

Managing Processes Chapter 5.

Process

- An instance of a computer program being executed by single or multiple threads.
- Has address space (memory pages) and data structures inside the kernel.
 - Address space has code, libraries, variables, stacks, other info.
 - PCB- data structure in the kernel with info needed to manage scheduling of that process.

Thread

- An instance of execution.
- Has its own stack and CPU context, but runs inside the process address space.
- Can run simultaneously on multiple CPUs.

Process

- Has unique PID (process ID).
- Assigned by kernel as processes are created.
- Runs within a Linux namespace.
- In Linux parent clones itself to create a child with a `fork()` system call.
- Process' PPID is creating processes PID.

Process Lifecycle

- To create a new process, a process forks itself to create an identical copy of self.
- That copy has its own PID and accounting info.
- Child gets zero return value from `fork()` and knows it's a child.
- Parent gets child's PID from `fork()` and knows he is a parent.
- Child uses `exec()` to start executing routine.
 - Several variations of `exec` command
- At boot time several processes are created by kernel:
 - Systemd with PID = 1;
- Parent has to ack the death of child process with `wait()` for process to disappear.
- If parent dies before child – child's is adopted by system that then executes `wait()`.

Signals

- Process level interrupt requests.(about 30)
- Can serve as means of communication between processes.
- Can be sent to suspend, kill, interrupt a process with Ctrl-C and Ctrl-Z.
- Can be killed by an admin.
- Can be sent by kernel at division by zero.
- Kernel notifies child of parent death.

If signal was received:

1. Receiving process calls a designated handler for that signal (if it had one)
 - a. Specifying a handling routine is called catching a signal.
 - b. After signal is caught, execution starts from the point at which signal was received.
2. Kernel takes default action on behalf of the process (varies).
 - a. Many signals terminate the process, some produce core dumps – QUIT, BUS, SEGV (process's memory image, can be useful for debugging).

`kill -l` displays all signals

# ^b	Name	Description	Default	Can catch?	Can block?	Dump core?
1	HUP	Hangup	Terminate	Yes	Yes	No
2	INT	Interrupt	Terminate	Yes	Yes	No
3	QUIT	Quit	Terminate	Yes	Yes	Yes
9	KILL	Kill	Terminate	No	No	No
10	BUS	Bus error	Terminate	Yes	Yes	Yes
11	SEGV	Segmentation fault	Terminate	Yes	Yes	Yes
15	TERM	Software termination	Terminate	Yes	Yes	No
17	STOP	Stop	Stop	No	No	No
18	TSTP	Keyboard stop	Stop	Yes	Yes	No
19	CONT	Continue after stop	Ignore	Yes	No	No
28	WINCH	Window changed	Ignore	Yes	Yes	No
30	USR1	User-defined #1	Terminate	Yes	Yes	No
31	USR2	User-defined #2	Terminate	Yes	Yes	No

- a. A list of signal names and numbers is also available from the **bash** built-in command `kill -l`.
b. May vary on some systems. See `/usr/include/signal.h` or `man signal` for more information.

Img src: course textbook, p. 95.

Please see description of signals on page 96 of the textbook.

Killing Processes

- Kill can send any signal, but usually TERM.
- Users can kill their process, admin can kill all.
 - **kill [-signal] pid**, where signal can be number or signal name
- TERM can be caught, blocked, or ignored
- KILL cannot be caught, so the command below kills the process
kill -9 pid
- Can kill process by name:
sudo killall httpd
- Can kill user's processes:

sudo pkill -u johnny

Monitoring Processes

- Use command **ps** for a snapshot of the system
- Many versions and flags exist from many lineages
- See processes running on the system:
 - **ps aux**
 - **ps lax**
- Determine pid of a process
 - **pidof /usr/bin/sshd**
 - **pgrep sshd**
- Real-time monitoring (1-2 sec updates)
 - **top**

Monitor processws

top – monitor resource use real time

htop – similar to top, but nicer utility that allows to sort processes by some attribute.

pstree – displays in a tree format

pgrep – returns matching PID

`pgrep ssh`

renice 19 pid – changes process priority to 19 (-20 to 19).

It is not Good to be Nice... If you are a process

- Scheduling priority determines how much CPU time process gets.
- Niceness – how nice you are planning to be to others on the system based on history.
- > niceness = low priority (you gonna be nice :)
- < niceness = high priority (you not gonna be nice :)
- Range is -20 to +19
- Default is 0
- Child inherits niceness from parent
- Owner can increase niceness, but not lower it.
- Low-running parents cannot have high running children.
- Root can set values arbitrarily.
- Set niceness at time of creation with **nice**, adjust later with **renice**.
nice -n 5 ~/bin/longtask (lowers by 5)

sudo renice -5 8829 (sets to -5)
sudo renice 5 -u johnny (setts to 5)

- Always use /usr/bin/nice

The /proc filesystem

- /proc – pseudo filesystem that contains info about the system state
- Info and stats generated by the kernel
- **ps** and **top** read info from /proc directory
- /proc/1 contains info about process with PID 1...
- Directories are created for every running process, files contain info about running process.
- Detailed info can be found here: <https://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html>

Runaway Processes

- Processes that consume more resources than expected – CPU time, disk, or nw.
- Interferes with operation of other processes and the system
- Need to be investigated.
- Use **ps**, **top** and **uptime** and check averages over 1, 5, and 15 min intervals.
- Use **df -h** to examine disk use by the mounted filesystem according to disk totals
du -h sums the sizes of all files in a directory

Zombies

- Zombie or a defunct process - process that has been completed, but its entry still remains in the process table due to lack of correspondence between the parent and child processes.
- All resources are deallocated, but entry in process table remains.
- Usually, a parent process keeps a check on the status of its child processes through the wait() function.
- Forgotten wait() can cause zombies, or a bug in the program.
- The **top** command displays a detailed view of the processes running on your system along with the memory and CPU resources they are using.
- Above the process list it will give stats about the tasks currently running:
Tasks: 264 total, 2 running, 189 sleeping, 0 stopped, 1 zombie.
- On command line ask for state, parentID, the process ID, the program that is running:
\$ ps axo stat,ppid,pid,comm | grep -w defunct
z+ 4256 4257 zombie <defunct>
- Zombies are dead and completed processes that do not use memory or CPU resources, BUT
- They still hurt the performance:
 - Use unique process ID assigned to them which comes from a limited pool of PIDs.
 - New processes may not be able to run because of the lack of available PIDs.

- To kill a zombie : signal a completed parent to run the wait() so that completed children can be called back with a SIGCHLD signal.
- OR: manually kill the parent that will kill all its children:

```
kill -s SIGCHLD PPID    or kill -9 PPID
```

Orphans

- An orphan process is a computer process whose parent process has finished or terminated, though it remains running itself.
- Any orphaned Linux process will be immediately adopted by the systemd process.
- This is called re-parenting and occurs automatically.
- Even though technically the orphan process has system as its *parent*, it is still called an orphan process since the process that originally created it no longer exists.
- Can happen unintentionally (parent process crashed), or intentionally (detached from user's session to run in the background).
- Created with SIGHUP.

Daemons

- Intentional orphans running in the background.
- To create: fork() and immediately exit, causing system to adopt the child.

Process with PID 0

- Starts first when system boots.
- Forks a child with PID 1, then becomes a swapper (aka "idle" process).

Process with PID 1

- adopts orphaned child processes.
- reaps zombie processes.
- handles signals for child processes.

Periodic processes

- Scheduled backups, db maintenance, night batch jobs.
- Daemon *cron* runs commands on a schedule.
- Destructive Greek God of time (Kronos) and the main Titan.
- Software utility, time-based job scheduler.
 - Starts at boot time and runs while system is up.
 - Runs periodically at fixed times, days, weeks, months, or intervals.
 - Uses – patches, updates, backups, motd, logwatch, logrotate, rootkit hunter.
 - Reads configuration files with lists of commands and times to be invoked
 - Command lines are executed with **sh**, but can be changed.
 - cron config file is called crontab (cron table)
 - Checks **/var/spool/cron**, **/etc/cron.d** and **/etc/anacron** for tasks

- To edit and restart the cron use `crontab -e`
- Each user can have a cron file, do not exit by default.
- Have to type all the code in there.
- Admin can limit who can create cron jobs: `/etc/cron.allow`
- Plain text files, named with user login name.
- cron uses user's UID to run commands
- saves log file
- there is a system wise crontab in `/etc/crontab` for sysadmins to change by hand
- directories **cron.daily**, **cron.monthly**, etc. contain scripts
- **/etc/cron.d** for sw to install crontab entries
-
- Anacron – like cron, but runs skipped jobs.
- systemd timers
 - Activate some systemd service on a predefined schedule
 - Timers has two files – schedule and unit to activate and details of what to run
 - Timers expressed in calendar terms and relative terms.
 - List timers: **systemctl--all list-timers**

Software Installation and Management Chapter 6

Sysadmin's job includes many tasks.

- Automating mass installation of Oss.
- Maintain custom configs.
- Patching and updating.
- Tracking licenses.
- Maintaining add-on software.

Localization – configuring off-the-shelf software package to conform to your local needs

System Installation

- Media (10 machines or less)
- Network install
- Imaging the subnet

Package Management

- Debian and Ubuntu low-level package management format is **.deb**.

- **dpkg** is tool that installs, uninstalls and queries packages.
 -- install, -- remove, -l
- To check if package installed:
 dpkg -l | grep *package*
- High level package management: **apt**
 - Can upgrade the entire system with software with one apt command
 - Is a wrapper for lower level commands apt-get and apt-cache
 - **/etc/apt/sources.list** tells apt where to get new software
 - “universe” component – larger world of open source Linux sw
 - “multiverse” – non-OS content, such as VMware tools, etc.