of the year, beginning 1900 -
- day of the year, 0-365
- non-zero flag indicates daylight savings

The program checks the system to determine if time zone and/or daylight savings time adjustments are appropriate whenever the local time is called for. (Adjustments for changes in these conversions in 1974 and 1975 are included.)

Timezone returns the time zone, measured in minutes westward from Greenwich, associated with its first argument. If the second argument is 0, the standard name is used. A non-zero flag indicates daylight savings time.

A table is built into the routine. If the required name does not appear on this table, the difference from GMT is produced. In Afghanistan, for example:

timezone(-60*4+30), 0)

would be appropriate because it is eastward of GMT +4:30. The string **GMT+4:30** is produced.

ERRORS

The return values point to static data whose content is overwritten by each call.

ALSO READ

time (2)

SUBROUTINE(S)

fclose, fflush - close or flush a stream

C

```
#include <stdio.h>

fclose(stream)
FILE *stream;

fflush(stream)
FILE *stream;
```

FUNCTION

Fclose causes any buffers for the named **stream** to be emptied and the file to be closed. Buffers allocated by the standard I/O system are freed. This function is performed automatically by calling **exit** (2).

Fflush causes any buffered data for the named output **stream** to be written to that file. The stream file remains open.

ERRORS

These routines return an **EOF** if **stream** is not associated with an output file, or if buffered data cannot be transferred to that file.

ALSO READ

close (2), fopen (3), setbuf (3)

SUBROUTINE(S)

feof, ferror, clearerr, fileno - stream status inquiries

C

```
#include <stdio.h>

feof(stream)
FILE *stream;

ferror(stream)
FILE *stream

clearerr(stream)
FILE *stream

fileno(stream)
FILE *stream
```

FUNCTION

The following functions are implemented as macros and cannot be redeclared:

Feof returns a non-zero when an end of file is read on the named input stream; otherwise a zero is returned.

Ferror returns a non-zero when an error occurs while reading or writing the named **stream. Error must be cleared by clearerr** or the error indication lasts until the stream is closed. A zero is returned if no error occurs.

Fileno returns an integer file descriptor associated with **stream** [see open (2)].

ALSO READ

fopen (3), open (2)

SUBROUTINE(S)

fabs, floor, ceil - absolute value and floor and ceiling functions

C

```
double floor(x)
double x;

double ceil(x)
double x;

double fabs(x)
double x;
```

FUNCTION

Fabs returns the absolute value **|x|**.

Floor returns the largest integer not greater than **x**.

Ceil returns the smallest integer not less than **x**.

ALSO READ

abs (3)

SUBROUTINE(S)

fopen, freopen, fdopen - open a stream

C

```
#include <stdio.h>

FILE *fopen(filename, type)
char *filename, *type;

FILE freopen(filename, type, stream)
char *filename, *type;
FILE *stream;

FILE *fdopen(fildes, type)
char *type;
```

FUNCTION

Fopen opens the file specified by "filename" and associates it with a stream. It returns a pointer to be used to identify the stream in subsequent operations.

Type is a character containing one of the following values:

a - append; begin new writing, or begin at end of current writing.

r - open for reading

w - create for writing

Freopen substitutes the named file in place of an open stream; it returns the original value of **stream**. The original value of **stream** is closed. This routine may also be used to attach preopened constant names - **stdin, stdout, stderr,** - to specified files.

Fdopen associates a stream with a file descriptor obtained from **creat, dup, open** or **pipe** (see section 2). The **type** of the stream must agree with the mode of the open file.

NOTES

If **filename** cannot be accessed, **fopen** and **freopen** return the NULL pointer.

ALSO READ

open (2), fclose (3)

SUBROUTINE(S)

fread, fwrite - buffered binary I/O

C

```
#include <stdio.h>

fread(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;

fwrite(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;
```

DESCRIPTION

Fread reads **nitems** of ***ptr** data from the named input stream into a block beginning at **ptr**. The return is the number of items actually read.

The standard output is line buffered if **stream** is stdin; any partial output line is flushed before any call to **read** (see section 2) to satisfy the fread call.

Fwrite appends **nitems** (maximum) of data to the named output stream. Data is of the type ***ptr** beginning at **ptr**. It returns the actual number of items written.

DIAGNOSTICS

A 0 is returned by **fread** and **fwrite** upon error or at end of file.

ALSO READ

read (2), write (2), fopen (3), getc (3), putc (3), puts (3), printf (3) and scanf (3).

SUBROUTINE(S)

fseek, ftell, rewind - reposition a stream

C

```
#include <stdio.h>

fseek(stream, offset, ptrname)
FILE *stream;
long offset;

long ftell(stream)
FILE *stream;

rewind(stream)
```

FUNCTION

Fseek sets the position of the next input or output operation on **stream**. This new position is at the signed distance, which is **offset** bytes from the beginning of the file, the current position, or at the end of the file, determined by whether **ptrname** has a value of 0, 1 or 2. This routine also undoes any effects of the **unget** routine.

Ftell returns the current value of the offset relative to the beginning of the file associated with the named **stream**. This routine provides the only foolproof way to obtain an **offset** for fseek.

Rewind(stream) = fseek(stream, 01, 0).

ERRORS

A -1 is returned by **fseek** for improper seeks.

ALSO READ

lseek (2), fopen (3)

SUBROUTINE(S)

getpwent, getpwuid, getpwnam, setpwent, endpwent - get password file entry

C

```
#include <pwd.h>

struct passwd *getpwent()

struct passwd *getpwuid(uid)
int uid;

struct passwd *getpwnam(name)
char *name;

int setpwent()

int endpwent()
```

DESCRIPTIONS

Each of the "get" routines - **getpwent, getpwuid** and **getpwnam** - each return a pointer to an object with the following structure:

```
struct passwd
    {
  char    *name
          *passwd,
  unsigned char uid, gid;
  char    *person, *dir, *shell;
    }
```

Each pointer contains the broken-out fields of a line in the password file.

Two fields, **pw_quota** and **pw_comment**, are unused. All the other fields have meanings which are described in passwd (5).

Routine Functions:

getpwent - reads the next line in the file; it will open the file, if necessary.

setpwent - rewinds the file

endpwent - closes the file

getpwuid - searches from the beginning of the file until a matching user ID is found, or until an end of file is encountered.

getpwnam - searches from the beginning of the file until a matching name is found, or until an end of file is encountered.

ERRORS

A zero (NULL pointer) is returned upon an error or at end of file.

NOTES

All information to be saved should be copied since it is contained only in a temporary area.

FILES

/etc/passwd

ALSO READ

getlogin (3), passwd (5)

SUBROUTINE(S)

gets, fgets - get a string from a stream

C

```
#include <stdio.h>

char *gets(s)
char *s;

char *fgets(s, n, stream)
FILE *stream;
```

FUNCTION

Gets reads a string into s from the standard input stream, stdin. A newline character terminates the string. This character is replaced in s by a NULL character; gets returns its argument.

Fgets reads n-1 characters (or up to a newline character) from the stream into the s string. A NULL character follows the last character read into s; fgets returns its first argument.

ERRORS

Both gets and fgets return the constant pointer (NULL) upon an error or at end of file.

NOTES

To maintain backward compatibility, fgets maintains a newline, gets deletes a newline.

ALSO READ

puts (3), getc (3), scanf (3), fread (3), ferror (3)

SUBROUTINE(S)

perror - system error messages

C

```
perror(s)
char *s;
```

FUNCTION

Perror displays error messages on the error file. These messages describe the last error encountered during a C program system call. The s argument string is printed, followed by a colon, the message and a newline. The argument string defines the name of the program in which the error occurred. The message is taken from errno (see Introduction to Section 2), which is set when errors have occurred but have not been cleared.

ALSO READ

Intro (2)

SUBROUTINE(S)

popen, pclose - initiates I/O to/from a process

C

```
# include <stdio.h>

FILE *popen(command, type)
char *command, *type;

pclose(stream)
FILE *stream;
```

FUNCTION

The arguments to **popen** are pointers to null-terminated strings. These strings contain: 1) a shell command line and 2) an I/O mode, either "r" for reading or "w" for writing. A pipe is created between the calling process and the command to be executed. The value returned is a stream pointer that may be used to write to the standard input of the command or to read from its standard output.

Streams opened by **popen** are closed by **pclose. Pclose** waits for the associated process to terminate, then returns the exit status of the command.

Since open files are shared, the type "r" command can be used as an input filter, the type "w" as an output filter.

ERRORS

Popen returns a 0 (NULL pointer) if the shell cannot be accessed, or if files or processes cannot be created.

Pclose returns -1 if **stream** is not associated with a command opened by **popen.**

NOTES

Filter input may be badly positioned if user attempts buffered reading before opening an input filter.

Similar problems with an output filter may be prevented by flushing the buffer, i.e., using **fflush.** [See **fclose** (3).]

READ ALSO

pipe (2), fopen(3), fclose (3), system (3), wait (2)

SUBROUTINE(S)

printf, fprintf, sprintf - formatted output conversion

C

```
# include <stdio.h>

printf(format [, arg]...)
char *format;

fprintf(stream, format [, arg] ...)
FILE *stream;
char *format;

sprintf(s, format [, arg ] ...)
char *s, format;
```

FUNCTION

Printf places output on **stdout** - the standard output stream.

Fprintf places output on **stream** - the named output.

Sprintf places output in the s string. This string is followed by a "\0" character.

These functions convert, format and print their arguments under the control of the first argument. This first argument will be a character string containing 1) plain characters that are copied to the output stream, and 2) conversion specifications which enable conversion and printing of the next **printf** argument.

The conversion specifications are introduced by the % character. The options listed below may follow this character:

A minus sign (-) specifying left adjustment of the converted value in the indicated field.

A string of digits specifying field width. Converted values with fewer characters than the indicated field width may be blank-padded to make up the difference, unless the field width begins with a 0, in which case it will be zero-padded.

A period (.) may separate the field width from the next string of digits.

An additional digit string may be added specifying a precision, or the number of digits that will appear after the decimal point. This is used for -e and -f conversions, or to specify the maximum

number of characters to be printed from a string. [An asterisk (*) may indicate field width or precision, in which case an integer argument indicates the field width or precision.]

A l may be specified with d, o, x, or u to indicate correspondence with a long integer argument. The same thing may be accomplished with a capitalized conversion code.

Any character may be given that specifies the type of conversion to be executed.

The conversion characters are listed and described below:

dox
Integer arg is converted to decimal, octal or hexadecimal notation.

f
Float or double arg is converted to decimal notation in the following format:

[-]ddd.ddd

The precision specification determines the amount of "d"s. A precision of 0 results in no digits; default is six digits.

e
Float or double arg is converted in the following format:

[-]d.ddde+dd

One digit before and one number after the decimal point equal the precision specification. By default, six digits are produced.

g
Float or double arg is printed in one of the formats listed above, depending upon which one specifies full precision in the least amount of space.

c
Character arg printed; NULL characters are ignored.

s
Character arg is defined as a character pointer; string characters are printed up to a NULL character, or if specified, up to point indicated by the precision specification.

%
Only the % is printed; arguments are not converted.

NOTES

Small or non-existent fields do not cause truncation. Fields are padded when the specified field exceeds the actual width.

Fields larger than 128 characters will fail.

Putc (3) prints the characters generated by printf.

ALSO READ

putc (3), scanf (3), ecvt (3)

PROGRAM(S)

puts, putchar, fputc, putw - put character or word on a stream

C

```
#include <stdio.h>

int putc(c, stream)
char c;
FILE *stream;

putchar(c)

fputc(c, stream)
FILE *stream;

putw(w stream)
FILE *stream;
```

FUNCTIONS

Putc appends c (character) to stream, the named output; it returns the character written.

Putchar(c) is the same as putc(c, stdout).

Fputc acts the same as putc but is a function rather than a macro. It can be used to save object text.

Putw appends w (word) to output stream and returns the word written. It does not assume or cause special alignment in the file.

Stdout (standard stream) is only buffered if the output refers to something other than a terminal (default). Setbuf (3) changes the default.

Stderr (also a standard stream) is unbuffered unconditionally (default). **Freopen** [see **fopen** (3)] causes it to become buffered; again, setbuf changes the default.

Unbuffered output streams appear on the destination file or terminal as soon as they are written. Buffered output streams save up characters and write them as a block. Blocks may be forced out prematurely with fflush [see **fclose** (3)].

ERRORS

These functions return the end of file constant upon error.

Ferror (3) should be used to detect **putw** errors.

Errors may occur some time after the initial call to **putc**.

NOTES

Since **putc** is implemented as a macro, it treats a **stream** argument improperly. Specifically,

```
putc(c, *f++);
```

does not work sensibly.

ALSO READ

fopen (3), fclose (3), getc (3), puts (3), printf (3), fread (3)

SUBROUTINE(S)

puts, fputs - put a string on a stream

C

```
#include <stdio.h>

puts(s)
char *s;

fputs(s, stream)
char *s;
FILE *stream;
```

FUNCTION

Puts copies s (null-terminated string) to **stdout**, the standard output stream. It also appends a newline character.

Fputs copies s to the named output, **stream.**

Neither of these routines copies the terminal NULL character.

NOTES

To maintain backward compatibility, **puts** appends a newline, **fputs** does not.

ALSO READ

fopen (3), gets (3), putc (3), printf (3), ferror (3), fwrite in fread (3)

SUBROUTINE(S)

scanf, fscanf, sscanf - formatted input conversion

C

```
#include <stdio.h>

scanf(format [, pointer]...)
char *format;

fscanf(stream, format [, pointer]...)
FILE *stream;
char *format;

sscanf(s, format [, pointer]...)
char *s, *format;
```

FUNCTION

Scanf reads from **stdin**, the standard input stream.

Fscanf reads from the named input, **stream**.

Sscanf reads from the s character string.

Each of these functions read and interpret characters according to a format, then store the results in its argument. A control string **format** and a set of pointer arguments indicating where the converted input is stored are arguments expected by these routines. These are described below.

Blanks, tabs or newlines, which match optional white space in the input.

Any ordinary character besides % that must match the next character of the input stream.

Optional conversion specifications, which consist of an assignment suppressing character (*), maximum field width number, a conversion character and the % character.

The control string usually contains conversion specifications that direct interpretation of the next input field. Unless assignment suppression was indicated (using the * character), the result is placed in the variable pointed to by the corresponding argument.

An input field is a string of non-space characters. The field extends to the next inappropriate character, or until the field width (if specified) is exhausted.

The interpretation of the input field is indicated by the conversion character. Corresponding pointer arguments are usually of a restricted type. A list and description of legal conversion characters follows:

%
A single percent sign (%) is expected. No assignment is done.

d
Decimal integer is expected. Corresponding argument should be integer pointer.

o
Octal integer expected. Corresponding argument should be integer pointer.

x
Hexadecimal integer expected. Corresponding argument should be integer pointer.

s
Character string expected. Corresponding argument should be a character pointer pointing to an array of characters. Array must be large enough to accept string and terminating "\0" character that will be added. A space character or newline terminates the input field.

c
Character expected. Corresponding argument should be a character pointer. The normal skip-over-space characters are suppressed. Use "%1s" to read the next non-space character.

Corresponding argument should refer to a character array and indicate number of characters read if a field width is given.

e, f
Floating point number is expected. Next field is converted and stored through the corresponding argument which should be a pointer to a **float**.

The format for floating point numbers is an optionally signed string of digits that may contain a decimal point followed by an optional exponent field consisting of E (or e), followed by an optionally signed integer.

[
Left bracket indicates a string not to be delimited by space characters. The string is defined by characters followed by a right bracket. A ^ as the first character after the left bracket defines the input field as all characters in the set of characters following the ^ sign. Absence of this sign means the input field is all the characters within the brackets. Corresponding argument must point to a character array.

The conversion characters d, o and x may be either capitalized or preceded by an l to indicate that a pointer to long rather than int is in the argument list. If these are preceded by h, indicates a pointer to short rather than int.

The e and f conversion characters may also be capitalized or preceded by l to indicate a pointer to double rather than float.

Scanf returns the number of input items successfully matched and assigned. This may also be used to decide how many items were actually found. The constant end of file is returned upon end of input. This is different from 0, which may be interpreted to mean no conversion was done. If conversion was intended, it was frustrated by an inappropriate character in the input. As an example:

```
int i; float x; char name[50];
scanf("%d%f%s", &i, &x, name);
```

with the input

```
25 54.32E-1 McCleary
```

assigns to i the value 25, x the value 5.432. Name contains "mccleary\0". Another example:

```
int i; float x; char name[50];
scanf("%2d%f%*d%[1234567890]", &i, &x, name
```

with the input

```
56789 0123 56a72
```

assigns 56 to i, 789.0 to x, skips "0123" and places the string "56\0" in name. The next call to getchar will return "a".

ERRORS

The success of matches and suppressed assignments can be determined only indirectly.

NOTES

In cases of missing or illegal data, **scanf** returns an end of file on end of input.

ALSO READ

atof (3), getc (3), printf (3)

SUBROUTINE(S)

setbuf - assign buffering to a stream

C

```
#include <stdio.h>

setbuf(stream, buf)
FILE *stream;
char *buf;
```

FUNCTION

Setbuf is used after a stream is opened, but before it is read or written to cause the **buf** character array to be used instead of an automatically allocated buffer.

If **buf** is NULL (constant pointer), I/O is completely unbuffered.

Size of array is determined by a the constant, BUFSIZ. An example:

```
char buf[BUFSIZ];
```

A buffer is obtained from **alloc** (3) upon the first **getc** or **putc** on the file. One exception is when the standard output is directed to a terminal, in which case it is line buffered. Normally, output streams directed to terminals and **stderr** (standard error stream) are not buffered.

When standard output is line buffered, it is flushed by **read** (2) each time data is read from the standard input.

ERRORS

By default, the standard error stream should be line buffered.

ALSO READ

fopen (3), getc (3), putc (3), alloc (3)

SUBROUTINES

stdio - standard buffered I/O package

C

```
#include <stdio.h>

FILE *stdin;
FILE *stdout;
FILE *stderr;
```

FUNCTION

This section describes two levels of buffered input/output utilities: 1) in-line macros designated by (3), and 2) higher level standard I/O utilities designated by (3s) or (3m).

The in-line macros, **getc** and **putc** handle characters quickly.

Routines such as **gets, fgets, printf, fprintf, fwrite, fread, puts** and **scanf** and fscanf use the in-line macros; both levels may be freely intermixed.

Files with this type of associated buffering are called **streams** and are declared to be a pointer to a defined type **FILE. Fopen** (3) has the ability to create descriptive data for streams. It returns a pointer to designate the stream in all further transactions. Normally, there are three open streams with constant pointers declared in the **include** file associated with a standard open file:

stdin - standard input file
stdout - standard output file
stderr - standard error file

A constant "pointer" (NULL 0) designates no stream.

An integer constant (end of file, or EOF -1) is returned upon error by any integer functions that deal with streams or upon end of file.

Routines that use the I/O package must have the header file with the pertinent macro definitions (<stdio.h>). The **include** file declares all functions and constants described in the sections designated by (3s); they need no further declaration.

The following constants and functions are used as macros:

feof ferror fileno getc getchar putc

It is dangerous to redeclare these names.

ERRORS

The **EOF** value is consistently returned, indicating that a **FILE** pointer has not been initialized with **fopen**. This means that input or output has been attempted on an output or input stream, or that a **FILE** pointer designates corrupt **FILE** data.

This use of the standard library has been changed to line buffer output to a terminal by default. It attempts to flush the output whenever a **read** (see Section 2) from the standard input is necessary. In most cases, this makes the utility more efficient. The actions themselves should be transparent unless it is used with programs which use standard I/O routines and **read** (2) to read from the standard input.

When large amounts of computation are done after outputting part of a line to a terminal, it is necessary to **fflush** (3) the standard output before continuing on so that the output appears.

ALSO READ

close (2), open (2), read (2), write (2)

SUBROUTINE(S)

system - issue a shell command

C

```
system(string)
char *string;
```

FUNCTION

System causes **string** to be given to **sh** (see shell - Section 1) as input just as if the string had been typed as a command at a terminal. Current process waits until the shell has completed then returns the exit status of the shell.

ERRORS

An exit status of 127 means the shell could not be executed.

ALSO READ

popen (3), exec (2), wait (2)

SUBROUTINE(S)

ttyname, isatty, ttyslot - find name of a terminal

C

char *ttyname(fildes)

isatty(fildes)

ttyslot()

FUNCTION

Ttyname returns a pointer to the null-terminated path name of the terminal associated with **fildes**, the file descriptor.

Isatty returns a 1 if the file descriptor is associated with a terminal; a 0 is returned if it isn't.

Ttyslot returns the number of the entry in the **ttys** file (see Section 5) for the control terminal of the current process.

ERRORS

Ttyname returns a 0 (NULL pointer) if the file descriptor does not describe a terminal that exists in the "/dev" directory.

Ttyslot returns a 0 if the "/etc/ttys" directory is inaccessible, or if it can't determine the control terminal.

NOTES

Return values point to static data whose content is overwritten by each call.

FILES

/dev/*
/etc/ttys

ALSO READ

??ioctl (2), ttys (5)

SUBROUTINE(S)

ttyname, isatty, ttyslot - find name of a terminal

C

```
char *ttyname(fildes)

isatty(fildes)

ttyslot()
```

FUNCTION

Ttyname returns a pointer to the null-terminated path name of the terminal associated with **fildes**, the file descriptor.

Isatty returns a 1 if the file descriptor is associated with a terminal; a 0 is returned if it isn't.

Ttyslot returns the number of the entry in the **ttys** file (see Section 5) for the control terminal of the current process.

ERRORS

Ttyname returns a 0 (NULL pointer) if the file descriptor does not describe a terminal that exists in the "/dev" directory.

Ttyslot returns a 0 if the "/etc/ttys" directory is inaccessible, or if it can't determine the control terminal.

NOTES

Return values point to static data whose content is overwritten by each call.

FILES

/dev/*
/etc/ttys

ALSO READ

stty (2), ttys (5)

Micronix Operating System reference manual

devices

MORROW

CONTENTS OF REFERENCE SECTION 4: DEVICES

NAME

cables - cables for modems, terminals, printers, and the network

DESCRIPTION

This document tells you how to build cables to connect the Decision to various devices. The presentation is rather technical. If you don't want to build one of these cables yourself, you may purchase it from your dealer or from Morrow, or your dealer may be able to build it for you from these specifications.

The first three serial ports on the Decision use the Wunderbus I/O motherboard, which (unlike the MULTIO expansion board) has no provisions for configuring the RS-232 connections via jumpers. Thus the cables must do this configuration internally. Also, the Wunderbus has an artwork error which assigns pin 5 as Carrier-Detect (RLSD) and pin 8 as Clear-To-Send (CTS). The cables detailed below take this into account.

Four serial cables and two parallel cables are discussed:

1. Decision to serial printer or terminal
2. Decision to Decision (network)
3. Decision to modem
4. MULTIO to modem

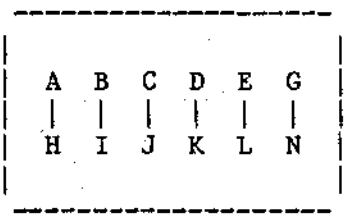
5. Decision to Centronics-compatible parallel printer
6. Decision to Diablo-compatible parallel printer

In general, the serial cables require two DB-25P male connectors (ITT Cannon or AMP) with shielded shells on both ends, and should be made with 22 guage shielded wire. The RS-232 standard specifies a maximum length of 25 feet (although much longer cables have been known to work). The parallel cables can be made with 50 conductor flat cable. They both have a 50 pin female flat cable connector (Ansley 609-5000M) on the computer end. Cable 6 has the same connector on the the printer end, while cable 5 uses a 36 pin female ribbon connector (Amphenol 57-40360).

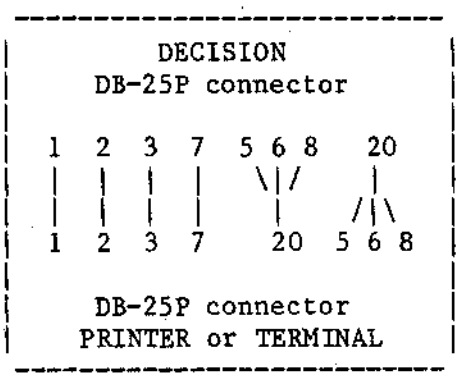
DECISION TO SERIAL PRINTER OR TERMINAL

The cable required to comunicate with a terminal or printer can be constructed so that it will work with both the Decision and the MULTIO board. This assumes that the hardware handshaking on the printer is via pin 20. (Most printers support this. One exception is the NEC Spinwriter, which uses pin 19 instead.)

The Decision's serial ports cannot be jumpered. On the MULTIO, the appropriate serial port should be jumpered as follows:



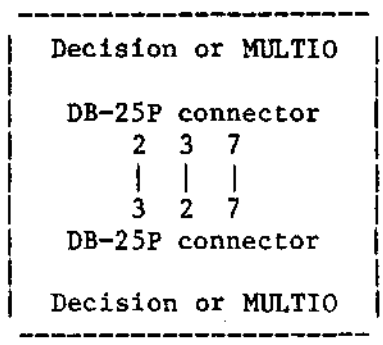
The cable requires 22 guage shielded 8 conductor wire. The pinout for both the MULTIO and the Decision is the same:



Note: For most terminals, only pins 2, 3, and 7 need be connected. The cable described here has the advantages of working interchangablly with printers and terminals, and of higher reliability at high baud rates.

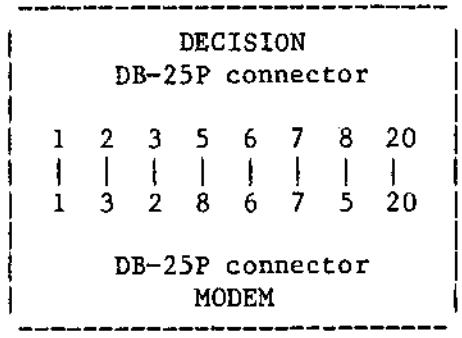
NETWORK

The Micronix network uses the serial ports on the Decision or the MULTIO. Since the first priority in designing this network was low cost, a simple unshielded 3 conductor cable is specified. (The network does extensive error checking, so noise on the cable will only slow it down.) The MULTIO should be jumpered as for #1 above. The cable should be constructed as follows:



DECISION TO MODEM

This requires 22 guage shielded 8 conductor wire:

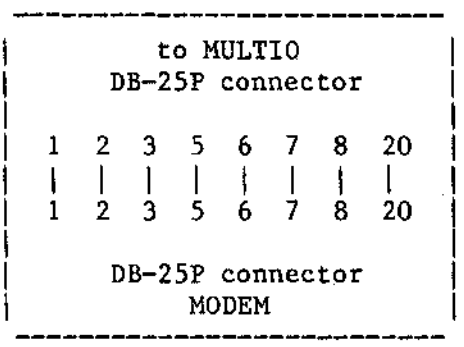


MULTIO TO MODEM

(Note: this discussion assumes that the serial ports on the MULTIO have been brought out to female D connectors on the back of the computer with special cables installed at the factory. If this has not been done, you can order these "MULTIO serial cables" from Morrow.)

Jumper the appropriate MULTIO serial port for modem communications as follows:

The cable from the computer to the modem requires 22 guage shielded 8 conductor wire:



DECISION TO PARALLEL CENTRONICS

This cable also works with the MULTIO expansion board, since its parallel port is identical with the Decision's. This is the most complex cable. It is available by special order from Morrow. Its pinout is:

```
DECISION
50 pin flat cable connector

 2  3  12 13 21 24 26 27 28 33 36 37 39 40 42 43 45
 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
16 12 32 31 1  36 10 11 13 5  3  2  4  6  7  8  9

36 pin female ribbon connector
CENTRONICS-COMPATIBLE PRINTER
```

or, sorted the other way:

```
DECISION
50 pin flat cable connector

21 37 36 39 33 40 42 43 45 26 27 3  28 2  13 12 24
 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 1  2  3  4  5  6  7  8  9  10 11 12 13 16 31 32 36

36 pin female ribbon connector
CENTRONICS-COMPATIBLE PRINTER
```

DECISION TO PARALLEL DIABLO

This cable also works with the MULTIO expansion board, since its parallel port is identical with the Decision's. All you need is a simple straight-through 50 conductor cable with 50-pin flat cable female connectors on both ends. It is the same cable that Morrow uses for its disk drives. If you need to build the cable, only 17 conductors are actually needed:

```
DECISION
50 pin flat cable connector

 2  3  12 13 21 24 26 27 28 33 36 37 39 40 42 43 45
 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 2  3  12 13 21 24 26 27 28 33 36 37 39 40 42 43 45

50 pin flat cable connector
DIABLO-COMPATIBLE PRINTER
```

NAME

djdma - Morrow Designs Disk Jockey Direct Memory Access floppy disk controller

DESCRIPTION

The Morrow Designs DJDMA is an S-100 floppy disk drive controller. This document describes the Micronix driver for the DJDMA. All access to the DJDMA is handled by the driver.

The DJDMA provides access to up to eight 8 inch and/or 5 1/4 inch floppy disk drives (limited to four of each type). At present, there may be only one controller board per system.

Currently, only IBM format 8" floppies and 10 hole, hard sectored 5 1/4" floppies are supported.

Diskettes used with Micronix must be formatted.

The following formats are supported. They are listed with the number of blocks per disk (a block is 512 bytes).

```
8",   128 byte sectors, single-sided,  487 blocks
8",   128 byte sectors, double-sided,  975 blocks
8",   256 byte sectors, single-sided,  975 blocks
8",   256 byte sectors, double-sided, 1950 blocks
8",   512 byte sectors, single-sided, 1125 blocks
8",   512 byte sectors, double-sided, 2250 blocks
8",  1024 byte sectors, single-sided, 1200 blocks
8",  1024 byte sectors, double-sided, 2400 blocks

5 1/4",  256 byte sectors, single-sided, 35 track,  160 blocks
5 1/4",  512 byte sectors, single-sided, 35 track,  330 blocks
5 1/4",  512 byte sectors, double-sided, 35 track,  660 blocks

5 1/4",  512 byte sectors, single-sided, 40 track,  380 blocks
5 1/4",  512 byte sectors, double-sided, 40 track,  760 blocks

5 1/4",  512 byte sectors, single-sided, 80 track,  780 blocks
5 1/4",  512 byte sectors, double-sided, 80 track, 1560 blocks
```

Note that because of reserved tracks (usually the first two), the available space is reduced slightly.

All 8 inch formats are standard 77 track.

While you gain approximately 7% in additional capacity with 1024 byte sectors on an 8 inch floppy diskette, the loss in transfer speed is considerable.

The optimum choice for Micronix floppy diskettes is 512-bytes, single or double sided, unless space considerations dictate otherwise.

The other formats are included for backward compatibility.

Alternate sectoring is optional and available by selecting the appropriate minor device number from the table below.

Bit 4 of the minor device number selects alternate sectoring.

(Alternate sectoring is recommended because it doubles the reading speed, but does not make much difference in writing speed.)

Note that alternate and non-alternate sectored media are incompatible (i.e., the sectors appear scrambled). The far utility (see Section 1) can read a CP/M formatted diskette under either alternate or straight sectoring, however.

The sector arrangement schemes fall into classes for even and odd sector per track values.

The sectors are numbered

0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15

for a 15 sector per track disk (for instance).

and

0, 2, 4, 6, 1, 3, 5, 7

for an 8 sector per track disk.

The major and minor device numbers listed below are used in conjunction with the mknod program.

See mknod in section 1 of the Micronix reference manual.

The major device number corresponding to the DJDMA is currently 2.

The minor device numbers have the following meanings:

minor	meaning
0	8" - drive 0
1	8" - drive 1
2	8" - drive 2
3	8" - drive 3
4	5 1/4" - drive 0
5	5 1/4" - drive 1
6	5 1/4" - drive 2
7	5 1/4" - drive 3
8	8" - drive 0 with alternate sectoring
9	8" - drive 1 with alternate sectoring
10	8" - drive 2 with alternate sectoring
11	8" - drive 3 with alternate sectoring
12	5 1/4" - drive 0 with alternate sectoring
13	5 1/4" - drive 1 with alternate sectoring
14	5 1/4" - drive 2 with alternate sectoring
15	5 1/4" - drive 3 with alternate sectoring

Note that there are 96-track 5 1/4" drives and 48-track 5 1/4" drives. For example, if you have only 48 track drives, you cannot read 96-track diskettes.

There are five different sets of delay constants available for stepping delay and head settle delay. The delay numbers for step time and head settle time were taken from real-world drive descriptions and are intended to be appropriate for the drive types listed below.

If you are using a drive not listed below, you should select the ones most closely suited to the types of drives you are using.

Refer to the following table:

seek	settle	minor	type	
8	8	0	SA 800	(8" drives)
3	15	16	SA 850	
5	15	0	Tandon	
20	20	16	SA 200	(5 1/4" drives)
40	10	32	SA 400	

The column entitled "seek" gives the seek time in milliseconds per track. "settle" is the disk drive head settle time in milliseconds. "minor" is the minor device addend, and "type" gives manufacturers drive types. "SA" stands for Shugart Associates.

To select a particular drive type, add in the value listed in the "minor" column of this table.

If you are familiar with binary notation, the drive type is encoded in bits 4 and 5 of the minor device number.

EXAMPLES

For a SA800 drive "A" with alternating sectoring the minor device number would be 8.

For a SA850 drive "A" with alternating sectoring the minor device number would be 24.

For a SA200 drive "A" with alternating sectoring the minor device number would be 28.

To create an appropriate special file for an SA200 (5 1/4" drive), the command would be:

```
mknod /dev/mfa b 2 28
```

SEE ALSO

mknod (1), far (1)

Disk Jockey / Direct Memory Access Floppy Disk Controller Technical Manual

NAME

djmem
- Morrow Designs DJDMA floppy controller memory device

DESCRIPTION

The djmem device provides access to the memory internal to the DJDMA floppy controller and to the funcions "sense status" and "execute controller routine". The djdma memory device was originally implemented to allow floppy diskette formatting under Micronix. It is conceivable that it may be put to other uses.

The djdma memory device driver ignores the minor device number. That is to say you may access only on djdma controller and there is only one way to access it.

The djdma and djmem drivers are mutually exclusive. If you attempt to use one while the other is in use you will get the error return "File or Device Busy".

A read on the djmem device transfers data from the djdma contoller's memory to the user's memory.

A write transfers from user memory to controler memory.

Seek may be used to position the pointer into controller memory.

The only safe locations to write are 0x1030 to 0x127f inclusive according to the djdma tech. reference manual.

Note that the "0x" preceding the preceding numbers indicates base 16.

At most 1024 bytes may be transferred in a single read/write call. Larger byte counts cause an "invalid argument" error.

So far the implementation has been very straightforward.

In order to pull off formatting a couple extra extended features are needed. These are implemented via the system calls stty() and gtty().

Gtty on the device caused a "sense status" command to be performed and the results returned.

Stty causes an "execute controller memory" command to be performed.

Gtty returns 4 bytes of meaningful status. Stty returns one byte of status.

Gtty and Stty are called by passing a pointer to a 6-byte structure in user memory. The user process fills out the first 2 bytes of memory, Gtty or Stty fill out the remaining 4 bytes.

For gtty the first 2 byte word represents the device number on which to perform the status operation.

For stty the first 2 byte quantity is the address in conroller memory to execute.

The user familiar with C will find the following structure declarations illuminating. Note that these declarations are available in /include/dj.h.

```
struct djstat
      {
      int drive;
      char status[4];
      };

struct djexec
      {
      char *address;
      char status [4];
      };
```

For gtty the drives are numbered as follows:

There are 8 possible drives. Four 5 inch and four 8 inch.

The 8 inch drives are numbered 0-3. The 5 inch drives are numbered 4-7.

The meanings of the 4 status bytes returned are described in the djdma technical reference manual.

The single status byte returned by the stty call has whatever meaning the controller routine called ascribes to it.

Note that the stty call is potentially dangerous. It is easily capable of dropping the whole system because through it you can do anything the conroller can do, which includes reading or writing any portion of memory.

This is also a security breach.

It is recommended then that the file /dev/djmem be highly restricted and write locked.

SEE ALSO

djdma(4), formatdj(1) intro(2), chmod(1), ls(1) DJDMA Techincal Reference Manual.

DEVICE

HDCA - Morrow Designs' Winchester disk controller

FUNCTION

The HDCA controls up to four 8 inch Winchester disk drives. Micronix currently supports 3 drive sizes:

m10	10 megabytes
m20	20 megabytes
m26	26 megabytes

The device names (in the /dev directory) for these disks depend on their position on the HDCA daisy-chain. Drive A is called "hda", drive B is called "hdb", etc. The major/minor device numbers are:

1/0	hda	1/1	hdb	1/2	hdc	1/3	hdd

Unlike the HDDMA drives, Micronix can automatically detect the size of the HDCA drives, so there is no need to indicate the drive size in the device name or number.

Micronix does not support a disk formatting program, so each drive must be formatted under CP/M. A sector skew of 6 is recommended, but is not critical. Since these drives are soft-sectored, there is no compatibility problem with different skews.

DEVICE

HDDMA - Morrow Designs' dma hard disk controller

FUNCTION

The HDDMA controls up to four 5 1/4 inch Winchester disk drives. Micronix currently supports 3 drive sizes:

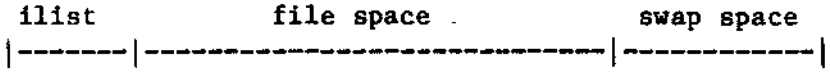
m5	5 megabytes
m10	10 megabytes
m16	16 megabytes

The device names (in the /dev directory) for these disks depend on their position on the HDDMA daisy-chain. An m5 attached as drive A is called "m5a", an m16 attached as drive D is called "m16d", etc. The major/minor device numbers are as follows:

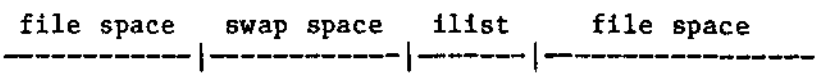
3/0	m5a	3/4	m10a	3/8	m16a
3/1	m5b	3/5	m10b	3/9	m16b
3/2	m5c	3/6	m10c	3/10	m16c
3/3	m5d	3/7	m10d	3/11	m16d

Micronix does not support a disk formatting program, so each drive must be formatted under CP/M. A sector skew of 6 is recommended, but is not critical. Since these drives are soft-sectored, there is no compatibility problem with different skews.

If you ever have to explore the surface of one of these drives without the intervention of Micronix, you should know that Micronix logically "rolls" the tracks by (number of tracks) / 2. Thus on an m5, which has 153 tracks, logical track 0 is physical track 76. The reason for this is that most Micronix systems have only one hard disk, which must serve as both the root device and the swap device. When Micronix sees that rootdev == swapdev, it uses the space above the file system as swap space. Thus, the logical disk looks like:



while the physical disk looks like:



In a busy system, the disk heads spend most of their time over the swap space and the ilist, so this gives a performance advantage.

NAME

io - Z80 I/O space device

DESCRIPTION

Under a multi-user operating system, it is not acceptable to allow user programs to execute certain instructions in an unregulated way. The Z80 instructions "in" and "out" fall into this category. It is, however, desirable to be able to perform these operations in special circumstances.

The "io" device allows users to access the Z80 I/O ports through the file system. Reading from the device causes the results of "in" instructions to be transferred to the user's memory space.

Writing to the device causes "out" instructions to be executed using the data in the user's memory space.

Seeking on the device sets the I/O address for subsequent reading/inputting or writing/outputting.

There are two classes of I/O device. One allows access to all of I/O space. The other allows access only to a single port.

Note that it would be extremely easy to crash the system through this avenue, and it is therefore recommended that the "/dev/io" file be heavily protected, perhaps mode 0 and owned by root.

Examples are in order:

To input a value from port 10:

```
f = open ("/dev/io", 0);

seek (f, 10, 0);

read (f, &data, 1);
```

The operation of the I/O device is dependent on the minor device number of the special file associated with it.

All 256 possible minor device numbers (0 - 255) have meaning.

When accessed through minor device zero, it is possible to select which I/O ports you access. This is the meaning of a seek on the io device file. For example, seeking to location 87 will cause subsequent reads and writes to the file to be interpreted as references to

io port 87.

Unlike other devices, the file offset does not progress with reads and writes. It may only be changed by seeking. It was thought that this arrangement was more in the spirit of I/O space accesses than the more Micronix-like alternative.

Non-zero minor device numbers limit access to that one I/O port whose address is the same as the minor device number. It is therefore possible to have I/O space selectively protected, while giving any user access to some certain port.

For example, if you wanted to allow free access to ports 77 and 78 by any user on the system, you would issue the following commands:

```
mknod /dev/io77 c 3 77

mknod /dev/io78 c 3 77

chmod a=rwx /dev/io77 /dev/io78
```

If the minor device number is not zero, seeks are simply ignored.

FILES

/dev/io - the device file

DEVICE

mem - core memory device

DESCRIPTION

This device reads and writes kernel memory (addresses 0 to FFFF). One must be exceedingly careful in protecting this device, since there are several hardware registers in this address range that will crash the system upon being read.

NAME

multio - Morrow Designs MULT/IO board interface

DESCRIPTION

The Micronix MULT/IO interface supports all features of the MULT/IO and Wunderbuss I/O boards. It will run unchanged in rack mount or table top models of the Decision I.

Serial ports and both Diablo and Centronics style parallel parallel protocols are supported. A cable adapter (available from Morrow Designs) is required to use the MULT/IO with a Centronics style parallel interface.

The MULT/IO board has three ACEs (Asyncronous Communication Elements) and 1 parallel port for I/O device connection. The ACEs communicate via standard RS232 serial interface. The parallel port is a standard Diablo (50-pin daisy-wheel) interface.

Each MULT/IO board must have its I/O port addresses set by means of switches on the circuit board. For proper operation, the base I/O address must be 48 (hex) for the first MULT/IO board.

In a Micronix system in which there are possibly many MULT/IO boards, the boards form a series. In the Decision I Table Top model, the Wunderbuss I/O motherboard functions as the first of the series. Micronix expects to find the Mult/IO boards (or Wunderbuss) as I/O (hex) locations 48, 58, 68, 78. The current incarnation of the Micronix Mult I/O drivers supports at most 4 Mult I/O boards (or 1 Wunderbus I/O and 3 Mult I/O boards).

In the following we talk about "minor device numbers".

Use "mknod" to create a new special file with a particular minor device number.

Use "ls -l" to view the minor device numbers of existing files.

Minor device numbers 0-3 refer to the Mult/IO board addresses at I/O location 48 hex.

Minor device numbers 4-7 refer to the Mult/IO board addresses at location 58 hex, and so on with 68, 78.

The first minor device number of each group of four refers to the parallel printer on that board. Minor device numbers 0, 4, 8, etc. refer to parallel

printers.

If the 16's bit is set in the minor device number, then the port will be treated as a Centronics rather than Diablo parallel interface. Minor device numbers 16, 20, 24, and 28 refer to Centronics parallel ports on the four MULT/IO boards.

You may mix Diablo and Centronics style parallel interfaces in the same system. It is possible to run Centronics interface printer(s) and Diablo interface printer(s) concurrently.

Sending data to the Mult I/O is fairly straightforward except for the case of the Diablo parallel interface.

Here the parallel data path is wider than one byte. In order to pass data to the driver through a byte stream, a special format was developed.

A conversion filter is provided which is capable of encoding an ordinary text file into a form suitable for the driver.

Bytes sent to the parallel printer device have the following format:

Each byte is stand-alone and has a meaning all by itself.

Bit-0 is the least significant bit and bit-7 is the most significant bit in this discussion.

There are two types of information one can send to the printer, motion commands and print commands.

If bit-7 is clear, the the byte is taken to mean a character to be printed. Otherwise, if bit-7 is set, the byte indicates a motion command.

If bit-6 is set, the byte indicates vertical motion.

If bit-5 is set, the byte indicates backwards motion (up or left).

Bits 0 - 4 (the low order 5 bits) are taken to be the magnitude of motion desired.

The unit of motion is 1/120" for horizontal motion and 1/48" for vertical motion.

If you want to move farther than 31 increments, you must send a series of motion bytes.

These are accumulated and optimized, then sent all at once.

Note that this scheme allows elaborate high-level interface. Even several different interfaces, each with its own set of escape sequences or what have you, all translate to common Mult/IO parallel printer motion and print byte formats.

SEE ALSO

diablo (1) - Diablo parallel high level conversion.

tty (4) - Description the tty interface.

mknod (1) - Make a new special file.

ls (1) - List files (including major/minor device numbers).

NAME

network - the Miconix network

DESCRIPTION

Micronix allows any number of Decisions to be interconnected in any pattern with simple, low cost hardware. Files can be copied from any source to any destination on the network (see cp in section 1), and any user may send mail to any other user (see mail in section 1). This is all done with ordinary serial I/O ports and 3-conductor unshielded cables. Error checking is extensive, so that cables may be cut and patched, and machines may be crashed and restarted, all with good assurance that all files will arrive intact. There is a penalty for this generality and low cost, however. The transfer rate is slow (about 40-50 bytes per second). But since all network activity is in the background, users need not wait idly for file transfers to finish.

Two-machine network

Let's begin by describing the simplest network: two machines with one cable between them. First, you must choose names for the two machines. These names are arbitrary; the only requirement is that each machine on the network must have a unique name. The names should also be short and easy to type. In this case, let's call them "sales" and "doc".

Next, you must buy or build a cable long enough to connect them. (See **cables** in section 4.) Then you must choose a free serial port on each machine (see ports in section 4). Let's say that you have choosen ttyD on doc, and ttyF on sales. The cable can be plugged in at this point -- either end to either machine.

Now you must edit three files on each machine. These files are:

/etc/ttys
/etc/netmap
/etc/rc

The /etc/ttys file tells which IO ports on the local machine are connected to the network. The /etc/netmap file gives the layout of the network. The /etc/rc file contains commands that are run every time the system goes multi-user. In this case, we will insert a command to start the "network daemon" (the background process that runs the network).

Miconix comes with **WordStar** (a screen editor) and **edit** (a line editor). You may use either to edit these files. Instructions for using edit on the ttys file are contained in the **ttys** document in section 5.

Here is what you must do to each file on each machine:

machine	file	action
doc	/etc/ttys	Find the line that begins with ttyD. Change it to read as follows: ttyD 1200 net-sales
doc	/etc/netmap	Change (or create) this file to read: doc sales
doc	/etc/rc	Add the following lines: echo Starting network netdaemon
sales	/etc/ttys	Find the line that begins with ttyF. Change it to read as follows: ttyF 1200 net-doc
sales	/etc/netmap	Change (or create) this file to read: sales doc
sales	/etc/rc	Add the following lines: echo Starting network netdaemon

You will have to bring the system down (with the down command) and then start it up again before these changes take effect. You can test the network by typing the following command from the doc machine:

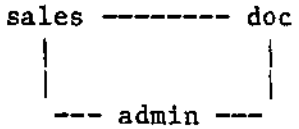
```
% cp /etc/ttys sales%/tmp
```

The /etc/ttys file should appear on the sales machine in the /tmp directory. See cp and mail in section 1 for more information on using the network.

Three-machine network

Now suppose we want to connect a third machine named "admin" to the network.

The new machine can be connected to any of the existing machines. It is only necessary to connect it to one of them, but connecting to more than one might speed up the network by providing a more direct path between machines. For example, you could connect the new machine as follows:



This requires a total of 6 IO ports, 2 on each machine. But messages

will be sent directly between any pair of machines. Or you could connect this way:

sales -------- doc -------- admin

This network uses only 4 ports: 1 on sales, 2 on doc, and 1 on admin. The disadvantage is that messages between sales and admin have to go through doc. This will be done automatically, but it takes longer. Since the emphasis in the design of this network is low cost, the 4 port alternative is recommended. (But if you have no other use for the ports, go ahead and use all 6.)

Let's assume that you are going to use the straight-line connection (the 4 port alternative). First, you must choose a free serial port on doc, say ttyE, and one on admin, say ttyC. Connect these ports with a network cable, and change the configuration files as follows:

	/etc/netmap	/etc/rc	/etc/ttys
sales	sales doc admin	no change	no change
doc	doc admin sales	no change	ttyE 1200 net-admin
admin	admin doc sales	netdaemon	ttyC 1200 net-doc

General networks

The only restriction on the shape of the network is a physical one: a seperate serial port is required at each end of each cable. Otherwise, the network can be a straight line, a star, a ring, or any pattern. The rules to follow in setting up the 3 configuration files are as follows:

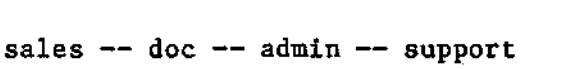
/etc/ttys -- Find the lines corresponding to each network port on the local machine. Put the word 1200 on each line, and in addition, put the word net-name, where "name" is the name of the machine to which that port is connected.

/etc/rc -- Put the command netdaemon in the file. It should be alone on a line. It should appear only once, no matter how many network ports there are on the machine. Optionally, it can be preceeded by an anouncement to remind the operator the network is being started.

/etc/netmap -- The name of the local machine should appear alone on the first line. Each succeeding line should begin with the name of a machine that is directly connected to the local machine. If there are additional machines on the network, their names should appear (in any order) following the name of the direct connection through which they can be reached. If there are several ways to reach an "indirect" machine, only one connection should be given: the one that leads to the machine via the shortest route.

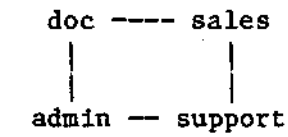
For example, let's look at several different networks, and a possible netmap file for the "doc" machine in each network.

Line



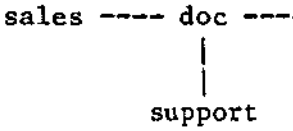

```
doc
sales
admin  support
```

Ring



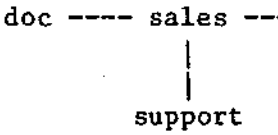

```
doc
sales
admin support
```

Star



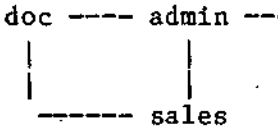

```
doc
sales
admin
support
```

Star




```
doc
sales admin support
```

Dipper




```
doc
sales
admin support
```

SEE ALSO

cp (1), mail (1), cables (4), ports (4)

NAME

null - the null device

DESCRIPTION

The null device has the curious property that everything written to it is completely ignored and thrown away. Reading from this device yields an end of file every time.

The minor device number is ignored. The major device number is zero in the current implementation.

This device may seem to be of limited usefulness, but don't get rid of it, the shell uses it to set up the standard input of background processes.

EXAMPLE

```
cat file > /dev/null
```

To read a file but discard the results.

NAME

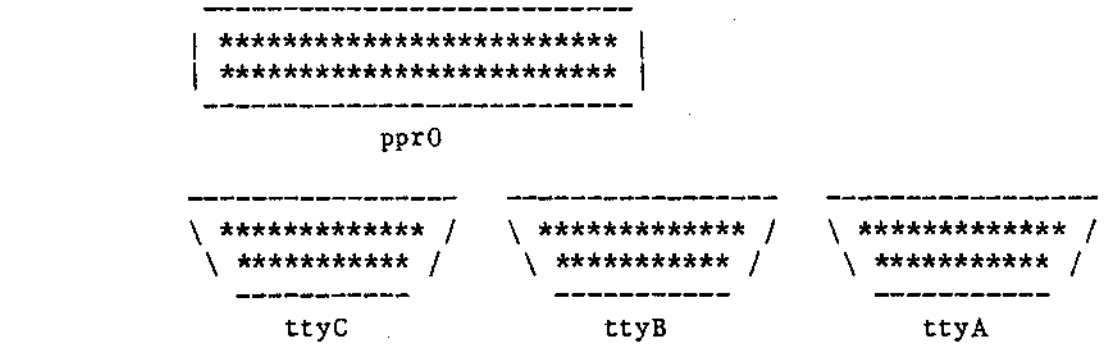
ports - how to identify the IO ports on the Decision

DESCRIPTION

The back of the Decision I has at least three "serial" ports and at least one "parallel" port. The serial ports look like thin sideways D's. They are called 25 pin female D connectors. (There is a smaller 15 pin female D connector that is not used.)

The parallel ports are 50 pin "flat cable" male connectors. The same kind of connector may also be used for disk drives, so you must look inside the computer to tell which is which. The connector is a parallel port if its cable goes to the "bus board" (on the tabletop model) or to a MULTIO board. It is a disk-drive port if its cable goes to a disk controller. (If you're not sure how to recognize a bus board, MULTIO board, or disk controller, you should ask your dealer for assistance.)

Once you have identified the serial ports and parallel ports, you should make sure that they are labeled. Here is a stylized picture of the back of a basic tabletop Decision I with the ports labeled:



The back of a rack-mount Decision looks slightly different, but the important thing is that on both models, the first serial port, called ttyA, is the right-most one (when looking at the computer from the back).

If you add a MULTIO board, its serial ports should be labeled as follows. Make sure that the board is plugged into the bus (with the component side of the board facing the the front of the computer), and that three flat cables are connecting the board to three D connectors at the back of the computer. (If you don't know how to do this, or if you aren't sure which MULTIO is the second one, you should ask your dealer for help.) Stand in back of the computer and look at the MULTIO. The three small connectors along the top of the

MULTIO are the serial "jacks"; the large one is the parallel jack. The D connector attached to the **right-most** (as seen from the back) serial jack should be labeled **ttyD**. The one attached to the **middle** serial jack should be labeled **ttyE**, and the D connector attached to the **left-most** (as seen from the back) serial jack should be labeled ttyF.

If you have a third or fourth MULTIO, then its serial ports should be labeled correspondingly as ttyG, ttyH, ttyI, and ttyJ, ttyK, ttyL.

The parallel ports attached to the second, third, and fourth MULTIOs should be labeled ppr1, ppr2, and ppr3. (Remember that the first one was called ppr0.)

NAMES

Under Micronix, the twelve possible serial ports are consistently refered to as ttyA through ttyL. (Some of the documentation that comes with the Decision I hardware refers to ttyA as the "console" or as "serial port 1", ttyB as "serial port 2", etc.) Names for the parallel ports are a little more involved. If ppr0 is connected to a parallel daisy-wheel printer, it is called "diab0". If it is connected via a special cable (available from Morrow) to a Centronics type printer, it is called "cent0".

IO board	references to serial ports	references to parallel port
first	ttyA ttyB ttyC	diab0 or cent0
second	ttyD ttyE ttyF	diab1 or cent1
third	ttyG ttyH ttyI	diab2 or cent2
fourth	ttyJ ttyK ttyL	diab3 or cent3

These names are arbitrary, by the way: they are not built into any program. But they are built into three other places:

1. They appear in the /dev directory as names for device files;

2. They appear in the /etc/ttys file in reference to these devices;

3. Finally, the curious program /bin/diablo (q.v.) may be linked to a name that matches a device name in /dev. If so, it will write to a parallel diablo printer attached to that device.

If you want to change any of these names, make sure to change all three occurrances.

SEE ALSO

lpr (1), diablo (1), multio (4), cables (4), printers (4), terminals (4), ttys (5)

NAME

printers - interfacing printers to Micronix

INTRODUCTION

This document explains what kinds of printers are supported by Micronix, how to plug them in, and how to test them. This is not a stand-alone document, however. You will have to refer to several others as you read. This will be easiest to do with the printed version of the Micronix Reference Manual. If you are reading this at your terminal, you can switch to another document by hitting DELETE or RUBOUT to get out of this one, and then typing the command "help" followed by the new document name. The related documents are:

ttys (5)

The document "configuration" in Section 5 (type the command "help configuration") explains how to configure Micronix for additional printers, automatic access from upm (the Micronix CP/M emulator), different baud rates, and hardware handshaking.

lpr (1)

The document "lpr" in Section 1 (type the command "help lpr") explains how to use the Micronix spooling program called lpr. This allows several users to print files without conflict.

upm (1)

The document "upm" in Section 1 (type the command "help upm") explains (amoung many other things) how to use printers from the Micronix CP/M emulator.

cables (4)

The document "cables" in Section 4 (type the command "help cables") explains how to build cables for various devices, including printers.

ports (4)

The document "ports" in Section 4 (type the command "help ports") explains how to identify the various I/O connectors on the Decision.

PRINTER INTERFACES

Micronix and the Decision come equipped to handle four types of printer interfaces. The interface refers to the physical connection, the cable, between the printer and the Decision. The interface also refers to the type of "handshaking" used by your printer. This is the method it uses to stop the flow of characters from the

computer when it gets behind. The four types of interfaces are:

1. Serial (or "RS-232") printers with hardware handshaking,

2. Serial printers with software ("XON XOFF") handshaking,

3. Parallel printers with Centronics-standard interfaces, and

4. Parallel printers with Diablo-standard interfaces.

If you have either the first or second type of printer, you're in luck. You can skip the rest of this section and start reading PLUGGING IN THE PRINTER. If you don't have one of these two types, or are unsure of which type you are using, read on.

Serial printers use either software handshaking or hardware handshaking. Software handshaking means that when the printer's buffer is full it sends a "byte" to the computer to temporarily stop the flow of characters. The printer sends a different byte when it is ready to receive again. The byte that Micronix recognizes for stopping transmission is X-OFF. Other aliases for X-OFF are control-S, DC3 and 13 hex. The byte that restarts output is X-ON, also known as control-Q, DC1 and 11 hex. If your printer uses X-ON and X-OFF, then you can skip ahead to the section PLUGGING IN YOUR PRINTER. If you have a serial printer that doesn't use X-ON and X-OFF (for example, it uses ETX and ACK instead), you'll have to try hardware handshaking.

Hardware handshaking makes use of one of the wires in the printer cable to start and stop output from the computer. When the printer wants to stop transmission, it lowers the voltage on this wire. To continue transmission, the voltage is raised again. Unfortunately, there is no real standard for which wire the printer should use. Hardware handshaking between a printer and the Decision will probably require a special cable.

If your printer requires hardware handshaking, you will have to do two things: (1) read the document "cables" in Section 4, and then make or purchase a suitable cable, and (2) read the document "configuration" in Section 4, and then tell Micronix that your printer needs hardware handshaking.

Parallel printers also require special cables (which you can either make or purchase from Morrow). Please read the "cables" document, then return to the next section and follow instructions for parallel printers.

PLUGGING IN YOUR PRINTER

In the discussion that follows, we will assume that you are going to plug a serial printer into the port called "ttyC", and/or a parallel printer into the port called "ppr0". If you don't know how to find these ports, please read the document called "ports". If you want to use other ports, you will have to tell Micronix which ports you are using. The document "configuration" tells how to do this.

The serial printer can only be plugged in one way, so you don't have to worry about which way to orient the connector. However, the 50 pin parallel connector is not "keyed": that is, it is quite possible to plug it in upside down. Read on for some guidance. On many cables, pin 1 is labeled by having a red stripe on that edge. Or, if you constructed your own cable, you should know where pin 1 is. The 50 pin connector on the back of your Decision has pin 1 on the left hand side when looking at the back of the printer. Plug the parallel cable into the Decision with the red stripe, and pin 1, toward the left.

TESTING YOUR PRINTER

Plug in your printer, turn it ON, and make sure that it is ON-LINE and set to "1200 baud". (These settings are usually governed by switches on the printer. Sometimes the switches are on the inside, especially the baud rate switches.) If you can't set your printer to 1200 baud, you will have to change Micronix to transmit at whatever rate is acceptable. The document called "configuration" explains how to do this.

For serial printers plugged into ttyC, type the command

```
echo A > /dev/ttyC
```

There are two different test commands for parallel printers. The difference between them is the way the device is named. For parallel daisy-wheel printers plugged into ppr0, type

```
echo A > /dev/diab0
```

and for Centronic style parallel printers plugged into

ppr0, use

```
echo A > /dev/cent0
```

Did you get an "A" on your printer? If not, remember that interfacing printers IS non-trivial. First, rule out the trivial problems by checking to see if:

1. the printer is turned on (and plugged in),

2. the printer is set to ON-LINE,

3. there is paper in it (many printers won't print unless they can sense a piece of paper in them), and

4. the cable is plugged in at both ends at the correct connector.

If you have a parallel cable, you can also try plugging it in upside down. After trying all of the suggestions, try the suggested command again, please. If you get a different letter than an "A", there may be some wires scrambled in your cable, or your serial printer may be set to the wrong baud rate.

If none of these suggestions helps, your cable is suspect. Home built cables should be rechecked and store bought cables checked for flaws. RS-232 cables, whether home-made or purchased, can be checked by using them to replace the cable connecting your terminal. Parallel cables can be checked for continuity.

Micronix Operating System reference manual

files

MORROW

CONTENTS OF REFERENCE SECTION 5: FILES

NAME

ascii - american standard code for information interchange

SYNOPSIS

cat /usr/pub/ascii

DESCRIPTION

The file /usr/pub/ascii serves as a cross reference guide for conversion between ascii character names and numeric values. Values are given in base 16 and base 8.

The file contains:

00	nul	01	soh	02	stx	03	etx	04	eot	05	enq	06	ack	07	bel
08	bs	09	ht	0a	nl	0b	vt	0c	np	0d	cr	0e	so	0f	si
10	dle	11	dc1	12	dc2	13	dc3	14	dc4	15	nak	16	syn	17	etb
18	can	19	em	1a	sub	1b	esc	1c	fs	1d	gs	1e	rs	1f	us
20	sp	21	!	22	"	23	#	24	$	25	%	26	&	27	'
28	(	29	)	2a	*	2b	+	2c	,	2d	-	2e	.	2f	/
30	0	31	1	32	2	33	3	34	4	35	5	36	6	37	7
38	8	39	9	3a	:	3b	;	3c	<	3d	=	3e	>	3f	?
40	@	41	A	42	B	43	C	44	D	45	E	46	F	47	G
48	H	49	I	4a	J	4b	K	4c	L	4d	M	4e	N	4f	O
50	P	51	Q	52	R	53	S	54	T	55	U	56	V	57	W
58	X	59	Y	5a	Z	5b	[	5c	\	5d	]	5e	^	5f	_
60	`	61	a	62	b	63	c	64	d	65	e	66	f	67	g
68	h	69	i	6a	j	6b	k	6c	l	6d	m	6e	n	6f	o
70	p	71	q	72	r	73	s	74	t	75	u	76	v	77	w
78	x	79	y	7a	z	7b	{	7c	|	7d	}	7e	~	7f	del

000	nul	001	soh	002	stx	003	etx	004	eot	005	enq	006	ack	007	bel
010	bs	011	ht	012	nl	013	vt	014	np	015	cr	016	so	017	si
020	dle	021	dc1	022	dc2	023	dc3	024	dc4	025	nak	026	syn	027	etb
030	can	031	em	032	sub	033	esc	034	fs	035	gs	036	rs	037	us
040	sp	041	!	042	"	043	#	044	$	045	%	046	&	047	'
050	(	051	)	052	*	053	+	054	,	055	-	056	.	057	/
060	0	061	1	062	2	063	3	064	4	065	5	066	6	067	7
070	8	071	9	072	:	073	;	074	<	075	=	076	>	077	?
100	@	101	A	102	B	103	C	104	D	105	E	106	F	107	G
110	H	111	I	112	J	113	K	114	L	115	M	116	N	117	O
120	P	121	Q	122	R	123	S	124	T	125	U	126	V	127	W
130	X	131	Y	132	Z	133	[	134	\	135	]	136	^	137	_
140	`	141	a	142	b	143	c	144	d	145	e	146	f	147	g
150	h	151	i	152	j	153	k	154	l	155	m	156	n	157	o
160	p	161	q	162	r	163	s	164	t	165	u	166	v	167	w
170	x	171	y	172	z	173	{	174	|	175	}	176	~	177	del

NAME

banner - the log-in banner

FUNCTION

The banner file is printed on each terminal just prior to the Name: prompt for the user login name. This file is inessential to system operation. It may be edited to the user's preference, or deleted.

FILES

/etc/banner - banner file.

NAME

core - core dump file

FUNCTION

A core file is produced by the system as a result of certain fatal signals; the offending program is then terminated. The file produced is named "core" in the current directory of the dumped program at the time of the fatal signal. The format of the core dump file is as follows:

The first 65,536 bytes are simply a "snapshot" of the program as it appeared in memory at the critical instant. Following this "memory image" is certain other pertinent information, mostly the contents of the Z80's registers:

```
    Task: Micronix task number
    Mask: The MPZ80 permission mode
    PC:   Program counter
    SP:   Stack pointer
Registers:
    af
    bc
    de
    hl
    zir
    zix
    ziy
    zaf
    zbc
    zde
    zhl
```

This section assumes previous knowledge of the Z80 processor.

NAME

device - printer spooler device description file

FUNCTION

The line printer spooler program searches for and follows the directives found in this file. The file is entered free-form, that is, order and spacing of specifiers given within this file are unimportant.

The following is a list of specifiers that may appear:

A baud rate given as a decimal number. Example baud rates: 9600, 1200, 300, 110, etc. All the standard baud rates are supported. The default baud rate is 9600.

An indication to use hardware handshaking, i.e., to transmit only when the clear-to-send pin is asserted. The indicator is simply the word "shake". By default, clear-to-send is ignored.

The name of the device to which to send the spooled files for printing. This must be the full path name of a character special file. Examples: /dev/ttyA, /dev/ttyB, /dev/ttyC, /dev/pprA, /dev/ttyD, etc. Default is /dev/lp.

FILES

/usr/spool/*/device

EXAMPLE

A complete typical "device" file follows:

```
/dev/ttyc 1200
```

NAME

directory - the format of directories.

FUNCTION

Micronix uses the standard UNIX version 6 directory format. A directory behaves like an ordinary file except that a user may write into a file, but cannot write into a directory.

A directory is distinguished from a file by a bit in the flag word of its i-node entry. Directory entries are 16 bytes long. The first word is the i-number of the file, represented by the entry if non-zero.

If the first word is zero, the entry is empty.

Bytes 2 through 15 of the entry represent the 14-character file name, null padded on the right. These bytes are not cleared for empty slots.

By convention, the first two entries in each directory are for "." and "..". The first is an entry for the directory itself. The second is for the parent directory.

The meaning of ".." is modified for the root directory of the master file system and for the root directories of removable file systems. In the first case, there is no parent, and in the second, the system does not permit off-device references. In both cases ".." has the same meaning as ".".

ALSO READ

filesystem (5)

NAME

dtab - the dump table

FUNCTION

Information on the times of a dump are kept in the file "/etc/dtab." This is a non-ASCII file presently used only by **td** (1). Each entry in **dtab** is eight bytes long and takes the form:

```
struct dtab
 {
 int     device,
         inode;

 long    date;
 };
```

There is an entry in **dtab** for each directory that has been dumped by **td**. Device is the major/minor device number of the directory in question (major device number in the high byte, minor in the low byte). Inode is the inode number of the directory. Date is the time of the last recorded dump of the directory in question expressed as a 4-byte quantity representing the number of seconds elapsed since Jan. 1, 1970 at midnight.

FILES

/include/dtab.h

ALSO READ

td (1)

NAME

filesystem - format of UNIX version 6 file systems

FUNCTION

Every file system storage volume, (hard or floppy disk) has a common format for certain vital information. Every such volume is divided into a certain number of 256 word (512 byte) blocks. Block 0 is unused.

Block 1 is the super block. Starting from its first word, the format of a super-block is as follows.

```
struct
      {
      unsigned
              isize,
              fsize,
              nfree,
              free [100],
              ninode,
              inode [100],

      char    flock,
              ilock,
              fmod;

      long    time;
      };
```

Isize is the number of blocks devoted to the i-list which starts just after the super-block, in block 2.

Fsize is the first block not potentially available for allocation to a file. These numbers are used by the system to check for a bad block number. If an impossible block number is allocated from the free list, or is freed, a diagnostic is written on the on-line console. Moreover, the free array is cleared to prevent further allocation from a presumably corrupted free list.

The free list for each volume is maintained as follows. The free array contains, in **free[1], ... , free[nfree-1]**, up to 99 numbers of free blocks. **Free[0]** is the block number of the head of a chain of blocks constituting the free list. The first word in each free-chain block is the number (up to 100) of free-block numbers listed in the next 100 words of the chain member. The first of these 100 blocks is the link to the next member of the chain.

To allocate a block: decrement **nfree** and the new block is **free[nfree]**.

If the new block number is 0, there are no blocks left and an error is given.

If **nfree** becomes 0, read in the block named by the new block number, replace **nfree** by its first word, and copy the block number in the next 100 words into the **free** array.

To free a block, check if **nfree** is 100; if so, copy **nfree** and the **free** array into it, write it out, and set **nfree** to 0.

In any event, set **nfree[nfree]** to the freed blocks's number and increment **nfree**.

Ninode is the number of free i-numbers in the inode array. To allocate an i-node: If **ninode** is greater than 0, decrement it and return **inode[ninode]**.

If **ninode** = 0, read the i-list and place the number of all free inodes (up to 100) into the **inode** array, then try again. To free an i-node, provided **ninode** is less than 100, place its number into **inode[ninode]** and increment **ninode**.

If **ninode** is already 100, do not bother to enter the freed i-node into any table. This list of i-nodes is used only to speed up the allocation process; the information as to whether the inode is really free or not is maintained in the inode itself.

Flock and **ilock** are flags maintained in the core copy of the file system while it is mounted; their values on disk are immaterial. The value of **fmod** on disk is likewise immaterial; it is used as a flag to indicate that the super-block has changed and should be copied to the disk during the next periodic update of file system information.

Time is the last time the super-block of the file system was changed and is a long representation of the number of seconds that have elapsed since midnight January 1st 1970 (GMT). During a reboot, the time of the super-block for the root file system is used to set the system's idea of the time.

I-numbers begin at 1 and the storage for i-nodes begins in block 2. Also, i-nodes are 32 bytes long, so 16 of them fit into a block. Therefore, i-node i is located in block (i + 31) / 16, and begins 32 * ((i + 31) (mod 16)) bytes form its start. I-node 1 is reserved for the

root directory of the file system, but no other i-number has a built-in meaning. Each i-node represents one file. The format of an i-node is as follows.

```
struct
      {
      int     flags;
      char    nlinks,
              uid,

              gid,
              size0;

      int     size1,
              addr[8],

      long    actime,
              modtime;
      };
```

The flags are as follows:

100000	i-node is allocated
060000	2-bit file type:
000000	plain file
040000	directory
020000	character-type special file
060000	block-type special file
010000	large file
004000	set user-ID on execution
002000	ignored
000400	read permission for owner
000200	write permission for owner
000100	execute permission for owner
000070	read, write, execute permission for group
000007	read, write, execute permission for others

Special files are recognized by their flags, not by i-number.

A block-type special file is one which can potentially be mounted as a file system, a character-type special file cannot, though it is not necessarily character-oriented. For special files the high byte of the first address word specifies the type of device and the low byte specifies one of several devices of that type. The device type numbers of block and character special files overlap.

The address words of ordinary files and directories contain the number of the blocks in the file (if it is small) or the number of indirect blocks (if the file is

large). Byte number N of a file is accessed as follows: N is divided by 512 to find its logical block number, use b as an example, in the file. If the file is small (flag 010000 is 0), b must be less than 8, and the physical block number is **addr[b]**.

If the file is large, b is divided by 256 to yield i. If i is less than 7, then **addr[i]** is the physical block number of the indirect block. The remainder from the division yields the word in the indirect block which contains the number of the block for the sought byte.

If i is equal to 7, the file has become extra large and **addr[7]** is the address of a first indirect block. Each word in this block is the number of a second level indirect block; each word in the second-level indirect blocks points to a data block. Notice that large files are not marked by any mode bit, but by having addr[7] non-zero. Even though this scheme allows for more than 33,554,432 (256 X 256 X 512) bytes per file, the length of files is stored in 24 bits. In practice, a file can be at most 16,777,216 bytes long.

For block b in a file to exist, it is not necessary that all blocks less than b exist. A zero block number in either address words of the i-node, or in an indirect block, indicates that the corresponding block has never been allocated. This type of missing block reads as if it contained all zero words.

SEE ALSO

icheck (1), dcheck (1), fsck (1)

NAME

group - the group file

FUNCTION

Each user name is associated with a group number. Associated with each file is a user ID number (the owner of the file) and a group ID number (the group of the file). These numbers are used in permission determinations.

The relationship between group numbers and names is established by the group file. The format of the group file is as follows:

Each line of the file defines one group name.

The group name is followed by a colon, then followed by the encrypted group password, if one exists. This is followed by a colon, which is followed by the group number (in decimal), followed by a colon, followed by the group comment field which may contain anything you desire, or nothing at all.

Here is a sample group file:

```
sales::4:
stock::5:
```

Normally group 0 is reserved for the super-user only. The **newuser** program makes additions to this file.

SEE ALSO

passwd(5)

NAME

motd - message of the day

FUNCTION

The message of the day file is printed on each users' terminal after the proper login sequence is completed.

The contents of this file are inessential to system operation. This file is intended to be updated frequently by the system administrator. It may be used to inform the users of changing system conditions or upcoming events. This file may be deleted from the system if the user so wishes.

SEE ALSO

banner (5), signon (5)

NAME

mtab - mounted file system table

FUNCTION

Mtab resides in the /etc directory and contains a table of devices "mounted" by the mount command. The umount command removes entries.

Each entry is 64 bytes long. The first 32 bytes are the null padded name of the place where the special file is mounted; the second 32 are the null padded name of the special file. The special file has all its directories stripped away, that is, everything through the last '/' is thrown away.

This table is present only so people can look at it. It does not matter to mount if there are duplicate entries, nor to umount if a name cannot be found.

FILES

etc/mtab

SEE ALSO

mount (1), umount (2)

NAME

passwd - password file

FUNCTION

The name passwd has become a misnomer, but is commonly found in UNIX documentation and we hesitate to change it here.

Passwd contains the following information:

name (login name, no upper case letters)
encrypted password
numerical user ID
numerical group ID
comment
home directory
command interpreter program name

This is an ASCII file. Each field within each user's entry is separated from the next by a colon. Each user is separated from the next by a newline. The encrypted passwords are kept in a separate file, the contents of the password entry are immaterial. This file resides in the "/etc" directory.

The first entry of the password file contains an entry for the super-user account. The name of this account is traditionally "root," but may be renamed to anything the user wishes.

EXAMPLE

root::0:0:super:/:/bin/sh

sally::10:1:sally:/a/sally:/bin/sh

FILES

/etc/passwd

SEE ALSO

login (5), passwd (5), group (5)

NAME

rc - multi-user start-up file

FUNCTION

The **init** (initialization) program looks for the "/etc/rc" file just before entering the multi-user mode. If such a file is found, and is readable, it is run as a shell script. That is, each of the commands in the file is executed as if from the keyboard.

Typically, it is used to clean out the temporary directory, remove locked files, start up daemons, etc. The user may place whatever he desires in the file.

FILES

/etc/rc

NAME

signon - file for the signon prompt

FUNCTION

The signon file is printed on the system console each time Micronix is booted. The contents of this file is inessential to system operation. It is merely there to display system-oriented information. It may be edited to suit the user, or deleted from the system.

FILES

/etc/signon

NAME

terminals - the terminal capabilities file

FUNCTION

The terminals file, always located in the /etc directory, contains the list of terminals and their capabilities. The information in this file is used by upm for terminal emulation. The information here is also available to other programs that make use of the features of intelligent terminals, such as, cursor addressing, clear screen, erase to end of screen, etc.

The terminals file is broken into descriptions of terminals by the terminal name lines, which always begin at the leftmost margin (column 1). The first word on the line is the "terminal abbreviation", which should be short and mnemonic. This should be followed by the full name of the terminal, which is free-form. This full name will appear automatically in the terminal menus presented by the reconfiguration program. See recon (1).

The lines that follow each terminal name line define the characters that the terminal uses for control of special effects (cursor movement, clearscreen, highlighting, etc.).

Terminals can be divided into three categories according to their capabilities. Grouping terminals into these three "levels" allows the use of only, at most, three different versions of terminal-configured software. The first level of terminals, the simplest level, is defined by terminals having the following capabilities:

LEVEL 1

Keyword	Description
cursor	string of characters to move cursor
home	home the cursor *
up	character(s) to move the cursor up *
down	character(s) to move the cursor down *
right	character(s) to move the cursor right *
left	character(s) to move the cursor left *
clear	clear the screen and home cursor

* (Optional, since most programs actually do not use.)

The keywords are used one per line, preceded by white-space (tabs or spaces) that distinguish the keywords from terminal names. The characters that complete the keyword definition follow, separated by more whitespace. A newline terminates each definition. The keywords that define the next two levels are:

LEVEL 2

Keyword	Description
ceos	clear to end of screen
high	begin highlighting *
low	return to normal intensity *

* (Optional, since most programs actually do not use.)

LEVEL 3

insline	insert a line
inschar	insert a character in a line *
delline	delete a line
delchar	delete a character *

* (Optional, since most programs actually do not use.)

The keywords must be spelled exactly as shown here, and must always be in lower case. When adding a terminal to the **terminals** file, try to define as many capabilities as possible. If you are deciding what level your terminal fits, your terminal must fulfill ALL of the categories in the lower levels before you can consider it as fitting in a higher level. Three other keywords may optionally be used: like, init and exit. The characters following init may be sent by a program to initialize a terminal; those following exit may be sent just before a program terminates.

The like keyword is used to include a previously defined terminal's keywords and their definitions for the terminal currently being defined. Differences between a terminal and one that it is "like" are adjusted by redefining keywords. You can only use three levels of like-ness, that is, the terminal you are describing can be "like" a previously defined terminal that is "like" another terminal, that is "like" some fully described terminal. The like keyword should be the first keyword used in a description; if you use it last, the keywords you were trying to redefined will be overwritten by the definition of the "like" terminal.

The keywords are followed by the characters that define them, separated by spaces. There are a number of abbreviations and operators that are used with the keyword definitions. The abbreviations that are recognized are:

ABBREVIATIONS RECOGNIZED

bs	backspace
cr	carriage return
esc	escape
ff	formfeed
nl	newline

null	0-byte
sp	space
tb	tab
vt	vertical tab

col	"
ascii	"
+	"
'	"
^	"
numbers	"

other all other single characters are sent literally

These abbreviations provide a way of including human readable characters, as opposed to using their octal equivalents in keyword definitions.

Several operators are also provided for keyword definitions. These operators modify the characters that follow them, or are used to include the row and column in cursor definitions.

OPERATORS

row + n	Send the line number plus an offset n as a single byte. The first row on the screen is row 0. Most terminals use row + 32.
col + n	Send the column number plus an offset n as a single byte. The first column on the screen is column 0. Most terminals use column + 32.
row ascii	Send the row number as a series of ascii digits (with leading zeros supressed). Used by ANSI-standard terminals.
col ascii	Send the column number as a series of ascii digits (with leading zeros supressed). Used by ANSI-standard terminals.
'	Send the following characters literally with no conversion. Eg, '0 is the same as 48 or x30 (see x below).
^	Convert the character that follows to the corresponding control code
xnn	Interpret the number nn that follows

as hexadecimal. (Any number of digits. Converts to one byte.)

Onnn — Interpret the number nnn that follows as octal. (Any number of digits. Converts to one byte.)

The row and col operators must always be followed by a plus sign and an offset (which may be zero), or by the word ascii. (Each of these must be separated by a space.) into their ascii values (between 48 and 57). The caret (^) indicates control keys, for example, ^H represents a backspace. If you prefer to use hexadecimal or octal notation, use x or 0 to indicate the type of conversion desired. Numbers are converted to a single byte (value 0 to 255). Strings or characters preceeded by a single quote (') are sent literally. For example, 0123 sends a sends the byte with the octal value 123 (this is the letter S), while '0 '1 '2 '3 sends the four ascii bytes 0, 1, 2, and 3.

EXAMPLE

```
mor20   Morrow 20
        like    proto3
        cursor  esc = col + 32 row + 32
        clear   ^Z
```

This example defines a terminal, abbreviated as mor20, as being "like" the proto3 definition, then goes on to redefine the cursor and clear keywords. The sequence of characters used to move the cursor is: an escape, an "=", the column number added to 32, and the row number added to 32. The character that clears the screen is a Control-Z.

If you need to create an entry for a terminal not in the terminals file, use the list of keywords for the three levels and find the character(s) used by your terminal for as many keywords as possible. These characters are usually buried in a table in your terminal's manual describing control sequences or special characters. You should always be able to find all the control sequences necessary for a level 1 terminal at a minimum. The names of the various functions, for example, home or highlight, will vary unpredictably from manual to manual.

NOTE

The shorter the terminals files is, the quicker it may be searched for matching terminal types. You may wish to rename (mv) the terminals file, and keep an edited version with just the terminals you normally use in the file /etc/terminals.

FILES

/etc/terminals

SEE ALSO

upm (1)

NAME

/etc/ttys - Micronix terminal and printer configuration

SUMMARY

(This is a technical summary. Explanatory examples follow.)

The /etc/ttys file specifies the Micronix terminal and printer configuration. It is a humanly-readable text file consisting of lines with whitespace-seperated words. Word order is not significant.

The first word on each line is the name of a "character special" device file (one of the files in the /dev directory). The rest of the words on the line apply to that device. The following words are meaningful:

1. Any valid baud rate: 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, or 19200. The system will set the baud rate as requested. This is done by the init process (a) when the system is powered up, (b) each time a user logs off, (c) each time the /etc/ttys file is changed (but only to devices that are not currently logged in). See **init, login, and update** in section 1 for more information.

2. The word "shake". This refers to RS-232 Clear-To-Send handshaking. The system will use this type of handshaking only if requested (since some devices do not support it). XON-XOFF (software) handshaking is done by default. The init process sets or resets this mode under the same circumstances as in #1 above. (See **printers** in section 4 for more information on handshaking.)

3. The word "login". When the system goes multi-user, a login process will be created for each device designated as a "login" terminal (see **init** in section 1).

4. The word "term=" followed by a terminal mnemonic. upm reads the ttys file and provides terminal emulation for the model of terminal named after the word "term=". The terminal named is matched against an entry in the /etc/terminals file, for the purpose of translating generic control codes into the exact codes expected by the terminal. (See **terminals** in section 5).

5. The word "lpr", or the name of any link to lpr (such as "dpr", "xpr", etc). When the lpr ("line printer") program is invoked, it first looks to see what name it was called by. It then reads the ttys file looking for its name. If found, it uses the device on the same line as its printer. Thus, several printers can be supported simply by keeping several copies of the lpr program under different names (see lpr in section 1).

6. The word "lst". When upm (the CP/M emulator) is invoked, it

reads the ttys file looking for the word "lst". If found, it uses the corresponding device as its LST: device (ie, its printer), unless it is told otherwise (by a command line argument, a command in the .upm file, or a command given in interactive mode). See upm in section 1.

7. A word beginning with "net-". The rest of the word is taken to be the network name of the machine to which that port is connected (not the local machine, but the machine at the other end of the cable). See network in section 4.

DON'TS

Don't use a login port for anything else. Don't specify "shake" or a baud rate for a parallel port (the "cent's" and the "diab's").

Don't specify one of the "diab's" as the lst device. (It takes a lot of code to drive the Diablo-type parallel printers, and this was left out of upm to save space.) If you want to use diab0 as the upm lst device, then you should do the following. Change to the /bin directory and make a link called "diab0" to the "diablo" program. Then change to your home directory and put "lst:|diab0" in your .upm file.

EXAMPLES

The file /etc/ttys (that is, the file found by starting in the root directory /, proceeding to the subdirectory etc, then to the file ttys) tells the system about its terminals and printers. More specifically, it tells which of the Decision's I/O connectors are attached to terminals, which are attached to printers, and what should be done with each of them.

The easiest way to understand it is to look at a sample ttys file:

```
ttyA  9600  login  term=tvi925
ttyB  9600  login  term=mor20
ttyC  1200  shake  lpr
ttyD  1200  net-sales
ttyF  1200  net-admin
ttyG  1200  login
ttyH
ttyI
ttyJ
ttyK
ttyL
cent0  xpr  lst
cent1
cent2
cent3
diab0
diab1
```

```
diab2
diab3
```

The first word on each line is the name of one of the I/O connectors on the back of the Decision, and also the name of the corresponding "character device" file in the /dev directory. By convention, the terminal that is attached to the I/O connector is also called by the same name. (So when you see "ttyA", you may have to infer from the context which of these three things we are talking about: the device file, the I/O connector, or the terminal itself.)

This example says that ttyA should be run at 9600 baud, and that it is a login terminal. When upm is invoked, it will send the correct control codes for a Televideo 925 (term=tvi925) to the terminal connected to ttyA, and use software configured for a ADM-31. The same is true for ttyB, except that upm will use control codes for the Morrow MD20 terminal.

TtyC should be run at 1200 baud, and needs hardware handshaking. (See printers in section 4 for more information on handshaking.) The "lpr" says that ttyC will be used by the lpr ("line printer") program (so it is presumably a printer).

TtyD and ttyE are network ports. Both run at 1200 baud. TtyD is connected to the machine named "sales", while ttyE is connected to "admin".

TtyF is a login device running at 1200 baud (so it is probably a modem). There isn't a "term=" entry on this line, so upm will not provide terminal emulation for ttyF.

The only other device that is is currently attached to the system is a Centronics-type printer called "xpr". It is attached to parallel port 0, and thus to the device named "cent0". (If it were a Diablo-type parallel printer, it would appear on the line with "diab0"). The "lst" means that the CP/M emulator upm will use this device as its LST: device (its printer), unless told otherwise by a later command.

CHANGING THE TTYS FILE

The recon program is used to recon-figure the ttys file. The recon program understands all the rules for using or modifying the ttys file, and will also interpret the /etc/terminals file for you.

Since this is an ordinary text file, it can also be changed with your favorite text editor. If you use WordStar, open the file in "non-document" mode. Here is a sample session with the Micronix line editor "edit".

Let's say that we want to change the "lst" device from cent0 to ttyC. First log in as "root" (since only the super-user is

allowed to change the ttys file). In the lines below, the '#' is the super-user shell prompt, and the ':' is edit's prompt.

```
# cd /etc                          Change directory to /etc.
# edit ttys                        Edit the ttys file.
"ttys" [reading] 19 lines          The editor prints this.
:/cent0                            Look for the line containing cent0.
cent0  xpr  lst                    The editor prints it.
:c                                 Change the line.
cent0  xpr                         You type the new line.
cent0  xpr                         The editor prints the new line.
:/ttyC                             Look for the line containing ttyC.
ttyC  1200  shake  lpr             The editor prints it.
:c                                 Change the line.
ttyC  1200  shake  lpr  lst        You type the new line.
ttyC  1200  shake  lpr  lst        The editor prints the new line.
:w                                 Write out the updated file.
"ttys" [writing] 19 lines          The editor prints this.
:q                                 Quit the editor.
#
```

(There are faster ways to use edit, but these are the easiest to explain.)

SEE ALSO init (1), login (1), lpr (1), recon (1), stty (1), update (1), upm (1), network (4), printers (4), ports (4), cables (4).

NAME

utmp - user information

FUNCTION

This file offers information on who is currently using Micronix. The file is binary. Each entry is described in the structure illustrated below:

```
struct utmp
      {
      char line[8],   /* typewriter name */
           name[8];   /* login name */

      long    time;   /* login time */
      };
```

Line is the name of tty (null padded) on which the user is logged in. Name is the (null padded) name of the user who logged in. Time is the time of login, expressed as a 32-bit quantity representing the number of seconds since the epoch (midnight, January 1st, 1970).

FILES

/etc/utmp
/include/utmp.h

SEE ALSO

NAME

wtmp - user login history

FUNCTION

This file records all logins and logouts. Its format is like **utmp** (5) with these exceptions: A null user name indicates a logout on the associated typewriter. Furthermore, the typewriter name ~ indicates the system was rebooted at the indicated time; the adjacent pair of entries with typewriter names | and } indicate the system-maintained time just before and just after a **date** command changed the systems' idea of the time. **Wtmp** is maintained by **login** (1) and **init** (1). Neither of these programs creates the file, so if is is removed, record-keeping is turned off.

FILES

/usr/adm/wtmp
/include/utmp.h

SEE ALSO

utmp (5), login (1), init (1), who (1)

Micronix Operating System reference manual

MORROW

other programs