Linux

Information Security Inc.

Contents

Brief history of UNIX/Linux (1957-1969)

- 1957 - BESYS (Bell Operating System) computing environment originally implemented as a batch processing OS at Bell Labs.
- 1962-1964 – GCOS (General Comprehensive Operating System).
- 1964 - Multics ("Multiplexed Information and Computing Service”),
- 1969 - Space Travel, an early computer game that simulated travel in the solar system was written by Ken Thompson for the Multics system.
- 1969 - AT&T made the decision to withdraw Multics in favor of GECOS.
- 1969 - Ken Thompson ported Space Travel to Fortran (previously FORTRAN, derived from Formula Translating System), to run on a GECOS system.

Brief history of UNIX/Linux (1969-1974)

- 1969 - Ken Thompson and Dennis Ritchie ported Space Travel to assembly language. Thompson and Ritchie wrote underlying code that eventually grew into what a punning colleague called UNICS (Uniplexed Information and Computing Service)--an 'emasculated Multics'. Some consider Space Travel the first Unix application program.
- 1971 – The first edition of UNIX was released (11/02/1971).
- 1972 - Ken Thompson and later Dennis Ritchie wrote and maintained the B programming language for use in the Multics project. Richie then re-wrote B and called the new language C. UNIX was re-written in C in 1972.
- 1974 – Thompson went to UC Berkeley to teach for a year. Bill Joy arrived as a new graduate student and created vi.

Brief history of UNIX/Linux (1977-1988)

- 1977 - UC Berkeley computer science professor Bob Fabry & his students led by Bill Joy released Berkeley UNIX under the official moniker BSD (Berkeley Software Distribution).

- 1987 - MINIX, a Unix-like system intended for academic use, was released by Andrew S. Tanenbaum

- 1988 – NeXTSTEP is a Unix operating system based on the Mach kernel, plus source code from BSD.

Brief history of UNIX/Linux (1991)

- 1991 - in Helsinki, Linus Torvalds began a project that later became the Linux kernel. He wrote the program specifically for the hardware he was using and independent of an operating system because he wanted to use the functions of his new PC with an 80386 processor. Development was done on MINIX using the GNU C compiler. This is still the main choice for compiling Linux today. The code however, can be built with other compilers, such as the Intel C Compiler.

- As Torvalds wrote in his book Just for Fun, he eventually ended up writing an operating system kernel. On 25 August 1991, he announced this system in a Usenet posting to the newsgroup "comp.os.minix.”:

Brief history of UNIX/Linux (1993-)

The rest is History:

- 1993 – FreeBSD was introduced.
- 1993 – The Debian Project was officially founded.
- 1994 – RedHat is introduced.
- 2004 – CentOS initial release.
- 2004 – Ubuntu initial release.

Brief history of UNIX/Linux

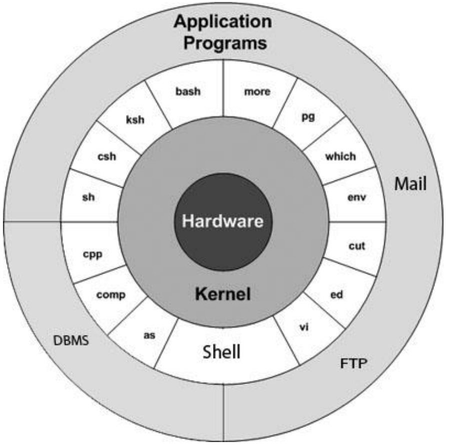
◎ Torvald's UseNet posting

"Hello everybody out there using minix –
I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).
I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)
PS. Yes – it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.
—Linus Torvalds "

Linux architecture

◎The Kernel

Primary function is to manage the computer's hardware and resources and allow other programs to run and use these resources.

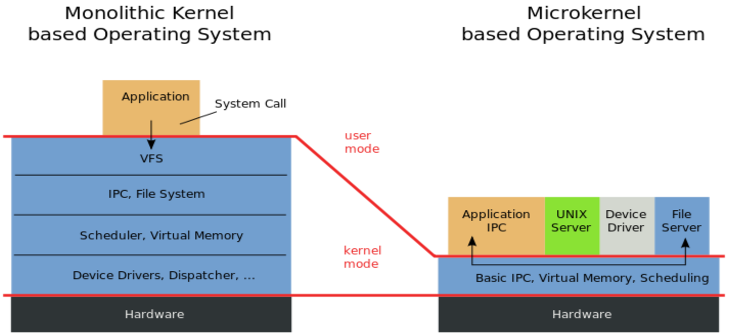


Linux architecture

◎ Types of kernels are:

- Monolithic kernel: Executes all the operating system instructions in the same address space to improve the performance of the system.

- Microkernel: Runs most of the operating system's background process in user space, to make the operating system more modular and, therefore, easier to maintain.



iSEC
information security inc.

Linux architecture

◎The daemon and Processes

- In multitasking computer operating systems, a daemon is a computer program that runs as a background process, rather than being under the direct control of an interactive user.
- In a Unix environment, the parent process of a daemon is often, but not always, the init process.
- UNIX Systems often start daemons at boot time and serve the function of responding to network requests, hardware activity, or other programs by performing some task. Daemons can also configure hardware (like udev), run scheduled tasks (like cron), and perform a variety of other tasks.
- When a process or daemon starts a file containing it's process ID (pid) is created. Usually in /var/run.
- Programs are stored on the disk. OS reads the file/program and load all/part of it in memory & CPU executes
- Process can be "running(R), stopped(T), sleeping(S), uninterruptible sleep – usually IO (D), defunct(Z) etc"

Linux architecture

◎Run levels

- "Runlevel" defines the state of the machine after boot.
- Conventionally, seven runlevels exist, numbered from zero to six.
- The BSD variants don't use the concept of run levels, although on some versions init provides an emulation of some of the common run levels.

Debian GNU/Linux runlevels[3]

ID	Description
S	Only run on boot (replaces `/etc/rc.boot`)
0	Halt
1	Single-user mode
2-5	Full Multi-user with console logins and display manager if installed
6	Reboot

Red Hat Linux/Fedora runlevels

Code	Information
0	Halt
1	Single-user text mode (without networking)
2	Not used (user-definable)
3	Full multi-user text mode
4	Not used (user-definable)
5	Full multi-user graphical mode (with an X-based login screen)
6	Reboot

Ubuntu runlevels[4]

Code	Information
0	Halt
1	Single-user mode
2	Graphical multi-user with networking
3-5	Unused but configured the same as runlevel 2
6	Reboot

Linux architecture

◎Cron

- Cron is a time-based job scheduler in a Unix-like OS.
- Cron is driven by a crontab (cron table) file, a configuration file that specifies shell commands to run periodically on a given schedule. Located in /etc/crontab, crontab is formatted as follows:

Entry	Description	Equivalent to
@yearly (or @annually)	Run once a year at midnight on the morning of January 1	0 0 1 1 *
@monthly	Run once a month at midnight on the morning of the first day of the month	0 0 1 * *
@weekly	Run once a week at midnight on Sunday morning	0 0 * * 0
@daily	Run once a day at midnight	0 0 * * *
@hourly	Run once an hour at the beginning of the hour	0 * * * *
@reboot	Run at startup	@reboot

```
# * * * * *  command to execute
# ┬ ┬ ┬ ┬ ┬
# │ │ │ │ │
# │ │ │ │ │
# │ │ │ │ └───── day of week (0 - 6) (0 to 6 are Sunday to Saturday, or use names; 7 is Sunday, the same as 0)
# │ │ │ └────────── month (1 - 12)
# │ │ └─────────────── day of month (1 - 31)
# │ └──────────────────── hour (0 - 23)
# └───────────────────────── min (0 - 59)
```

Linux architecture

◎File and directory permissions

Each file and directory has three permission groups:

- Owner
- Group
- All users

Each permission group has three permission types:

- Read
- Write
- Execute

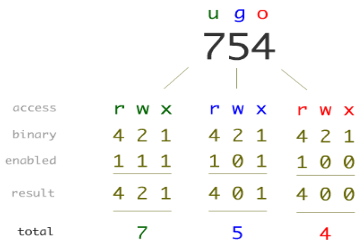
ls -l <file/dir> can be used to view permissions for file/dir

Eg. $ ls -l hello.py

-rwxrwxr-x 1 ash ash 228 May 19 16:24 hello.py

The following chart describes basic file/directory permissions (last 3 digits in octal notation):

Symbolic Notation	Octal Notation	English
----------	0000	no permissions
---x--x--x	0111	execute
--w--w--w-	0222	write
--wx-wx-wx	0333	write & execute
-r--r--r--	0444	read
-r-xr-xr-x	0555	read & execute
-rw-rw-rw-	0666	read & write
-rwxrwxrwx	0777	read, write, & execute



Linux architecture

◎File and directory permissions (contd…)

- Special file and directory permission bits include (first digit in octal notation):
- setuid bit(4) - allows you to specify which user a certain program is executed as.
- setgid bit(2) - allows you to enforce what group ownership a directory (and all it's subdirectories and files) have.
- sticky bit(1) (also known as the "Save Text Attribute" bit) - is set only on a directory and specifies that only the owner of a file can delete their own file within the directory regardless of the group or other's "writable" status.
- File types are depicted by preceding the permissions by one of these characters:
- - = Regular File
- d = Directory
- l = Symbolic Link
- b = Block Special Device
- c = Character Device
- s = Unix Socket (local domain socket)
- p = Named Pipe

Linux architecture

◎ File and directory permissions (contd…)

- 'umask' controls default file and directory permissions

- Read example ('1' in binary position = disabled):

```
$ umask
        0002
$ umask -S
           u=rwx,g=rwx,o=rx
```

- Set example:

```
$ umask u-x,g=r,o+w
```

How umask (in octal) translates to full (r, w, x) permissions.

Mask digit (octal)	Mask digit (binary)	Mask digit negated (binary)	Logical AND with "rwx" request[6]
0	000	111	rwx
1	001	110	rw-
2	010	101	r-x
3	011	100	r--
4	100	011	-wx
5	101	010	-w-
6	110	001	--x
7	111	000	---

Practical Linux

◎Bash shell (Bourne Again Shell)

- A Shell is a text interface to the OS
- Accessed through a ‘terminal’ application
- Takes in input as commands
- Bash is the shell used on most common distributions
- Prompt: Shell is ready to accept commands

```
adi@aadi-ubuntu:~$
root@adi-ubuntu:~#
```

- Common format: <username>@<machine>:<cwd>$ (# if root). This is not standard. Customizable by setting the PS1 environment variable.
- Shell environment variables can be declared in .bashrc or .profile
- Completion using the [Tab] key is supported in the bash shell, but not all shells.

Practical Linux

◎Binaries and Man (Manual) files

- Compiled programs are stored as ‘binary files’
- Binary file is a file that contains data as sequences of bits that do not represent plain text
- Binary files are used for images, sound files, executable files and compressed data
- Bin files execute very quickly
- Common linux utilities (cp, ls etc) are binary executables stored on disk
- Man file: Use by the ‘man’ utility to view documentation about UNIX commands and functions.

  Eg. `$ man ls`

  Display the manual page for the item (program) ls
- PATH variable used to tell shell which directories to search for executables

```
# echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
```

- To invoke a script or un-compiled program: `./script_or_program_name.ext`
  - Requirements: I. Must have execute bit set (`chmod +x`), II. script must include hash bang as the first line. (eg. `#!/usr/bin/python`)
  - Scripts may be run with interpreter, without the above requirements. eg. `# sh test.sh`

Practical Linux

◎Commands: ps

- Ps command used for viewing a snapshot of the processes running on the system. Provides detailed information for the current processes such as user, pid, cpu usage, memory, process name etc.

- ‘top’ for real time updated status

- ‘aux’ flags gives more useful info than basic ps invoke.

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2   4456  2264 ?        Ss   09:56   0:01 /sbin/init
root         2  0.0  0.0      0     0 ?        S    09:56   0:00 [kthreadd]
```

- Process can be in different states such as ‘D’ (uninterruptible sleep – usually IO), R(running), S(sleeping), Z(defunct), etc.

Practical Linux

◎Commands: netstat

- Command provides different network related information such as network sockets, interface information, routing table etc.

```
$ netstat -tunlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
```

- Network interface info (interface MTU can be found here)

```
$ netstat -i
Kernel Interface table
Iface   MTU Met   RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0       1500 0     73062      0      0 0         72013      0      0      0 BMRU
```

- netstat –r for routing information

Practical Linux

◎Commands: Tcpdump

- Utility to capture, write and read a pcap file (libpcap)

- Capture example (STOUT):

```
$ sudo tcpdump -n -i eth0 host 10.66.18.53
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
18:33:33.755369 ARP, Request who-has 10.66.18.53 tell 10.66.18.169, length 28
18:33:33.756022 ARP, Reply 10.66.18.53 is-at 00:90:0b:23:e8:74, length 46
18:33:33.756034 IP 10.66.18.169 > 10.66.18.53: ICMP echo request, id 32048, seq 1, length 64
18:33:33.756194 IP 10.66.18.53 > 10.66.18.169: ICMP echo reply, id 32048, seq 1, length 64
```

- Write to a file:

```
$ sudo tcpdump -n -i eth0 host 10.66.18.53 -w test.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
4 packets captured
4 packets received by filter
0 packets dropped by kernel
```

- Read from a file:

```
$ tcpdump -n -r test.pcap
reading from file test.pcap, link-type EN10MB (Ethernet)
18:39:11.057826 ARP, Request who-has 10.66.18.53 tell 10.66.18.169, length 28
18:39:11.058386 ARP, Reply 10.66.18.53 is-at 00:90:0b:23:e8:74, length 46
18:39:11.058395 IP 10.66.18.169 > 10.66.18.53: ICMP echo request, id 32071, seq 1, length 64
18:39:11.058588 IP 10.66.18.53 > 10.66.18.169: ICMP echo reply, id 32071, seq 1, length 64
```