Linux from Scratch

Version 20180906.1

Ross Alexander

September 6, 2018

Contents

1 Brief History of Unix and Linux

1.1 Multics

In 1964 a ambitious program to create a new operating system was founded called Multics. By 1969 Bell Labs (owned by AT&T) pulled out. A number of people within Bell Labs then went on to implement many of the ideas on a Digital PDP-8. This was originally called Unics, later renamed to Unix.

1.2 The C Language

Originally written in PDP-8 assembly the developers first created a high level language called B, when then went on to become C. Between 1970 and 1975 Unix was mostly migrated from from assembly to C. By having the OS written in a high level language allowed it to be ported to different CPU architectures without a complete rewrite.

1.3 BSD

Copies of Unix source [last was Version 7, 1979] were publically available. The University of California Berkeley contributed a large amount of additional code and improvements between 1979 and 1991. When AT&T were allowed to sell Unix a long legal fight ensued. Eventually it was settled in 1994, with AT&T developing a proprietary version, called System III Unix, while Berkeley publically released 4.4BSD as source code. This was free of any AT&T code. By then Linux had become the leading open source alternative.

1.4 Minix [1987]

Written by Andrew Tanenhaum to accompany his textbook *Operation Systems: Design and Implementation*, Minix was written for PC/AT and syscall compatible with Seventh Edition Unix. Minix was a different kernel design from historical Unix but remained compatiable with at the userspace level.

1.5 Linux

Started by Linus Torvalds in 1991 as a replacement kernel for Minix (it kept the Minix filesystem and was syscall compatible), originally was a toy OS on the Intel 80386 (the first x86 processor with a paged memory manager, released 1985). Initially it relied on the Minix userspace, the GNU and BSD tools were soon ported so by 1992 it was free of Minix code.

2 Virtualisation

2.1 QEMU

QEMU is an open source CPU and hardware emulator, covering a large number of different CPU architectures and hardware platforms. Specifically it emulates the Intel Q35 chipset [ICH9], released in 2007. This supports multiple CPUs with multiple cores, USB 2.0, PCIe 1.1 and AHCI SATA controllers. BIOS is either via LPC and SPI buses.

2.2 KVM

The Kernel-based Virtual Machine is a Linux specific virtualisation platform. It utilises either the Intel VT or AMD-V hardware extensions to do hardware acceleration of the virtualation emulation, especially around memory faults and interrupt handling. Generally used in conjuction with QEMU (or QEMU with KVM) to do the platform emulation.

2.3 Firmware (BIOS)

The original IBM PC came with a large (at the time) firmware called the Basic I/O System (BIOS). Originally stored on ROM chips, the BIOS was hard mapped into the top 64K of the 8086 memory address space. When the CPU booted it started with the address specified by the reset vector (0xFFFF0). The BIOS would then initialize any additional hardware, run the Power On Self-Test (POST) before trying to find a bootable device.

Currently QEMU uses the SeaBIOS code [from the coreboot project] as it default firmware for the x86 and x64 platforms (since it runs in 16-bit mode it is the same for both 32-bit and 64-bit CPUs).

3 UEFI

When Intel developed the IA-64 architecture the x86 BIOS was not compatible so it had to create a new firmware platform, called Extensible Firmware Interface (EFI). This was later back ported to the X64 architecture as the Univeral Extensible Firmware Interface (UEFI). Unlike the original BIOS UEFI comes in both 32-bit and 64-bit versions.

QEMU currently supports (or rather the way around) the tianocore Open Virtual Machine Firmware (OVMF) UEFI code. UEFI firmware generally support booting BIOS based MBR systems if necessary.

4 Components of Unix

4.1 Bootloader

The bootloader is the glue between the firmware and kernel. On some systems it is possible to have the firmware load the kernel directly however bootloaders generally allow considerable flexibility. For Linux on x64 the most popular boot loader is GRUB. GRUB2 can be built to be loaded from the original 16-bit BIOS (as a 16-bit executable) or as UEFI executable (also as a PXE bootloader). This allows a standard booting experience regardless of the firmware or initial boot method.

4.2 Kernel

The kernel is heart of the Operating System. It manages memory, multiprocessing, devices, network and file systems. The kernel interacts with user processes via syscalls, sockets, device files and pseudo file systems.

4.2.1 File Systems

A file system is a collection of files, normally on block storage. Examples of this are FAT, NTFS or EXT4. Other types of file systems include tmpfs (which is based in memory), NFS or SMB for network based file systems, or pseudo file systems such as proc or sysfs. Specfically Unix requires a file system (known as the root file system) to load its initial process [init] from.

4.2.2 Devices

The kernel contains code to initial and control various hardware devices, such as the PCI bus, disk controllers, network interface cards and I/O devices such as serial ports or VGA controllers. These can be load modules or compiled directly into the kernel image. The kernel must have enough drivers builtin to be able to find and mount the root file system.

4.3 Kernel Processes

The kernel may create a number of kernel processes. These are not normally controllable from the user environment.

4.4 Userspace

Once the kernel has mounted the root file system it loads an excutable image from it [the root file system] and starts it as process ID 1. This can be a shell for debugging but normally it is specialised inititialisation process, often called init. Unix creates new processes by forking an existing running process. The forked process is initially identical to its parent except for return code from the fork system call. Normally the child process will then load a new executable image over top of itself using the exec syscall.

4.4.1 The Shell

The default Unix command intepreter, originally the Thompson Shell, then the Mashey Shell and finally the Bourne Shell. There are a number of clones of the Bourne Shell and several alternative shells including the Bourne Again shell, Korn shell, the C shell and tcsh.

4.5 Development environment

Apocryphally the first program written using hand coded binary was an assembler. The second program was the same assembler, written in assembly. Assemblers take text file with assembly neumonics and convert them to binary while automatically tracking the mapping of names [symbols] to addresses.

As programs grew in size they became too large to be assembled as a single file, either because the the assembler didn't have enough memory or the coder needed to split the program up to make it managable.

To handler this assemblers produced binary output in an intermediate format known as an object format. This contained both the code and data as well as symbol tables and relocation information. Each piece of assembly is assembled into an object file and a separate program, known as a linker, which combined all the code and data into an executable program, using the symbol table and relocation information to correct symbol addresses.

Rather than having to individually link the object file of standardized functions the object files can be collected together into an archive file. The linker would then extract any referenced symbols out of the library into the final executable without including the full library.

With the C language a standard set of functions was developed. This was collected into a library (libc.a). All userland C programs generally are linked against libc.a. The original unix executable format (known as a.out) did not support runtime (aka dynamic) linking. So if libc.a was updated, every executable had to be relinked to pick up the new code.

Dynamic linking involves either the kernel or a userland program linking the executable into memory at runtime. After a number of attempts Linux converged on the ELF (Executable Link Format) standard. The ELF format convered object files, shared (aka dynamic) libraries and executables. When a dynamically linked executable is execed by the kernel it loads an additional program into memory and executes that instead. This code is the dynamic linker (/lib/ld-linux.so.2 for 32-bit executables and /lib64/ld-linux-x64-64.so.2 for 64-bit executables).

In addition to doing the dynamic linking at runtime the dynamic linker (in the guise of libdl.so) can be used to explicitly load additional shared libraries during the execution of program. This allows programs to extend functionality without having to relink the base executable. These are often known as modules, so for example php has modules for accessing oracle, postgresql and mysql. If a new database is required a module can be written and created as a shared library and php can be instructed to load that shared library at runtime without having to modify the php executable itself.

The C library NSS (Name Service Switch) and PAM (Pluggable Authentication Modules) are examples of functionality which rely on dynamically loading shared libraries to work.

5 Tools

The local environment requires a full build toolchain. This is quite extensive. To avoid having to go through the individual building of each package they are all pre-installed in their own subdirectory.

5.1 NBD

The kernel Network Block Driver allows a userspace daemon to present a block device over the network (using TCP), which the kernel treats as a local block device (similar to a local disk). This allows VM image files to presented back to the local OS. The actual VM image format can be hidden from the OS, so that VMDK or QCOW2 formatted images look like a flat raw device back to the kernel.

The kernel NBD driver supports partitions so each partition can be mounted as required. The advantage of NBD over using a loop device is that image itself doesn't have to be raw.

6 VM Image File

QEMU supports several image formats include VMDK and VHDX. Its native format is QCOW2 but for simplicity the examples are using RAW. The QEMU tool qemu-img can convert between the various formats so a raw image can be converted to VMDK if required.

6.1 Creating an image

To create a new VM image file qemu-img is used. In this example a 5GB raw disk will be created.

```
qemu-img create -f raw lfs/lfs.raw 5G
```

Make sure nbd is enabled (it is built as a kernel module).

```
modprobe nbd
```

Create an nbd daemon.

```
qemu-nbd -c /dev/nbd0 -f raw lfs/lfs.raw
```

Check device NBD is present.

```
bash-4.4# lsblk /dev/nbd0
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
nbd0  43:0    0   5G  0 disk
```

Create GPT partition (label) using parted. UEFI requires disks have a GPT to be bootable (except when doing BIOS emulation).

```
parted -s /dev/nbd0 mklabel gpt
```

Check GPT label has been created.

```
bash-4.4# parted -s /dev/nbd0 print
Model: Unknown (unknown)
Disk /dev/nbd0: 5369MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start  End  Size  File system  Name  Flags
```

6.2 EFI System Partition (ESP)

EFI requires any boot devices to have an EFI System Partition. This is normally the first partition on a disk but does not have to be. It does need to be FAT formatted. The documentation states that it should be FAT32 and a minimum size of 100MB. The minimum size a FAT32 partition can be is 32MB. In this example the partition is 36MB. GPT supports partition names.

```
parted -s /dev/nbd0 mkpart EFISYS 2M 38M
```

In addition the partition needs to be flagged as the ESP.

```
parted -s /dev/nbd0 set 1 esp
```

The partition then needs to be formatted. FAT supports file system names and these can be used to find the find the file system without knowing its device / partition.

```
bash-4.4# mkfs -t fat -F 32 -n EFISYS /dev/nbd0p1
mkfs.fat 4.1 (2017-01-24)
```

6.3 Boot partition

The boot partition isn't strictly required but does enable additional flexibility in boot options. It normally only contains the kernel image, GRUB configuration and modules and possibly an initrd (init RAM disk).

This partition is normally a POSIX compatible file system rather than FAT. As performance isn't required using EXT4 is a good choice. It can be as small as 10M but 100MB gives space for multiple kernels.

```
parted -s /dev/nbd0 mkpart boot 38M 136M
```

The file system label is /boot.

```
bash-4.4# mkfs -t ext4 -L /boot /dev/nbd0p2
mke2fs 1.44.2 (14-May-2018)
Discarding device blocks: done
Creating filesystem with 96256 1k blocks and 24096 inodes
Filesystem UUID: b9f50581-fc13-4107-a2d7-f92a9862a1cf
Superblock backups stored on blocks:
        8193, 24577, 40961, 57345, 73729

Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done
```

6.4 Root partiton

The remaining disk will become the root partition.

```
parted -s /dev/nbd0 mkpart / 136M 5G
```

For this example EXT4 is used for the root file system.

```
bash-4.4# mkfs -t ext4 -L / /dev/nbd0p3
mke2fs 1.44.2 (14-May-2018)
Discarding device blocks: failed - Input/output error
Creating filesystem with 1277184 4k blocks and 319488 inodes
Filesystem UUID: 38b62fa3-51d4-4024-9f2d-d64a7f36eff2
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
```

6.5 Final disk layout

```
bash-4.4# lsblk /dev/nbd0
NAME     MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
nbd0      43:0    0    5G  0 disk├─
nbd0p1  43:1    0   34M  0 part├─
nbd0p2  43:2    0   94M  0 part└─
nbd0p3  43:3    0  4.9G  0 part
```

6.6 Disconnect nbd

```
bash-4.4# qemu-nbd -d /dev/nbd0
/dev/nbd0 disconnected
```

7 OVMF and QEMU

The OVMF image comes in two parts, OVMF_CODE.fd and OVMF_VARS.fd. The code is static and can be shared between VMs. The VARS should be unique to each VM as it is updated by the running firmware (and potentially the OS).

```
cp OVMF_VARS.fd lfs/OVMF_VARS.fd
```

7.1 QEMU script

For testing is a easiest to run qemu on the command line. In production this would be done using a daemon like libvirtd or a full virtualisation environment like Openstack.

Rather than having a virtual console this uses a virtual serial console, which is connected to stdio (so the running shell). If run from a daemon then the serial console can be a telnet server.

```
bash-4.4# more lfs.sh
NAME=lfs
IMG=${NAME}/${NAME}.raw
OVMF=./OVMF_CODE.fd
VARS=${NAME}/OVMF_VARS.fd

qemu-system-x86_64 \
    -enable-kvm \
    -machine q35 \
    -m 1024 -cpu host \
    -drive unit=0,if=pflash,format=raw,readonly,file=$OVMF \
    -drive unit=1,if=pflash,format=raw,file=$VARS \
    -drive index=0,if=none,id=hd,file=$IMG,format=raw \
    -device virtio-scsi-pci,id=scsi \
    -device scsi-hd,drive=hd \
    -netdev type=tap,id=net0,ifname=$NAME,script=/locker/vm/ovs-up.sh,downscript=/locker/vm/ovs-down.sh \
    -device virtio-net-pci,netdev=net0,mac=00:16:3E:00:00:01 \
    -nographic \
    -monitor tcp:localhost:4445,server,nowait,telnet \
    -serial stdio
```

Because there is nothing on the disk initially UEFI will start in its interface shell. It can see all three partitions and the FAT32 file system.

```
UEFI Interactive Shell v2.2
EDK II
UEFI v2.70 (EDK II, 0x00010000)
Mapping table
      FS0: Alias(s):HD0a0b:;BLK1:
          PciRoot(0x0)/Pci(0x2,0x0)/Scsi(0x0,0x0)/HD(1,GPT,F9BF897A-4A17-4C30-89FE-2E29FE6F8E64,0x1000,0x11000)
```

```
    BLK0: Alias(s):
        PciRoot(0x0)/Pci(0x2,0x0)/Scsi(0x0,0x0)
    BLK2: Alias(s):
        PciRoot(0x0)/Pci(0x2,0x0)/Scsi(0x0,0x0)/HD(2,GPT,DA961AA1-00D4-44D0-8014-24895DDC779F,0x12000,0x2F000)
    BLK3: Alias(s):
        PciRoot(0x0)/Pci(0x2,0x0)/Scsi(0x0,0x0)/HD(3,GPT,41455D11-2E9A-4697-B3FA-535F89BABC9A,0x41000,0x9BE800)



Press ESC in 5 seconds to skip startup.nsh or any other key to continue.
Shell>
```

7.2 Install GRUB boot loader

Create GRUB EFI image. The image will include an embedded file system with all the GRUB modules inside it so it isn't strictly necessary to force include the part_gpt and part_msdos. This also embeds a GRUB configuration file inside the embedded file system so it will be executed by GRUB.

```
grub-mkstandalone -O x86_64-efi -o grub/grub.efi --modules="part_gpt part_msdos" "/boot/grub/grub.cfg=./grub/grub.cfg"
```

Connect nbd0 to image.

```
qemu-nbd -c /dev/nbd0 -f raw lfs/lfs.raw
```

Create mountpoint for root file system and mount rootfs.

```
mkdir lfs/root
mount /dev/nbd0p3 lfs/root/
```

Create mountpoint for /boot on the rootfs and mount /boot.

```
mkdir lfs/root/boot
mount /dev/nbd0p2 lfs/root/boot/
```

Create mountpoint for ESP under /boot/efi.

```
mkdir lfs/root/boot/efi
mount /dev/nbd0p1 lfs/root/boot/efi
```

Create grub subdirectory for configuration file.

```
mkdir lfs/root/boot/grub
```

UEFI by default looks for the boot image /EFI/BOOT/BOOTX64.EFI. Because this is a FAT filesystem it is not case sensitive.

```
mkdir -p lfs/root/boot/efi/EFI/BOOT/
cp grub/grub.efi lfs/root/boot/efi/EFI/BOOT/BOOTX64.EFI
```

7.2.1 grub.cfg

Use blkid to find partition UUID of the root partition.

```
blkid /dev/nbd0p2
/dev/nbd0p2: LABEL="/boot" UUID="b9f50581-fc13-4107-a2d7-f92a9862a1cf" TYPE="ext4" PARTLABEL="boot" PARTUUID="da961aa1-00d4-44d0-8014-24895ddc779f"
```

Create grub.cfg based on kernel (4.17.5) and root partition.

```
search.fs_label / root
search.fs_label /boot boot

menuentry "Linux 4.17.5 [dash]" {
    linux ($boot)/vmlinuz-4.17.5 root=PARTUUID=41455d11-2e9a-4697-b3fa-535f89babc9a rootfstype=ext4 console=ttyS0 init=/bin/dash
    boot
}

menuentry "Linux 4.17.5" {
    linux ($boot)/vmlinuz-4.17.5 root=PARTUUID=41455d11-2e9a-4697-b3fa-535f89babc9a rootfstype=ext4 console=ttyS0
    boot
}
menuentry "Halt" {
    halt
}
menuentry "Reboot" {
    reboot
}
menuentry "Exit" {
    exit
}
```

Unmount everything and diconnect nbd.

```
umount lfs/root/boot/efi
umount lfs/root/boot/
umount lfs/root/
qemu-nbd -d /dev/nbd0
```

Booting the kernel will cause it to panic as there is nothing on the root file system.

```
[    0.792449] Kernel panic - not syncing: No working init found.  Try passing init= option to kernel. See Linux Documentation/admin-guide/init.rst
 for guidance.
[    0.793725] CPU: 0 PID: 1 Comm: swapper/0 Not tainted 4.17.5 #1
[    0.794266] Hardware name: QEMU Standard PC (Q35 + ICH9, 2009), BIOS 0.0.0 02/06/2015
[    0.794978] Call Trace:
[    0.795221]  dump_stack+0x5c/0x80
[    0.795538]  ? rest_init+0x59/0xb5
[    0.795859]  panic+0xd2/0x223
[    0.796141]  ? do_execveat_common+0x96/0x770
[    0.796535]  ? rest_init+0xb5/0xb5
[    0.796847]  kernel_init+0xed/0xf4
[    0.797162]  ret_from_fork+0x22/0x40
[    0.798114] Kernel Offset: 0x1ee00000 from 0xffffffff81000000 (relocation range: 0xffffffff80000000-0xffffffffbfffffff)
[    0.799097] ---[ end Kernel panic - not syncing: No working init found.  Try passing init= option to kernel. See Linux
 Documentation/admin-guide/init.rst for guidance. ]---
```

8 Packages

skel-dirs Bespoke package with the basic directory structure, including symbolic links for /bin, /sbin, /lib & /lib64 to their /usr counterparts.
glibc The GNU C library. A number of shared libraries (including libc & libm) and the dynamic loader.
libcap Shared library support Linux Capabilities (effectively process ACLs).
libacl Shared library for POSIX ACLs on files. Requires ACLs to be enabled on individual file systems to be effective.
libattr Shared library for extended file attributes. Xattrs allow for bespoke attributes (metadata) on files, provided the file system supports it.
ncurses Includes libtinfo, for terminal (tty) handling, libncurses for terminal based programs, and libforms for advanced text windows etc.
readline Shared libraries for CLI input, including libreadline and libhistory.
dash The Debian Almquist shell. A clone of the Bourne Shell with a focus on having a small footprint.
bash The Bourne Again Shell. A fully functioned clone of the Bourne shell. The standard shell for Linux.
coreutils A huge collection of essential unix utilities. Originally developed by the GNU project as alternatives to the standard utilities on commerical unix varients, these are the default on Linux.
util-linux A large collection of unix utilities which are not covered by coreutils. Some of these a linux specific but many are standard utilities.

8.1 Booting into dash

```
[    0.793514] EXT4-fs (sda3): mounted filesystem with ordered data mode. Opts: (null)
[    0.794231] VFS: Mounted root (ext4 filesystem) readonly on device 8:3.
[    0.794913] devtmpfs: mounted
[    0.795874] Freeing unused kernel memory: 1220K
[    0.796293] Write protecting the kernel read-only data: 16384k
[    0.797716] Freeing unused kernel memory: 2020K
[    0.800750] Freeing unused kernel memory: 1668K
[    1.441165] clocksource: tsc: mask: 0xffffffffffffffff max_cycles: 0x33e45f795f8, max_idle_ns: 440795261487 ns
[    1.440439] tsc: Refined TSC clocksource calibration: 3600.013 MHz
/bin/dash: 0: can't access tty; job control turned off
#
```

8.1.1 devtmpfs

The devtmpfs is the first pseudo file system mounted. For userland to communicate with devices special files called device files are used. These are not real files, they map to kernel devices using major and minor device numbers. A device file can be either a character file, such as a tty, or a block device such as disk.

Historically these device files had to be manually created (normally under /dev). The actual device numbers had to found in the kernel documentation, there was no way to create them automagically.

An attempt to solve this by exporting these as a psuedo file system (devfs) ended in failure as people couldn't agree on UIDs and permissions for the devices. Linus declared it a userland issue and killed devfs. The solution reached was to create an API from the kernel to userland exporting kernel events and a userland daemon to handle these events as it wanted. This lead to the development of udev. However a minimal /dev is required for init (or systemd) to function correctly. To solve this a minimal /dev file system is exported by the kernel. It is effectively a tmpfs (in memory temporary file system) so can be overwritten once udevd starts. Rather than complicated UID and permissions everything is UID and GID 0 and with a few exceptions 600 (rw owner only).

```
# ls -l /dev/
total 0
drwxr-xr-x 2 0 0      60 Jul 31 13:03 bsg
crw------- 1 0 0  10, 234 Jul 31 13:03 btrfs-control
crw------- 1 0 0   5,   1 Jul 31 13:03 console
crw------- 1 0 0  10,  61 Jul 31 13:03 cpu_dma_latency
crw-rw-rw- 1 0 0   1,   7 Jul 31 13:03 full
crw------- 1 0 0  10, 228 Jul 31 13:03 hpet
crw------- 1 0 0  10, 183 Jul 31 13:03 hwrng
crw-r--r-- 1 0 0   1,  11 Jul 31 13:03 kmsg
crw------- 1 0 0   1,   1 Jul 31 13:03 mem
crw------- 1 0 0  10,  58 Jul 31 13:03 memory_bandwidth
crw------- 1 0 0  10,  60 Jul 31 13:03 network_latency
crw------- 1 0 0  10,  59 Jul 31 13:03 network_throughput
crw-rw-rw- 1 0 0   1,   3 Jul 31 13:03 null
crw------- 1 0 0  10, 144 Jul 31 13:03 nvram
crw------- 1 0 0   1,   4 Jul 31 13:03 port
crw-rw-rw- 1 0 0   5,   2 Jul 31 13:03 ptmx
crw------- 1 0 0 251,   0 Jul 31 13:03 ptp0
crw-rw-rw- 1 0 0   1,   8 Jul 31 13:03 random
crw------- 1 0 0  10,  62 Jul 31 13:03 rfkill
crw------- 1 0 0 253,   0 Jul 31 13:03 rtc0
brw------- 1 0 0   8,   0 Jul 31 13:03 sda
brw------- 1 0 0   8,   1 Jul 31 13:03 sda1
brw------- 1 0 0   8,   2 Jul 31 13:03 sda2
brw------- 1 0 0   8,   3 Jul 31 13:03 sda3
crw------- 1 0 0  10, 231 Jul 31 13:03 snapshot
crw-rw-rw- 1 0 0   5,   0 Jul 31 13:03 tty
crw------- 1 0 0   4,   0 Jul 31 13:03 tty0
crw------- 1 0 0   4,   1 Jul 31 13:03 tty1
crw------- 1 0 0   4,  10 Jul 31 13:03 tty10
```

```
crw------- 1 0 0   4,  11 Jul 31 13:03 tty11
crw------- 1 0 0   4,  12 Jul 31 13:03 tty12
crw------- 1 0 0   4,  13 Jul 31 13:03 tty13
crw------- 1 0 0   4,  14 Jul 31 13:03 tty14
crw------- 1 0 0   4,  15 Jul 31 13:03 tty15
crw------- 1 0 0   4,  16 Jul 31 13:03 tty16
crw------- 1 0 0   4,  17 Jul 31 13:03 tty17
crw------- 1 0 0   4,  18 Jul 31 13:03 tty18
crw------- 1 0 0   4,  19 Jul 31 13:03 tty19
crw------- 1 0 0   4,   2 Jul 31 13:03 tty2
crw------- 1 0 0   4,  20 Jul 31 13:03 tty20
crw------- 1 0 0   4,  21 Jul 31 13:03 tty21
crw------- 1 0 0   4,  22 Jul 31 13:03 tty22
crw------- 1 0 0   4,  23 Jul 31 13:03 tty23
crw------- 1 0 0   4,  24 Jul 31 13:03 tty24
crw------- 1 0 0   4,  25 Jul 31 13:03 tty25
crw------- 1 0 0   4,  26 Jul 31 13:03 tty26
crw------- 1 0 0   4,  27 Jul 31 13:03 tty27
crw------- 1 0 0   4,  28 Jul 31 13:03 tty28
crw------- 1 0 0   4,  29 Jul 31 13:03 tty29
crw------- 1 0 0   4,   3 Jul 31 13:03 tty3
crw------- 1 0 0   4,  30 Jul 31 13:03 tty30
crw------- 1 0 0   4,  31 Jul 31 13:03 tty31
crw------- 1 0 0   4,  32 Jul 31 13:03 tty32
crw------- 1 0 0   4,  33 Jul 31 13:03 tty33
crw------- 1 0 0   4,  34 Jul 31 13:03 tty34
crw------- 1 0 0   4,  35 Jul 31 13:03 tty35
crw------- 1 0 0   4,  36 Jul 31 13:03 tty36
crw------- 1 0 0   4,  37 Jul 31 13:03 tty37
crw------- 1 0 0   4,  38 Jul 31 13:03 tty38
crw------- 1 0 0   4,  39 Jul 31 13:03 tty39
crw------- 1 0 0   4,   4 Jul 31 13:03 tty4
crw------- 1 0 0   4,  40 Jul 31 13:03 tty40
crw------- 1 0 0   4,  41 Jul 31 13:03 tty41
crw------- 1 0 0   4,  42 Jul 31 13:03 tty42
crw------- 1 0 0   4,  43 Jul 31 13:03 tty43
crw------- 1 0 0   4,  44 Jul 31 13:03 tty44
crw------- 1 0 0   4,  45 Jul 31 13:03 tty45
crw------- 1 0 0   4,  46 Jul 31 13:03 tty46
crw------- 1 0 0   4,  47 Jul 31 13:03 tty47
crw------- 1 0 0   4,  48 Jul 31 13:03 tty48
crw------- 1 0 0   4,  49 Jul 31 13:03 tty49
crw------- 1 0 0   4,   5 Jul 31 13:03 tty5
crw------- 1 0 0   4,  50 Jul 31 13:03 tty50
crw------- 1 0 0   4,  51 Jul 31 13:03 tty51
crw------- 1 0 0   4,  52 Jul 31 13:03 tty52
crw------- 1 0 0   4,  53 Jul 31 13:03 tty53
crw------- 1 0 0   4,  54 Jul 31 13:03 tty54
crw------- 1 0 0   4,  55 Jul 31 13:03 tty55
crw------- 1 0 0   4,  56 Jul 31 13:03 tty56
crw------- 1 0 0   4,  57 Jul 31 13:03 tty57
crw------- 1 0 0   4,  58 Jul 31 13:03 tty58
crw------- 1 0 0   4,  59 Jul 31 13:03 tty59
crw------- 1 0 0   4,   6 Jul 31 13:03 tty6
crw------- 1 0 0   4,  60 Jul 31 13:03 tty60
crw------- 1 0 0   4,  61 Jul 31 13:03 tty61
crw------- 1 0 0   4,  62 Jul 31 13:03 tty62
crw------- 1 0 0   4,  63 Jul 31 13:03 tty63
crw------- 1 0 0   4,   7 Jul 31 13:03 tty7
```

```
crw------- 1 0 0   4,   8 Jul 31 13:03 tty8
crw------- 1 0 0   4,   9 Jul 31 13:03 tty9
crw------- 1 0 0   4,  64 Jul 31 13:03 ttyS0
crw------- 1 0 0   4,  65 Jul 31 13:03 ttyS1
crw------- 1 0 0   4,  66 Jul 31 13:03 ttyS2
crw------- 1 0 0   4,  67 Jul 31 13:03 ttyS3
crw-rw-rw- 1 0 0   1,   9 Jul 31 13:03 urandom
crw------- 1 0 0   7,   0 Jul 31 13:03 vcs
crw------- 1 0 0   7,   1 Jul 31 13:03 vcs1
crw------- 1 0 0   7, 128 Jul 31 13:03 vcsa
crw------- 1 0 0   7, 129 Jul 31 13:03 vcsa1
crw------- 1 0 0  10,  63 Jul 31 13:03 vga_arbiter
crw-rw-rw- 1 0 0   1,   5 Jul 31 13:03 zero
```

8.1.2 procfs

Historically to find out what has happening in the kernel a userland process would read the map of kernel symbols (created when the kernel was linked together) and then read /dev/kmem at that address. This was considered a hack so the procfs file system was created (originally done in solaris I think) to export various details of the running kernel, including all the processes.

```
mount -t proc none /proc
```

```
# ls proc
1    200  581  642        cmdline      iomem       misc          sys
10   202  594  646        consoles     ioports     modules       sysrq-trigger
11   267  617  647        cpuinfo      irq         mounts        sysvipc
12   288  618  7          crypto       kallsyms    mtrr          thread-self
13   3    621  713        devices      kcore       net           timer_list
14   32   622  714        diskstats    key-users   pagetypeinfo  tty
15   4    627  725        dma          keys        partitions    uptime
17   405  630  8          driver       kmsg        schedstat     version
196  5    633  9          execdomains  kpagecount  self          vmallocinfo
197  508  634  acpi       fb           kpageflags  slabinfo      vmstat
199  509  637  buddyinfo  filesystems  loadavg     softirqs      zoneinfo
2    553  638  bus        fs           locks       stat
20   580  641  cgroups    interrupts   meminfo     swaps
```

8.1.3 sysfs

Along with the udev work the kernel exports a file system exposing its internal device tree and kernel parameters.

```
mount -t sysfs none /sys
```

```
# ls /sys
block  class  devices   fs          kernel  power
bus    dev    firmware  hypervisor  module
```

8.1.4 Killing the kernel

Exiting the shell will cause the kernel to panic (since there must always be a PID 1). Under normal circumstances the system should not be booting into a shell, instead it should use a specialised application, generally known as init.

```
# exit
[ 8259.407986] Kernel panic - not syncing: Attempted to kill init! exitcode=0x00000000
[ 8259.407986]
[ 8259.408884] CPU: 0 PID: 1 Comm: dash Not tainted 4.17.5 #1
[ 8259.409370] Hardware name: QEMU Standard PC (Q35 + ICH9, 2009), BIOS 0.0.0 02/06/2015
[ 8259.410072] Call Trace:
[ 8259.410320]  dump_stack+0x5c/0x80
[ 8259.410638]  panic+0xd2/0x223
[ 8259.410916]  ? security_file_free+0x1d/0x30
[ 8259.411298]  do_exit.cold.25+0x4e/0xfb
[ 8259.411647]  do_group_exit+0x35/0xa0
[ 8259.411976]  __x64_sys_exit_group+0xf/0x10
[ 8259.412351]  do_syscall_64+0x55/0x440
[ 8259.412689]  entry_SYSCALL_64_after_hwframe+0x44/0xa9
[ 8259.413149] RIP: 0033:0x7efc01185676
[ 8259.413478] RSP: 002b:00007ffce5defe48 EFLAGS: 00000206 ORIG_RAX: 00000000000000e7
[ 8259.414162] RAX: ffffffffffffffda RBX: 0000000000000004 RCX: 00007efc01185676
[ 8259.414804] RDX: 0000000000000000 RSI: 000000000000003c RDI: 0000000000000000
[ 8259.415450] RBP: 0000000000412680 R08: 00000000000000e7 R09: ffffffffffffff80
[ 8259.416088] R10: 000000000000021f R11: 0000000000000206 R12: 0000000000402340
[ 8259.416738] R13: 00007ffce5df0120 R14: 0000000000000000 R15: 0000000000000000
[ 8259.417928] Kernel Offset: 0x24a00000 from 0xffffffff81000000 (relocation range: 0xffffffff80000000-0xffffffffbfffffff)
[ 8259.418911] ---[ end Kernel panic - not syncing: Attempted to kill init! exitcode=0x00000000
[ 8259.418911]  ]---
```

9 SysVinit

One of the original purposes of Unix was to allow multiple users to connect to the system simultaneously. In the days before Ethernet this was one using teletype terminals connected by a serial line concentrator. Each serial line became a device in /dev, for example /dev/ttyS0, /dev/ttyS1 etc. For each device to be useful a getty process is required to be connected to it so init is heavily focused on spawning (and respawning) getty processes.

As unix envolved the whole initialization process became more complicated and unmanagable so in Unix System V (from AT&T) a new init paradigm based on runlevels was created. This was carried forward into Linux with the sysvinit package. However the package doesn't actually come with any init scripts (often known as rc scripts) so each distribution still had to maintain their once basic initialisation.

9.1 rc.S

The rc.S (shell) script is hand written. It is a minimal script which mounts the necessary kernel file systems, starts udevd to populate /dev, does a fsck (file system check) on the root file system and starts syslogd to start capturing logs.

The additional packages are needed for this (some are dependencies).

sysvinit A reimplentation of the System V init program.
sysvinit-scripts Bespoke sysvinit configuration and rc scripts.
gawk The GNU implementation of awk. Used in the rc scripts.
grep The fatest grep in the west. Used in the rc scripts.
eudev A fork of udev for those that don't like systemd.
pcre The Perl Compatible Regular Expression library. Used by grep.
mpfr The multi-precision floating-point library. Used by awk.
gmp The GNU multi-precision library. Supports extended floating-point function across are large number of CPU architectures. Required by mpfr.
e2fsprogs Utlities for the EXT2/3/4 file systems, such as mkfs.ext4 and fsck.ext4.
kmod Collection of utilities to support dynamic loading of kernel modules, such as modprobe, insmod, lsmod and rmmod.
zlib A compression library, came out of the gzip.
gzip A patent free replacement for the compress utility.
xz A compression utility based on LZMA, which is also the basis the LZIP & 7-zip.
inetutils A collection of network based utilities. Includes syslogd so included to assist in debugging.
pciutils Provides details of what is connected to the PCI(e) bus.
procps Utilities for checking processes, such as ps and tree.
vim A vi editor clone.
skel-files A bespoke collection of configuration files, such as /etc/passwd, /etc/group and /etc/shadow.

9.1.1 Login

The login utilities are built with PAM (Pluggable Authentication Modules) support so Linux PAM is required. The package skel-files contains additional missing symbolic links and configuration files to allow root login (for example /etc/passwd and /etc/-

pam.d/others).

9.1.2 Example (with kernel loglevel=3)

The startup is minimal but quick.

```
INIT: version 2.90 booting
System initialization starting
fsck from util-linux 2.32.1
e2fsck 1.44.2 (14-May-2018)
/: clean, 7141/319488 files, 131236/1277184 blocks
Starting syslogd
INIT: Entering runlevel: 3
Entering runtime 3

(none) login:
Password:
-bash: warning: setlocale: LC_ALL: cannot change locale (en_GB.utf8): No such file or directory
(none) 16:40:18 ~ #
```

Init also handles shutdown and reboot.

```
(none) 16:41:28 ~ # halt

Broadcast message from root@(none) (ttyS0) (Tue Jul 31 16:41:31 2018):

The system is going down for system halt NOW!
INIT: Switching to runlevel: 0
INIT: added agetty on ttyS0 with id S0
INIT: Sending processes configured via /etc/inittab the TERM signal
Running shutdown script /etc/rc.d/rc.0:
Unmounting local file systems.
Remounting root filesystem read-only.
[  120.464656] reboot: Power down
```

10 /etc configuration files

10.1 /etc/fstab

10.2 /etc/hosts

10.3 /etc/machine-id

10.4 /etc/motd

10.5 /etc/issue

10.6 /etc/profile

10.7 /etc/ssh/

11 Networking

With IPv4 and IPv6 the world is divided into gatways and hosts. Gateways are generally have static addressing and dynamic routing, where as hosts often have dynamic addressing with static routing (routing is effective static within each DHCP scope). Cloud services such as Azure do not allow static addressing, hosts must be running DHCP.

11.1 IPv6 SLAAC

With IPv6 an additional mechanism is StateLess Address AutoConfiguration was created. This requires a gateway to send IMCPv6 Router Advertisements. Hosts can then create a local node address based their MAC address (or potentially with a RNG) (provided the link prefix <= 64). Limited DNS configuration can also be included in RAs but the linux kernel ignores this (userland daemons may act on this).

11.2 DHCP

DHCPv4 and DHCPv6 are similar but not identical. They use different port ranges and the options are more equivilant rather than equal. Because DHCPv6 does not support a default gateway option hosts require RAs to determine their default gateway.

11.3 Host networking

11.3.1 Netdevs

The core concept in Linux networking is the netdev (network device). All physical NICs are netdevs but virtual interfaces such as the loopback interface, bridges, bonds and sub interfaces are also netdevs. Some netdevs are can often be both L2 (ethernet) and L3 devices.

11.3.2 TAP/TUN

TAP/TUN [net] devices are virtual interfaces which connect the network stack to running processes. A TAP device is a virtual ethernet interface, so it has its own MAC address where as the TUN device is purely L3, carrying either IPv4 or IPv6.

QEMU uses TAP devices to attach virtual NICs within the guest to the host network stack.

11.3.3 Linux Bridge

The original linux bridge worked as a standard ethernet bridge (including support for Spanning Tree Protocol). A limitation was the bridge removed any outer VLAN tagging so if you wanted to have VMs on multiple VLANs you needed create VLAN sub interfaces on the physical interfaces, for example eth2.200, and then create a separate bridge for VLAN 200, add eth2.200 into it and then any VM TAP interfaces into that specific bridge. Only large systems this lead to a large number of netdevs.

One solution was to use OpenVSwitch (OVS). This is a openflow compatible virtual switch with supported VLANs natively. It allows for multiple briding domains within a single OVS instance.

More recently an extension to the Linux bridge code has allowed bridges to VLAN aware. This allows allows physical interfaces to be added into the bridge as trunk interfaces and TAP interfaces are access ports.

11.4 Host configuration

For this example OVS is used. A single bridge instance (called broadcom for historical reasons) with two internal VLAN interfaces.

```
bash-4.4# ovs-vsctl show
84c7a1c3-ce51-4ccf-a29e-db6685b12954
    Bridge broadcom
        Port broadcom
            Interface broadcom
                type: internal
        Port "vlan101"
            tag: 101
            Interface "vlan101"
                type: internal
        Port "vlan100"
            tag: 100
            Interface "vlan100"
                type: internal
    ovs_version: "2.9.2"
```

Internal OVS ports are netdevs so show up as interfaces. These have been allowed IPv6 addresses from a Hurricane Electric /48 allocation, which is routed via an 6over4 tunnel from HE.

```
11: vlan100: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether 16:e4:6e:12:64:7f brd ff:ff:ff:ff:ff:ff
    inet6 2001:470:688f:100::1/64 scope global
```

```
        valid_lft forever preferred_lft forever
    inet6 fe80::14e4:6eff:fe12:647f/64 scope link
        valid_lft forever preferred_lft forever

10: vlan101: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether 66:02:8c:2d:27:75 brd ff:ff:ff:ff:ff:ff
    inet6 2001:470:688f:101::1/64 scope global
        valid_lft forever preferred_lft forever
    inet6 fe80::6402:8cff:fe2d:2775/64 scope link
        valid_lft forever preferred_lft forever
```

11.4.1 QEMU

QEMU creates a TAP interface and then passes the name of the interface to a script. It does not allow any additional information so currently the ifup and ifdown scripts have static bridge and vlan settings.

The tap interface.

```
44: lfs: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master ovs-system state UNKNOWN mode DEFAULT group default qlen 1000
    link/ether 86:3e:be:ce:b1:26 brd ff:ff:ff:ff:ff:ff
```

Added to OVS.

```
bash-4.4# ovs-vsctl show
84c7a1c3-ce51-4ccf-a29e-db6685b12954
    Bridge broadcom
        Port lfs
            tag: 101
            Interface lfs
```

11.5 Guest

11.5.1 virtio-net

QEMU supports a network adapter similar to vmware's VMNET3.

```
(none) 16:10:56 ~ # lspci
00:00.0 Host bridge: Intel Corporation 82G33/G31/P35/P31 Express DRAM Controller
00:01.0 VGA compatible controller: Device 1234:1111 (rev 02)
00:02.0 SCSI storage controller: Red Hat, Inc. Virtio SCSI
00:03.0 Ethernet controller: Red Hat, Inc. Virtio network device
00:1f.0 ISA bridge: Intel Corporation 82801IB (ICH9) LPC Interface Controller (rev 02)
00:1f.2 SATA controller: Intel Corporation 82801IR/IO/IH (ICH9R/DO/DH) 6 port SATA Controller [AHCI mode] (rev 02)
00:1f.3 SMBus: Intel Corporation 82801I (ICH9 Family) SMBus Controller (rev 02)

(none) 16:17:47 ~ # lspci -v -s 0:0:3
00:03.0 Ethernet controller: Red Hat, Inc. Virtio network device
    Subsystem: Red Hat, Inc. Device 0001
    Flags: bus master, fast devsel, latency 0, IRQ 23
    I/O ports at 60a0 [size=32]
    Memory at 91001000 (32-bit, non-prefetchable) [size=4K]
    Memory at 800004000 (64-bit, prefetchable) [size=16K]
    Expansion ROM at 91040000 [disabled] [size=256K]
    Capabilities: [98] MSI-X: Enable+ Count=3 Masked-
    Capabilities: [84] Vendor Specific Information: VirtIO: <unknown>
    Capabilities: [70] Vendor Specific Information: VirtIO: Notify
    Capabilities: [60] Vendor Specific Information: VirtIO: DeviceCfg
    Capabilities: [50] Vendor Specific Information: VirtIO: ISR
    Capabilities: [40] Vendor Specific Information: VirtIO: CommonCfg
    Kernel driver in use: virtio-pci
```

Using lsmod we can the module is loaded.

```
(none) 16:19:53 ~ # lsmod
Module                  Size  Used by
vfat                   20480  1
fat                    69632  1 vfat
mousedev               24576  0
kvm_amd                94208  0
kvm                   622592  1 kvm_amd
irqbypass              16384  1 kvm
input_leds             16384  0
intel_agp              20480  0
psmouse               106496  0
intel_gtt              24576  1 intel_agp
atkbd                  28672  0
agpgart                36864  2 intel_agp,intel_gtt
virtio_net             45056  0
evdev                  20480  0
```

11.5.2 Network packages

iana-etc-2.30 /etc/services and /etc/protocols
libmnl-1.0.4 Netlink support library, used by iproute2.
libelf-0.8.13 Library to process ELF shared libraries and executables.
iproute2-4.17.0 Netlink based utilites for configuring the linux network stack.
dhcp-4.4.1 The ISC dhcpd server and dhclient software.
openssl-1.0.2o A SSL/TLS library and utility package.
openssh-7.6p1 An SSH client and server package.

11.5.3 ioctl vs netlink

The original method of configuring was using the ioctl system call. To configure at interface ifconfig was used, with route for routing information and arp to look at the neighbours.

In Linux this was been completely replaced by the netlink protocol, which is a general pub/sub system for userland processes to commicate with the kernel (and potentially each other). While the ioctl method still works the software using it had not been maintained is nearly a decade.

11.6 iproute2

The iproute2 package is the kitchen sink of linux networking. All the basic functionality is in a single program called ip. This will configure links (ip link), interfaces (ip addr) and routes (ip route). ARP is visible via ip neighbors. The netstat program is replaced by ss (socket statistics).

11.6.1 Manual configuration

Historically unix systems have been servers, normally based server rooms with hardwired networking. The networking therefore rarely changed and was normally static, so configuration could based on fixed settings, either directly in configuration files or via other configuration file, for example using the IP address of the hostname from /etc/hosts, along with /etc/netmaks to get the prefix.

With iproute2 it is possible to configure the network with using a single batch file. However other network settings such as DNS or NTP need to be specified in other files, /etc/resolv.conf and /etc/ntp.conf in these cases.

```
link set lo up
link set eth0 up
addr add 2001:470:688f:101::20/64 dev eth0
```

11.6.2 DHCP

There are multiple DHCP clients for linux. The ISC DHCP package includes dhclient, which works well with both the ISC DHCPD and their laster Kea DHCP server. A standalone DHCP client daemon called dhcpcd supports IPv4, DHCPv6 and IPv6 SLAAC.

Within the last few years full featured network configuration packages have developed. These often include their own local DNS resolvers and NTP clients. Some examples are Network Manager, connman, wicd, wicked and systemd-networkd.

12 Systemd

The sysvinit system has a number of weeknesses. It doesn't handle dependency tracking at all so all rc scripts are executed in series. Should a script hang then the whole boot process can grind to a halt. It also doesn't have any real service control, once a process daemonizes init effectively loses control of it.

There have been quite a few attempts to replace init (runit, initng, upstart) with limited success. The latest is systemd. Systemd is not universially liked, and it has become very large (some would say bloated). The converse is that almost no bespoke code is required to get a fully functioning system.

Systemd uses linux containers to manage services. Any child processes of a service cannot escape the container so systemd has full visibility of all the processes belonging to that service.

The cost of having everything init PID 1 is that it requires are considerable number of shared library dependencies.

1. libseccomp
2. cryptosetup
3. libgcrypt
4. iptables
5. libidn
6. lz4
7. LVM2
8. libgpg-error
9. argon2
10. json-c
11. libnl
12. libpcap
13. expat
14. dbus
15. pcre2
16. systemd

12.1 /etc/machine-id

Systemd requires every system to have a unique identify. This is based on UUID [Universally Unique Identifer].

```
dbus-uuidgen > /etc/machine-id
```

12.2 /etc/fstab

The fsck and remount of systemd require /etc/fstab to populated.

```
UUID=9255cd5a-5672-4c03-be82-ba81c7c01664       /       ext4    defaults        1 1
UUID=8d48024e-1b8c-4177-bf30-462c29aeedbb       /boot   ext4    defaults        1 1
UUID=3E35-C6FC  /boot/efi       vfat    defaults        1 1
```