Installing and Using Research Unix Version 7 In the OpenSIMH PDP-11/45 and 11/70 Emulators

Revision 3.2

by Will Senn

Created December 7, 2015
Last Updated May 23, 2024

This is document revision 3.2. This note is intended to document the process of installing Unix v7 in an OpenSIMH PDP-11/45 emulated environment and running Unix v7 multi-user and multi-session in an OpenSIMH PDP-11/70 emulated environment[1]. There is a similar entry that documents the process of running Unix v6 in OpenSIMH[2]. The primary motivator for this revision is the formation of the Open SIMH steering committee and their taking on the maintenance and care of SIMH going forward. This document assumes you are able to use the command-line skillfully[3]

First, the reader will be led through the restoration of a pristine v7 instance from tape to disk. Next, the reader will be led through adding a regular user, making the system multi-user capable. Then, the reader will be shown how to make the system multi-session cable allowing multiple simultaneous sessions. Finally, the system will be put to use with hello world, DMR style, and the learn system will be enabled.

Changelog

```
[3.2 - 20240517] Fixed mktape scripts EOF, EOM markers based on discussion with Clem Cole
[3.1 - 20221029] Switched from DCI to DZ and upped the baud rate to 9600 for telnet
[3.0 - 20221028] Updated to OpenSIMH and made various minor changes to bring it up to date
[2.1 - 20220203] Changed baud rate to 1200, from 9600 to allow telnet from DCI lines
[2.0 - 20220103] Made significant changes to both the document contents and installation process
[1.8 - 20211231] Added some additional comments and cleaned up in a few places
[1.7 - 20211230] Fixed another typo, updated some urls, and added clarifying comments
[1.6 - 20171026] Fixed a typo and added clarification about 0 vs O in STTY command
[1.5 - 20171014] Added section describing multiple sessions and made some minor edits
[1.4 - 20171014] Added Appendix C - a notes section
[1.3 - 20171014] Added binmode to perl script to support windows and updated tested environments
[1.2 - 20171012] Minor wording corrections
[1.1 - 20171011] Created PDF version
```

Acknowledgements

My Dad - Ted Senn - He introduced me to the world of Digital Equipment computers oh so long ago.

TUHS - I wouldn't have a clue about any of this without the patient folks on the TUHS mailing list.

OpenSIMH - None of this would work without the wonder of emulators and this is my sim of choice.

Clem Cole - Our many correspondences and talks helped greatly to fill in the gaps in my understanding related to Unix, PDP, and tech generally.

[1] Originally based on the blog entry `https://decuser.blogspot.com/2015/12/installing-and-using-research-unix.html`

[2] `http://decuser.blogspot.com/2022/10/installing-and-using-research-unix.html`

[3] PDFs are notoriously unreliable about copying and pasting. If pasting doesn't seem to work correctly, either type your text in directly or paste it into an editor first and then copy and paste from there, after making any necessary fixups.

Contents

[4] Suggestion by Warren Young

0 Preliminaries

0.1 Cross references

- Open SIMH Project
  `https://opensimh.org/`
- Additional Software for Open SIMH
  `https://opensimh.org/software/`
- Setting up Unix - Seventh Edition
  `https://www.tuhs.org/Archive/Documentation/PUPS/Setup/v7_setup.html`
- The Unix Heritage Society
  `https://www.tuhs.org/`
- The PDP Unix Preservation Society
  `https://minnie.tuhs.org/PUPS/`
- The Earlier Computer History Simulation Project
  `http://simh.trailing-edge.com/`
- Hellwig Geisse's package, notes, and utilities (mktape and exfs)
  `https://homepages.thm.de/~hg53/pdp11-unix/`
- Warren Toomey's work on restoration and archiving
  `https://minnie.tuhs.org/`

This note follows the approach laid out in the author's original blog entry for Unix Version 6, *Installing and Using Research Unix Version 6 in OpenSIMH PDP-11/40 Emulator*[2]. It is written for anyone wishing to bring a v7 instance into life on modern hardware.

The bits that will be used are binary copies of original tapes that folks have donated to The Unix Heritage Society, hereafter referred to as TUHS. They are available to the public and represent the best known source of 'original bits'.

0.2 Prerequisites

- A working Host - I have used Mac OS on versions ranging from 10.9-12.6, as well as FreeBSD version 10.2-13.1 and Linux Mint 19-21.3. I feel confident that it would work with any host capable of running Open SimH, but I haven't tried them all.
- The license - I am using the Caldera Unix Enthusiasts license available at `https://www.tuhs.org/Archive/Caldera-license.pdf`
- Open SimH PDP-11 Emulator - I have used versions 3+, this note is based on using the latest code as of 20240517 available at `https://github.com/open-simh/simh`. See Appendix A for basic build instructions.
- A distribution tape image - I am using a tape constructed from Keith Bostic's tape records which appear to be closest to the bits of the original distribution. Available at `https://www.tuhs.org/Archive/Distributions/Research/Keith_Bostic_v7/`
- A working Python or Perl instance

0.3 Tested Environments

These notes are based on having aria2c, curl, wget or another downloading tool, a working OpenSIMH environment, and a C-compiler installed, such as clang or gcc. The configurations that have been tested directly are:

This note - v 3.2:

- Linux Mint 21.3 "Virginia" running on an IBM ThinkCentre m92p i7-Quad Core with 32GB RAM

Previous note - v 3.1:

- MacOS 12.6 Monterrey running on a Mac Pro Xeon 6-Core with 32GB RAM and XCode and Macports installed
- MacOS 12.6 Monterrey running on a MacBook Pro i7 Quad Core with 16GB RAM and XCode and Macports installed

Previous note - v2.1:

- MacOS 10.14.6 Mojave running on a MacBook Pro i7 Quad Core with 16GB RAM and XCode and Macports installed
- MacOS 10.13 High Sierra running on a MacBook Pro i7 Quad Core with 16GB RAM and XCode and Homebrew installed
- Mac OS X 10.11.1 El Capitan and MacOS 10.12.x Sierra running on a MacBook Pro i7 Quad Core with 16GB RAM and XCode and Homebrew (for gcc) installed
- FreeBSD 10.2, 11, 12, and 13, running on a Dell Optiplex 755 Core 2 Quad with 8GB RAM with gcc48 installed
- Windows 8.1 Enterprise running virtually in Linux Mint 18.2 on a HP Elite Desk 800 i5-6500 3.2 GHz with 16 GB RAM and Git bash with Unix tools installed
- Linux Mint 20.2 Uma on an IBM Thinkpad T430 Dual Core i5-3320M with 16GB Ram with gcc9.3.0 installed
- Raspbian 2017-09-07-stretch running on a Raspberry Pi2 Model B

1 Installation

In this section, the process of (re) creating the distribution tape image, booting from it, restoring it onto the system's hard disks, booting from disk, and saving a backup copy of the working system are described in detail. The resultant system will be a pristine multi-user Unix v7 system, which will serve as a baseline for additional configuration and extension.

1.1 Preparing the tape image

This section describes creating a workspace, downloading the tape records, and constructing a bootable tape image.

1. Create a set of working directories for v7 (I use /retro-workarea/v7 as the base):

```
mkdir -p ~/retro-workarea/v7/{dist,work,save}
cd ~/retro-workarea/v7/dist
```

2. Get a copy of Keith Bostic's tape records and the author's mktape.pl [5]

```
aria2c https://www.tuhs.org/Archive/Distributions/Research/Keith_Bostic_v7/f0.gz
aria2c https://www.tuhs.org/Archive/Distributions/Research/Keith_Bostic_v7/f1.gz
aria2c https://www.tuhs.org/Archive/Distributions/Research/Keith_Bostic_v7/f2.gz
aria2c https://www.tuhs.org/Archive/Distributions/Research/Keith_Bostic_v7/f3.gz
aria2c https://www.tuhs.org/Archive/Distributions/Research/Keith_Bostic_v7/f4.gz
aria2c https://www.tuhs.org/Archive/Distributions/Research/Keith_Bostic_v7/f5.gz
aria2c https://www.tuhs.org/Archive/Distributions/Research/Keith_Bostic_v7/f6.gz
aria2c https://www.tuhs.org/Archive/Distributions/Research/Keith_Bostic_v7/filelist
aria2c https://www.tuhs.org/Archive/Distributions/Research/Keith_Bostic_v7/mktape.pl
aria2c https://www.tuhs.org/Archive/Distributions/Research/Keith_Bostic_v7/mktape.py
```

3. Make the mktape.pl or mktape.py script runnable

```
chmod u+x mktape.pl
chmod u+x mktape.py
```

4. Change into the working directory and unpack the six tape sections.

```
cd ../work
cp ../dist/* .
gunzip f?.gz
```

5. Run the mktape.pl script.

   Create the tape image[6] using the mktape.pl script for use with OpenSIMH. The OpenSIMH simulator will emulate a TU10 MagTape controller. We need to create a suitable tape image to load into the simulated TU10. We do this by combining each of the tape records along with their lengths and appropriate tape marks into a single file, named v7.tap. Alternatively, you can use the author's mktape.py, a more full featured script [7] i.e. mktape.py –v7 to build the v7 tape.

[5] The source code for the script is found in the appendices

[6] Or download it from `https://www.tuhs.org/Archive/Distributions/Research/Keith_Bostic_v7/v7.tap.gz`

[7] The source code for the script is found in the appendices

```
./mktape.pl
f0: 8192 bytes = 16 records (blocksize 512 bytes)
f1: 7168 bytes = 14 records (blocksize 512 bytes)
f2: 512 bytes = 1 records (blocksize 512 bytes)
f3: 11264 bytes = 22 records (blocksize 512 bytes)
f4: 11264 bytes = 22 records (blocksize 512 bytes)
f5: 2068480 bytes = 202 records (blocksize 10240 bytes)
f6: 9594880 bytes = 937 records (blocksize 10240 bytes)
```

6. To confirm that the correct number of records and block sizes were used, take a look at the filelist file we downloaded and compare the block size and number of records written to those that the mktape.pl script reported.

```
cat filelist
file 0: block size 512: 16 records
file 0: eof after 16 records: 8192 bytes
file 1: block size 512: 14 records
file 1: eof after 14 records: 7168 bytes
file 2: block size 512: 1 records
file 2: eof after 1 records: 512 bytes
file 3: block size 512: 22 records
file 3: eof after 22 records: 11264 bytes
file 4: block size 512: 22 records
file 4: eof after 22 records: 11264 bytes
file 5: block size 10240: 202 records
file 5: eof after 202 records: 2068480 bytes
file 6: block size 10240: 937 records
file 6: eof after 937 records: 9594880 bytes
file 7: block size 63:
```

7. To confirm that this tape is actually byte identical to the author's version (which is identical to the one on TUHS), use openssl to check the sha1 digest.

```
openssl sha1 v7.tap
SHA1(v7.tap)= 8056d35a2cb6529330f26db5754e858c9eab0462
```

8. Cleanup the working files.

```
rm f* mktape.pl mktape.py
ls
v7.tap
```

With a working distribution tape image, we are ready to fire up the OpenSIMH simulator and install v7 onto an RP06 diskpack.

1.2 Create a OpenSIMH ini file for the first boot

In the original v6 blog entry, I showed both a manual and and ini file method of booting the simulation. In this note, I will only show the ini method. However, I will describe the contents of the file. Just be aware that any commands in the ini file can be entered into the simulator interactively, if so desired.

Below is the ini file that will be used during the installation of v7. After the system is installed, a modified version of the ini file will be used for subsequent boots.

The file sets the cpu type and allows it to idle. A virtual RP06 moving head disk pack is attached to the simulator and associated with a file on the host as rp0, another is added as rp1, then a virtual TU10 magtape is added and associated with the distribution file we compiled on the host as tm0.

This ini file gives us the following initial configuration: A PDP-11/45 with two identical, empty, RP06 disk packs, and a TU10 magtape with our v7 distribution tape ready to load.

We use cat to create the tape.ini file.

```
cat > tape.ini <<"EOF"
set cpu 11/45
set cpu idle
set rp0 rp06 noautosize
att rp0 rp06-0.disk
set rp1 rp06 noautosize
att rp1 rp06-1.disk
att tm0 v7.tap
boot tm0
EOF
```

1.3 Boot from the tape image

I highly recommend following along in Dennis Ritchie's *Setting up Unix - Seventh Edition* `https://www.tuhs.org/Archive/Documentation/PUPS/Setup/v7_setup.html` (also found in Vol II of the 7th edition of the UNIX Programmer's Manual) as you work through the rest of this document. *Setting up Unix* is authoritative for installing Unix v7. With the work we have done up to this point, if you were to leave out the boot tm0 command from the ini file, the instructions from the manual can be followed verbatim (talk about durable documentation). However, I will be explaining as we go, so you may want to wait until you have worked along with these instructions before trying to rely solely on the official document.

1. Start the OpenSIMH PDP-11 Simulator with the tape.ini file.

   The simulator will start, process our ini file, and boot the tape. The simulator will ask if we want to overwrite the last tracks of each rp06. This will write a bad blocks table to the last tracks of each of the disks. Answer y and press enter for both RP0 and RP1.

   ```
   pdp11 tape.ini

   PDP-11 simulator Open SIMH V4.1-0 Current        git commit id: 43963943
   Disabling XQ
   /home/wsenn/retro-workarea/v7/work/tape.ini-4> att rp0 rp06-0.disk
   %SIM-INFO: RP0: Creating new file: rp06-0.disk
   /home/wsenn/retro-workarea/v7/work/tape.ini-6> att rp1 rp06-1.disk
   %SIM-INFO: RP1: Creating new file: rp06-1.disk
   /home/wsenn/retro-workarea/v7/work/tape.ini-7> att tm0 v7.tap
   %SIM-INFO: TM0: Tape Image 'v7.tap' scanned as SIMH format
   %SIM-INFO: contains 11701760 bytes of tape data (1214 records, 8 tapemarks)
   Boot
   :
   ```

   We don't have to key in a boot routine because OpenSIMH has a built in tape rom that works fine. However, if we wanted to, we could key in the TU10 boot routine as described in *Setting up Unix* manually and run it with the same effect.

1.4 Create a filesystem on the first RP06

The simulator has started from the tape image and displays the word Boot followed by a carriage return and a colon prompt. At this point, no operating system is present. However a tape is loaded and a standalone program

called *tm* is also loaded and available. If we run *tm*, it will run a program directly from the tape, indexed by the tape controller and file on the tape. It is a zero based index. We want to run the 4th file on the tape, which is a standalone version of *mkfs* in order to create a filesystem on the RP06 diskpack. The target device name is *hp* for the rp06. This time the index refers to the controller and the partition. We want to create a filesystem on the first controller's first partition. This will prepare our disk for a root filesystem.

```
: tm(0,3)
file sys size: 5000
file system: hp(0,0)
isize = 1600
m/n = 3 500
Exit called
Boot
:
```

1.5 Restore the root filesystem from tape

Again the Boot and : prompt will be displayed. Now that a filesystem is prepared, we will use another program from the tape, *restor*, to populate it. The standalone program *restor* will take a tape file as input and a disk device as output. The sixth file on the tape is a dump of *rp0* (a root filesystem). The same conventions as above apply regarding the indexes. Press return after *Last chance before scribbling on disk.* to proceed.

```
: tm(0,4)
Tape? tm(0,5)
Disk? hp(0,0)
Last chance before scribbling on disk. [press enter]
End of tape
Boot
:
```

1.6 Boot the root filesystem

At this point, a root filesystem is available. We can boot Unix from the root. Using the indexing scheme described above, we can load and run the hptmunix (hp and tm drivers are included) kernel from the root filesystem.

```
: hp(0,0)hptmunix
mem = 177344
#
```

The system comes up single user. It also comes up with UPPERCASE characters and is slow to print output to the console, which is annoying and weird. We could enter multi-user mode and it would fix this and other annoyances, but because of our remaining low level tasks, it is simpler and more effective to stay in single user mode. In order to fix the annoying console issues while we complete the installation, we will use stty to set lowercase and to add no delays to newlines or carriage returns. Make sure that you type the number `0`, not the letter `O`, in the STTY command below. This will make our console snappy to respond and allow us to use both upper and lower case letters in the terminal. Note: <DEL> does not work in the current environment, if you make a mistake typing, use `#` to remove the last character from the input buffer (doesn't rubout the character on screen, but it does remove it from input). That is if you type lsa and then `#`, you will see lsa# on screen, but the terminal will only see ls. Also, ^C doesn't interrupt programs in the v7 world, use ^DEL instead. This may require tweakage of your terminal settings, e.g. Konsole has keybindings for DEL+Modifiers, these need to be disabled in your profile for Unix to receive them.

```
# STTY -LCASE NL0 CR0
```

1.7 Remove unused kernels and clean up

Typing hp(0,0)hptmunix is a pain, let's shorten it to unix by copying the hptmunix kernel, saving it just in case we muck it up later, and get rid of the extra unused kernels in the root filesystem.

```
# cp hptmunix unix
# rm hphtunix rphtunix rptmunix
# sync; sync; sync
# ls *ix
hptmunix
unix
```

1.8 Create device files

In order to use Unix to complete the installation, we will need to create a number of special files to represent our hardware devices. Special files serve as interfaces between the user and the underlying devices. They are the magic that allows the Unix system to present nearly all hardware to the user as simple files. The special file abstraction maps a filename to a memory vector that points to a limited set of common I/O operations (read, write, getchar, putchar, etc). The device drivers that implements these I/O operations are either part of the operating system, as is the case for all of our devices, or are supplied as add-ons. The makefile in /dev contains sections for the most common devices and serves as a template for us to determine what devices we need to instantiate. **DO NOT** use these actual values. Later, we will modify them before using them.

```
The rp06 section in that file looks like this:
rp06:
/etc/mknod rp0 b 6 0
/etc/mknod swap b 6 1
/etc/mknod rp3 b 6 7
/etc/mknod rrp0 c 14 0
/etc/mknod rrp3 c 14 7
chmod go-w rp0 swap rp3 rrp0 rrp3
```

The syntax for mknod from man with edits is: `/etc/mknod name [c][b] major minor`

The first argument is the name of the entry. The second is b if the special file is block-type (buffered, block sized I/O, slower) or c if it is character-type (unbuffered, byte sized I/O, faster, aka raw). The last two arguments are numbers specifying the major device type and minor device (unit, drive, line-number).

For our purposes, we obtain the major device number from the makefile (6 for block-type rp and 14 for character-type rp). The minor device is a number represented by a byte that combines the block device index and the partition.

In order to know what partitions to use requires a bit of detective work, but I will just tell you that partition 7 is the largest available partition for us to use on the disk. The reference for the curious is the manpage HP(4), where the original authors describe the rp06 partition scheme.

So, here are the devices we will need to create special files for:

- The first rp06 partition 0 is root, partition 1 is swap, we will leave the rest unused for the time being
- The second rp06, partition 7 will be used for /usr
- The magtape tu10.

We will manually create block and character devices for each rp06, but we will use make to create the tape device.

According to the makefile, the rp06 major devices are 6 and 14, respectively for block and character devices. The minor numbers are:

- for the first drive, drive 0's first partition, it is 00000000, decimal 0

- for the first drive's second partition, it is 00000001, decimal 1
- although, we aren't using it, the first drive's seventh partition would be 00000111, decimal 7
- for the second drive, drive 1's seventh partition, it is 00001111, decimal 15

1. Create nodes for RP06 devices.

   Using this information results in the following commands, which we run to create the special files for the rp06 and give them appropriate permissions.

```
# cd /dev
# /etc/mknod rp0 b 6 0
# /etc/mknod swap b 6 1
# /etc/mknod rp3 b 6 15
# /etc/mknod rrp0 c 14 0
# /etc/mknod rrp3 c 14 15
# chmod go-w rp0 swap rp3 rrp0 rrp3
# sync; sync; sync
```

   Now, rp0 refers to disk 0, partition 0 (the root device), swap refers to disk 0, partition 1, rp3 refers to disk 1, partition 6 (/usr), rrp0 refers to the raw disk 0, partition 0, and rrp3 refers to the raw disk 1, partition 6.

2. Create nodes for TU10 device.

   We use make to create the tape special files (regular device, rewinding device, and non-rewinding device) and set appropriate permissions.

```
# make tm
/etc/mknod mt0 b 3 0
/etc/mknod rmt0 c 12 0
/etc/mknod nrmt0 c 12 128
chmod go+w mt0 rmt0 nrmt0
```

   At this point you should have the following special files in /dev:

```
# ls -l
total 2
crw--w--w- 1 root    0,  0 Dec 31 19:02 console
crw-r--r-- 1 bin     8,  1 Jan 10 15:40 kmem
-rw-rw-r-- 1 bin       775 Jan 10 15:26 makefile
crw-r--r-- 1 bin     8,  0 Jan 10 15:39 mem
brw-rw-rw- 1 root    3,  0 Dec 31 19:02 mt0
crw-rw-rw- 1 root   12,128 Dec 31 19:02 nrmt0
crw-rw-rw- 1 bin     8,  2 Jan 23 16:35 null
crw-rw-rw- 1 root   12,  0 Dec 31 19:02 rmt0
brw-r--r-- 1 root    6,  0 Dec 31 19:02 rp0
brw-r--r-- 1 root    6, 15 Dec 31 19:02 rp3
crw-r--r-- 1 root   14,  0 Dec 31 19:02 rrp0
crw-r--r-- 1 root   14, 15 Dec 31 19:02 rrp3
brw-r--r-- 1 root    6,  1 Dec 31 19:02 swap
crw-rw-rw- 1 bin    17,  0 Jan 10 15:40 tty
```

1.9 Create a filesystem on the second RP06 to hold /usr

With the devices attached and working and the special files representing them available, it is time to create a filesystem for the /usr partition and copy files from tape into the filesystem. We will use mkfs to create the filesystem and icheck to check the result.

```
# cd /
# etc/mkfs /dev/rp3 322278
isize = 65496
m/n = 3 500
# icheck /dev/rp3
/dev/rp3:
files      2 (r=1,d=1,b=0,c=0)
used       1 (i=0,ii=0,iii=0,d=1)
free  314088
missing    0
```

1.10 Restore the /usr filesytem from the distribution tape

We will use dd to move the tape to the appropriate starting point (skipping 6 files and setting the tape to point at the seventh file, a dump of rp3). Then we will use restor to restor the files in that tape file. Note that we are reading from nrmt0, the non-rewinding tape device, to position the tape and rmt0, the rewinding tape device, to restore the content.

```
# dd if=/dev/nrmt0 of=/dev/null bs=20b files=6
202+80 records in
202+75 records out
# restor rf /dev/rmt0 /dev/rp3
last chance before scribbling on /dev/rp3. [press enter]
end of tape
```

1.11 Mount /usr and copy the boot block onto the first RP06

Finally, copy a boot block from the restored /usr to the first block of our first disks' root partition. This will allow us to boot from the disk.

```
# /etc/mount /dev/rp3 /usr
# dd if=/usr/mdec/hpuboot of=/dev/rp0 count=1
0+1 records in
0+1 records out
```

1.12 Shut down gracefully

The installation is complete. It is a good idea to ensure that the superblocks of our filesystems are written (sync'ed) to disk before we virtually turn the power off.

```
# sync; sync; sync
```

Halt the Unix system by typing CTRL-E.

```
# CTRL-E
Simulation stopped, PC: 002306 (MOV (SP)+,177776)

Exit the simulator by tying q at the sim prompt:
sim> q
Goodbye
```

1.13 Test the restored system

Next, we will create an ini file that is appropriate for normal booting. We'll also upgrade the system to a beefier PDP-11 in the process, log in as root and test the pristine system.

1.13.1 Create a normal single session boot OpenSIMH ini file and boot normally

The following ini file is similar to the initial boot file, with the following changes. First, it includes comments and creates a PDP-11/70 with 2 Megs of memory. The second change is the removal of the tape, this is optional. The last change is that we boot directly from the rp06 instead of from tape.

```
cat > nboot.ini << "EOF"
echo
echo After Disabling XQ is displayed type in boot
echo and at the : prompt type in hp(0,0)unix
echo
set cpu 11/70
set cpu 2M
set cpu idle
set tto 7b
set rp0 rp06 noautosize
att rp0 rp06-0.disk
set rp1 rp06 noautosize
att rp1 rp06-1.disk
boot rp0
EOF
```

When running, note that the simulator does not initially provide a recognizable prompt, just type boot and the familiar colon prompt should appear.

```
pdp11 nboot.ini

PDP-11 simulator Open SIMH V4.1-0 Current        git commit id: 43963943

After Disabling XQ is displayed type in boot
and at the : prompt type in hp(0,0)unix

Disabling XQ
boot
Boot
: hp(0,0)unix
mem = 2020544
#
```

1.13.2 Log into multi-user mode as root

Rather than continuing on to single user mode, it is just a matter of pressing CTRL-D to obtain a properly configured multi-user mode console. root is the username and password.

```
# CTRL-D

# RESTRICTED RIGHTS: USE, DUPLICATION, OR DISCLOSURE
IS SUBJECT TO RESTRICTIONS STATED IN YOUR CONTRACT WITH
WESTERN ELECTRIC COMPANY, INC.
WED DEC 31 20:02:48 EST 1969

login: root
Password:
You have mail.
#
```

1.13.3 Read mail

Read mail to assure yourself that something (mail) works :)! When you're satisfied quit mail by typing `x` to keep the message or `d` to delete the message.

```
# mail
From bin Thu Jan 11 19:28:15 1979
Secret mail has arrived.

? x
# sync; sync; sync
```

Halt the Unix system by typing CTRL-E.

```
# CTRL-E
Simulation stopped, PC: 002306 (MOV (SP)+,177776)

Exit the simulator by tying q at the sim prompt:
sim> q
Goodbye
$
```

1.14 Backup the working pristine system

Save a pristine working copy of the system before adding a regular user.

```
cd ..
tar cvjf save/v7-pristine.tar.bz2 work

ls -lh save/
total 5.8M
-rw-rw-r-- 1 wsenn wsenn 6.5M May 23 11:12 v7-pristine.tar.bz2
```

2 Create a user and set sane TTY options

2.1 Start the simulator

```
cd work
pdp11 nboot.ini

PDP-11 simulator Open SIMH V4.1-0 Current        git commit id: 43963943

After Disabling XQ is displayed type in boot
and at the : prompt type in hp(0,0)unix

Disabling XQ
boot
Boot
: hp(0,0)unix
mem = 2020544
#
```

2.2 Log into multi-user mode as root

```
# CTRL-D

# RESTRICTED RIGHTS: USE, DUPLICATION, OR DISCLOSURE
IS SUBJECT TO RESTRICTIONS STATED IN YOUR CONTRACT WITH
WESTERN ELECTRIC COMPANY, INC.
WED DEC 31 20:02:48 EST 1969

login: root
Password:
You have mail.
#
```

2.3 Create a .profile for root with sane TTY defaults

Add an stty line to .profile that sets the erase character to `^h` and kill to `^u` , rather than the defaults `#` and `@` respectively. This change assumes that `^h` is the delete character. See the Appendix C for a note on this, if you are on a Mac. After you create the root .profile, logout by typing `^d`. Also, remember that the interrupt key is ^DEL, rather than the more typical ^C as discussed previously.

```
echo 'stty erase "^h" kill "^u" nl0 cr0 9600'> .profile
# ^d
```

2.4 Add a regular user

Adding a new user account for normal operations requires a litte bit of work. Add a line to `/etc/passwd`, create a home directory, change the ownership of the new directory to the new user, login as the user, change the password (optional), and set sane TTY defaults.

Editing `/etc/passwd` is not difficult, however, you want to be careful and try not to make a mistake because it could prevent you from being able to login again. To learn more about the file, `man 5` passwd will tell you all about it. Basically the file consists of entries, one for each user, with 7 fields delimited by colons. The fields are:

- name (login name, contains no upper case)

- encrypted password
- numerical user ID
- numerical group ID
- GCOS job number, box number, optional GCOS user-id
- initial working directory
- program to use as Shell

In this case, I will use `wsenn` for the user name, an empty password field, `100` for the user id (it's not taken yet and it reserves double digit ids for services), `3` for the group id (bin, mirroring the dmr user), an empty GCOS field, `/usr/wsenn` for the user's home directory, and an empty shell field (defaults to /bin/sh).

1. Login as root.

```
login: root
Password:
You have mail.
#
```

2. edit /etc/passwd noting the existing users.

```
# ed /etc/passwd
141
1,$p
root:VwL97VCAx1Qhs:0:1::/:
daemon:x:1:1::/:
sys::2:2::/usr/sys:
bin::3:3::/bin:
uucp::4:4::/usr/lib/uucp:/usr/lib/uucico
dmr::7:3::/usr/dmr:
a
wsenn::100:3::/usr/wsenn:
.
w
167
q
```

3. Create a home directory for the user and change the owner.

```
# mkdir /usr/wsenn
# chown wsenn /usr/wsenn
# chgrp bin /usr/wsenn
```

4. Login as the new user, change the password and create a new .profile.

```
# login wsenn
$ passwd
Changing password for wsenn
New password:
Retype new password:
$ echo 'stty erase "^h" kill "^u" nl0 cr0 9600'> .profile
$ stty
speed 9600 baud
erase = '#'; kill = '@'
even odd -nl echo -tabs
# ^d
```

5. Logout and back in again as the new user.

```
# CTRL-D

login: wsenn
Password:
$ stty
speed 9600 baud
erase = '^H'; kill = '^U'
even odd -nl echo -tabs
$ who
wsenn    console Dec 31 19:07
```

2.5 Shutdown

The system is now operational for single sessions. To exit, write the superblock and halt Unix, then exit the simulation.

```
$ sync; sync; sync
$ CTRL-E
Simulation stopped, PC: 002306 (MOV (SP)+,177776)

Exit the simulator by tying q at the sim prompt:
sim> q
Goodbye
```

2.6 Backup the working multi-user system

Save a working copy of the system before adding multi-session support.

```
cd ..
tar cvjf save/v7-multi-user.tar.bz2 work

ls -lh save/
total 9.7M
-rw-rw-r-- 1 wsenn wsenn 6.5M May 23 11:20 v7-multi-user.tar.bz2
-rw-rw-r-- 1 wsenn wsenn 6.5M May 23 11:12 v7-pristine.tar.bz2
```

3 Set up multi-session support

In this section the system is configured to allow more than one user to login at the same time. One user at a time can log in to the console, and with the following changes, up to 16 more sessions can be running simultaneously over telnet. A DZ-11 with 16 lines is added to the OpenSIMH configuration and configured in the system.

3.1 Boot from the multi-user nboot.ini file

```
cd work
pdp11 nboot.ini

PDP-11 simulator Open SIMH V4.1-0 Current        git commit id: 43963943

After Disabling XQ is displayed type in boot
and at the : prompt type in hp(0,0)unix

Disabling XQ
boot
Boot
: hp(0,0)unix
mem = 2020544
#
```

3.2 Log into multi-user mode as root

Rather than continuing on to single user mode, it is just a matter of pressing CTRL-D to obtain a properly configured multi-user mode console. root is the username and password.

```
# CTRL-D

# RESTRICTED RIGHTS: USE, DUPLICATION, OR DISCLOSURE
IS SUBJECT TO RESTRICTIONS STATED IN YOUR CONTRACT WITH
WESTERN ELECTRIC COMPANY, INC.
WED DEC 31 20:02:48 EST 1969

login: root
Password:
You have mail.
#
```

3.3 Reconfigure the system to support 16 DZ-11 lines

In order to connect to the system via the OpenSIMH DZ device through telnet, unix needs to support the DZ-11 device and have tty's available and listening. Once these changes have been made, use mboot.ini to boot the system. In the future, if you want to boot without the DZ-11 lines enabled it is critically important that you first disable the ttys in /etc/ttys, then use nboot.ini and the unix kernel to boot the system.

Add dz support to mkconf.c

```
# cd /usr/sys/conf
# ed mkconf.c
14422
249a
"dz",
0, 300, CHAR+INTR,
" dzin; br5+%d.\ndzou; br5+%d.",
".globl _dzrint\ndzin: jsr r0,call; jmp _dzrint\n",
".globl _dzxint\ndzou: jsr r0,call; jmp _dzxint\n",
"",
" dzopen, dzclose, dzread, dzwrite, dzioctl, nulldev, dz_tty,",
"",
"int dzopen(),dzclose(),dzread(),dzwrite(),dzioctl();\nstruct tty dz_tty[];",
.
45a
"dz",
.
w
14740
q

# cc mkconf.c
# mv a.out mkconf
```

Configure the system with a dz

```
# cp hptmconf myconf
# echo dz >> myconf
# mkconf < myconf
console at 60
clock at 100
clock at 104
parity at 114
tm at 224
hp at 254
dz at 300
```

Rebuild and deploy a new multi-session capable kernel

```
# rm l.o c.o
# make
as - -o l.o l.s
cc  -c c.c
ld -o unix -X -i l.o mch.o c.o ../sys/LIB1 ../dev/LIB2

# sum unix
43924   108

Save the kernel:
mv unix /munix
```

Enable 16 ttys

```
# ed /etc/ttys
266
2,17s/./1/
w
266
q

# sed -n '1,17p' /etc/ttys

14console
10tty00
10tty01
10tty02
10tty03
10tty04
10tty05
10tty06
10tty07
10tty08
10tty09
10tty10
10tty11
10tty12
10tty13
10tty14
10tty15
```

Determine the major device number of dz from c.c:

```
# cat /usr/sys/conf/c.c |grep dz
int    dzopen(), dzclose(), dzread(), dzwrite(), dzioctl();
struct  tty     dz_tty[];
dzopen, dzclose, dzread, dzwrite, dzioctl, nulldev, dz_tty,      /* dz = 19 */
```

It's 19, use that to create the devices:

```
/etc/mknod /dev/tty00 c 19 0
/etc/mknod /dev/tty01 c 19 1
/etc/mknod /dev/tty02 c 19 2
/etc/mknod /dev/tty03 c 19 3

/etc/mknod /dev/tty04 c 19 4
/etc/mknod /dev/tty05 c 19 5
/etc/mknod /dev/tty06 c 19 6
/etc/mknod /dev/tty07 c 19 7

/etc/mknod /dev/tty08 c 19 8
/etc/mknod /dev/tty09 c 19 9
/etc/mknod /dev/tty10 c 19 10
/etc/mknod /dev/tty11 c 19 11

/etc/mknod /dev/tty12 c 19 12
/etc/mknod /dev/tty13 c 19 13
/etc/mknod /dev/tty14 c 19 14
/etc/mknod /dev/tty15 c 19 15
chmod 640 /dev/tty??
```

Check the special files were properly created

```
# ls -l /dev/tty??
crw-r----- 1 root   19,  0 Dec 31 19:10 /dev/tty00
crw-r----- 1 root   19,  1 Dec 31 19:10 /dev/tty01
crw-r----- 1 root   19,  2 Dec 31 19:10 /dev/tty02
crw-r----- 1 root   19,  3 Dec 31 19:10 /dev/tty03
crw-r----- 1 root   19,  4 Dec 31 19:10 /dev/tty04
crw-r----- 1 root   19,  5 Dec 31 19:10 /dev/tty05
crw-r----- 1 root   19,  6 Dec 31 19:10 /dev/tty06
crw-r----- 1 root   19,  7 Dec 31 19:10 /dev/tty07
crw-r----- 1 root   19,  8 Dec 31 19:10 /dev/tty08
crw-r----- 1 root   19,  9 Dec 31 19:10 /dev/tty09
crw-r----- 1 root   19, 10 Dec 31 19:10 /dev/tty10
crw-r----- 1 root   19, 11 Dec 31 19:10 /dev/tty11
crw-r----- 1 root   19, 12 Dec 31 19:10 /dev/tty12
crw-r----- 1 root   19, 13 Dec 31 19:10 /dev/tty13
crw-r----- 1 root   19, 14 Dec 31 19:10 /dev/tty14
crw-r----- 1 root   19, 15 Dec 31 19:10 /dev/tty15
```

3.4 Shutdown

The system is now 'fully' operational for multiple sessions. To exit, write the superblock and halt Unix, then exit the simulation.

```
$ sync; sync; sync
$ CTRL-E
Simulation stopped, PC: 002306 (MOV (SP)+,177776)

Exit the simulator by tying q at the sim prompt:
sim> q
Goodbye
```

3.5 Create a multi-session capable normal boot OpenSIMH ini file

The following ini file is similar to the single session boot file, with the addition of lines enabling and configuring the DZ-11 device with 16 lines listening on the host system on port 2222. You can change the port to any unused port.

```
cat > mboot.ini << "EOF"
echo
echo After Disabling XQ is displayed type in boot
echo and at the : prompt type in hp(0,0)munix
echo
set cpu 11/70
set cpu 2M
set cpu idle
set tto 7b
set rp0 rp06 noautosize
att rp0 rp06-0.disk
set rp1 rp06 noautosize
att rp1 rp06-1.disk
set dz en
set dz line=16
att dz 2222
boot rp0
EOF
```

3.6 Boot from the multi session ini file

```
pdp11 mboot.ini

PDP-11 simulator Open SIMH V4.1-0 Current        git commit id: 43963943

After Disabling XQ is displayed type in boot
and at the : prompt type in hp(0,0)munix

Disabling XQ
/home/wsenn/retro-workarea/v7/work/mboot.ini-15> att dz 2222
%SIM-INFO: Listening on port 2222
boot
Boot
: hp(0,0)munix
mem = 2018496
# RESTRICTED RIGHTS: USE, DUPLICATION, OR DISCLOSURE
IS SUBJECT TO RESTRICTIONS STATED IN YOUR CONTRACT WITH
WESTERN ELECTRIC COMPANY, INC.
WED DEC 31 19:11:35 EST 1969

login: root
Password:
You have mail.
#
```

3.7 Telnet into the V7 instance

On the host system, open two additional terminal windows and telnet into the V7 instance on both.

```
telnet localhost 2222
Trying ::1...
Connected to localhost.
Escape character is '^]'.


Connected to the PDP-11 simulator DCI device, line 0

??lo?i?:?
```

Ignore the characters that look like trash and press enter to get a login prompt and login

```
login: dmr
$
```

```
telnet  localhost 2222
Trying ::1...
Connected to localhost.
Escape character is '^]'.


Connected to the PDP-11 simulator DCI device, line 1

??lo?i?:?
login: root
Password:
You have mail.
# who
root     console Dec 31 19:13
dmr      tty00   Dec 31 19:12
root     tty01   Dec 31 19:12
```

To end your sessions, type `^d to logout of the unix session and ^]` and `q` to exit the telnet session, then `^e` and `q` to end the OpenSIMH session.

3.8 Backup the working multi-session system

Save a working copy of the system before proceeding.

```
cd ..
tar cvjf save/v7-multi-session.tar.bz2 work

ls -lh save/
total 17M
-rw-rw-r-- 1 wsenn wsenn 6.6M May 23 11:32 v7-multi-session.tar.bz2
-rw-rw-r-- 1 wsenn wsenn 6.5M May 23 11:20 v7-multi-user.tar.bz2
-rw-rw-r-- 1 wsenn wsenn 6.5M May 23 11:12 v7-pristine.tar.bz2
```

From this point onward, you should use mboot.ini and the munix kernel to boot your system [8]. If you want to use nboot.ini and the unix kernel, be sure to edit /etc/ttys and disable the ttys that you have enabled.

[8] Once you have enabled ttys in /etc/ttys, you will need to boot into a system that supports them.

4 Miscellaneous

In this section, it is assumed that you are booted into a fully multi-user, multi-session capable Unix v7 instance and have started multi-user mode (you have pressed CTRL-D at the login: prompt).

4.1 login as dmr and say hello, world[9]

In honor of the user dmr, login to the system as dmr, set some sane TTY options, and compile this, most well known program, as it was originally written, straight from the seminal work *The C Programming Language* by Brian W. Kernighan and dmr himself, Dennis M. Ritchie.

1. Login as dmr with no password.

```
login: dmr
$
```

2. Create a sane profile for the dmr account as well, log out, and back in again.

```
echo 'stty erase "^h" kill "^u" nl0 cr0 9600'> .profile

$ ^D
login: dmr

$
```

3. Create hello.c

   Use cat, instead of ed, to create the file. In this case ^D indicates end of file and isn't logging you out of the shell.

```
cat > hello.c
main()
{
printf("hello, world\n");
}
^D
```

4. Compile hello.c

```
$ cc -o hello hello.c
$ ./hello
hello, world
```

4.2 Setup learn(1)

Learn is a tutorial system that was included in early versions of Unix. It is installed by default, but is not in a working state. This section shows how to get it set up and working properly. Although there is no learn(1) man page, Volume II of the Programmer's Manual contains an informative article by Brian Kernighan entitled, *LEARN – Computer-Aided Instruction on UNIX (Second Edition)*. Learn comes with 6 Computer Aided Instruction (CAI) modules:

- basic file handling commands

[9] Suggestion by Warren Young

- the UNIX text editor *ed*
- advanced file handling
- the *eqn* language for typing mathematics
- the "–ms" macro package for document formatting
- the C programming language

1. login as the user you created above

```
login: wsenn
Password:
$
```

2. build learn

```
# cd /usr/src/cmd/learn
# make
cc -O -c copy.c
cc -O -c dounit.c
cc -O -c learn.c
cc -O -c list.c
cc -O -c mem.c
cc -O -c makpipe.c
cc -O -c maktee.c
cc -O -c mysys.c
cc -O -c selsub.c
cc -O -c selunit.c
cc -O -c start.c
cc -O -c whatnow.c
cc -O -c wrapup.c
cc -n -s -o learn -O copy.o dounit.o learn.o list.o mem.o  makpipe.o maktee.o \
mysys.o selsub.o selunit.o  start.o whatnow.o wrapup.o
cc -O -s -n tee.c -o tee
cc -O -s -n lcount.c -o lcount
cp learn /bin
cp tee /usr/lib/learn
cp lcount /usr/lib/learn
rm learn tee lcount *.o
Now do 'make lessons' if you need to extract the lesson archives
Then do 'make play; make log' to make playpen and log directories
```

3. make lessons

   Ignore the Error messages. They are because the files don't yet exist.

```
# make lessons
rm -r /usr/lib/learn/files
rm: /usr/lib/learn/files nonexistent
*** Error code 1 (ignored)
mkdir /usr/lib/learn/files
(cd /usr/lib/learn/files; ar x ../files.a)
rm -r /usr/lib/learn/editor
rm: /usr/lib/learn/editor nonexistent
*** Error code 1 (ignored)
mkdir /usr/lib/learn/editor
(cd /usr/lib/learn/editor; ar x ../editor.a)
rm -r /usr/lib/learn/morefiles
rm: /usr/lib/learn/morefiles nonexistent
*** Error code 1 (ignored)
mkdir /usr/lib/learn/morefiles
(cd /usr/lib/learn/morefiles; ar x ../morefiles.a)
rm -r /usr/lib/learn/macros
rm: /usr/lib/learn/macros nonexistent
*** Error code 1 (ignored)
```

```
mkdir /usr/lib/learn/macros
(cd /usr/lib/learn/macros; ar x ../macros.a)
rm -r /usr/lib/learn/eqn
rm: /usr/lib/learn/eqn nonexistent
*** Error code 1 (ignored)
mkdir /usr/lib/learn/eqn
(cd /usr/lib/learn/eqn; ar x ../eqn.a)
rm -r /usr/lib/learn/C
rm: /usr/lib/learn/C nonexistent
*** Error code 1 (ignored)
mkdir /usr/lib/learn/C
(cd /usr/lib/learn/C; ar x ../C.a)
```

4. Create the playpen and log directories These are required to run the learn program. Ignore the messages about nonexistent directories, they just mean the directories don't exist yet.

```
make play; make log
rm -r /usr/lib/learn/play; mkdir /usr/lib/learn/play; chmod +w /usr/lib/learn/play
rm: /usr/lib/learn/play nonexistent
rm -r /usr/lib/learn/log; mkdir /usr/lib/learn/log; chmod +w /usr/lib/learn/log
rm: /usr/lib/learn/log nonexistent
```

4.3 Test learn

1. Start learn

```
$ learn
These are the available courses -
files
editor
morefiles
macros
eqn
C
If you want more information about the courses,
or if you have never used 'learn' before,
type 'return'; otherwise type the name of
the course you want, followed by 'return'.
files
```

2. Choose files

```
files
If you were in the middle of this subject
and want to start where you left off, type
the last lesson number the computer printed.
To start at the beginning, just hit return.

This course will help you learn about basic file
handling commands.  You should first understand
the special characters # and @:
# cancels the previous character typed;
@ cancels the line being typed.
If you make a typing mistake, you can use these characters to
correct it before you finish the line and the computer won't
ever know about it.  For example, what will the
computer really receive if you type

st#he@
thf#e

at it?  Reply "answer WORD" where WORD is the word
as it will be interpreted.  For example, if
you think it will get 'dog', type

answer dog

If you think it will receive the word "bark", type

answer bark

Don't forget to leave a space between "answer" and the word
and to hit RETURN at the end of the line.
Don't use any quotation marks in your answer.
$
```

3. Interact with learn

   Learn has its own shell environment. While it is running, it will prompt you with $. This looks like a normal shell prompt, but learn only understands learn commands.

```
$ answer bark

Sorry, that's not right.  Do you want to try again?  y
Try the problem again.
$ answer the

Good.  Lesson 0.1a (0)

You should also understand a few simple commands.
When UNIX types a "$" at you, you can type a command.
For example, if you type "date" the computer will tell
you the current date and time. If you see "$" and type "who",
UNIX will tell you who is logged on at present.  There are
many other commands, too.  You must type a RETURN at the
end of each command line.

Try the "date" command now: find out what
date it is, and after the computer has responded, type "ready".
And don't forget the RETURN!
$
```

4. Exit learn

   This is a bit tricky. As discussed above, learn has it's own shell environment. To exit it requires an interrupt signal. This is the ^DEL key combination discussed earlier.

```
$ ^DEL
Interrupt.
Want to go on?  n
Bye.
$
```

4.4 Backup the working system

Save a working copy of the system any time you make significant changes.

```
cd ..
tar cvjf save/v7-working-system.tar.bz2 work

ls -lh save/
total 25M
-rw-rw-r-- 1 wsenn wsenn 6.6M May 23 11:32 v7-multi-session.tar.bz2
-rw-rw-r-- 1 wsenn wsenn 6.5M May 23 11:20 v7-multi-user.tar.bz2
-rw-rw-r-- 1 wsenn wsenn 6.5M May 23 11:12 v7-pristine.tar.bz2
-rw-rw-r-- 1 wsenn wsenn 6.6M May 23 12:04 v7-working-system.tar.bz2
```

4.5 Celebrate

Celebrate your successful installation and first use of a classic operating system of the 1970's and a job well done. Try out mail to communicate between users asynchronously and write to communicate synchronously.

Appendices

Appendix A Build OpenSIMH from git repository

In 2022, SIMH became OpenSIMH and a steering committee was formed:

1. Clem Cole
2. Richard Cornwell
3. Paul Koning
4. Timothe Litt
5. Seth Morabito
6. Bob Supnik

More information about the OpenSIMH project, the steering committee and its goals can be found on the project's website at `https://opensimh.org/about/`.

The following instructions are meant to get a working pdp11 simulator built from source.

1. Install Dependencies (this varies between oses). I use Linux Mint, so I use APT to install dependencies. On Mac use macports or homebrew. On linux, use your favorite package manager to add any dependencies identified at the time you try to build. On Debian based systems, this is typically apt, on Redhat and its kin, it's yum, zypper or something similar put on top of rpm. On Windows, use cygwin or ming.

   ```
   sudo apt install libpcap-dev libpcre3-dev vde2 libsdl2-dev libsdl2-ttf-dev
   libedit-dev libvdeplug-dev
   ```

2. Download the source code

   ```
   git clone https://github.com/open-simh/simh.git open-simh-newest
   ```

3. Compile the simulator

   You don't need SDL for this note, but it is a dependency for pdp11 that you may want to have. The build script will warn you about it, but it's up to you whether or not you want to go to the trouble of satisfying the dependency. I haven't needed, but YMMV.

   ```
   cd open-simh-newest
   make clean
   make pdp11
   ```

4. Install the simulator

   ```
   cp BIN/pdp11 ~/bin/pdp11
   ```

Appendix B mktape.py source code

```
cat > mktape.py <<"EOF"
#!/usr/bin/env python

# Written by Will Senn. Create a bootable SimH tap file to install the system.
# Originally Based on Hellwig Geisse's mktape.c and inspired by various Perl scripts
# Argument handling borrowed from Allen Garvin's mk-dist-tape.py
#
# created 20171012.1924
# coverted to python 20220120.1019 in prep for a better generalized version
# generalized by adding argument handling

# ./mktape.py -o v7.tap f0:512 f1:512 f2:512 f3:512 f4:512 f5 f6
# equivalent to ./mktape.py -7
# shasum v7.tap
# e6188335c0c9a3e3fbdc9c29615f940233722432  v7.tap
# ./mktape.py -o bsd42.tap stand:512 miniroot rootdump srcsys.tar usr.tar vfont.tar
#    src.tar new.tar ingres.tar
# shasum bsd42.tap
# 8810274832cadf01083eeb228368896048b35a5a  bsd42.tap

import sys
import struct
import argparse

EOF = b"\x00\x00\x00\x00"
EOM = b"\xFF\xFF\xFF\xFF"
files = []
blkszs = []

# add arguments for default v7 tape creation, and 4.2bsd
parser = argparse.ArgumentParser(description="Create distribution tapes for simh")
parser.add_argument("--v7", "-7", action='store_true',
    help="build v7 tape (requires tape files in dir)")
parser.add_argument("--bsd42", action='store_true',
    help="build bsd42 tape (requires tape files in dir)")
parser.add_argument("--bsd43", action='store_true',
    help="build bsd43 tape (requires tape files in dir)")
parser.add_argument("--blocksize", "-b", type=int,
    help="set default block size (default 10240)", default=10240)
parser.add_argument("-output", "-o", metavar="FILE",
    help="output to file (if omitted, sends to stdout)")
parser.add_argument("file", nargs="*",
    help="files to add in order (append with :bs where bs is block size)")
args = parser.parse_args()

if args.v7:
    filelist = ["f0:512", "f1:512", "f2:512", "f3:512", "f4:512", "f5", "f6"]
    blocksize = 10240
    outfile = "v7.tap"
```

```
elif args.bsd42:
    filelist = ["stand:512", "miniroot", "rootdump", "srcsys.tar",
        "usr.tar", "vfont.tar", "src.tar", "new.tar", "ingres.tar"]
    blocksize = 10240
    outfile = "bsd42.tap"
elif args.bsd43:
    filelist = ["stand:512", "miniroot", "rootdump", "usr.tar",
        "srcsys.tar", "src.tar", "vfont.tar", "new.tar", "ingres.tar",
        "install.tar"]

    blocksize = 10240
    outfile = "bsd43.tap"

else:
    blocksize = args.blocksize
    if args.output:
        outfile = args.output
    else:
        outfile = "out.tap"
        filelist = args.file

for f in filelist:
    if ":" in f:
        fn = f.split(":")[0]
        bs = f.split(":")[1]
        if not bs.isdigit():
            parser.print_help()
            print(f"\nERROR: blocksize {bs} in {f} is not a positive integer")
            sys.exit(1)
        bs = int(bs)
    else:
        fn = f
        bs = blocksize
        files.append(fn)
        blkszs.append(bs)
try:
    fdout = open(outfile, "wb")
except IOError as e:
    print(f"{outfile}: errno {e.errno}: {e.strerror}")
    sys.exit(1)


for f in range(0, len(files)):
    file = files[f]
    blocksize = blkszs[f]
    packedlen = struct.pack("<I", blocksize)
    try:
        fdin = open(file, "rb")
    except IOError as e:
        print(f"{file}: errno {e.errno}: {e.strerror}")
        sys.exit(1)
    blockswritten = 0
    bytes = fdin.read(blocksize)
```

```
    while(bytes):
        buffer = bytes
        if len(bytes) < blocksize:
            buffer += b"\x00" * (blocksize - len(bytes))
        fdout.write(packedlen)
        fdout.write(buffer)
        fdout.write(packedlen)
        blockswritten += 1
        bytes = fdin.read(blocksize)
    fdin.close()
    fdout.write(EOF)

    print(f"file {f}: block size {blocksize}: {blockswritten} records")
    print(f"file {f}: eof after {blockswritten} records: {blocksize * blockswritten} bytes")
fdout.write(EOF)
fdout.write(EOM)
```

Appendix C mktape.pl source code

```
cat > mktape.pl <<"EOF"
#!/usr/bin/perl
use strict;
# Written by Will Senn. Create a bootable SimH tap file to install the system.
# Inspired by various Perl scripts and based on Hellwig Geisse's mktape.c
#
# modified 20220517 tape compatibily issue EOF/EOM marker -fix Clem Cole
# modified 20171012 binmode required for windows use

my @files = ("f0", "f1", "f2", "f3", "f4", "f5", "f6");
my @blkszs = (512, 512, 512, 512, 512, 10240, 10240);

my $outfile = "v7.tap";

# EOF is end of a tape file.
my $EOF = "\x00\x00\x00\x00";
# EOM end of physical medium (tape)
my $EOM = "\xFF\xFF\xFF\xFF";

open(OUTFILE, ">$outfile") || die("Unable to open $outfile: $!\n");
binmode(OUTFILE);
for(my $i = 0; $i <= $#files; $i++) {
    my ($bytes, $blocksize, $buffer, $packedlen, $blockswritten, $file) = 0;

    $file = $files[$i];
    $blocksize = $blkszs[$i];
    $packedlen = pack("V", $blocksize);

    open(INFILE, $file) || die("Unable to open $file: $!\n");
    binmode(INFILE);
    while($bytes = read(INFILE, $buffer, $blocksize)) {
        $buffer .= $bytes < $blocksize ? "\x00" x ($blocksize - $bytes) : "";
        print OUTFILE $packedlen, $buffer, $packedlen;
        $blockswritten++;
    }
    close(INFILE);
    print OUTFILE $EOF;
    printf "%s: %d bytes = %d records (blocksize %d bytes)\n", $file,
    $blockswritten * $blocksize, $blockswritten, $blocksize;
}
# write logical EOT (i.e. second EOF in a row per ANSI spec and UNIX driver)
print OUTFILE $EOF;
# now write physical EOM
print OUTFILE $EOM
EOF
```

Appendix D Various Notes

This section has helpful notes that don't fit anywhere else. They are generally system or situation specific so YMMV.

- Configure the Mac terminal so that the delete key provides backspace functionality.

  Open up terminal preferences and click on the default profile, or create a custom profile, click the Keyboard tab, and click + to add a key customization. Select Key: <-Delete, Modifier: None, Action: Send Text:, put cursor in the text box under Send Text: and type CTRL-h. It will be replaced with \010 (ASCII code for backspace). This change combined with the stty erase '^h' change we made for dmr and root will cause the terminal to behave as one might expect - type helli, hit delete, and the cursor will back up over the i. It won't delete the i from the screen, but when you type o, it will replace the i onscreen and in the input buffer with o.