SETTING UP UNIX—Sixth Edition

An OpenSIMH Laboratory for Reading Lions

Ken Thompson and Dennis Ritchie's *Setting Up UNIX—Sixth Edition* described the installation of Sixth Edition UNIX on contemporary PDP-11 hardware. Hereafter, this system is referred to as V6. This guide follows their general approach using OpenSIMH, with the narrower purpose of constructing the PDP-11/40 environment assumed by John Lions' *A Commentary on the UNIX Operating System*. It is not a complete guide to V6 system administration, nor a guide to working through Lions' commentary. Its object is to produce a small, reproducible V6 installation containing the running kernel, complete source tree, and system documentation needed to read Lions against the system he describes. It is intended to be used with Lions' commentary and source booklet, including my updated and expanded edition of *UNIX Operating System Source Code Level Six*, which preserves Lions' source-sheet numbering while adding indexes, cross-references, and file locations for a running V6 system. Where choices are required, the guide follows the V6 distribution and selects the configuration most closely corresponding to that discussed by Lions. All of this takes place on an ordinary modern computer: OpenSIMH simulates the PDP-11/40 in software, and the operator works entirely from a terminal window on the host. Any errors or omissions are my own.

What This Guide Produces

Following this guide produces a complete, restorable laboratory for reading Lions against a running V6 system. The completed laboratory includes:

- an OpenSIMH simulation of a PDP-11/40;
- a V6 kernel rebuilt for the simulated machine;
- the complete operating-system and program source tree;
- the complete documentation filesystem;
- a device configuration corresponding to the kernel, special files, and simulated peripherals;
- a verified baseline archive from which the completed system can be restored; and
- a working correspondence among Lions' commentary, the numbered source listing, and the files of the installed system.

System Requirements

- A Unix-like host (Linux, BSD, or macOS) with a Unix shell
- A C compiler
- curl
- OpenSIMH 4.x (https://opensimh.org; source at https://github.com/open-simh/simh)

License

The V6 distribution and other ancient UNIX releases are made available for personal, noncommercial use under the Caldera license (https://www.tuhs.org/Archive/Caldera-license.pdf). Obtain a copy and keep it with the laboratory materials.

The Laboratory

John Lions' *A Commentary on the UNIX Operating System* was prepared for students studying V6 on a PDP-11/40. The original course materials circulated as two volumes: *UNIX Operating System Source Code Level Six* and *A Commentary on the UNIX Operating System*. They were later published together as Lions' *Commentary on UNIX 6th Edition with Source Code*. Lions' laboratory comprised the PDP-11/40 and its peripheral equipment, the installed V6 system and source code, and such materials as the *UNIX Programmer's Manual, UNIX Operating System Source Code Level Six*, and *A Commentary on the UNIX Operating System*.

Reproducing that laboratory physically is now difficult. The PDP-11/40, controllers, disk and tape equipment, terminal, and printer together formed a substantial and expensive installation. An RK05 disk drive occupied 10.5 inches of rack space and accepted a removable cartridge containing a single 14-inch disk platter. On a PDP-11, each cartridge stored about 2.5 megabytes. A working system required several drives, together with the associated controllers, cabling, cabinets, and media. Surviving equipment is now scarce, delicate, and largely confined to museums and private collections.

The laboratory constructed here recreates that instructional setting, but its components take different forms. The PDP-11/40 and its peripheral devices are supplied by OpenSIMH. The V6 distribution tape is represented by a tape-image file, and each removable RK05 cartridge by a disk-image file. The terminal is provided by a terminal program running on the modern host computer. The *UNIX Programmer's Manual*, *UNIX Operating System Source Code Level Six*, and *A Commentary on the UNIX Operating System* are represented by modern digital copies. This guide provides the procedure by which these parts are assembled into a working environment.

The modern host computer stands outside the reconstructed system. It stores the laboratory files, runs OpenSIMH, and provides the terminal window through which the operator controls the simulator and communicates with V6. OpenSIMH supplies the virtual PDP-11/40, its memory, controllers, and peripheral devices. V6 then runs on that virtual machine. Commands entered in the terminal may therefore be directed first to the host shell, then to OpenSIMH, and finally to the V6 shell. The changing prompts shown throughout this guide indicate which environment is receiving input.

The correspondence with Lions' laboratory is direct. Lions' system had physical RK05 drives connected through an RK11 controller; this laboratory has simulated RK05 drives connected through a simulated RK11. Lions' removable cartridges contained 14-inch platters; ours are files stored on the host. His distribution arrived on magnetic tape; ours is a tape-image file attached to a simulated TM11 controller. Paper-tape input and output and line-printer output are represented by host files. The cabinets, motors, reels, and removable media have disappeared, but the processor and device interfaces presented to V6 remain those of the original system. V6 does not know the difference.

The completed installation occupies three nearly full RK05 disk packs. The first contains the executable system, the root filesystem, and the operating-system source. The second contains the remaining program source. The third contains the manuals and other documents intended for processing with roff or nroff. The first disk alone is sufficient to boot and operate V6. Reading Lions, however, calls for access to the complete source, commands, libraries, manuals, and supporting documents. The laboratory therefore installs all three filesystems.

Preparing the Host

This section prepares the modern host for constructing the laboratory: creating a workspace, obtaining the V6 distribution tape image, and building the `enblock` and `deblock` utilities needed to present that tape to OpenSIMH.

Create a working directory

The examples in this guide use `~/retro-workarea/v6`:

```
$ mkdir -p ~/retro-workarea/v6
$ cd ~/retro-workarea/v6
```

Obtain the distribution tape

Download the raw V6 distribution tape image:

```
$ curl -O https://www.tuhs.org/Archive/Distributions/Research/Ken_Wellsch_v6/
v6.tape.gz
```

Download Wolfgang Helbig's V6 fixes together with the source code for the `enblock` and `deblock` utilities:

```
$ curl -O https://www.tuhs.org/Archive/Distributions/Research/Bug_Fixes/V6enb/
v6enb.tar.gz
```

Unpack the archives

```
$ gunzip v6.tape.gz
$ tar xf v6enb.tar.gz
```

The working directory should now contain the raw distribution tape image, `v6.tape`, the v6enb tarball, `v6enb.tar.gz`, and the extracted `v6enb` directory containing the utility source code and related files.

Build the utilities

The distribution tape is archived as a raw sequence of 512-byte tape records. OpenSIMH expects a structured tape-image format that records the length of each record and represents tape marks explicitly. The `enblock` utility converts the raw tape image into the format expected by OpenSIMH. The `deblock` utility performs the reverse conversion.

Change into the source directory and compile the utilities:

```
$ cd v6enb
$ cc -std=gnu89 -Wno-implicit-function-declaration -Wno-implicit-int \
  -Wno-builtin-declaration-mismatch enblock.c -o enblock
$ cc -std=gnu89 -Wno-implicit-function-declaration -Wno-implicit-int \
  deblock.c -o deblock
```

These programs use conventions accepted by older C compilers, including implicit function declarations, implicit `int` return types, and declarations that conflict with modern built-in prototypes. The `-std=gnu89` option selects a compatible C dialect, while the warning options suppress diagnostics related to those historical conventions.

Copy the resulting programs into the working directory and return to it:

```
$ cp ??block ..
$ cd ..
```

Prompts and Environments

Commands in this guide are entered in several environments. The `$` prompt denotes the host shell, `sim>` the OpenSIMH monitor, `=` the standalone installation program, `@` the V6 disk bootstrap, and `#` the V6 super-user shell. `CTRL+E` suspends the simulated processor and returns control to OpenSIMH. `CTRL+D` signals end-of-file to programs reading from the terminal, such as cat.

Preparing the Distribution Tape

Preserve the downloaded `v6.tape` file unchanged. It is the master copy from which the OpenSIMH tape image can always be recreated.

Convert the raw tape image into an OpenSIMH-formatted tape image:

```
$ ./enblock < v6.tape > dist.tap
```

The resulting file, `dist.tap`, is the OpenSIMH representation of the V6 distribution tape that will be attached to the simulated TM11 tape controller during the installation of V6. The `deblock` utility is not required during installation but may later be used to recover a raw tape image from an OpenSIMH-formatted tape.

Preparing the PDP-11

We will begin constructing the simulated machine with a minimal PDP-11/40 configuration consisting of a TU10 magnetic-tape drive and four RK05 disk drives. This is an installation configuration rather than the completed laboratory system; the remaining devices will be added after V6 is running.

Create an OpenSIMH initialization file for the tape bootstrap:

```
$ cat > tboot.ini << "EOF"
set cpu 11/40

set tm0 locked
attach tm0 dist.tap

set rk0 en noautosize
set rk1 en noautosize
set rk2 en noautosize
set rk3 en noautosize

attach rk0 rk0
attach rk1 rk1
attach rk2 rk2
attach rk3 rk3

d cpu 100000 012700
d cpu 100002 172526
d cpu 100004 010040
d cpu 100006 012740
d cpu 100010 060003
d cpu 100012 000777

g 100000
EOF
```

The initialization file selects a PDP-11/40, attaches the converted distribution tape to the first TU10 drive, creates four RK05 disk images, deposits the six-word TU10 bootstrap at address `100000`, and executes it.

The tape is attached read-only:

```
set tm0 locked
```

This protects `dist.tap` from accidental modification.

The `noautosize` setting ensures that each newly created disk image remains configured as an RK05:

```
set rk0 en noautosize
```

The first three disk images will later hold the binary, source, and documentation filesystems.

The six `d cpu` commands deposit 16-bit PDP-11 words, expressed in octal, at successive memory addresses. Together they form the short TU10 bootstrap that reads the first tape record into memory at address `0`. The final command begins execution at `100000`.

Start OpenSIMH with the initialization file:

```
$ pdp11 tboot.ini
```

OpenSIMH will display output similar to:

```
PDP-11 simulator Open SIMH V4.1-0 Current        git commit id: a1f57fa3
Disabling XQ
tboot.ini-4> attach tm0 dist.tap
%SIM-INFO: TM0: Tape Image 'dist.tap' scanned as SIMH format
tboot.ini-11> attach rk0 rk0
%SIM-INFO: RK0: Creating new file: rk0
tboot.ini-12> attach rk1 rk1
%SIM-INFO: RK1: Creating new file: rk1
tboot.ini-13> attach rk2 rk2
%SIM-INFO: RK2: Creating new file: rk2
tboot.ini-14> attach rk3 rk3
%SIM-INFO: RK3: Creating new file: rk3
```

OpenSIMH may display the absolute path to `tboot.ini` rather than the shorter filename shown here.

The bootstrap reads the first record of the distribution tape into memory beginning at address `0`. Its final instruction branches to itself, leaving the processor in a loop when the tape operation is complete.

Press `CTRL+E` to halt the processor and return to the OpenSIMH command prompt:

```
Simulation stopped, PC: 100012 (BR 100012)
sim>
```

The tape has advanced by one record, and the bootstrap has entered its terminating loop. The standalone installation program now resides in memory beginning at address `0`, ready to copy the distribution onto disk.

Making a Disk From Tape

The standalone installation program is now loaded at address `0`. It provides the utilities needed to copy the distribution from tape to disk without first booting V6.

The distribution tape contains 12,100 records of 512 bytes followed by a tape mark. Within the standalone installation program, these records are addressed by tape offset. The binary, source, and documentation filesystems begin at offsets `101`, `4101`, and `8101` respectively.

The following procedure assumes the TU10 tape drive and RK05 disk are configured in `tboot.ini`. Within the simulated PDP-11, `tm0` is the tape drive and `rk0` is the first RK05 drive; on the host, their contents are supplied by the files `dist.tap` and `rk0`.

Start the standalone installation program

At the OpenSIMH prompt, begin execution at address `0`:

```
sim> g 0
=
```

The tape will rewind, and the `=` prompt indicates that the standalone installation environment is ready.

Copy the RK05 bootstrap to disk

Use `tmrk` to copy the RK05 bootstrap from tape offset `100` to disk block `0`:

```
=tmrk
disk offset
```

```
0
tape offset
100
count
1
```

The operation should complete after a brief delay while the tape advances to offset `100` and copies one record to disk block `0`.

Copy the binary filesystem to disk

Use `tmrk` again to copy 3,999 records of the binary filesystem, beginning at tape offset `101, to` the disk beginning at block `1`:

```
=tmrk
disk offset
1
tape offset
101
count
3999
=
```

This operation will take noticeably longer as the tape advances through the binary filesystem.

The first use of `tmrk` copies the RK05 bootstrap program from tape offset `100` to disk block `0`. The second copies the binary filesystem from tape offsets `101` through `4099` to disk blocks `1` through `3999`. Together, these operations create the bootable binary RK05 disk represented by the host file `rk0`.

The source and documentation filesystems, beginning at tape offsets `4101` and `8101` respectively, will be installed later, after V6 is running and `dd` is available.

Leave OpenSIMH

Press `CTRL+E` to halt the simulation and return to the OpenSIMH command prompt:

```
Simulation stopped, PC: 137274 (TSTB @#177560)
sim>
```

Then quit OpenSIMH:

```
sim> q
Goodbye
```

Booting V6

OpenSIMH provides built-in bootstrap routines for its simulated disk controllers. The preferred command for the RK05 is `sim> boot rk0`

We will use the RK05 configuration assumed by Lions. Create boot.ini:

```
$ cat > boot.ini << "EOF"
set cpu 11/40
set cpu idle
set tto 7b

set tm0 locked
```

```
attach tm0 dist.tap

set rk0 en noautosize
set rk1 en noautosize
set rk2 en noautosize
set rk3 en noautosize

attach rk0 rk0
attach rk1 rk1
attach rk2 rk2
attach rk3 rk3

attach ptr ptr.txt
attach ptp ptp.txt
attach lpt lpt.txt

dep system sr 173030
boot rk0
EOF
```

This establishes the OpenSIMH hardware configuration used for the remainder of the laboratory: a PDP-11/40, four RK05 drives, a PC11 paper-tape reader and punch, an LP11 line printer, and the console terminal. The TU10 magnetic-tape drive is retained so that the remaining distribution filesystems can be installed.

`set tto 7b` sets terminal output to seven bits, as expected by V6.

The first three RK05 drives will hold the binary, source, and documentation filesystems. The fourth is left available for additional filesystems or later experimentation.

The PC11 reader and punch are attached to the host files `ptr.txt` and `ptp.txt`. Output sent to the LP11 is written to `lpt.txt`.

The command `dep system sr 173030` initializes the PDP-11 switch register. The value `173030` is the conventional switch-register setting used when booting the distributed V6 system.

The final command `boot rk0` invokes OpenSIMH's RK05 bootstrap, which reads disk block `0` into memory and executes it.

Start OpenSIMH:

```
$ pdp11 boot.ini
```

Output will resemble:

```
PDP-11 simulator Open SIMH V4.1-0 Current        git commit id: a1f57fa3
Disabling XQ
boot.ini-6> attach tm0 dist.tap
%SIM-INFO: TM0: Tape Image 'dist.tap' scanned as SIMH format
boot.ini-18> attach ptr ptr.txt
%SIM-INFO: PTR: creating new file
boot.ini-19> attach ptp ptp.txt
%SIM-INFO: PTP: creating new file
boot.ini-20> attach lpt lpt.txt
%SIM-INFO: LPT: creating new file
@
```

Now follow the indicated dialogue, where `@` and `#` are prompts:

```
@rkunix
mem = 1035
```

```
RESTRICTED RIGHTS

Use, duplication or disclosure is subject to
restrictions stated in Contract with Western
Electric Company, Inc.
#
```

The `mem` message gives the memory available to user programs. The `#` is the V6 shell prompt and indicates that the system is running with superuser privileges.

Before proceeding, disable uppercase terminal mapping:

```
# STTY -LCASE
```

V6 is now running. The supplied `rkunix` kernel is sufficient to boot the root disk, but it is not yet configured for all of the hardware attached by OpenSIMH. In the next section the system will be reconfigured and rebuilt for the laboratory machine.

Reconfiguration

The V6 system now running is the supplied `rkunix` configuration. It is sufficient to boot the root disk, but it does not correspond exactly to the machine prepared in OpenSIMH. We will rebuild the kernel for the PDP-11/40 configuration used in this laboratory: RK05 disks, a TU10 magnetic-tape drive, a PC11 paper-tape reader and punch, an LP11 line printer, and the KL11 console terminal assumed by Lions.

Display the file `/usr/sys/run`:

```
# cat /usr/sys/run
chdir ken
cc -c -O *.c
ar r ../lib1
rm *.o
...
```

This Shell file contains the commands used to recompile the system source, install the resulting objects in the system libraries, and build kernels for the RK, RP, and HP disk configurations. We will use it as a guide rather than execute it unchanged.

Change to the configuration directory, compile `mkconf`, and rename the resulting program:

```
# chdir /usr/sys/conf
# cc mkconf.c
# mv a.out mkconf
```

The `mkconf` program generates a kernel configuration from a list of device classes. Run it and specify the devices configured in `boot.ini`:

```
# ./mkconf
rk
tm
pc
lp
done
```

The RK controller supports the four RK05 disks attached as `RK0` through `RK3`. The TU10 remains attached as `TM0` for installation work. The PC11 supplies the paper-tape reader and punch, and the LP11 supplies the line printer. The KL11 console is included automatically and is therefore not entered separately.

`mkconf` creates two files:

`l.s`, containing the interrupt and trap vectors;
`c.c`, containing the block and character device-switch tables and the root and swap-device definitions.

Examine them:

```
# cat l.s
/ low core

br4 = 200
br5 = 240
...

# cat c.c
/*
 */

int     (*bdevsw[])()
...
```

The generated interrupt vectors should agree with the standard OpenSIMH configuration. No changes are required for the machine used here.

The PDP-11/40 machine-dependent assembly code is contained in `m40.s`. Lions discusses this file together with the trap and interrupt mechanisms represented in `l.s`. The alternative `m45.s` is used for PDP-11/45 and PDP-11/70 systems and is not required here.

The kernel is constructed from:

```
m40.s       PDP-11/40 machine-language assist
l.s         trap and interrupt vectors
c.c         device configuration and switch tables
../lib1     system object library
../lib2     device-driver object library
```

Assemble the machine-language assist and trap-vector files, compile the configuration table, and link the kernel:

```
# as m40.s
# mv a.out m40.o
# cc -c c.c
# as l.s
# ld -x a.out m40.o c.o ../lib1 ../lib2
# mv a.out /unix
```

The assembler places its result in `a.out`. After assembling `m40.s`, that file is therefore renamed `m40.o`. Assembling `l.s` then places its object code in a new `a.out`, which is supplied as the first input to `ld`.

Confirm that the new kernel has been installed:

```
# ls -l /unix
-rwxrwxrwx  1 root    29516 Oct 10 12:34 /unix
```

The exact size will vary with a different set of selected drivers.

Before booting the new kernel, force delayed filesystem output to disk:

```
# sync
```

At the next bootstrap prompt, it may be loaded with `@unix`
The supplied `rkunix` should be retained until the new kernel has been shown to boot successfully.

Special Files

The kernel now contains the selected device drivers, but programs access those drivers through special files in `/dev`. Initially, the directory contains:

```
# ls /dev
kmem
mem
null
tty8
```

`/dev/tty8` is the special file for the KL11 console. `/dev/mem` provides access to physical memory, `/dev/kmem` provides access to kernel memory, and `/dev/null` discards all data written to it.

The generated `c.c` defines the block- and character-device switch tables used by the kernel to dispatch device operations. `bdevsw` is an array whose entries are arranged in groups of four for each block-device class: open, close, strategy, and the address of the device's request queue. `cdevsw` is an array whose entries are arranged in groups of five for each character-device class: open, close, read, write, and terminal control. In both arrays, each group corresponds to one device class, and the group's zero-based position is that device's major number.

An unsupported device class, or an individual operation not supplied by a configured driver, is represented by `nodev`, a stub routine that reports the device as unavailable. `nulldev`, by contrast, performs no operation.

The block-device switch table is:

```
int     (*bdevsw[])()
{
        &nulldev,       &nulldev,       &rkstrategy,    &rktab, /* rk */
        &nodev,         &nodev,         &nodev,         0,      /* rp */
        &nodev,         &nodev,         &nodev,         0,      /* rf */
        &tmopen,        &tmclose,       &tmstrategy,    &tmtab, /* tm */
        &nodev,         &nodev,         &nodev,         0,      /* tc */
        &nodev,         &nodev,         &nodev,         0,      /* hs */
        &nodev,         &nodev,         &nodev,         0,      /* hp */
        &nodev,         &nodev,         &nodev,         0,      /* ht */
        0
};
```

Each four-element group corresponds to one block-device class. Counting the groups from zero gives the device's block major number. Thus the RK05 occupies block major number `0` and the TU10 occupies block major number `3`. The remaining entries retain their historical positions even though their operations are represented by `nodev`.

The character-device switch table is:

```
int     (*cdevsw[])()
{
        &klopen,    &klclose,   &klread,    &klwrite,   &klsgtty,    /* console */
        &pcopen,    &pcclose,   &pcread,    &pcwrite,   &nodev,      /* pc */
        &lpopen,    &lpclose,   &nodev,     &lpwrite,   &nodev,      /* lp */
        &nodev,     &nodev,     &nodev,     &nodev,     &nodev,      /* dc */
        &nodev,     &nodev,     &nodev,     &nodev,     &nodev,      /* dh */
        &nodev,     &nodev,     &nodev,     &nodev,     &nodev,      /* dp */
        &nodev,     &nodev,     &nodev,     &nodev,     &nodev,      /* dj */
        &nodev,     &nodev,     &nodev,     &nodev,     &nodev,      /* dn */
```

```
        &nulldev,  &nulldev,  &mmread,   &mmwrite,  &nodev,     /* mem */
        &nulldev,  &nulldev,  &rkread,   &rkwrite,  &nodev,     /* rk */
        &nodev,    &nodev,    &nodev,    &nodev,    &nodev,     /* rf */
        &nodev,    &nodev,    &nodev,    &nodev,    &nodev,     /* rp */
        &tmopen,   &tmclose,  &tmread,   &tmwrite,  &nodev,     /* tm */
        &nodev,    &nodev,    &nodev,    &nodev,    &nodev,     /* hs */
        &nodev,    &nodev,    &nodev,    &nodev,    &nodev,     /* hp */
        &nodev,    &nodev,    &nodev,    &nodev,    &nodev,     /* ht */
        0
};
```

Each five-element group corresponds to one character-device class. Again, counting the groups from zero gives the device's character major number. Thus mem occupies character major number `8`, the raw RK05 interface occupies character major number `9`, and the raw TU10 interface occupies character major number `12`.

These switch tables define the major numbers recognized by the kernel. mknod combines these major numbers with device-specific minor numbers to create special files through which programs access individual devices.

- RK05 disk driver: Major number 0 identifies the RK05 driver. Minor numbers 0 through 3 select drives rk0 through rk3.
- TU10 tape driver: Major number 3 identifies the TU10 driver. Minor number 0 selects the first tape drive.
- PC11 paper tape reader/punch: Character major number 1 identifies the PC11 paper-tape driver. Minor number 0 selects the installed reader/punch.
- LP11 line printer: Character major number 2 identifies the LP11 driver. Minor number 0 selects the installed printer.

Create block special files for the four RK05 disks and the TU10 tape drive:

```
# /etc/mknod /dev/rk0 b 0 0
# /etc/mknod /dev/rk1 b 0 1
# /etc/mknod /dev/rk2 b 0 2
# /etc/mknod /dev/rk3 b 0 3
# /etc/mknod /dev/mt0 b 3 0
```

Create the corresponding raw character interfaces:

```
# /etc/mknod /dev/rrk0 c 9 0
# /etc/mknod /dev/rrk1 c 9 1
# /etc/mknod /dev/rrk2 c 9 2
# /etc/mknod /dev/rrk3 c 9 3
# /etc/mknod /dev/rmt0 c 12 0
```

The raw interfaces permit direct, unbuffered transfers between a process and the device, bypassing the block buffer cache.

Create special files for the PC11 and LP11:

```
# /etc/mknod /dev/ppt c 1 0
# /etc/mknod /dev/lp0 c 2 0
```

The PC11 is attached by OpenSIMH to `ptr.txt` and `ptp.txt`. Output sent to the LP11 is written to `lpt.txt`.

No additional terminal special files are required. The laboratory has only the KL11 console, already represented by `/dev/tty8`.

Set appropriate access modes:

```
# chmod 640 /dev/*rk*
```

```
# chmod 640 /dev/*mt*
# chmod 640 /dev/*ppt*
# chmod 640 /dev/*lp*
```

Examine the resulting directory:

```
# ls -l /dev
total 0
crw-rw-r--  1 bin     8,  1 May 13 20:01 kmem
crw-r-----  1 root    2,  0 Oct 10 12:39 lp0
crw-rw-r--  1 bin     8,  0 May 13 20:01 mem
brw-r-----  1 root    3,  0 Oct 10 12:38 mt0
crw-rw-rw-  1 bin     8,  2 May 13 20:01 null
crw-r-----  1 root    1,  0 Oct 10 12:38 ppt
brw-r-----  1 root    0,  0 Oct 10 12:38 rk0
brw-r-----  1 root    0,  1 Oct 10 12:38 rk1
brw-r-----  1 root    0,  2 Oct 10 12:38 rk2
brw-r-----  1 root    0,  3 Oct 10 12:38 rk3
crw-r-----  1 root   12,  0 Oct 10 12:38 rmt0
crw-r-----  1 root    9,  0 Oct 10 12:38 rrk0
crw-r-----  1 root    9,  1 Oct 10 12:38 rrk1
crw-r-----  1 root    9,  2 Oct 10 12:38 rrk2
crw-r-----  1 root    9,  3 Oct 10 12:38 rrk3
crw--w--w-  1 root    0,  0 Oct 10 12:39 tty8
#
```

The newly created entries show the expected major and minor device numbers in the fifth and sixth columns.

The simulated hardware, kernel configuration, and special files now agree: a PDP-11/40 with four RK05 drives, a KL11 console, a PC11 paper-tape reader/punch, an LP11 line printer, and a TU10 magnetic-tape drive used for software distribution.

Installing the Remaining Filesystems

The root disk contains only the root filesystem and operating system. The source code and documentation remain on the distribution tape. Copy these onto the second and third RK05 disks.

Use `dd` with the raw character interfaces to copy the source and documentation filesystem images directly from the tape:

```
# dd if=/dev/rmt0 of=/dev/rrk1 count=4000 skip=4100
4000+0 records in
4000+0 records out

# dd if=/dev/rmt0 of=/dev/rrk2 count=4000 skip=8100
4000+0 records in
4000+0 records out
```

Create a mount point for the documentation filesystem:

```
# mkdir /usr/doc
```

Mount the source and documentation filesystems:

```
# /etc/mount /dev/rk1 /usr/source
# /etc/mount /dev/rk2 /usr/doc
```

Verify that both have been mounted successfully:

```
# ls /usr/source
as
c
cref
fort
iolib
m6
mdec
rat
run
s1
s2
s3
s4
s5
s7
salloc
sno
tmg
yacc

# ls /usr/doc
as
bc
beg
c
ctut
ed
hel
iolib
iosys
man
rat
secur
start
unix
yacc
```

Arrange for the filesystems to be mounted automatically during startup by appending the following lines to `/etc/rc`. After entering the second line, press `CTRL+D` at the beginning of a new line to signal end-of-file and return to the shell:

```
# cat >> /etc/rc
/etc/mount /dev/rk1 /usr/source
/etc/mount /dev/rk2 /usr/doc
^D
```

Updating `df`

The distributed `df` command is configured for the original installation's default disks. Modify its device list to report the four RK05 drives used by the laboratory.

Change to the source directory:

```
# chdir /usr/source/s1
```

Edit `df.c` using `ed`.

The standard editor supplied with V6 is `ed`, a line-oriented editor. Commands operate on numbered lines rather than on a visual display. A leading line number specifies the line on which the command acts, so `5p` prints line 5 and `5d` deletes it. A line number by itself makes that line current, `i` enters insert mode before the current line, a single period (`.`) ends insertion, `w` writes the file, and `q` quits. The line numbers in the following session assume the unmodified distribution source.

The following transcript shows only the commands entered; `ed` responses, including the initial and written byte counts and the output of commands such as `5p` are omitted.

```
# ed df.c
5p
5d
4i
"/dev/rk0",
"/dev/rk1",
.
7i
"/dev/rk3",
.
```

To examine the edited array, print lines 1 through 9:

```
1,9p
char    *dargv[]
{
        0,
"/dev/rk0",
"/dev/rk1",
        "/dev/rk2",
"/dev/rk3",
        0
};
```

The inserted lines appear flush left while the original lines retain their tabs: `ed` inserts text exactly as typed. The difference is cosmetic and does not affect compilation.

Write the modified file and quit:

```
w
q
```

Compile and test the revised program:

```
# cc df.c
# ./a.out
/dev/rk0 995
/dev/rk1 935
/dev/rk2 1691
/dev/rk3 bad free count
0
```

The first three entries show the number of free blocks on the root, source, and documentation filesystems. The fourth RK05 has not yet been initialized as a filesystem, so its superblock does not contain a valid free-block list. `df` therefore reports `bad free count`; the following `0` is the number of free blocks counted before the check failed.

Install it:

```
# cp a.out /bin/df
```

Verify the installed command:

```
# df
/dev/rk0 993
/dev/rk1 935
```

```
/dev/rk2 1691
/dev/rk3 bad free count
0
```

The root filesystem now reports two fewer free blocks because compiling and installing the revised command consumed additional disk space.

Verifying the Filesystems

Verify the integrity of the three installed filesystems. The fourth RK05 (`rk3`) is intentionally omitted because it has not yet been initialized with a filesystem. `icheck` and `dcheck` operate on the raw character interfaces (`rrk0`–`rrk2`) rather than the block devices (`rk0`–`rk2`) so that they examine the disk directly, bypassing the block buffer cache.

```
# icheck /dev/rrk0
/dev/rrk0:
spcl      16
files    294
large     96
direc     25
indir     96
used    2920
free     993
# dcheck /dev/rrk0
/dev/rrk0:
# icheck /dev/rrk1
/dev/rrk1:
spcl       0
files    596
large     98
direc     34
indir     98
used    2978
free     935
# dcheck /dev/rrk1
/dev/rrk1:
# icheck /dev/rrk2
/dev/rrk2:
spcl       0
files    337
large     69
direc     25
indir     69
used    2222
free    1691
# dcheck /dev/rrk2
/dev/rrk2:
#
```

`icheck` reports filesystem statistics and should report no duplicate blocks, bad blocks, or free-list errors. Its free-block counts should agree with those reported by `df`. `dcheck` should print only the device name. Any additional output indicates a directory or link-count inconsistency.

Flush all pending filesystem updates:

```
# sync
# sync
# sync
```

Return to the OpenSIMH monitor and exit:

```
CTRL+E
sim> q
Goodbye
```

Operating the Completed Laboratory

The laboratory is now assembled: the configured kernel, source tree, documentation, and simulated devices are in place. Normal use begins by starting OpenSIMH, booting the installed kernel, and entering V6 as root.

During installation, `boot.ini` was configured for single-user startup and attached the distribution tape. Restore normal operation by editing the file in the editor of your choice (`vi`), commenting out these installation settings: add a semicolon (`;`) at the beginning of each line:

```
...
;set tm0 locked
;attach tm0 dist.tap
...
;dep system sr 173030
...
```

Save the file, then start the simulator.

```
$ pdp11 boot.ini
@
```

Load the V6 kernel:

```
@unix
```

When the login prompt appears, log in as root. The distributed system has no password for the root account. No additional user accounts or civil-time configuration are required for this laboratory. The kernel's clock, user-identification, protection, and accounting mechanisms are present, but their site-administration procedures lie outside the purpose of this guide.

```
login: root
#
```

Disable carriage return and line feed delays for more responsive terminal output:

```
# stty cr0 nl0
```

The laboratory is now fully installed. The root disk, source tree, and documentation are available, providing the PDP-11/40 environment assumed by John Lions in *A Commentary on the UNIX Operating System*.

Return to the OpenSIMH monitor and exit:

```
CTRL+E
sim> q
Goodbye
```

Backing Up the Completed System

At this point the installation is complete. Before experimenting with the system, create a backup of the working disk images. Restoring these files is much faster than repeating the installation procedure.

Create a directory to hold the baseline system:

```
$ mkdir baseline-v6
$ cp boot.ini baseline-v6/
$ cp rk? baseline-v6/
```

Archive the directory:

```
$ tar cvzf baseline-v6.tar.gz baseline-v6
```

You may then remove the temporary directory:

```
$ rm -rf baseline-v6
```

To restore the baseline system later:

```
$ tar xvzf baseline-v6.tar.gz
$ cd baseline-v6
$ pdp11 boot.ini
```

OpenSIMH may create new paper-tape and line-printer output files when it starts:

```
%SIM-INFO: PTR: creating new file
%SIM-INFO: PTP: creating new file
%SIM-INFO: LPT: creating new file
```

The restored system should then boot normally and reproduce the completed laboratory:

```
@unix

login: root
# ls /usr/source
…
# stty cr0 nl0
# ls /usr/doc
...
# df
/dev/rk0 994
/dev/rk1 935
/dev/rk2 1691
/dev/rk3 bad free count
0
```

The root filesystem reports one more free block than at backup time, a result of startup housekeeping performed by `/etc/rc`.

Return to the OpenSIMH monitor and exit:

```
CTRL+E
sim> q
Goodbye
```

If you followed this procedure exactly, the extracted `baseline-v6` directory is now inside the working `v6` directory. After verifying the restored system, it is safe to remove the extracted copy and continue working with the original installation:

```
$ cd ~/retro-workarea/v6
$ rm -rf baseline-v6
```

Congratulations. You now have a complete and restorable V6 laboratory for reading Lions' *A Commentary on the UNIX Operating System*.

Concluding Remarks

You now have this manual, Lions' commentary, the source listing, the system manuals, the configured kernel, and the running machine available together. The PDP-11/40 selection determines the machine-dependent code in `m40.s`; the attached devices determine the configuration represented in `c.c` and `l.s`; and the resulting `/unix` is the kernel executing while you read Lions' account of it. From this point you may study the kernel, rebuild individual commands, experiment with the source, or restore the baseline system whenever you wish to return to a known starting point. The files of the numbered listing are those installed under `/usr/sys:` sheet 15's `main.c` is `/usr/sys/ken/main.c`.

This laboratory intentionally omits user administration, mail, networking, routine peripheral use, and broader system customization. Those belong to operating a V6 installation; this guide is concerned only with constructing the environment required to read, modify, and rebuild the system discussed by Lions.

Good hunting.

Will Senn
Godley, Texas
July 2026

References

Lions, J. (1977). *Commentary on the UNIX Operating System, with Source Code*. Peer-to-Peer Communications.

Thompson, K., & Ritchie, D. M. (1975). *Setting Up UNIX—Sixth Edition*. Bell Laboratories.

Ritchie, D. M., & Thompson, K. (1974). The UNIX time-sharing system. *Communications of the ACM*, 17(7), 365–375.

Senn, W. (2022). *UNIX Operating System Source Code Level Six* (Rev. 1.4; adapted from J. Lions' 1977 source booklet). https://decuser.github.io/assets/pdf/unix/lions-v6-source-code-letter-v1.4.pdf

UNIX Programmer's Manual: Sixth Edition. (1975). Bell Laboratories.

Software and Archives

- OpenSIMH PDP-11 Simulator – https://opensimh.org/
- The Unix Heritage Society (TUHS) Archives – https://www.tuhs.org/

Laboratory Materials

- V6 distribution tape (`v6.tape`)
- Wolfgang Helbig's V6 fixes (`v6enb`)
- OpenSIMH 4.x
- John Lions' Commentary
- *UNIX Operating System Source Code Level Six*, updated and expanded edition (rev. 1.4)