

Solaris-Minix 2.0.0 User Guide for COP 4600

© 1998 Timothy A. Davis
CISE Department
University of Florida

Revised: January 2000

This document describes how to run, modify, and compile Solaris-Minix 2.0.0 on the CISE departmental computers. See *Operating Systems: Design and Implementation, A. S. Tanenbaum, Prentice-Hall, 2nd edition*, and *Smx - the Solaris port of Minix, P. Ashton*, for more details. The latter document, as well as this document, are located in `/cise/class/cop4600/minix/doc`.

Registering for a Minix Directory

Before you may begin working with Minix, you must sign up for a Minix directory by visiting <http://www.cise.ufl.edu/~mconnor/cop4600/> and filling out the form. This form requires you to have already obtained a valid CISE account. For instructions on obtaining a CISE account, visit the computer lab in room CSE 114 and ask the person(s) at the help desk for assistance.

After completing the online registration form, you will be placed on a waiting list for a Minix directory. Every few days (depending on the number of persons on the waiting list), the list is sent to the system administrators to be processed. The system administrators will create a directory for you, located at `/cise/class/cop4600/class/<username>`, in which your copy of Minix will later reside. For instance, if your username is `mconnor`, your Minix directory will be `/cise/class/cop4600/class/mconnor`. You should check for this directory every day or two. You may or may not receive an email notifying you when your directory has been created.

Registering for a Minix directory is a very important step, which should be completed within the first couple weeks of class. In addition to creating a Minix directory for you, registration will place you in the class roster that will be used to create the grade spreadsheet.

Setting Up Your Minix Directory

Run *minix_setup*

The command `minix_setup` places a copy of Solaris-Minix 2.0.0 in your COP 4600 directory and configures it for your use. It also creates a *symbolic link* to this directory from your home directory, with the name `cop4600`. To get to your COP 4600 directory, you can either type

```
cd /cise/class/cop4600/class/$user
```

or the more simple form,

```
cd ~/cop4600
```

A few rules regarding the use of you Minix directory:

- Use it for COP 4600 only. All other material will be deleted, *possibly without warning!* The directory will be deleted at the end of the semester.
- Keep your directory protected, so that other students can't read it. This is the default, but if you mess up the protections, type `minix_protect` to restore them.

Running Minix

```
Type cd ~/cop4600/src/tools  
Run minix  
Login as root
```

The `minix` command

From your COP 4600 directory, `cd` to the `src/tools` directory, and type `minix`. You can do this in any sized X-window, but 80-by-24 is best if you want to use an editor in Minix. Note that you won't need to compile any code inside Minix. In fact, there is no compiler inside Minix. The Minix source code editing and compilation are done outside of Minix (in `src/kernel`, etc.). You will see something like the following when you run `minix`:

```
% minix  
lock file: /cise/class/cop4600/class/davis/lockfile  
RAM file is /tmp/minixAAAa001r0, of size 3407872 bytes  
Loading module kernel from ./image ... ok  
Loading module mm from ./image ... ok  
Loading module fs from ./image ... ok  
Loading module inet from ./image ... ok  
Loading module init from ./image ... ok  
  
Minix 2.0.0 Copyright 1997 Prentice-Hall, Inc.  
  
eth_init0: got reply for wrong port  
arp.c: unable to open ethernet  
ip.c: unable to open eth port  
  
Memory size = 3328K   MINIX = 1176K   RAM disk = 0K   Available = 2152K  
  
/dev/hdx1 is read-only mounted on /usr/bin  
/dev/hdx2 is read-only mounted on /usr/man  
Starting standard daemons: update.  
-----  
Welcome to Solaris-Minix, modified for COP 4600 by Prof. Tim Davis, Sept. 1997.  
Valid logins are root (superuser), bin (owner of binaries), and ast. No  
passwords are required. To exit minix, type "shutdown" as root or bin.  
-----  
  
Solaris-Minix Release 2.0 Version 0  
  
noname login:
```

Ignore the ethernet error messages. Log in as `root` (you can also log in as `bin` or `ast`). No passwords are required. You have access to three Minix file systems while running Minix. Typed `df` and you'll see:

```
# df
```

Device	Inodes total	Inodes used	Inodes free	Blocks total	Blocks used	Blocks free	Mounted on	V Pr
/dev/hdx0	1376	189	1187	4096	3078	1018	/	2 rw
/dev/hdx1	3424	212	3212	10240	9792	448	/usr/bin	2 ro
/dev/hdx2	688	382	306	2048	1443	605	/usr/man	2 ro

The three file systems shown above are three single *files* in Solaris. The root file system (4 Mbytes, or 4096 1K Blocks) is your own copy, in `disks/root`. You can both read and write to this file system. If you corrupt this file system, you can get a fresh copy with the command (typed while *outside* Minix):

```
cp /cise/class/cop4600/minix/disks/root ~/cop4600/disks
```

The other two, `/usr/bin` and `/usr/man`, are read-only (see the last column of the `df` output). You share them with all other students. Type `cd ; dir -R | more` to see everything in these three file systems (`dir` is an alias for `ls`). Everything you see in this directory listing is in one of the three Solaris files, `disks/root`, `disks/usrbin`, or `disks/man`. Typing `ps -alx` will show you all the tasks, server processes, and user processes:

```

  F S UID    PID  PPID  PGRP   SZ      RECV TTY  TIME CMD
 10 W  0      0    0    0   192     ANY  ?  0:00 TTY
 10 W  0      0    0    0   192     ANY  ?  0:00 DP8390
 10 W  0      0    0    0   192     CLOCK ?  0:00 SYN_AL
  0 R  0      0    0    0   192     ?    ?  0:08 IDLE
 10 W  0      0    0    0   192     ANY  ?  0:00 PRINTER
 10 W  0      0    0    0   192     ANY  ?  0:01 HDDISK
  0 R  0      0    0    0   192     ?    ?  0:00 MEMORY
 10 W  0      0    0    0   192     ANY  ?  0:00 CLOCK
 10 W  0      0    0    0   192     ANY  ?  0:00 SYS
  0 R  0      0    0    0   192     ?    ?  0:00 HARDWAR
 10 W  0      0    0    0   104     ANY  ?  0:00 MM
 10 W  0      0    0    0   616     MEMORY ?  0:00 FS
 10 W  0      0    0    0   224     ANY  ?  0:00 INET
 10 S  0      1    0    0    40     (wait) MM ?  0:00 INIT
 10 S  0     29    1   29    64     (wait) MM co 0:00 -sh
 10 W  0     18    1    0    24     MM    ?  0:00 update
 10 S  0     30    1   30   32     (TTY) FS c1 0:00 getty
 10 S  0     31    1   31   32     (TTY) FS c2 0:00 getty
 10 S  0     32    1   32   32     (TTY) FS c3 0:00 getty
 10 W  0     35   29   29   152     FS    co 0:00 ps -alx

```

The first ten are the system tasks. A "task" is Tanenbaum's name for a system thread running inside the kernel code. They all share the same code, and are thus the same size (192K). The next three (MM, FS, and INET) are server processes. They receive all the system calls made by user processes. `INIT` is the first user process, and is the ancestor of all other user processes. Next you see `-sh`, which is the shell used by `root`; `update`, which flushes the disk cache every 30 seconds; three `getty`'s, which are waiting for someone to type `mlogin`; and finally the `ps -alx` command. Type `man ps` for a description of the output.

You can use an X-window of any size to run Minix. If you change the X-window size to 90-by-67, for example, then type (in Minix):

```
# stty rows 67
# stty cols 90
```

Changing the number of rows is useful primarily for reading "man" pages in Minix. There are four Minix editors: `elvis` (with synonyms `vi`, `ex`, and `view`), `elle` (with synonym `emacs`), `uemacs` (Micro Emacs), and `mined`, the MINix EDitor. Only `mined` tolerates the larger windows properly. Fortunately, you will almost never need to edit any files inside Minix. Even if you do, you could

`sunwrite` them from Minix to Solaris, edit them in Solaris, and then `sunread` them back into Minix (see below). You can find out about these editors from the `man` pages while running Minix (try typing `man mined`, for example).

You can have multiple logins into one Minix instance. For example, go to another X-window in Solaris and type `mlogin $user` where `$user` can be typed as is, or you can type your login name instead. If I type this command, I get:

```
Connected to host davis

Solaris-Minix  Release 2.0 Version 0

noname login:
```

Note the "hostname" is `davis` in this case, since this is my copy of Minix. The name of your copy of Minix is defined in `yoursrc/tools/.minix` configuration file. No one but you will be able to log into your copy of Minix. Log in as `bin`, and type `who`:

```
$ who
root      console  Fri Sep 12 17:20
bin       ttycl    Fri Sep 12 17:24
```

To log out, hit control-D. To terminate an `mlogin` session, type control-Q control-U control-I control-T, in sequence. To terminate the entire Minix instance, including all `mlogin` sessions, type `shutdown` as either the user `root` or the user `bin`, or type `halt` as the user `root`. The `reboot` and `shutdown -R` commands restart Minix (on a PC), but cause Solaris-Minix to shutdown. To avoid corrupting your root file system, do not terminate Minix in any other way except by using `shutdown`, `halt`, or `reboot`.

Please note that while you may use multiple `mlogin`'s to run multiple shells within the same Minix instance, you should never run multiple `minix`'s at the same time. The `minix` command should prevent you from doing this accidentally, but you should take caution to avoid this on your own.

The `syschecks` and `killminix` commands

You can only run one copy of Minix at a time (running two of them would corrupt your root file system). If you try to do so, you will get an error message, telling you on what computer it thinks the other copy is running on. To shut down Minix safely, log into that computer, and then type `mlogin $user`. Log in as `root` and type `halt`. **This is the safest way of shutting down Minix.**

If this fails, then you can locate the process id's of the Minix simulator with the `syschecks` command and then kill them with the `kill -9 pid1 pid2` command, where `pid1` and `pid2` are the process id's of the two processes. The `syschecks` command takes a while, since it interrogates every computer in the CISE network that you can normally log in to (you can use the `syscheck` command to find just those Minix processes on the computer you are currently logged into). You will also need to delete your lock file, with the command

```
cd ; cd cop4600 ; rm lockfile
```

Do this as a last resort, since `kill` will leave your root file system corrupted. Alternatively, you can type `killminix`. This command uses the `syschecks` command to find your Minix processes, and then kills them for you. It also deletes your lock file. Do this as a last

resort, since `killminix` will leave your root file system corrupted.

If your root file system is corrupted, you can get a fresh copy by typing:

```
cp /cise/class/cop4600/minix/disks/root ~/cop4600/disks
```

This is not as serious as it looks, since none of your Minix source code is located in your root file system. Even your Minix user programs are normally written outside Minix itself. Typically, the only things you lose are the files that you have previously `sunread` into the root file system - which are normally just binary executables, copies of which are in your normal Solaris directory (usually `cop4600/src/tools`).

If Minix is killed, the next time you run Minix you may get a message:

```
The system was not properly shut down.  Checking file systems.
```

This is followed by a report on what `fsck`, the file system checker, has found. The `fsck` program is usually able to repair any damage done.

The `syschecks` and `killminix` commands only check those CISE Solaris computers that students normally have access to. It's possible that they could miss one, if a new computer is installed or if you have access beyond that of a typical student. Use these commands with caution.

[Revision Note: The `syschecks` and `killminix` commands rely on a pre-determined list of hostnames within the CISE network. Since the original writing of this document, many hostnames have changed or been added to the network, meaning these computers will not be checked by `syschecks` or `killminix`. For example, `sand` is not listed among the computers checked. For this reason, it is advised that you avoid use of `syschecks` and `killminix` and take a more responsible approach: keep track of which systems you may have left runaway Minix processes on and remove those processes yourself.]

Runaway Minix Processes

Occasionally when running Minix, it will hang (or freeze) or enter an infinite loop, and you will be forced to terminate Minix abruptly. Sometimes, when dialing in from home, you may be disconnected while running Minix. Whatever the circumstance, there are going to be occasions on which you do not exit Minix properly (using the `halt` or `shutdown` commands from within Minix). When this happens, you should immediately log into the computer you were previously logged in to (the one on which you were running Minix) and attempt to shutdown Minix correctly using the `mlogin` command as described above. If this fails, you should run the `minix_cleanup` command to kill all 'runaway' Minix processes. Failure to do this may result in the system administrators banning you from using many CISE computers.

It is your responsibility to share the computing resources provided by CISE. Since Minix consumes a considerable amount of processor time and memory, many runaway Minix processes could affect the performance of the entire system. If you suspect that you have some runaway Minix processes, you should attempt to locate them and kill them immediately. You can do this using the `minix_cleanup` command mentioned just a moment ago, or by using the `ps -u $user` command to view all processes owned by you. Minix processes will appear with the name 'minix' and usually appear in pairs of two. You must kill both processes to remove runaway Minix processes. Actually, if you kill the Minix process with the most 'TIME', it will kill both processes and remove your lockfile for you. After killing runaway Minix processes, you may have to also remove your lockfile, located at `~/cop4600/lockfile`.

Note: Runaway Minix processes will appear with a '?' in the TTY column. Processes that are currently running in another window appear with a named terminal, such as 'pts/0'.

Getting Help

To get help inside Minix, use the `man` or `man -k keyword` commands. For example, `man printf` tells you about the `printf` library routine, and `man sunread` tells you about how to read Solaris files into your Minix file system (more on this later). Typing `man -k shell` gives you a list of all the man pages with the keyword `shell`. Use `man -s 1 ash` to get the Section 1 man page on the `ash` shell, for example. The command `man man` will tell you how to use `man`. Sometimes a command and a library routine have the same name. Try typing `man -k write`. There is a `write` system call (`write` to a file) and a `write` command (send a message to another user on their screen). The `man write` command will give you the entry in the first Section it finds (Section 1, in this case). If you want the other one (the `write` system call), you have to give the section number (`man -s 2 write`).

The `man` command presents information to you via the `more` program, which allows you to page through a long document (or output of a program). Type `q` to exit `more`, a space to go down one page, and `b` to go back a page. You can view the output of a command by typing the command, followed by `| more`, on the same line (as in `dir -R | more`, for example). The `more` command can make use of any sized X-window, if `stty` is used to set the window size. I've set up synonyms `less` and `l` for `more`, as well.

Your Source Code for the Minix Operating System

The class assignments will require you to modify the Minix source code. The contents of your `cop4600` directory are described below. Be aware that the source code in your `cop4600` is for Solaris-Minix 2.0.0, not Minix 2.0.0. The source code listed in the back of the textbook is Minix 2.0.0. The changes are discussed in the *Smx - the Solaris port of Minix* document.

- Backup

This contains any backups of any Minix source code files you have modified. Backups are made via the `minix_backup` command (no automatic backups are made).

- disks

This contains your copy of the root file system, `disks/root`, as well as symbolic links to the shared `/usr/bin` and `/usr/man` file systems. The original copy of `disks/root` is `/cise/class/cop4600/minix/disks/root`.

- include

This contains your copies of the system-wide include files for the Minix operating system, the Minix library calls, and any Minix user programs. You will typically have to modify some of these include files. The original copy of this directory is `/cise/class/cop4600/minix/include`.

- lib

Your copy of the compiled C library routines. Source code for these compiled libraries is located in `src/lib`. The original copy of this directory is `/cise/class/cop4600/minix/lib`.

- `src`

Your `src` directory holds your copy of all of the Minix source code. The original directory is `/cise/class/cop4600/minix/src`. The subdirectories of `src` are described below.

- `src/commands`

This contains the source code for all Minix commands (such as `more`, `ls`, `who`, etc.). In the current setup, you have a symbolic link to a read-only directory that you share with all other students. You normally won't be modifying any of this code. If you do, you'll have to make your own copy first.

- `src/etc`

This contains copies of the contents of the `/etc` directory in your root file system. You won't normally edit any files in this directory. These files also appear in the `/etc` directory in your Minix root file system.

- `src/fs`

This contains the file system source code, for the FS server process.

- `src/inet`

This contains the network system source code, for the INET server process.

- `src/kernel`

This contains the source code for the Minix kernel, including the system tasks and the process management layer beneath them.

- `src/lib`

This contains the source code for the C library routines that are available to a Minix user program. For some projects, you will change a few of these files to add new system calls to Minix. This directory includes:

- `src/lib/ansi`, some of the ANSI C library routines.
- `src/lib/curses`, a screen handling package.
- `src/lib/editline`, a line editing package.
- `src/lib/fphook`, for printing floating-point numbers.
- `src/lib/ip`, network access routines.
- `src/lib/liby`, yacc library routines.
- `src/lib/math`, mathematical library routines.
- `src/lib/other`, other system library routines.
- `src/lib/posix`, all system calls.
- `src/lib/stdio`, the C standard I/O library.
- `src/lib/sunsyscall`, all system calls (these just call the `src/lib/posix` versions).
- `src/lib/syslib`, used by Minix operating system, and a few commands in `src/commands`.

- `src/lib/sun4`, C run time routines.
- `src/mm`

This contains the memory manager source code, for the MM server process.

- `src/tools`

This contains the files needed to combine the `src/kernel/kernel`, `src/mm/mm`, `src/fs/fs`, `src/inet/inet`, and `src/tools/init` files, and the compiled libraries in `lib`, into the single `src/tools/image` file. You won't normally edit any code in this directory, but this is the only place you can recompile all of your Minix operating system (the `src/tools/image` file).

Compiling Your Copy of Minix

```
Type cd ~/cop4600/src/lib
Run mmake
Type cd ~/cop4600/src/tools
Run mmake
```

The Minix operating system consists of five binary files: `src/kernel/kernel`, the operating system kernel, `src/mm/mm`, the memory manager, `src/fs/fs`, the file system, `src/inet/inet`, the network manager, and `src/tools/init`, the ancestor of all user processes. These five files are combined (along with your compiled libraries) to form the binary `src/tools/image` file, which is used to run Minix.

To compile all of them (except the libraries), `cd` to `src/tools` and type `mmake`. You can also compile individual components while in the related source directories, *but you must always then type `mmake` in `src/tools`!* To run Minix with this new image, type `minix` in your `src/tools` directory, which uses `src/tools/.minix` as a configuration file. If you want to run Minix with my copy of the unmodified, original image, then type `minix orig` while in your `src/tools` directory. This will use your root file system `disks/root`, but my "clean" version of Minix (`/cise/class/cop4600/minix/src/tools/image`).

Typing `mmake` in `src/lib` will recompile all the C library routines in the `src/lib` subdirectories. If you modify the `include` files, you may wish to recompile the C library routines as well, since they depend on the `include` files. Since the Minix operating system (`image`) depends on the C library routines, you must recompile `image` if you recompile the C library routines.

Exploring the Source Code of the Minix Operating System

There are four commands that you will find useful for exploring the Minix source code, in addition to the normal Solaris editors available in the CISE network. The following are Solaris commands, not commands that you run while inside Minix. You do not have to be in your COP 4600 directory to type these commands (with the exception of `mgrep` when no directory is specified).

- `mgrep`

This command searches your entire Minix source code, or a portion of it, for a specified text pattern. The usage of the command is `mgrep pattern directory`, where `pattern` is a text string to search for, and `directory` is an optional argument specifying where to look. The following options are available for the `directory` argument:

- `.` a search is made in the current directory and all subdirectories
- `-m` a search is made in the `src/kernel`, `src/mm`, `src/fs`, `src/inet`, `src/tools`, and include directories, and all subdirectories
- `-l` a search is made in the `src/lib` directory, and all subdirectories
- `-ml` do both `-m` and `-l`
- `-lm` do both `-m` and `-l`
- `(none)` do both `-m` and `-l`
- `other` a search is made in the `\other` directory, and all subdirectories

For example, to search the Minix source code (excluding the `src/lib` directory) for the string `NR_PROCS` type the command:

```
mgrep NR_PROCS -m
```

To search just the `src/kernel` directory for the same string, use

```
mgrep NR_PROCS src/kernel
```

The current version of `mgrep` command creates a temporary file by the name `.tempfile_mgrep` in your `cop4600` directory, for each file it compares. You can thus only run one `mgrep` command at a time (otherwise, the single temporary file can become corrupted). Fixing this (and `minix_diff`, below) is on my to-do list.

- `minix_diff`

This command can compare any portion of your copy of the Minix source code with the original Minix source code in `/cise/class/cop4600/minix/src` and `/cise/class/cop4600/minix/include`. For a description of the output format, see `man diff`. Lines from your code are preceded with a `<` character, and the related lines from the original code are preceded with a `>` character. The command

```
minix_diff src/fs src/kernel
```

compares code inside just those two directories, for example. With no arguments, everything is compared (`src` and `include`, except for `src/test` and `src/commands`). If you want to compare your test code with the original test code, type `minix_diff src/test`. Like `mgrep`, this command uses a temporary file, `.tempfile_diff`. You can only run one diff-type command at a time (`minix_diff`, `mdiff`, or `ldiff`).

- `mdiff`

The `mdiff` command compares all source code for the Minix image, except `src/lib`. Short-hand for `minix_diff include src/kernel src/fs src/mm src/inet src/tools`.

- `ldiff`

The `ldiff` command compares all library source code. Short-hand for `minix_diff src/lib`.

Compiling Minix User Programs, `sunread` and `sunwrite`

Type `cd ~/cop4600/src/tools`
 Create your C program (let's call ours `example.c`)
 Type `mcc example.c -o example`
 Run `minix`
 Login as `root`
 Type `sunread example > example`
 Type `chmod +x example`
 Run `example`

Suppose you want to run the following `hello.c` program in Minix:

```
#include <stdio.h>
main (int argc, char **argv)
{
    printf ("hello\n") ;
}
```

A copy of this program is located in your `src/tools` directory. Compile the program with the command `mcc hello.c -o hello`. Next, while running Minix you can type the command `sunread hello > hello` to read in the compiled `hello` program. You can't read it into `/usr/bin` or `/usr/man`, since those are read-only file systems. Next, make the program executable with the command `chmod +x hello`. Then type `hello` to run the program. There is an analogous `sunwrite` command, which writes a file from a Minix file system into a Solaris file system. Type `man sunread` and `man sunwrite`, in Minix, for more information.

NOTE: if you recompile your libraries (`mmake` in `src/lib`), the newly recompiled libraries *will not be used* until you then recompile the code that uses those libraries. So if you recompile your libraries, then recompile Minix, the Minix test code (if needed), and any Minix user programs that you've written.

Reinstalling a Fresh Copy of Minix

Run `minix_backup`
 Run `minix_DESTROY`
 Run `minix_setup`

IMPORTANT: `minix_DESTROY` will erase all files in your Minix directory (except for those in your Minix Backup directory). Before running `minix_DESTROY`, you should run `minix_backup` to copy all of the files you've modified to the Minix Backup directory.

- `minix_backup`

This command finds all of your source files in `src` and `include` that differ from the original Minix distribution, and copies them into a new directory with the name `Backup/date`, where `date` is the date and time that the backup was made. Your source files in `src` and `include` are unchanged.

NOTE: this does not save a copy of your root file system. It only saves all files ending in `.c`, `.h`, or `.s`, all Makefile's, and every file in `src/etc`. If you want to save the `disks/root` file, or other files, then copy or move them into the backup directory yourself.

- `minix_DESTROY`

This command DESTROYS your `src`, `include`, `lib`, and `disks` directories. It will not work properly if you are currently running Minix. It prompts you before destroying everything, and leaves your other subdirectories (such as `Backup`) intact. Use it after running `minix_backup`, to save your modified source code. Once you have destroyed your old copy, reinstall Minix with the `minix_setup` command.

Restoring a Backup of Minix

First, reinstall a fresh copy of Minix (see above)

Type `cd ~/cop4600/Backup/date` (where `date` is the name of the directory corresponding to the date/time of the backup you wish to restore)

Type `cp -R * ../..`

Before restoring a backup of Minix, you must first reinstall a fresh copy of Minix, because `minix_backup` only copies files that were modified and not the entire Minix directory. Once you have a fresh copy of Minix, change to the directory corresponding to the particular backup you wish to restore, and copy all of the files from the backup over the existing ones using the following command:

```
cp -R * ../..
```