How Linux Works

Disk and Filesystem

2 types of partition table.

- MBR and GPT (Globally Unique Identifier Partition Table)

Disk and Partition Geometry

Filesystems

The File System in User Space (FUSE) feature allows user-space filesystems in Linux. Like /sys and /proc.

A traditional Unix filesystem has two primary components: a pool of data blocks where you can store data and a database system that manages the data pool. The database is centered around the inode data structure. An **inode** is a set of data that describes a particular file, including its type, permissions, and—perhaps most importantly—where in the data pool the file data resides

Init

The init program is a user-space program Its main purpose is to start and stop the essential service processes on the system, but newer versions have more responsibilities.

System V is traditionally used for init for most linux distribution but now most linx distributions uses systemd.

systemd is goal oriented. You define a target that you want to achieve, along with its dependencies, and when you want to reach the target. systemd satisfies the dependencies and resolves the target. systemd can also defer the start of a service until it is absolutely needed.

- start some services only when needed
- systemd incorporates a number of standard Unix services such as **cron** and **inetd**.

It does not just operate processes and services; it can also mount filesystems, monitor network sockets, run timers, and more. Each type of capability is called a **unit type**, and each specific capability is called a **unit**.

- Service units. Control the traditional service daemons on a Unix system.
- Mount units. Control the attachment of filesystems to the system.
- Target units. Control other units, usually by grouping them.

To accommodate the need for flexibility and fault tolerance, systemd offers a myriad of dependency types and styles.

- **Requires** Strict dependencies. When activating a unit with a Requires dependency unit, systemd attempts to activate the dependency unit. If the dependency unit fails, systemd deactivates the dependent unit.

- **Wants**. Dependencies for activation only. Upon activating a unit, systemd activates the unit's Wants dependencies, but it doesn't care if those dependencies fail.

- **Requisite**. Units that must already be active. Before activating a unit with a Requisite dependency, systemd first checks the status of the dependency. If the dependency has not been activated, systemd fails on activation of the unit with the dependency.

- **Conflicts**. Negative dependencies. When activating a unit with a Conflict dependency, systemd automatically deactivates the dependency if it is active. Simultaneous activation of two conflicting units fails.

To activate units in a particular order, you can use the following dependency modifiers:

- **Before**. The current unit will activate before the listed unit(s). For example, if Before=bar.target appears in foo.target, systemd activates foo.target before bar.target.

- **After**. The current unit activates after the listed unit(s).

If a **conditional dependency** in a unit is false when systemd tries to activate the unit, the unit does not activate, though this applies only to the unit in which it appears.

- ConditionPathExists=p: True if the (file) path p exists in the system.
- ConditionPathIsDirectory=p: True if p is a directory.
- ConditionFileNotEmpty=p: True if p is a file and it's not zero-length.

systemd Configuration

globally configured, usually */usr/lib/systemd/system* and a system configuration directory usually */etc/systemd/system*.

Unit Files

section names in brackets ([]) and variable and value assignments (options) in each section.

```
[Unit]
Description=OpenSSH server daemon
After=syslog.target network.target auditd.service

[Service]
EnvironmentFile=/etc/sysconfig/sshd
ExecStartPre=/usr/sbin/sshd-keygen
ExecStart=/usr/sbin/sshd -D $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID

[Install]
WantedBy=multi-user.target
```

The [Install] section in the sshd.service unit file is used to activate systemd's **WantedBy** and **RequiredBy** dependency options.

When you enable a unit, systemd reads the [Install] section; in this case, enabling the sshd.service unit causes systemd to see the **WantedBy** dependency for multi-user.target. In response, systemd creates a symbolic link to sshd.service in the system configuration directory.

The [Install] section is usually responsible for the .wants and .requires directories in the system configuration directory (/etc/systemd/system).

You'll interact with systemd primarily through the **systemctl** command.

An Example Socket Unit and Service

echo.socket
[Unit]
Description=echo socket

[Socket]
ListenStream=22222
Accept=yes

echo@.service
[Unit]
Description=echo service

[Service]
ExecStart=-/bin/cat
StandardInput=socket

> systemctl start echo.socket
> telnet localhost 22222

System V

There are two major components to a typical System V init installation: a central configuration file and a large set of boot scripts augmented by a symbolic link farm. The configuration file **/etc/inittab** is where it all starts.

```
id:5:initdefault:
l5:5:wait:/etc/rc.d/rc 5
```

This small line triggers many other programs. In fact, rc stands for run commands. The 5 in this line tells us that we're talking about runlevel 5. The commands are probably either in /etc/rc.d/rc5.d or /etc/rc5.d.

Shutdown

1. init asks every process to shut down cleanly.
2. If a process doesn't respond after a while, init kills it, first trying a TERM signal.
3. If the TERM signal doesn't work, init uses the KILL signal on any stragglers.
4. The system locks system files into place and makes other preparations for shutdown.
5. The system unmounts all filesystems other than the root.
6. The system remounts the root filesystem read-only.
7. The system writes all buffered data out to the filesystem with the sync program.
8. The final step is to tell the kernel to reboot or stop with the reboot(2) system call. This can be done by init or an auxiliary program such as reboot, halt, or power off.

Initial RAM Filesystem

The problem stems from the availability of many different kinds of storage hardware. Storage controller drivers that distributions can't include all of them in their kernels, so many drivers are shipped as loadable modules.

The workaround is to gather a small collection of kernel driver modules along with a few other utilities into an archive. The boot loader loads this archive into memory before running

the kernel. Upon start, the kernel reads the contents of the archive into a temporary RAM filesystem (the initramfs), mounts it at /, and performs the user-mode handoff to the init on the initramfs. Then, the utilities included in the initramfs allow the kernel to load the necessary driver modules for the real root filesystem. Finally, the utilities mount the real root filesystem and start true init.

On distributions that use systemd, you'll typically see an entire systemd installation there with no unit configuration files and just a few udevd configuration files.

System Configuration

/etc
Most system configuration files on a Linux system are found in /etc.

What kind of configuration files are found in /etc? The basic guideline is that customizable configurations for a single machine, such as user information (/etc/passwd) and network details (/etc/network), go into /etc.

System Logging

Most Linux distributions run a new version of syslogd called rsyslogd that does much more than simply write log messages to files. The base rsyslogd configuration file is /etc/rsyslog.conf.

Network

Networking is the practice of connecting computers and sending data between them.

A computer transmits data over a network in small chunks called packets, which consist of two parts: a header and a payload.

Kernel Network Interfaces

The Linux kernel maintains its own division between the two layers(physical and the Internet layer) and provides communication standards for linking them called a (kernel) network interface.

NetworkManager is a daemon that the system starts upon boot. Its job is to listen to events from the system and users and to change the network configuration based on a bunch of rules.

Upon startup, NetworkManager gathers all available network device information, searches its list of connections, and then decides to try to activate one. Here's how it makes that decision for Ethernet interfaces:

1. If a wired connection is available, try to connect using it. Otherwise, try the wireless connections.
2. Scan the list of available wireless networks. If a network is available that you've previously connected to, NetworkManager will try it again.
3. If more than one previously connected wireless networks are available, select the most recently connected.

You also want to configure the localhost interface early in the boot process because basic system services often depend on it. Most distributions keep NetworkManager away from localhost.

The Transport Layer

There are several ways to automatically configure networks in Linux-based systems.

When using **TCP**, an application opens a connection (not to be confused with NetworkManager connections) between one port on its own machine and a port on a remote host.

The important thing to know about the ports is that the client picks a port on its side that isn't currently in use, but it nearly always connects to some well-known port on the server side.

TCP is popular as a transport layer protocol because it requires relatively little from the application side. An application process only needs to know how to open (or listen for), read

from, write to, and close a connection. To the application, it seems as if there are incoming and outgoing streams of data; the process is nearly as simple as working with a file.

UDP is a far simpler transport layer than TCP. It defines a transport only for single messages; there is no data stream. At the same time, unlike TCP, UDP won't correct for lost or out-of-order packets. In fact, although UDP has ports, it doesn't even have connections! One host simply sends a message from one of its ports to a port on a server, and the server sends something back if it wants to.

DHCP

When you set a network host to get its configuration automatically from the network, you're telling it to use the Dynamic Host Configuration Protocol (DHCP) to get an IP address, subnet mask, default gateway, and DNS servers.

Although there are many different kinds of network manager systems, nearly all use the Internet Software Consortium (ISC) ***dhclient*** program to do the actual work.

NAT

The basic idea behind NAT is that the router doesn't just move packets from one subnet to another; it transforms them as it moves them. Hosts on the Internet know how to connect to the router, but they know nothing about the private network behind it.

Linux distributions such as OpenWRT appeared for routers, with OpenWRT installed, the manufacturer and age of the hardware don't really matter. This is because you're using a truly open operating system on the router that doesn't care what hardware you use as long as your hardware is supported.

Firewalls

A firewall is a software and/or hardware configuration that usually sits on a router between the Internet and a smaller network, attempting to ensure that nothing "bad" from the Internet harms the smaller network.

In Linux, you create firewall rules in a series known as a *chain*. A set of chains makes up a *table*.

You'll normally work primarily with a single table named filter that controls basic packet flow. There are three basic chains in the filter table: INPUT for incoming packets, OUTPUT for outgoing packets, and FORWARD for routed packets.

Each firewall chain has a default policy that specifies what to do with a packet if no rule matches the packet. The policy for all three chains in this example is ACCEPT, meaning that the kernel allows the packet to pass through the packet-filtering system. The DROP policy tells the kernel to discard the packet. To set the policy on a chain, use iptables -P like this:

> # iptables -P FORWARD DROP

> # iptables -A INPUT -s 192.168.34.0/24 -p tcp --destination-port 25 -j DROP

There are two basic kinds of firewall scenarios: one for protecting individual machines (where you set rules in each machine's INPUT chain) and one for protecting a network of machines (where you set rules in the router's FORWARD chain). In both cases, you can't have serious security if you use a default policy of ACCEPT and continuously insert rules to drop packets from sources that start to send bad stuff. You must allow only the packets that you trust and deny everything else.

For example, say your machine has an SSH server on TCP port 22. There's no reason for any random host to initiate a connection to any other port on your machine, and you shouldn't give any such host a chance. To set that up, first set the INPUT chain policy to DROP:

> # iptables -P INPUT DROP
> # iptables -A INPUT -p icmp -j ACCEPT
> # iptables -A INPUT -s 127.0.0.1 -j ACCEPT
> # iptables -A INPUT -s my_addr -j ACCEPT

ARP

When a machine boots, its ARP cache is empty. So how do these MAC addresses get in the cache? It all starts when the machine wants to send a packet to another host. If a target IP address is not in an ARP cache, the following steps occur:

1. The origin host creates a special Ethernet frame containing an ARP request packet for the MAC address that corresponds to the target IP address.
2. The origin host broadcasts this frame to the entire physical network for the target's subnet.
3. If one of the other hosts on the subnet knows the correct MAC address, it creates a reply packet and frame containing the address and sends it back to the origin. Often, the host that replies is the target host and is simply replying with its own MAC address.
4. The origin host adds the IP-MAC address pair to the ARP cache and can proceed.

Wireless Ethernet

In principle, wireless Ethernet ("WiFi") networks aren't much different from wired networks. Much like any wired hardware, they have MAC addresses and use Ethernet frames to transmit and receive data, and as a result the Linux kernel can talk to a wireless network interface much as it would a wired network interface. Everything at the network layer and above is the same; the main differences are additional components in the physical layer such as frequencies, network IDs, security, and so on.

For most wireless security setups, Linux relies on a daemon called wpa_supplicant to manage both authentication and encryption for a wireless network interface.

Network Applications

These are some other common network servers that you might find running on your system:

- httpd, apache, apache2 Web servers
- sshd Secure shell daemon (see 10.3 Secure Shell (SSH))
- postfix, qmail, sendmail Mail servers
- cupsd Print server
- nfsd, mountd Network filesystem (file-sharing) daemons

- smbd, nmbd Windows file-sharing daemons (see Chapter 12)
- rpcbind Remote procedure call (RPC) portmap service daemon

Secure Shell (SSH)

One of the most common network service applications is the secure shell (SSH), the de facto standard for remote access to a Unix machine.

OpenSSH has three host key sets: one for protocol version 1 and two for protocol 2. Each set has a public key (with a .pub file extension) and a private key (with no extension). Do not let anyone see your private key, even on your own system, because if someone obtains it, you're at risk from intruders.

SSH version 1 has RSA keys only, and SSH version 2 has RSA and DSA keys. RSA and DSA are public key cryptography algorithms.

> # ssh-keygen -t rsa -N '' -f /etc/ssh/ssh_host_rsa_key
> # ssh-keygen -t dsa -N '' -f /etc/ssh/ssh_host_dsa_key

One traditional way to simplify the use of servers is with the inetd daemon, a kind of superserver designed to standardize network port access and interfaces between server programs and network ports. After you start inetd, it reads its configuration file and then listens on the network ports defined in that file. As new network connections come in, inetd attaches a newly started process to the connection.

A newer version of **inetd** called **xinetd** offers easier configuration and better access control, but xinetd itself is being phased out in favor of **systemd.**

netstat is a basic network service debugging tool that can display a number of transport and network layer statistics.

lsof can track open files, but it can also list the programs currently using or listening to ports.
> # lsof -i

tcpdump puts your network interface card into promiscuous mode and reports on every packet that crosses the wire.

Remote Procedure Call

RPC stands for remote procedure call, a system residing in the lower parts of the application layer. It's designed to make it easier for programmers to access network applications by leveraging the fact that programs call functions on remote programs (identified by program numbers) and the remote programs return a result code or message.

The server is called **rpcbind**, and it must be running on any machine that wants to use RPC services.

> $ rpcinfo -p localhost

Network Security

There are two important kinds of vulnerabilities to worry about: direct attacks and clear-text password sniffing.

Here are a few basic rules of thumb:

- Run as few services as possible.
- Block as much as possible with a firewall.
- Track the services that you offer to the Internet.
- Use "long-term support" distribution releases for servers.
- Don't give an account on your system to anyone who doesn't need one.
- Avoid installing dubious binary packages.

Sockets

When a process connects to a Unix domain socket, it behaves almost exactly like a network socket: It can listen for and accept connections on the socket, and you can even choose between different kinds of socket types to make it behave like TCP or UDP.

Developers like Unix domain sockets for IPC for two reasons. First, they allow developers the option to use special socket files in the filesystem to control access, so any process that

doesn’t have access to a socket file can’t use it. And because there’s no interaction with the network, it’s simpler and less prone to conventional network intrusion.

You can view a list of Unix domain sockets currently in use on your system with

```
> lsof -U
```