

building a tiny Linux system

- Linux most widely-used operating system
 - 24 official supported hardware platforms
 - also many variants, including Android, FireOS & ChromeOS
- lets look at compiling and running a tiny linux system from source
- we'll introduce some useful tools (make, qemu, mkfs, ...)
- and important concepts (kernel, filesystems, emulators, ...)

building a tiny Linux system - tools needed

- all the software we'll be using is open source and already installed on CSE servers
- if you want to try out these examples on your own machine, on Debian install these packages:
 - the package names may be different on other systems (Ubuntu, WSL, ...)

```
$ sudo apt update
$ sudo apt install bc bison build-essential curl flex fuse2fs libelf-dev
↪ libssl-dev qemu-system-x86
```

- build-essential is a meta-package which installs many other packages - for tools often used in building software
 - e.g. gcc, make

- qemu is an open-source emulator
 - can emulate instruction sets including x86, MIPS, 32-bit ARMv7, ARMv8, PowerPC, RISC-V, ...
 - uses clever techniques to make emulation fast but still has significant overhead
- can emulate execution of a single (linux) program - user mode emulation
 - e.g. can emulate MIPS code on an x86 box
 - operating system is still the host operating system
 - e.g. can't run windows executables on linux
- can emulate an entire (virtual) machine - system emulation
 - including CPU, RAM, display, disks, network ...
 - can boot another operating system, e.g. can emulate Windows on a Linux box
 - often more convenient for development than real hardware
 - we'll build some tiny linux systems and run them with qemu
- qemu also provides virtualization - virtual machine with (almost) no overhead
 - virtualization needs hardware support & OS support
 - virtual machine must be same architecture (e.g. x86) as host
 - other virtualization implementations: virtualbox, vmware, Microsoft Virtual PC

running ARM code on x86 with qemu - step 1 cross compiler

- a cross compiler is a compiler which generates machine code for a different platform
 - e.g. a cross compiler might run on x86 (Intel) but generate code for MIPS
 - allows you to build software on a powerful general purpose machine (e.g. laptop)
 - and deploy to special-purpose machine e.g. a router

```
$ cat >hello.c <<eof
#include <stdio.h>
int main(void) {
    printf("hello from ARM code\n");
}
eof
# cross-compile C to ARM machine code on our x86 box
$ aarch64-linux-gnu-gcc-12 -static hello.c -o hello
$ file hello
hello: ELF 64-bit LSB executable, ARM aarch64
# we can't run hello directly, it's not x86 machine code
$ ./hello
-bash: ./hello: cannot execute binary file: Exec format error
```

qemu user mode example - simple test program

```
// print the nth-prime
#include <stdio.h>
#include <stdlib.h>
int is_prime(int number) {
    for (int possible_factor = 2; possible_factor < number; possible_factor++)
        if (number % possible_factor == 0) {
            return 0;
        }
    return 1;
}
int main(int argc, char *argv[]) {
    int n = atoi(argv[1]);
    int n_primes = 0;
    for (int number = 2; n_primes < n; number++) {
        n_primes += is_prime(number);
        if (n_primes == n) {
            printf("%d\n", number);
            return 0;
        }
    }
}
```

```
# a simple program to print the nth prime
$ gcc -O3 nth_prime.c -o nth_prime
$ file nth_prime
nth_prime: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
↳ dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, ...
$ time ./nth_prime 10000
104729
real    0m0.643s
```

qemu user mode example - a complication - dynamic linking

- **nth_prime** is dynamically linked
- **nth_prime** does not contain the code for the C library (e.g. printf)
- when run **nth_prime** is linked with C library code by the operating system
 - this allows operating system to share C library code in RAM between many programs
 - can upgrade (bug/security) fix library without recompiling programs which use library
 - saves disk space
- **ldd** is useful program - shows how linking will occur

```
$ file nth_prime
nth_prime: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically 1
$ ldd nth_prime
linux-vdso.so.1 (0x00007ffd09966000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f3d61a5f000)
/lib64/ld-linux-x86-64.so.2 (0x00007f3d61c6a000)
# nth_prime only links in two libraries, other programs may link many
$ ldd /bin/obs |wc -l
263
```

qemu user mode example - static compilation

- dynamic linking offers large benefits but breaks if we run the program without libraries present
 - this is problem for emulation and in some other contexts
- gcc/clang has a **-static** option to include all library code in the binary (will be larger)

```
# cross-compile C to ARM machine code instead on our x86 box
$ aarch64-linux-gnu-gcc-12 -static -O3 nth_prime.c -o nth_prime.arm
# nth_prime.arm contains ARM machine code
$ file nth_prime.arm
nth_prime.arm: ELF 64-bit LSB executable, ARM aarch64, ...
# we are on x86 so we can't run nth_prime.arm directly but qemu-arm64 can emulate
$ ./nth_prime.arm 10000
-bash: ./nth_prime.arm: cannot execute binary file: Exec format error
$ time qemu-arm64 ./nth_prime.arm 10000
104729
real    0m1.251s
```

- note emulation is 50% slower than our previous native execution

- **curl** lets you interact from command line with web and other http/https servers
 - curl has many other options & features, **wget** provides similar functionality
 - Andrew likes curl better than wget - but little difference for most tasks
 - busybox provides a cut-down wget

```
# fetch a file
$ curl -O https://cgi.cse.unsw.edu.au/~cs2041/examples.zip
# get other info
$ curl -I https://unsw.edu.au
HTTP/1.1 200 OK
Server: Apache/2.4.34 (Red Hat) OpenSSL/1.0.1e-fips PHP/5.6.25
X-Powered-By: PHP/5.6.25
# send data to web server
$ curl -X PUT -H 'content-type: txt/plain' https://google.com
# send cookies to web server
$ curl -b 'id=42' https://google.com
....
```

virtual machines

- qemu can simulate an entire computer - a virtual machine
- use for virtual machines include:
 - running a different OS, e.g running windows VM on linux
 - running potentially malicious software safely in isolation
- can hosting multiple VM on one physical machine
- possible to move running VMs between physical machines

virtual machine example - running ARM OpenWrt on x86

- OpenWRT is popular linux-based operating system for routers and other embedded devices

```
# download openwrt kernel with built-in ramdisk
$ curl \
  -O "https://downloads.openwrt.org/releases/22.03.6/targets/armvirt/32/openwrt-22.03.6-armvirt-32-zImage-initramfs-kernel.bin"
$ ls -l openwrt-kernel
-rw-r--r-- 1 andrewt andrewt 3156752 Jul 27 11:02 openwrt-kernel
$ file openwrt-kernel
openwrt-kernel: Linux kernel ARM boot executable zImage (little-endian)
$ qemu-system-arm -nographic -M virt -kernel openwrt-kernel
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 4.14.63
               (buildbot@builds-03.infra.lede-project.org)
[    0.000000] CPU: ARMv7 Processor [414fc0f0] revision 0 (ARMv7), cr=10c5387d
...

      _ _ _ _ _ 
     |         | .----- .----- .----- .----- | | | | | |
     |   -   | |         |         |         |         |
     |_____| | |         |         |         |         |
           | | W I R E L E S S   F R E E D O M
           |_|

root@OpenWrt:/# uname -a
Linux OpenWrt 5.10.201 #0 SMP Sat Aug 04 18:18:57 1997 armv7l GNU/Linux
root@OpenWrt:/# poweroff
root@OpenWrt:/# [ 130.887967] reboot: Power down
```

- tar - widely used tool to create or extract archive files (in tar format)
- An archive file captures metadata and contents of multiple files and directories as a single file
- often incorporates compression
- example file formats include:
 - **tar** - general purpose, Unix-like systems
 - **zip** - general purpose, many platforms, includes compression
 - **deb** - software packages, Debian-family Linux distributions
 - **ar** - used for libraries of relocatable binaries (.o files)
 - **shar** - usually software packages, Unix-like systems
 - self-extracting shell-script!
 - **cpio** - mostly obsolete, general purpose, Unix-like systems, some remaining niche uses (Linux kernel ramdisks)

tar - example

```
# capture files in assignment directory tree
# -c create an archive
# -f archive filename
# -z compress with gzip
$ tar -zcf assignment.tar.gz assignment
$ cp assignment.tar.gz /tmp
$ cd /tmp
# extract files from archive
# -x create an archive
# -v (verbose) - print filenames when extracting
# -f archive filename
$ tar -xvf assignment.tar.gz
...
```

tar - compression options

tar uses compression formats available as external programs and compatible with these programs.

- xz/unxz (-J option to tar)
 - algorithm Lempel–Ziv–Markov-chain
 - good level of compression
 - slow to compress but uncompression fast
- bzip2/bunzip2 (-j option to tar)
 - algorithm: Burrows–Wheeler algorithm
 - faster to compress than xz but compression level not as good
- gzip/gunzip (-z option to tar)
 - algorithm: DEFLATE
 - compression level not as good as bzip2
 - very widely available on Unix-like machines
 - used for HTTP compression

- Linux source is over 1.3 gigabytes
 - 80,000 separate files, 30 million lines of C source
 - compressed with xz only 136 Mb
 - tar recognises the xz compression automatically - don't need to use unxz

```
$ curl -sO https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.9.9.tar.xz
$ ls -l linux-6.9.9.tar.xz
-rw-r--r-- 1 andrewt andrewt 144131116 Jul 25 13:06 linux-6.9.9.tar.xz
$ tar xf linux-6.9.9.tar.xz
$ cd linux-6.9.9
$ find . -type f | wc -l
84283
$ find . -name '*.ch'|xargs cat|wc -l
33730066
# can download and extract with single pipeline, but must specify compression
$ curl https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.9.9.tar.xz|tar
↵ -Jxf -
```

configuration

- portable software often has one or more initial configuration steps
 - user specifies how they want system build
 - scripts set up build for this platform
- Linux uses make + custom scripts for configuration

```
$ find . -name Makefile|wc -l
2945
$ find . -name '*.sh'|wc -l
868
```

- The configuration step for Linux creates a custom file named **.config**

```
$ make defconfig
...
# configuration written to .config
$ wc .config
5273 16698 140452 .config
```

a smaller configuration

- make defconfig builds a .config including all Linux code (drivers) for this platform
- This includes many many feature we don't need for our experiments with a tiny linux system.
- Lets download a .config which only enables the sort features we need for our experiments
- This will make our build faster and our kernel (Linux compiled binary) smaller

```
$ curl -sL
↵ "https://raw.githubusercontent.com/buildroot/buildroot/master/board/qemu/x86_
↵ >.config
$ make olddefconfig
# configuration written to .config
```

compiling Linux

- we can compile the entire Linux kernel just by running `make` - but even on a fast machine takes 8 minutes!
- **make -j** where possible runs builds in parallel
 - possible because of the specification of dependencies in Makefiles
- **make -j8** 7x faster on a machine with 8+ cores

```
$ time make
...
Kernel: arch/x86/boot/bzImage is ready  (#1)
...
real    8m19.669s
$ make clean
$ time make -j8
...
Kernel: arch/x86/boot/bzImage is ready  (#1)
...
real    1m11.791s
```

compiling Linux - success

```
$ cd ..
$ cp linux-6.9.9/arch/x86/boot/bzImage mykernel
$ ls -l mykernel
-rw-r--r-- 1 andrewt andrewt 6075392 Apr  9 09:15 mykernel
```

- we have an operating system for our tiny system

side note: make - incremental builds

- **make** only rebuilds where files have changed
 - typically we work on only a few files, so make can save a lot of build time

```
# make a (silly) change to 1 of the kernel's 30,000 sources files
$ echo '// I miss VGA' >> arch/x86/boot/video-vga.c
$ time make -j8
...
CC      arch/x86/boot/video-vga.o
...
real    0m12.709s
```

make only rebuilt the part of the of the kernel that depended on `video-vga.c`

- makefiles also provide a simple way to specify convenient actions
 - a **make clean** rule is common - removes intermediate compiled files
 - handy to save disk space with you are finished building
 - building Linux kernel create 400mb of intermediate compiled files

```
$ du -s .
1914036 .
$ make clean
...
$ du -s .
1583284 .
```

running our kernel in a Virtual Machine with QEMU

```
# -append passes options to the kernel
# -display none -serial stdio runs the virtual machine within our terminal
$ qemu-system-x86_64 -kernel mykernel -append 'console=ttyS0' -display none
↳ -serial stdio
Linux version 6.4.8 (andrewt@localhost) (gcc (Debian 12.3.0-5) 12.3.0, ...
Command line: console=ttyS0
x86/fpu: x87 FPU will use FXSAVE
signal: max sigframe size: 1040
BIOS-provided physical RAM map:
...
VFS: Cannot open root device "(null)" or unknown-block(0,0): error -6
Please append a correct "root=" boot option; here are the available
↳ partitions:
---[ end Kernel panic - not syncing: VFS: Unable to mount root fs on
↳ unknown-block(0,0) ]---
```

Our kernel boots but we need to give it a filesystem

File Systems

- a filesystem is a method used to organize storage of files & directories on a storage device - a storage device (e.g. SSD) typically provides an array of fixed-size blocks
 - 512 KB is a typical blocks size
- a filesystem must track for each file where it is stored
 - in other words which blocks are used for the file
- a filesystem must also track free space - which blocks are unused

- **ext4** - mostly widely used general-purpose Linux filesystem
- **ext2/ext3** - ext4 predecessors with less features but still often used
- **brtfs** - copy-on-write filesystem with interesting features
- **zfs** - filesystem with interesting features including can span disks
- **ntfs** default Windows filesystem - can be accessed from Linux
- **vfat** - older Windows filesystem
 - widely used for removable devices such as SD cards and USB keys
- **nfs** - network filesystem used to provide remote access to files
- **sshfs** - remote filesystem layered on ssh
 - you can use to access your CSE files at home

Creating A Virtual Disk

- we'll create a virtual ext2 filesystem
- ext2 and many other file systems store data in an array of blocks
 - blocks often 4096 bytes
- normally blocks provided by physical disk via a special file, e.g

```
$ ls -l /dev/sda
brw-rw---- 1 root disk 8, 0 Jun 18 18:17 /dev/sda
```

- we'll use a normal file instead for our virtual (pretend) disk

```
# fallocate is an convenient and efficient way to create a 128 Mb file
$ fallocate -l 128M mydisk
$ ls -l mydisk
-rw-r--r-- 1 andrewt andrewt 20971520 Jul 27 12:06 mydisk
```

Creating A Filesystem with mkfs

- mkfs creates an initial filesystem with no files or directories
- **Beware:** mkfs overwrites (destroys) contents of disk - potential huge data loss!

```
$ mkfs.ext2 mydisk
mke2fs 1.47.0 (4-Aug-1997)
Creating filesystem with 20480 1k blocks and 5112 inodes
Filesystem UUID: a59742a0-464e-4b07-9919-6cb8d01ca8be
Superblock backups stored on blocks:
    8193

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
```

```
# -drive file=mydisk,format=raw tells qemu to use our virtual disk
# root=/dev/sda tells Linux to use the virtual disk for its root filesystem
$ qemu-system-x86_64 -drive file=mydisk,format=raw -kernel mykernel -append
  ↪ 'root=/dev/sda rw console=ttyS0' -display none -serial stdio
...
EXT4-fs (sda): mounting ext2 file system using the ext4 subsystem
Run /sbin/init as init process
Run /etc/init as init process
Run /bin/init as init process
Run /bin/sh as init process
Kernel panic - not syncing: No working init found. Try passing init= option
  ↪ to kernel.
...
```

- our kernel mounts the filesystem but can not find init
- init is the first program run by unix-like systems, it starts other programs

A Toy Init

```
$ cat >myinit.c <<eof
#include <stdio.h>
#include <linux/reboot.h>
#include <sys/reboot.h>

int main(void) {
    printf("hello from myinit\n");
    reboot(LINUX_REBOOT_CMD_POWER_OFF);
}
eof
$ gcc myinit.c -o myinit
$ ls -l myinit
-rwxr-xr-x 1 andrewt andrewt 15952 Jul 27 12:40 myinit
$ ldd myinit
    linux-vdso.so.1 (0x00007ffea6f8b000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f170da75000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f170dc80000)
# above binary is dynamically linked which is inconvenient
# because it needs libraries to run, so lets make a static binary
$ gcc myinit.c -static -o myinit
$ ls -l myinit
-rwxr-xr-x 1 andrewt andrewt 736360 Jul 25 13:13 myinit
```

mount - mount a filesystem

- mount makes the files/directories of a filesystem available below a mount point
 - mount point must be an existing directory
- umount reverses this.

```
$ mkdir mnt
$ sudo mount mydisk mnt
$ ls -l mnt
..
$ umount mnt
```

- distributions typically include a helper program to mount & unmount removable devices.
- mount needs root privileges, fine on your own computer not possible at CSE
- we'll **fuse2fs** instead - which can be run by a regular user, including at CSE

copy myinit to the filesystem on our virtual disk

```
$ ls -l mnt
total 0
$ fuse2fs -o fakeroot mydisk mnt
# our new file system contains only 1 directory: lost+found
# which is pre-created for filesystem repair
$ ls -l mnt
drwx----- 2 root    root      12288 Apr 10 17:17 lost+found
# copy myinit to the filesystem on our virtual disk
$ cp myinit mnt/myinit
$ ls -l mnt
drwx----- 2 root    root      12288 Apr 10 17:17 lost+found
-rwxr-xr-x 1 andrewt andrewt 73352 Apr 10 17:20 myinit
$ umount mnt
$ ls -l mnt
total 0
```

running our kernel with our init

```
# init=/myinit tells linux to run /myinit as the initial process
# quiet tells linux not to print diagnostic information as it boots
$ qemu-system-x86_64 -drive file=mydisk,format=raw -kernel mykernel -append
↪ 'init=/myinit root=/dev/sda rw console=ttyS0 quiet' -display none -serial
↪ stdio
hello from myinit
reboot: Power down
$
```

- we've successfully booted and shutdown our tiny system
- now we need more programs to run on our tiny system

Summary - Commands to Build a Tiny Virtual Machine

```
$ curl -s https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.9.9.tar.xz | tar
↪ -Jxf -
$ curl -sL
↪ "https://raw.githubusercontent.com/buildroot/buildroot/master/board/qemu/x86_
↪ >linux-6.9.9/.config
$ make -C linux-6.9.9 olddefconfig
$ make -C linux-6.9.9 -j32
$ cp linux-6.9.9/arch/x86/boot/bzImage mykernel
$ fallocate -l 128M mydisk
$ mkfs.ext2 mydisk
$ gcc myinit.c -static -o myinit
$ mkdir mnt
$ fuse2fs -o fakeroot mydisk mnt
$ cp myinit mnt/myinit
$ umount mnt
$ qemu-system-x86_64 -drive file=mydisk,format=raw -kernel mykernel -append
↪ 'init=/myinit root=/dev/sda rw console=ttyS0 quiet' -display none -serial
↪ stdio
```

Busybox

- busybox provides simplified versions of 270+ Unix utilities
- widely used on embedded & other systems, due to small footprint
- easy to download an already compiled busybox

```
$ curl -sO
↳ https://busybox.net/downloads/binaries/1.35.0-x86_64-linux-musl/busybox
$ ls -l busybox
-rwxr-xr-x 1 andrewt andrewt 1131168 Jul 28 13:42 busybox
$ chmod +x busybox
$ file busybox
busybox: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically
↳ linked, stripped
```

Building Busybox

- also easy to compile from source

```
$ curl -sL https://busybox.net/downloads/busybox-1.36.1.tar.bz2 | tar -jxf -
$ cd busybox-1.36.1
$ find . -type f | wc -l
5233
$ find . -name '*.ch'|xargs cat|wc -l
314808
$ make defconfig
....
$ sed -i 's/# CONFIG_STATIC is not set/CONFIG_STATIC=y/' .config
# workaround busybox bug
$ sed -i 's/CONFIG_TC=y/CONFIG_TC=n/' .config
$ make -j8
....
$ cd ..
$ cp busybox-1.36.1/busybox busybox
$ ls -l busybox
-rwxr-xr-x 1 andrewt andrewt 2302040 Jul 27 10:18 busybox
$ file busybox
busybox: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically
↳ linked ,..
```

Busybox

- busybox provides simplified versions of 400+ Unix utilities
- you can run these commands as command-line arguments to busybox

```
$ busybox echo hello from Busybox
hello from Busybox
$ busybox cp myinit.c myinit.c.backup
$ busybox wc myinit.c.backup
      8          14      162 myinit.c.backup
```

- more often busybox is run via a link, it looks at argv[0] and behaves appropriately

```
$ ln -s busybox echo
$ ln -s busybox cp
$ ln -s busybox wc
$ ./echo hello from Busybox
hello from Busybox
$ ./cp myinit.c myinit.c.backup
$ ./wc myinit.c.backup
      8      14      162 myinit.c.backup
```

Busybox - argv[0] trick - how it might work in Python

```
#!/usr/bin/python3
import os, shutil, sys
myname = os.path.basename(sys.argv[0])
if myname == "echo":
    print(*sys.argv[1:])
elif myname == "cp":
    shutil.copyfile(sys.argv[1], sys.argv[2])
elif myname == "rm":
    for pathname in sys.argv[1:]:
        os.unlink(pathname)
else:
    print(f"Unknown name: {myname}", file=sys.stderr)
```

source code for argv0.py

Programs Provided by Busybox

```
$ busybox --list | wc -l
402
$ busybox --list
[ [ acpid add-shell addgroup adduser adjtimex arch arp arping ascii ash awk
  base32 base64 basename bc beep blkdiscard blkid blockdev bootchartd brctl
  bunzip2 bzip2 cal cat chat chattr chgrp chmod chown chpasswd chpst
  chroot chrt chvt cksum clear cmp comm conspy cp cpio crc32 crond crontab
  cryptpw cttyhack cut date dc dd deallocvt delgroup deluser depmod devmem
  df dhcprelay diff dirname dmesg dnsd nsdomainname dos2unix dpkg dpkg-deb
  du dumpkmap dumpleases echo ed egrep eject env envdir envuidgid ether-wake
  expand expr factor fakeidentd fallocation false fatattr fbset fbsplash
  fdflush fdformat fdisk fgconsole fgrep find findfs flock fold free
  freeramdisk fsck fsck.minix fsfreeze fstrim fsync ftpd ftpget ftpput fuser
  getfattr getopt getty grep groups gunzip gzip halt hd hddparm head hexdump
  hexedit hostid hostname httpd hush hwclock i2cdetect i2cdump i2cget i2cset
  i2ctransfer id ifconfig ifdown ifenslave ifplugd ifup inetd init insmod
  install ionice iostat ip ipaddr ipcalc ipcrm ipcscs iplink ipneigh iproute
  iprule iptunnel kbd_mode kill killall killall5 klogd last less link
  linux32 linux64 linuxrc ln loadfont loadkmap logger login logname logread
  losetup lpd lpq lpr ls lsattr lsmod lsosf lspci lsscsi lsusb lzcat lzma
  lzop makdevs makemime man mds5sum mdev mesg microcom mim mkdir mkdosfs
  mke2fs mkfifo mkfs.ext2 mkfs.minix mkfs.vfat mknod mkpasswd mkswap mktmp
  modinfo modprobe more mount mountpoint mpstat mt mv nameif nanddump
  nandwrite nbd-client nc netstat nice nl nmeter nohup nologin nproc nsenter
  nslookup ntpd od openvt partprobe passwd paste patch pgrep pidof ping
  ping6 pipe_progress pivot_root pkill pmapp popmaildir poweroff powertop
  printenv printf ps pscan pstree pwd pwdx raidautorun rdate rdev readahead
  readlink readprofile realpath reboot reformime remove-shell renice reset
  resize resume rev rm rmdir rmdir route rpm rpm2cpio rtcwake run-init
  run-parts runlevel runsv runsvdir rx script scriptreplay sed seedrng
  sendmail seq setarch setconsole setfattr setfont setkeycodes setlogcons
  setpriv setserial setsid setuidgid sh sha1sum sha256sum sha3sum sha512sum
  showkey shred shuf slattach sleep smemcap softlimit sort split ssl_client
  start-stop-daemon stat strings stty su sulogin sum sv svc svlogd svok
  swapoff swapon switch_root sync sysctl syslogd tac tail tar taskset tcpsvd
  tee telnet telnetd test tftp tftpd time timeout top touch tr traceroute
  traceroute6 tree true truncate ts tsort tty ttysize tuncctl ubiattach
  ubidetach ubimkv ubirename ubirmvol ubirsvol ubiupdatevol udhccp udhccp6
  udhccp udpsvd uevent umount uname unexpand uniq unix2dos unlink unlzma
  unshare unxz unzip uptime users usleep uudecode uuencode vconfig vi vlock
  volname w wall watch watchdog wc wget which who whoami whois xargs xxd xz
  xzcat yes zcat zcip
```

Adding Busybox to the Filesystem on our Virtual Disk

```
$ fuse2fs -o fakeroot mydisk mnt
$ mkdir mnt/bin
# add busybox to our virtual disk
$ cp busybox mnt/bin/
# create 400+ links to busybox on virtual disk
$ for p in $(./busybox --list);do ln -s busybox mnt/bin/$p; done
$ umount mnt
# init=/bin/sh says run busybox's shell as the first process
$ qemu-system-x86_64 -drive file=mydisk,format=raw -kernel mykernel -append
↳ 'init=/bin/sh root=/dev/sda rw console=ttyS0 quiet' -display none -serial
↳ stdio
...
/ # ls -l /bin/sh
lrwxrwxrwx    1 517      517              7 Jul 28 04:57 /bin/sh -> busybox
/ # ls -l /bin/cat
lrwxrwxrwx    1 517      517              7 Jul 28 04:57 /bin/cat -> busybox
# we can now run 403 programs all of which are really busybox
/ # ls -l /bin | wc -l
403
```

Summary for Building & Installing Busybox

```
$ curl -sL https://busybox.net/downloads/busybox-1.36.1.tar.bz2|tar -jxf -
$ make -C busybox-1.36.1 defconfig
$ sed -i 's/# CONFIG_STATIC is not
↳ set/CONFIG_STATIC=y;/s/CONFIG_TC=y/CONFIG_TC=n/' busybox-1.36.1/.config
$ make -C busybox-1.36.1 -j8
$ fuse2fs -o fakeroot mydisk mnt
$ mkdir mnt/bin
$ cp -p busybox-1.36.1/busybox mnt/bin/busybox
$ for p in $(mnt/bin/busybox --list);do ln -s busybox mnt/bin/$p; done
$ umount mnt
$ qemu-system-x86_64 -drive file=mydisk,format=raw -kernel mykernel -append
↳ 'init=/bin/sh root=/dev/sda rw console=ttyS0 quiet' -display none -serial
↳ stdio
...
```

Some Special Linux Filesystems

- some commands need special “virtual” files provided by Linux, Unix philosophy *Everything is a file*

```
/ # ps
PID  USER      TIME  COMMAND
ps: can't open '/proc': No such file or directory
/ # sleep 5 &
/ # /bin/sh: can't open '/dev/null': No such file or directory
```

- **/dev** - pathnames for hardware devices.
- **/proc** - special filesystem with information about processes
- **/sys** - special filesystem with information about system

These are virtual filesystems provided by Linux.

Mounting Virtual filesystems

```
/ # mkdir /dev /proc /sys
/ # mount -t devtmpfs none /dev
/ # mount -t proc none /proc
/ # mount -t sysfs none /sys
/ # cat /proc/cpuinfo
processor      : 0
vendor_id     : AuthenticAMD
cpu family    : 15
model         : 107
model name    : QEMU Virtual CPU version 2.5+
...
/ # ls -l /dev/sda /dev/null /dev/random /dev/zero
crw-rw-rw-    1 0      0          1,    3 Jul 28 05:25 /dev/null
crw-rw-rw-    1 0      0          1,    8 Jul 28 05:25 /dev/random
brw-----    1 0      0          8,    0 Jul 28 05:25 /dev/sda
crw-rw-rw-    1 0      0          1,    5 Jul 28 05:25 /dev/zero
```

- **c** indicates character device - read/write bytes
- **b** indicates block devices - read/write blocks

some interesting “virtual” devices

- **/dev/null**
 - writes do nothing (are ignored)
 - reads return nothing
- **/dev/zero**
 - reads return bytes containing 0
- **/dev/urandom**
 - reads return random bytes

some interesting “virtual” devices

```
$ xxd /dev/null
$ xxd /dev/zero|head
00000000: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000010: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000020: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000030: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000040: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000050: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000060: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000070: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000080: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000090: 0000 0000 0000 0000 0000 0000 0000 0000 .....
$ xxd /dev/urandom|head
00000000: 16a0 e6c3 16ac d43d a258 de1f 9ba8 a24b .....=.X....K
00000010: 5661 3488 a6e4 fa2f 4204 fb80 16ad e0ec Va4..../B.....
00000020: 9eb2 e60b 7267 1250 1954 47a4 75bb 79bc ....rg.P.TG.u.y.
00000030: 410c 334a b38d 4801 550a 2c83 35a3 e30d A.3J..H.U.,.5...
00000040: d603 31b7 6062 6c54 3b6f 0e00 bd4a 765a ..1.`bLT;o...JvZ
00000050: cfe6 d39c 89ba 9a02 66cd 5044 f417 30da .....f.PD..0.
00000060: eba5 df10 223a b963 7ad4 160c ae06 7660 ....":.cz....V`
00000070: a4c4 352e 0252 615c 8e17 7b30 2e48 6fb3 ..5..Ra\..{0.Ho.
00000080: f9d5 d4f7 2913 73a9 3042 07de dde5 3537 ....).s.0B....57
00000090: 686c fba5 f41a b66a 7b68 2d48 3077 c672 hL.....j{h-H0w.r
```

- *process* is an instance of an executing program
- on Unix-like systems each process had a unique number (pid)
 - pid's are smallish non-negative integer
- **ps** ... show process information
- **kill** ... send a signal to a process
 - typically used to terminate a process
 - signal 9 always terminates process

```
# using kill to logout
$ ps
  PID TTY          TIME CMD
 344024 pts/2    00:00:00 bash
 346137 pts/2    00:00:00 ps
$ kill -9 344024
```

Tools for Managing Processes

- **pgrep** ... print PIDs of processes matching selection criteria
 - **pkill** send a signal to processes matching selection criteria
- **killall** ... also send a signal to a process with particular names
 - less powerful but more widely available than **pkill**
- **top** ... real-time monitoring of running process
 - or more easy to use **htop**

```
# find processes with python in their name
$ pgrep python
10787
20060
25975
27475
# kill processes with 2 consecutive vowels (don't do this!)
$ pkill -9 '[aeiou][aeiou]'
# kill all programs named teams or zoom
$ killall zoom teams
```

Running 100 Processes Inside our Virtual Machine

- **qemu** is one process (runing program) but it can simulate a machine running 100 processes

```
/ # cat >repeat.sh <<'eof'
#!/bin/sh
for j in $(seq 10)
do
    echo "Hello $j from process $1"
    sleep 10
done
eof
/ # for i in $(seq 100); do ./repeat.sh $i & done
....
```

Adding Networking to our System

- qemu's **-nic** option adds a virtual network card to our virtual machine
- take COMP3331/9331 to understand the networking commands

```
$ qemu-system-x86_64 -nic user,model=virtio-net-pci -drive
↪ file=mydisk,format=raw -kernel mykernel -append 'init=/bin/sh
↪ root=/dev/sda rw console=ttyS0 quiet' -display none -serial stdio
/ # mkdir etc
/ # echo "nameserver 10.0.2.3" >/etc/resolv.conf
/ # ifconfig eth0 up 10.0.2.15
/ # route add default gw 10.0.2.2
/ # ifconfig eth0 up 10.0.2.15
# check networking works by getting today's weather
/ # wget -O- http://wttr.in
Connecting to wttr.in (5.9.243.187:80)
writing to stdout
Weather report: Sydney, Australia
...
# we now have networking, lets install a Linux distribution
```

Summary for Setting up Special Filesystems & Networking

```
$ qemu-system-x86_64 -nic user,model=virtio-net-pci -drive
↪ file=mydisk,format=raw -kernel mykernel -append 'init=/bin/sh
↪ root=/dev/sda rw console=ttyS0 quiet' -display none -serial stdio
/ # mkdir -p /dev /proc /sys /etc
/ # mount -t devtmpfs none /dev
/ # mount -t proc none /proc
/ # mount -t sysfs none /sys
/ # echo "nameserver 10.0.2.3" >/etc/resolv.conf
/ # ifconfig eth0 up 10.0.2.15
/ # route add default gw 10.0.2.2
/ # ifconfig eth0 up 10.0.2.15
```

Linux Distributions

- A distribution packages the Linux kernel with many other programs.
- Hundreds of linux distributions
- distributions used by CSE students include:
 - Debian - classic distro, used for CSE systems, runs on many architectures
 - Ubuntu - popular distro, based on Debian
 - Mint - based on Ubuntu
 - Arch - lightweight rolling release
 - Alpine - lightweight distro popular for containers/VMs

- One of the oldest Linux distributions (1993)
- widely used & available for many platforms.
 - 10 CPU architecture officially supported
 - 14+ more CPU architecture unofficially supported
- Stable - new release every 2 yrs.
- Many derivative distributions
- packages total 300+ million lines of code

Linux Packages

- A packages contains files that make up an application
- And build scripts to install/remove application.
- May contain metadata for managing the package.
- Used to install new applications onto a system
- Debian uses the **.deb** format (as do Ubuntu, Mint)
 - other distributions use other formats, e.g **rpm** (Red Hat) and **pkg** (Arch)
- distributions have their own package management tools
- Debian has
 - **dpkg** - low level tool - you probably don't need to use directly
 - **apt** - command-line tool - you probably want to use this
 - in scripts use **apt-get** (same options)
 - **aptitude** & **synaptic** - high level GUI tools

The Alpine Linux Distribution

- small, simple and secure
- good for environments low in memory and storage,
- uses busybox, musl, OpenRC, system
 - debian uses dash, glibc, systemd (more features, bigger footprint)
- used primarily for embedded systems, container & virtual machines
- less widely used, less architecture supported
- package format is apk (based on tar)

Installing Alpine on our Tiny VM using the Network

```
$ qemu-system-x86_64 -nic user,model=virtio-net-pci -drive
↪ file=mydisk,format=raw -kernel mykernel -append 'init=/bin/sh
↪ root=/dev/sda rw console=ttyS0 quiet' -display none -serial stdio
/ # mkdir /etc/apk
/ # apk_url=http://mirror.aarnet.edu.au/pub/alpine/latest-stable/main
/ # echo $apk_url >/etc/apk/repositories
/ # wget -qO- $apk_url/x86_64/apk-tools-static-2.14.6-r3.apk|tar -vzxf -
↪ sbin/apk.static
/ # rm /bin/* # remove our manually installed busybox
/ # apk.static -U --allow-untrusted --initdb add alpine-base
fetch
↪ http://mirror.aarnet.edu.au/pub/alpine/latest-stable/main/x86_64/APKINDEX.tar.gz
(1/25) Installing alpine-baselayout-data (3.4.3-r1)
(2/25) Installing musl (1.2.4-r1)
...
(24/25) Installing libc-utils (0.7.2-r5)
(25/25) Installing alpine-base (3.18.2-r0)
Executing busybox-1.36.1-r2.trigger
OK: 10 MiB in 25 packages
/ # sed -i 's/^tty/#tty/;s/^#ttyS0//' /etc/inittab
```

<https://www.cse.unsw.edu.au/~cs2041/25T1/>

COMP(2041|9044) 25T1 — Linux

55 / 73

Installing Alpine on our Tiny VM using the Network (continued)

```
/ # mkdir -p /etc/network
/ # cat >/etc/network/interfaces <<eof
auto lo
iface lo inet loopback

auto eth0
iface eth0 dhcp
eof
/ # rc-update add networking boot
/ # poweroff
```

<https://www.cse.unsw.edu.au/~cs2041/25T1/>

COMP(2041|9044) 25T1 — Linux

56 / 73

Booting Alpine Linux on our Tiny VM

```
$ qemu-system-x86_64 -nic user,model=virtio-net-pci -drive
↪ file=mydisk,format=raw -kernel mykernel -append 'root=/dev/sda rw
↪ console=ttyS0 quiet' -display none -serial stdio
...
Welcome to Alpine Linux 3.18
Kernel 6.4.8 on an x86_64 (/dev/ttyS0)
localhost login: root
...
localhost:~# cat /etc/os-release
NAME="Alpine Linux"
ID=alpine
VERSION_ID=3.18.2
PRETTY_NAME="Alpine Linux v3.18"
HOME_URL="https://alpinelinux.org/"
BUG_REPORT_URL="https://gitlab.alpinelinux.org/alpine/aports/-/issues"
```

<https://www.cse.unsw.edu.au/~cs2041/25T1/>

COMP(2041|9044) 25T1 — Linux

57 / 73

Installing An Alpine Package

```
~# apk update
fetch
↳ http://mirror.aarnet.edu.au/pub/alpine/latest-stable/main/x86_64/APKINDEX.tar.gz
v3.20.2-19-ga9bc4d9eff9
↳ [http://mirror.aarnet.edu.au/pub/alpine/latest-stable/main]
OK: 5521 distinct packages available
localhost:~# apk add python3
(1/17) Installing libbz2 (1.0.8-r5)
(2/17) Installing libexpat (2.5.0-r1)
(3/17) Installing libffi (3.4.4-r2)
...
(14/17) Installing python3 (3.12.3-r1)
(15/17) Installing python3-pycache-pyc0 (3.12.3-r1)
(16/17) Installing pyc (3.12.3-r1)
(17/17) Installing python3-pyc (3.12.3-r1)
Executing busybox-1.36.1-r2.trigger
OK: 53 MiB in 42 packages
localhost:~# python3
Python 3.12.3 (main, Apr 18 2024, 07:52:31) [GCC 13.2.1 20240309] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

<https://www.cse.unsw.edu.au/~cs2041/25T1/>

COMP(2041|9044) 25T1 — Linux

58 / 73

/etc/passwd - user “database”

User information in **/etc/passwd**

Password hashes in **/etc/shadow**

Every user has unique number: **uid**

```
$ sed 2q /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
$ sed 2q /etc/shadow
root:$6$YiSiP7Pehz8aoe...../:18379:0:99999:7:::
daemon*:18362:0:99999:7:::
```

- better to not edit **/etc/passwd** & **/etc/shadow** directly
- instead use (distribution-specific) tools, e.g on Debian-like systems:
 - add users with **adduser**
 - remove users with **deluser**

<https://www.cse.unsw.edu.au/~cs2041/25T1/>

COMP(2041|9044) 25T1 — Linux

59 / 73

/etc/group - group database

Group information in **/etc/group**

```
$ head /etc/group
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:
tty:x:5:
```

- Each group has unique number: **gid**
- Better to not edit **/etc/group** directly
- instead use (distribution-specific) tools, e.g on Debian-like systems:
 - Add users to groups with **adduser**
 - Also **addgroup** **delgroup**

<https://www.cse.unsw.edu.au/~cs2041/25T1/>

COMP(2041|9044) 25T1 — Linux

60 / 73

- Many system actions require root (uid == 0)
- **su** allows you to execute commands as root or any other user.
- **sudo** allows command to be run as root
- Use cautiously - easy to damage system with commands run as root.
- Edit sudo config file /etc/sudoers with **visudo**

```
# Adding user to sudo group should allow them to run sudo
$ adduser andrewt sudo
```

fsck - repair a file-system

- Power failure or other unexpected events may leave a filesystem in inconsistent state.
- **fsck** (file system check) checks and repairs a file-system.

```
# copy a random byte into our virtual disk to simulate corruption
$ dd if=/dev/urandom of=mydisk bs=1 seek=1024 count=1 conv=notrunc
$ fuse2fs -o fakeroot mydisk mnt
mydisk: The ext2 superblock is corrupt.
Please run e2fsck -fy mydisk.
# repair file system
$ fsck.ext2 mydisk
e2fsck 1.46.4 (18-Aug-2021)
ext2fs_open2: The ext2 superblock is corrupt
fsck.ext2: Superblock invalid, trying backup blocks...
mydisk was not cleanly unmounted, check forced.
...
Pass 1: Checking inodes, blocks, and sizes
Deleted inode 13 has zero dtime. Fix? yes
...
# filesystem will now mount
$ fuse2fs -o fakeroot mydisk mnt
```

- File system should not be in use (unmounted)
- **Beware:** dangerous operation - have backups!

running a program with a different root directory

- Linux **namespace** is the view of a system resource given to a process
- **namespaces** include filesystem (mount), processes (PID) and network
- Linux unshare system call allows a process to be run with a different **namespace**
 - e.g allows process to be run with a different root directory

```
$ mkdir tiny_root/
$ echo "hello from tiny root" >tiny_root/hello.txt
$ ls tiny_root
hello.txt
$ unshare --map-root-user --root=tiny_root /bin/sh
unshare: failed to execute /bin/sh: No such file or directory
# failed because no /bin/sh once root directory changed
$ ls tiny_root/bin/sh
ls: cannot access 'tiny_root/bin/sh': No such file or directory
```

running a program with a different root directory

```
# add busybox to tinyroot
$ mkdir tiny_root/bin
$ cp busybox tiny_root/bin/
# create links for the 400 programsu busybox provides
$ for p in $(./busybox --list); do ln -s busybox tiny_root/bin/$p;done
$ ls -l tiny_root/bin/sh
lrwxrwxrwx 1 andrewt andrewt 7 Apr 18 10:14 tiny_root/bin/sh -> busybox
$ unshare --map-root-user --root=tiny_root /bin/sh
# now in shell which sees tiny_root as /
$ ls /
bin
hello.txt
$ cat /hello.txt
hello from tiny root
```

Linux containers

- Linux containers use namespaces to run processes in a different “world”
- you can run a process with effectively:
 - different root directory
 - e.g. /usr/bin can look different to process
 - e.g different process ids
 - e.g different view of network
 - different uid
 - specified resource limits
- allows a program to be run which needs different packages to those installed

Docker & Podman - Tools for Running Containers

- Docker popular set of tools for working with Linux containers
 - easy to use, but heavyweight (requires daemon)
- Podman open source equivalent for Red Hat
 - other open source tools may be useful depending on needs
- both use a union file system in clever way
 - overlays images so base images can be reused
- can specify dependencies for a program and produce self-contained image
 - e.g. can specify program requires python-3.9
- can run image on any Linux, Windows or OSX system with Docker installed
 - independent of what software is installed on that platform
- **hub.docker.com** provides sharing similar to **github.com**
- Docker great for many purposes, but heavyweight (requires daemon)
- good to experiment with over the term-break
 - containers hard to use at CSE
 - disk space issues if every student has containers
 - docker not available - requires root
 - podman doesn't work with NFS
 - easy to install docker or podman on your own machine

Running Shells Command Inside and Outside a Container

```
$ whoami
andrewt
$ pwd
/home/andrewt
$ grep andrewt /etc/passwd
andrewt:x:517:517:andrewt,,,:/home/andrewt:/bin/bash
# or podman run -it --rm alpine
$ docker run -it --rm alpine
...
/ # whoami
root
/ # pwd
/
/ # find /|wc -l
65716
/ # grep andrewt /etc/passwd
/ # ls /home/andrewt
ls: /home/andrewt: No such file or directory
```

<https://www.cse.unsw.edu.au/~cs2041/25T1/>

COMP(2041|9044) 25T1 — Linux

67 / 73

Mounting A Directory Inside a Container

```
$ ls /home/z1234567
hello.txt
$ cat /home/z1234567/hello.txt
Hello COMP(2041|9044)
# or podman run -it -v /home/z1234567:/mnt alpine
$ docker run -it -v /home/z1234567:/mnt alpine
/ # ls /mnt
hello.txt
/ # cat /mnt/hello.txt
Hello COMP(2041|9044)
/ # echo hello from a container >/mnt/new.txt
/ # exit
$ cat /home/z1234567/new.txt
hello from a container
```

<https://www.cse.unsw.edu.au/~cs2041/25T1/>

COMP(2041|9044) 25T1 — Linux

68 / 73

Specifying a Container Image with a Dockerfile

- Docker has a specification language for creating containers images
 - series of commands in a file (usually) named **Dockerfile**
- **FROM** command - use pre-existing Docker image used as starting point
 - often image with linux distribution installed (Linux Alpine, Ubuntu, Debian ...)
 - and perhaps an application such as webserver or database (nginx, postgres, ...)
- **ADD** command - copy files into image
- **RUN** command - run shell commands as part of building image
- **ENTRYPOINT** command - run command when container starts

<https://www.cse.unsw.edu.au/~cs2041/25T1/>

COMP(2041|9044) 25T1 — Linux

69 / 73

Example Dockerfile

```
# use Alpine Linux as the base of our image
FROM alpine
RUN \
    apk update &&\
    # install apache web server
    apk add apache2 curl &&\
    # a configuration file Apache needs
    echo ServerName my-web-server >/etc/apache2/conf.d/localhost.conf
# add a file to the image for our webserver to serve
ADD hello.txt /var/www/localhost/htdocs/hello.txt
# run web-server plus shell when container started
ENTRYPOINT \
    /usr/sbin/httpd &&\
    ash &&\
    killall httpd
```

source code for Dockerfile

Example - Building and Running the Container

```
# 127.0.0.1 is a special network address for the local machine
$ curl http://127.0.0.1/hello.txt
curl: (7) Failed to connect to 127.0.0.1 port 80 after 0 ms: Connection
  ↪ refused
$ docker build -t my_apache .
$ docker run -it --rm my_apache
/ # wget -O- http://127.0.0.1/hello.txt
hello from inside a container
```

In another window try to access web server in container

```
$ curl http://127.0.0.1/hello.txt
curl: (7) Failed to connect to 127.0.0.1 port 80 after 0 ms: Connection
  ↪ refused
$
```

Example - Making a Network Port visible Outside the Container

```
$ curl http://127.0.0.1:6789/hello.txt
curl: (7) Failed to connect to 127.0.0.1 port 6789 after 0 ms: Connection
  ↪ refused
$ docker run -it -p 6789:80 --rm my_apache
/ # tail -f /var/log/apache2/access.log
172.17.0.1 - - [04/Aug/1997:01:15:18 +0000] "GET /hello.html HTTP/1.1" 200 9
  ↪ "-" "curl/7.88.1"
```

In another window

```
$ curl http://127.0.0.1:6789/hello.txt
hello from a webserver inside a container
```

Example - Supplying Webpages from Outside the Container

```
$ mkdir /tmp/content
$ echo I love shell > /tmp/content/webpage.txt
$ docker run -it -p 6789:80 -v /tmp/content:/var/www/localhost/htdocs --rm
  ↪ my_apache
/ # tail -f //var/log/apache2/access_log
172.17.0.1 - - [03/Aug/1997:01:15:18 +0000] "GET /5.html HTTP/1.1" 200 9 "-"
  ↪ "curl/7.88.1"
```

In another window

```
$ curl http://127.0.0.1:6789/webpage.txt
I love shell
```