

Operating Systems - Report on the Linux Systems

Samuel, Andersson, Johan Dahlberg, Eric Falheim,
Camilla Heiding, Xuan Hoang, Amer Hodzic

Today

Contents

1	Linux History	3
1.1	The making of Linux	3
1.2	The kernel	3
1.3	Linux licensing	3
2	Kernel Modules	4
3	Process Management	5
3.1	Process attributes	5
3.2	Signals	6
3.3	Foreground and background	6
4	Scheduling	7
4.1	User-mode scheduling	7
4.2	Kernel-mode scheduling	7
5	Memory Management	7
6	File Systems	7
6.1	Files	7
6.2	inode	8
6.3	Mounting	8
6.4	File System Hierarchy Structure	8
6.5	ext3 Filesystem and Journaling	9
7	Input and Output	9
7.1	STDIN, STDOUT and STDERR	9
7.2	Redirection and Pipes	9
8	Interprocess Communication	9
8.1	Shared memory	9
8.2	Pipes	9
8.3	Read-write to files	9
9	Network Communications	9
10	Security	9
11	Suggested topics to add	9

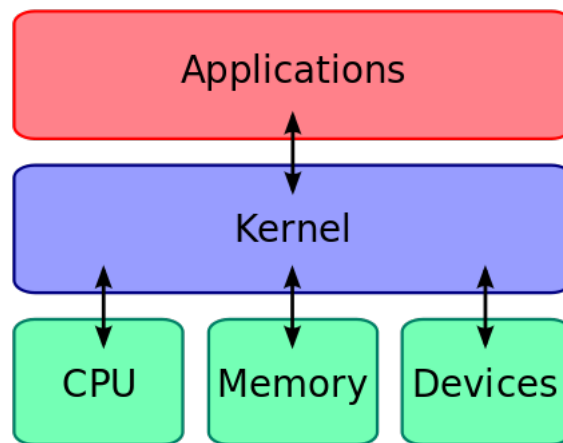
1 Linux History

1.1 The making of Linux

git gud - Linus Torvalds.

1.2 The kernel

The kernel is the core of the operating system and make communication possible to the hardware. It can exist many kernels in the same system. And in the case of a failure, when updating a kernel, you can always boot the system with an older version that was working.



Kernel interfacing hardware and user-space.

1.3 Linux licensing

The Linux kernel is distributed under the GNU General Public License (GPL). This means that anyone can download the kernel for free to use however they choose. If you modify the kernel and make a derivative of it, however, it must be distributed with the same licensing terms. So whenever anyone modifies the code and adds their own standard programs and tools in a packaged form, it could be called a new distribution of the system. This is the reason for the extremely many different "flavors" of Linux.

2 Kernel Modules

A kernel module is a driver that can be loaded into the kernel dynamically, at boot-time or run-time, to make communication between a device and hardware possible (via the kernel). This makes the kernel more lightweight due to the fact that unused modules can be unloaded from the kernel, which frees up RAM. Whenever a new module is loaded, there is no need to rebuild the kernel or reboot the system.

All modules can be found in the `/lib/modules/` folder. Working with modules is easy in Linux. There are commands you can run from the CLI to load, unload and list modules.

- `lsmod` - lists the loaded modules.
- `modprobe [-r] module_name` - will load or unload named module depending on the flag `-r`.
- `insmod module_name` - same as using `modprobe` without `-r` flag.
- `rmmod module_name` - same as using `modprobe` with `-r` flag.

3 Process Management

A process is a program in execution and it has its own address space if it's spawned with the `fork()`. Processes can also be cloned with parameters, effectively telling the child whether it should share the parent's filesystem, memory space, signal handlers and open files. Each process executes instructions sequentially, but can contain many threads. In Linux this is called a task. All threads share the same address space, and can therefore communicate via shared-memory.

3.1 Process attributes

A process is associated with a Process ID (PID) on creation, which will be unique for this process. Every process is spawned by a parent process (PPID) with a `fork`, and on creation the child-process memory space is identical to the parent's, but separate. Who spawned the process and what group they belonged to is stored in Real User ID (RUID), Effective User ID (EUID), Real Group ID (RGID) and Effective Group ID (EGID).

The process has its own priority which can be set from -20 (low) to 20 (high), by default it depends on recent CPU usage. In addition to this every process can be nice to other processes by hugging less resources. This value can be set between -20 to 19. The TTY attribute tells us what terminal the process is connected to.

1	[]	4.0%	Tasks: 101, 234 thr; 1 running								
2	[]	8.3%	Load average: 0.40 0.42 0.22								
3	[]	3.4%	Uptime: 00:05:25								
4	[]	9.3%									
Mem	[]	917M/3.85G									
Swp	[]	0K/701M									
PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1138	eric	20	0	3982M	303M	119M	S	13.2	7.7	0:24.21	/usr/bin/gnome-shell
977	eric	20	0	723M	131M	76444	S	5.3	3.3	0:07.36	/usr/lib/xorg/Xorg vt1 -displayfd 3 -auth /run/user/1000/gdm/Xauthorit
1153	eric	20	0	3982M	303M	119M	S	2.0	7.7	0:02.66	/usr/bin/gnome-shell
1155	eric	20	0	3982M	303M	119M	S	2.0	7.7	0:02.61	/usr/bin/gnome-shell
1532	eric	20	0	942M	72544	40880	S	2.0	1.8	0:03.39	/usr/bin/python3 /usr/bin/guake -p
1152	eric	20	0	3982M	303M	119M	S	2.0	7.7	0:02.86	/usr/bin/gnome-shell
1154	eric	20	0	3982M	303M	119M	S	2.0	7.7	0:02.68	/usr/bin/gnome-shell
987	eric	20	0	723M	131M	76444	S	1.3	3.3	0:01.97	/usr/lib/xorg/Xorg vt1 -displayfd 3 -auth /run/user/1000/gdm/Xauthorit
3033	eric	20	0	42984	5576	3896	R	0.0	0.1	0:00.14	htop
1	root	20	0	219M	9140	6832	S	0.0	0.2	0:01.24	/sbin/init splash
736	root	20	0	568M	18652	15652	S	0.0	0.5	0:00.12	/usr/sbin/NetworkManager --no-daemon
1141	eric	20	0	3982M	303M	119M	S	0.0	7.7	0:00.14	/usr/bin/gnome-shell
1283	eric	20	0	432M	23256	17940	S	0.0	0.6	0:00.17	/usr/lib/gnome-settings-daemon/gsd-wacom
1424	eric	20	0	217M	8592	7852	S	0.0	0.2	0:00.03	/usr/lib/ibus/ibus-engine-simple
1176	eric	20	0	370M	10068	8412	S	0.0	0.2	0:00.11	ibus-daemon --xim --panel disable
1174	eric	20	0	370M	10068	8412	S	0.0	0.2	0:00.20	ibus-daemon --xim --panel disable
1171	eric	20	0	3982M	303M	119M	S	0.0	7.7	0:00.01	/usr/bin/gnome-shell
311	root	19	-1	95048	15856	14904	S	0.0	0.4	0:00.23	/lib/systemd/systemd-journald
329	root	20	0	46644	4904	3124	S	0.0	0.1	0:00.28	/lib/systemd/systemd-udev
406	systemd-t	20	0	142M	3340	2784	S	0.0	0.1	0:00.00	/lib/systemd/systemd-timesyncd
403	systemd-t	20	0	142M	3340	2784	S	0.0	0.1	0:00.03	/lib/systemd/systemd-timesyncd
405	systemd-r	20	0	70620	5464	4904	S	0.0	0.1	0:00.03	/lib/systemd/systemd-resolved
836	root	20	0	175M	17684	9752	S	0.0	0.4	0:00.00	/usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-triggers
695	root	20	0	175M	17684	9752	S	0.0	0.4	0:00.09	/usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-triggers
712	root	20	0	107M	2032	1816	S	0.0	0.1	0:00.00	/usr/sbin/rqbalance --foreground

3.2 Signals

A signal in linux is telling the process that an external event has happened of some sort. The process can handle most signals in it's own way. If it doesn't, it will terminate.

The command `kill -signal_number` can be used to send a signal to a process. The most common signals to terminate a process are `SIGTERM(15)`, `SIGHUP(1)` and `SIGKILL(9)`, where the latter are more "aggressive" and considered more dirty ways to terminate a process.

```
eric@eric-VirtualBox:~$ kill -l
1) SIGHUP      2) SIGINT      3) SIGQUIT      4) SIGILL      5) SIGTRAP
6) SIGABRT     7) SIGBUS      8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT  17) SIGCHLD    18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN    22) SIGTTOU    23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF    28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS     34) SIGRTMIN   35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX
```

3.3 Foreground and background

A process that's spawned, for example by a shell, can either take over the thread of execution and thereby halting the process spawning it. This is said to be a process running in the foreground. You can also make a parent spawn a child process run in the background by appending the character `&` after the command. This will spawn the new process, but it will not take control over the shell spawning it.

```
eric@eric-VirtualBox:~$ jobs
eric@eric-VirtualBox:~$ sleep 1000&
[1] 3133
eric@eric-VirtualBox:~$ sleep 2000&
[2] 3134
eric@eric-VirtualBox:~$ sleep 3000&
[3] 3135
eric@eric-VirtualBox:~$ jobs
[1]   Running                  sleep 1000 &
[2]-  Running                  sleep 2000 &
[3]+  Running                  sleep 3000 &
eric@eric-VirtualBox:~$ fg %2
sleep 2000
^Z
[2]+  Stopped                  sleep 2000
eric@eric-VirtualBox:~$ jobs
[1]   Running                  sleep 1000 &
[2]+  Stopped                  sleep 2000
[3]-  Running                  sleep 3000 &
eric@eric-VirtualBox:~$ bg %2
[2]+  sleep 2000 &
eric@eric-VirtualBox:~$ jobs
[1]   Running                  sleep 1000 &
[2]-  Running                  sleep 2000 &
[3]+  Running                  sleep 3000 &
eric@eric-VirtualBox:~$ kill 3134
eric@eric-VirtualBox:~$ jobs
[1]   Running                  sleep 1000 &
[2]-  Terminated              sleep 2000
[3]+  Running                  sleep 3000 &
eric@eric-VirtualBox:~$ kill -9 3133
eric@eric-VirtualBox:~$ jobs
[1]-  Killed                    sleep 1000
[3]+  Running                  sleep 3000 &
eric@eric-VirtualBox:~$ fg %3
sleep 3000
^C
eric@eric-VirtualBox:~$
```

4 Scheduling

4.1 User-mode scheduling

Completely fair scheduler (CFS).

4.2 Kernel-mode scheduling

First come first served (FCFS) and Round Robin (RR).

5 Memory Management

6 File Systems

6.1 Files

Everything is stored as files on Linux. Even directories, devices, sockets, pipes and symbolic links. When showing information about a file you see something like "-rwxr-xr-x". The first character determines the file's type. For regular files, a '-' is shown, and for directories a 'd' is shown. The remainder of the string determines the user-group-universal access-rights. For the example above, the user got read-write-execute rights, whereas the group and everyone else can just read that file. This adds a level of protection to the system so only the processes and users with access-rights can read/modify/execute the file.

```
drwxr-xr-x 16 eric eric 4096 nov 27 18:22 ./
drwxr-xr-x  3 root root 4096 nov 27 16:31 ../
eric@eric-VirtualBox:/$ ll
total 108
drwxr-xr-x 24 root root 4096 nov 27 16:34 ./
drwxr-xr-x 24 root root 4096 nov 27 16:34 ../
drwxr-xr-x  2 root root 4096 nov 27 16:33 bin/
drwxr-xr-x  3 root root 4096 nov 27 16:34 boot/
drwxrwxr-x  2 root root 4096 nov 27 16:31 cdrom/
drwxr-xr-x 18 root root 4080 nov 27 18:21 dev/
drwxr-xr-x 121 root root 12288 nov 27 18:26 etc/
drwxr-xr-x  3 root root 4096 nov 27 16:31 home/
lrwxrwxrwx  1 root root  32 nov 27 16:34 initrd.img -> boot/initrd.img-5.0.0-36-generic
lrwxrwxrwx  1 root root  32 nov 27 16:30 initrd.img.old -> boot/initrd.img-5.0.0-23-generic
drwxr-xr-x 21 root root 4096 nov 27 16:32 lib/
drwxr-xr-x  2 root root 4096 aug  5 20:58 lib64/
drwx----- 2 root root 16384 nov 27 16:29 lost+found/
drwxr-xr-x  2 root root 4096 aug  5 20:58 media/
drwxr-xr-x  2 root root 4096 aug  5 20:58 mnt/
drwxr-xr-x  2 root root 4096 aug  5 20:58 opt/
dr-xr-xr-x 204 root root  0 nov 27 18:21 proc/
drwx----- 3 root root 4096 aug  5 21:09 root/
drwxr-xr-x 27 root root 780 nov 27 18:22 run/
drwxr-xr-x  2 root root 12288 nov 27 16:34 sbin/
drwxr-xr-x 11 root root 4096 nov 27 16:56 snap/
drwxr-xr-x  2 root root 4096 aug  5 20:58 srv/
dr-xr-xr-x 13 root root  0 nov 27 19:01 sys/
drwxrwxrwt 15 root root 4096 nov 27 19:00 tmp/
drwxr-xr-x 11 root root 4096 aug  5 21:03 usr/
drwxr-xr-x 14 root root 4096 aug  5 21:11 var/
lrwxrwxrwx  1 root root  29 nov 27 16:34 vmlinuz -> boot/vmlinuz-5.0.0-36-generic
lrwxrwxrwx  1 root root  29 nov 27 16:34 vmlinuz.old -> boot/vmlinuz-5.0.0-23-generic
```

6.2 inode

An inode in UNIX-like systems are data-structures for files and directories. It could be simplified to a node in a tree. It would contain information about itself, it's parent-node and a list of child-nodes.

6.3 Mounting

There are no disk-drives in Linux, instead all devices are mounted on mounting points in the file-system. So installing a new hard-disk or inserting a USB-drive is treated the same way. By mounting the device on an inode, you can treat the new device as a part of the whole file-system.

6.4 File System Hierarchy Structure

The filesystem begins at the root /, where directories that divide the system logically is placed. Some examples:

- /bin - Contains binaries for the system. Easily accessed by the system via environment variable \$PATH.
- /boot - Files for booting the system, including the kernel.
- /dev - Contains the system's devices.
- /etc - Configuration files for the system.
- /lib - Modules, software libraries and information databases.
- /mnt - Mounting point for external devices.
- /net - Mounting point for remote file-systems.
- /home - Home directory containing every user's own home folder on the system.
- /proc - Process file system
- /sbin - Binaries used by the administrator and the system.
- /usr -
- /tmp - A temporary directory that is emptied periodically, or when the system is shut down.

6.5 ext3 Filesystem and Journaling

ext3 stands for Third Extended Filesystem. Prior to ext3, there was ext2, which was considered a good filesystem with a few problems. If there was a power-outage and there was writings performed to the ext2 filesystem, data could be corrupted and a disk-recovery were being performed over everything in the filesystem. ext3 introduced a journal which held meta-data on the latest write operations. This made every write more resource expensive, but in the case of a power-outage, only the affected files would need to be considered for recovery. In the ext3 filesystem the maximum file-size is set to 2TB and the filesystem is capped at 32TB.

7 Input and Output

7.1 STDIN, STDOUT and STDERR

7.2 Redirection and Pipes

8 Interprocess Communication

8.1 Shared memory

8.2 Pipes

8.3 Read-write to files

9 Network Communications

10 Security

11 Suggested topics to add

- file descriptors
-