

Process control

CIS 2230 Linux System Administration

Lecture 14

Steve Ruegsegger

Review

- What do these permissions represent for files or dir? r, w, x
- What are the 3 *types* of users which can get their own permissions?
- What do these mean? What are the octals?
 - `rw-rw-r--`
 - `rwX-r--r--`
- What do these do?
 - `$ chmod g+X *.sh`
 - `$ chmod w=r *.lab`
- What is the command to change a user owner? Group owner?
- Describe what the sticky bit does? When would you use it?
- What does the umask do?

What is a Process?

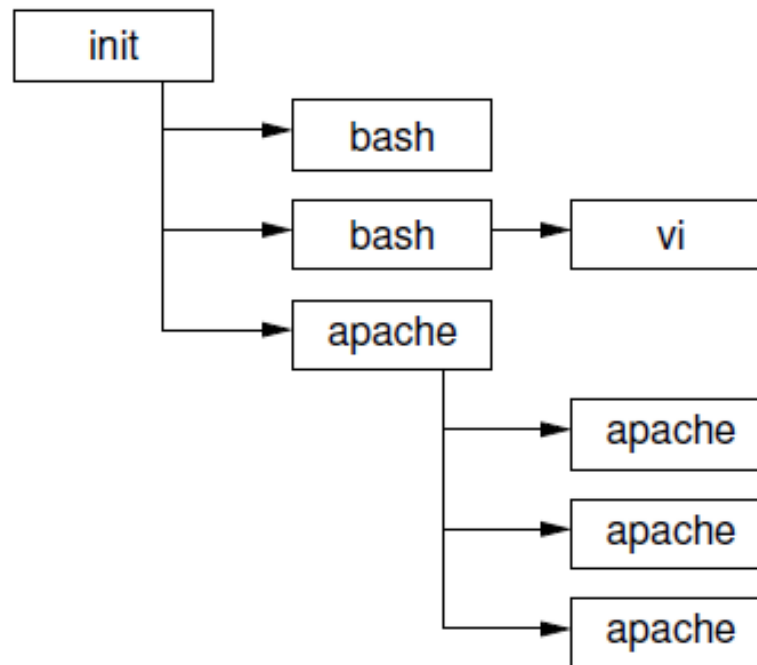
- The kernel considers each program running on your system to be a process
- A process lives as it executes, with a lifetime that may be short or long
- A process is said to 'die' when it terminates
- The kernel identifies each process by a number known as a process ID, or **pid**
- The kernel keeps track of various properties of each process

Process Properties

- A process has a user ID (**uid**) and a group ID (**gid**), which together specify what permissions it has
- A process has a parent process ID (**ppid**) — the pid of the process that created it
 - The kernel starts an `init` process with pid 1 at boot-up
 - Every other process is a descendant of pid 1
- Each process has its own working directory, initially inherited from its parent process.
- There is an environment for each process — a collection of named environment variables and their associated values.
 - A process's environment is usually inherited from its parent process

Parent and Child Processes

- The `init` process is the ancestor of all other processes
- Note: Ubuntu has used `systemd` rather than `init` for a couple of years.



(Apache starts many child processes so that they can serve HTTP requests at the same time)

Job control

- Most shells offer job control: The ability to stop, restart, and background a running process
- The shell lets you put `&` on the end of a command line to start it in the background
- Or you can hit **Ctrl+Z** to suspend a running foreground job
- Suspended and backgrounded jobs are given “job numbers” by the shell
- These “job numbers” can be given to the shell’s job control built-in commands
- Job control commands include `jobs`, `fg`, and `bg`
- `jobs` command displays jobs in the current shell

`fg` and `bg`

- **`fg`:**
 - Brings a background job into the foreground
 - Re-starts a suspended job, running it in the foreground
 - `fg %1` will foreground job number 1
 - `fg` with no arguments will operate on the current job
- **`bg`:**
 - Re-starts a suspended job, running it in the background
 - `bg %1` will background job number 1
 - `bg` with no arguments will operate on the current job
 - For example, after running `gedit` and suspending it with `Ctrl+Z`, use `bg` to start it running again in the background

examples

```
steve-oneiric@oneiric-w500:~$ emacs  
^Z
```

```
[1]+  Stopped                  emacs  
steve-oneiric@oneiric-w500:~$ bg  
[1]+  emacs &
```

```
steve-oneiric@oneiric-w500:~$ gedit  
^Z  
[2]+  Stopped                  gedit
```

```
steve-oneiric@oneiric-w500:~$ jobs  
[1]-  Running                  emacs &  
[2]+  Stopped                  gedit  
steve-oneiric@oneiric-w500:~$ bg %2  
[2]+  gedit &
```

```
steve-oneiric@oneiric-w500:~$ jobs  
[1]-  Running                  emacs &  
[2]+  Running                  gedit &  
steve-oneiric@oneiric-w500:~$
```

Process Monitoring: `ps`

- The `ps` command gives a snapshot of the processes running on a system at a given moment in time
- Defaults are limited
 - Normally shows a fairly brief summary of each process
 - Normally shows only processes that are both owned by the current user and attached to the current shell terminal
- 3 'sets' of options:
 - `ps` in Linux uses a mixture of options with one of three syntaxes:
 - 1) Traditional BSD `ps`: a single letter with no hyphen
 - 2) Unix98 `ps`: a single letter preceded by a hyphen
 - 3) GNU: a word or phrase preceded by two hyphens

Ubuntu \$ man ps

- Ubuntu decides to give all 3 syntaxes

PS(1) Linux User's Manual PS(1)

NAME

ps - report a snapshot of the current processes.

SYNOPSIS

ps [options]

DESCRIPTION

ps displays information about a selection of the active processes. If you want a repetitive update of the selection and the displayed information, use top(1) instead.

This version of ps accepts several kinds of options:

- 1 UNIX options, which may be grouped and must be preceded by a dash.
- 2 BSD options, which may be grouped and must not be used with a dash.
- 3 GNU long options, which are preceded by two dashes.

Options of different types may be freely mixed, but conflicts can appear. There are some synonymous options, which are functionally identical, due to the many standards and ps implementations that this ps is compatible with.

ps Options

- `ps` has *many* options
- Some of the most commonly used are:

Option	Description
<code>a</code>	Show processes owned by other users
<code>f</code>	Display process ancestors in a tree-like format
<code>u</code>	Use the 'user' output format, showing usernames and process start times
<code>w</code>	Use a wider output format. Normally, each line of output is truncated; each use of the <code>w</code> option makes the 'window' wider
<code>x</code>	Include processes which have no controlling terminal
<code>-e</code>	Show information on all processes
<code>-l</code>	Use a 'long' output format
<code>-f</code>	Use a 'full' output format
<code>-C cmd</code>	Show only processes named <code>cmd</code>
<code>-U user</code>	Show only processes owned by <code>user</code>

System V

BSD

ps Unix (sys V) format

- `ps` defaults to displaying only the processes with the same user ID as the current user and associated with the same terminal as the invoker.
- I often use `ps -ef` or `ps -elf`

`-f` = full format

`-F` = 'really full' format

`-l` = long format

`-e` = everyone



```
$ ps -ef | grep steve
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
steve	1832	1	0	14:40	?	00:00:00	/usr/lib/gvfs//gvfs-
fuse-							
steve	2769	1	0	14:45	?	00:00:00	/bin/sh
/usr/bin/libreoff							
steve	2770	2769	0	14:45	?	00:00:00	
/usr/lib/libreoffice/prog							
steve	2783	2770	3	14:45	?	00:04:51	
/usr/lib/libreoffice/prog							

ps BSD format

- Alternatively, we can use BSD format `ps aux`

```
steve-oneiric@oneiric-w500:~$ ps aux | grep steve
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
steve	1832	0.0	0.0	31588	2296	?	Ss1	14:40	0:00	/usr/lib/gvfs//
steve	2769	0.0	0.0	2040	512	?	S	14:45	0:00	/bin/sh /usr/bi
steve	2770	0.0	0.0	33432	3040	?	S1	14:45	0:00	/usr/lib/libreo
steve	2783	3.3	3.1	403864	125060	?	S1	14:45	5:57	/usr/lib/libreo
steve	7733	90.0	0.0	2040	512	pts/3	R	19:39	0:16	/bin/sh ./forever

userid	proc id	%cpu	mem util %	virt. mem size	non- swap phys Mem	tty	state (running, multithreaded sleeping)	start time	CPU time	command
--------	---------	------	---------------	----------------------	-----------------------------	-----	--	---------------	-------------	---------

(Resident set size)
* VSZ includes RSS



Know what these cols mean!

Which one is taking the most CPU?
Which one is taking the most memory?

Here's a trick for grepping `ps -elf` and getting the header

- If I use `ps -elf` a lot, which I do, and `grep` for keywords, which I do, then I don't get the header. :(
- Here's a trick using an alias

1) Edit or append to `.bash_aliases`

```
$ cat >> .bash_aliases  
alias psef='ps -elf | head -1 && ps -elf'
```

1) Source

```
$ . .bash_aliases
```

1) Use

```
$ psef | grep steve
```

Process Display: `ps tree`

- Displays a snapshot of running processes
- Always uses a tree-like display (a little like `ps -f`)
 - But by default, it shows only the name of each command
- Normally shows all processes
 - Specify a pid as an argument to show a specific process and its descendants.
 - Specify a username as an argument to show process trees owned by that user

Process Monitoring: `top`

- `$ top` shows full-screen, **continuously-updated** snapshots of process activity
 - Waits a short period of time between each snapshot to give the illusion of real-time monitoring
 - It's like running a sorted `ps` every sec.
- Processes are displayed in descending order of how much processor time they're using
- Also displays system uptime, load average, CPU status, and memory information
- **Format:** `$ top -d[delay] -n [iterations] [process]`
 - `$ top -d1 -n5`
- Usually, I just type `$ top` until I hit 'q' or ^C to quit.

Top cols

CPU % per state: User, system, nice, idle, IOwait, hardware IRQ, software interrupt, steal time

4Gb
6Gb

```

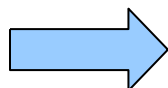
steve-oneiric@oneiric-w500: ~
top - 14:30:48 up 3:49, 3 users, load average: 0.74, 0.37, 0.17
Tasks: 192 total, 1 running, 191 sleeping, 0 stopped, 0 zombie
Cpu(s): 8.6%us, 53.0%sy, 0.0%ni, 18.3%id, 19.9%wa, 0.0%hi, 0.2%si, 0.0%st
Mem: 4014852k total, 3805832k used, 209020k free, 30752k buffers
Swap: 6291640k total, 0k used, 6291640k free, 1997200k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
  4628 steve-on   20   0 1463m 1.1g 1.1g  S   102  29.1   2:36.17 VirtualBox
  4759 steve-on   20   0  162m  12m 9780  S    6   0.3    0:00.18 gnome-screensho
 1794 steve-on   20   0  433m  91m  23m  S    5   2.3    4:07.56 compiz
 1186 root       20   0  193m  36m  17m  S    4   0.9    4:54.84 Xorg
 1758 steve-on   20   0   6912 3204  916  S    2   0.1    0:20.68 dbus-daemon
 2045 steve-on   20   0 36204  15m 7296  S    1   0.4    0:00.98 zeitgeist-daemo
 3962 steve-on   20   0  134m  62m  27m  S    1   1.6    1:10.78 acroread
 2039 steve-on   20   0 51208 5180 4152  S    0   0.1    0:01.09 zeitgeist-datah
 2067 steve-on   20   0  144m   9m 7064  S    0   0.3    0:01.55 bamfdaemon
 2071 steve-on   20   0  182m  26m  10m  S    0   0.7    0:25.91 unity-panel-ser
 2121 steve-on   20   0 37276  16m 9276  S    0   0.4    0:01.38 applet.py
 2177 steve-on   20   0 52588 4044 3428  S    0   0.1    0:00.12 unity-music-dae
 3854 steve-on   20   0  196m  21m  12m  S    0   0.6    0:08.57 gnome-terminal
 4492 root       20   0     0     0     0  S    0   0.0    0:00.30 kworker/0:0
 4734 steve-on   20   0  256m  17m  11m  S    0   0.4    0:00.76 gnome-screensho
     1 root       20   0  3336 1896 1264  S    0   0.0    0:00.71 init
     2 root       20   0     0     0     0  S    0   0.0    0:00.00 kthreadd

```

CPU utilization

- us = User = working on user programs (good)
- sy = System = overhead (ok)
- ni = Nice = programs which are deliberately being 'nice' (good)
- id = Idle = CPUs waiting to be asked to do something (good)
- hi/si = interrupts (usually very low)
- st = stolen CPU time by virtual machines
- wa = IOWait = CPUs need something from the Hard Drive (bad)



```

steve-oneiric@oneiric-w500: ~
top - 14:30:48 up 3:49, 3 users, load average: 0.74, 0.37, 0.17
Tasks: 192 total, 1 running, 191 sleeping, 0 stopped, 0 zombie
Cpu(s): 8.6%us, 53.0%sy, 0.0%ni, 18.3%id, 19.9%wa, 0.0%hi, 0.2%si, 0.0%st
Mem: 4014852k total, 3805832k used, 209020k free, 30752k buffers
Swap: 6291640k total, 0k used, 6291640k free, 1997200k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
  4628 steve-on   20   0 1463m 1.1g 1.1g  S   102  29.1   2:36.17 VirtualBox
  4759 steve-on   20   0  162m  12m 9780  S    6   0.3    0:00.18 gnome-screensho
  1794 steve-on   20   0  433m  91m  23m  S    5   2.3    4:07.56 compiz
  1186 root        20   0  193m  36m  17m  S    4   0.9    4:54.84 Xorg
  1758 steve-on   20   0   6912 3204  916  S    2   0.1    0:20.68 dbus-daemon
  2045 steve-on   20   0 36204  15m 7296  S    1   0.4    0:00.98 zeitgeist-daemo
  3962 steve-on   20   0   134m  62m  27m  S    1   1.6    1:10.78 acroread
  2039 steve-on   20   0 51208 5180 4152  S    0   0.1    0:01.09 zeitgeist-datah

```

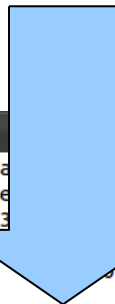


Know what these CPU states mean!

Cols from bottom rows

Process id, user, priority (to run next), nice, virtual memory, status, %cpu, %mem, cpu_time, command

Sorted by descending
%CPU



```

steve-oneiric@oneiric-w500: ~
top - 14:30:48 up 3:49, 3 users, load average: 0.74, 0.37, 0.17
Tasks: 192 total, 1 running, 191 sleeping, 0 stopped, 0 zombie
Cpu(s): 8.6%us, 53.0%sy, 0.0%ni, 18.3%id, 0.0%wa, 0.0%hi, 0.2%si, 0.0%st
Mem: 4014852k total, 3805832k used, 1209020k free, 30752k buffers
Swap: 6291640k total, 0k used, 6291640k free, 1997200k cached
  
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
4628	steve-on	20	0	1463m	1.1g	1.1g	S	102	29.1	2:36.17	VirtualBox
4759	steve-on	20	0	162m	12m	9780	S	6	0.3	0:00.18	gnome-screensho
1794	steve-on	20	0	433m	91m	23m	S	5	2.3	4:07.56	compiz
1186	root	20	0	193m	36m	17m	S	4	0.9	4:54.84	Xorg
1758	steve-on	20	0	6912	3204	916	S	2	0.1	0:20.68	dbus-daemon
2045	steve-on	20	0	36204	15m	7296	S	1	0.4	0:00.98	zeitgeist-daemo
3962	steve-on	20	0	134m	62m	27m	S	1	1.6	1:10.78	acoread
2039	steve-on	20	0	51208	5180	4152	S	0	0.1	0:01.09	zeitgeist-datah
2067	steve-on	20	0	144m	9m	7064	S	0	0.3	0:01.55	bamfdaemon
2071	steve-on	20	0	182m	26m	10m	S	0	0.7	0:25.91	unity-panel-ser
2121	steve-on	20	0	37276	16m	9276	S	0	0.4	0:01.38	applet.py
2177	steve-on	20	0	52588	4044	3428	S	0	0.1	0:00.12	unity-music-dae
3854	steve-on	20	0	196m	21m	12m	S	0	0.6	0:08.57	gnome-terminal
4492	root	20	0	0	0	0	S	0	0.0	0:00.30	kworker/0:0
4734	steve-on	20	0	256m	17m	11m	S	0	0.4	0:00.76	gnome-screensho
1	root	20	0	3336	1896	1264	S	0	0.0	0:00.71	init
2	root	20	0	0	0	0	S	0	0.0	0:00.00	kthreadd

PID: A process's process ID number.

USER: The process's owner.

PR: The process's priority. The lower the number, the higher the priority.

NI: The nice value of the process, which affects its priority.

VIRT: How much virtual memory the process is using.

RES: How much physical RAM the process is using, measured in kilobytes.

SHR: How much shared memory the process is using.

S: The current status of the process (zombied, sleeping, running, uninterruptedly sleeping, or traced).

%CPU: The percentage of the processor time used by the process.

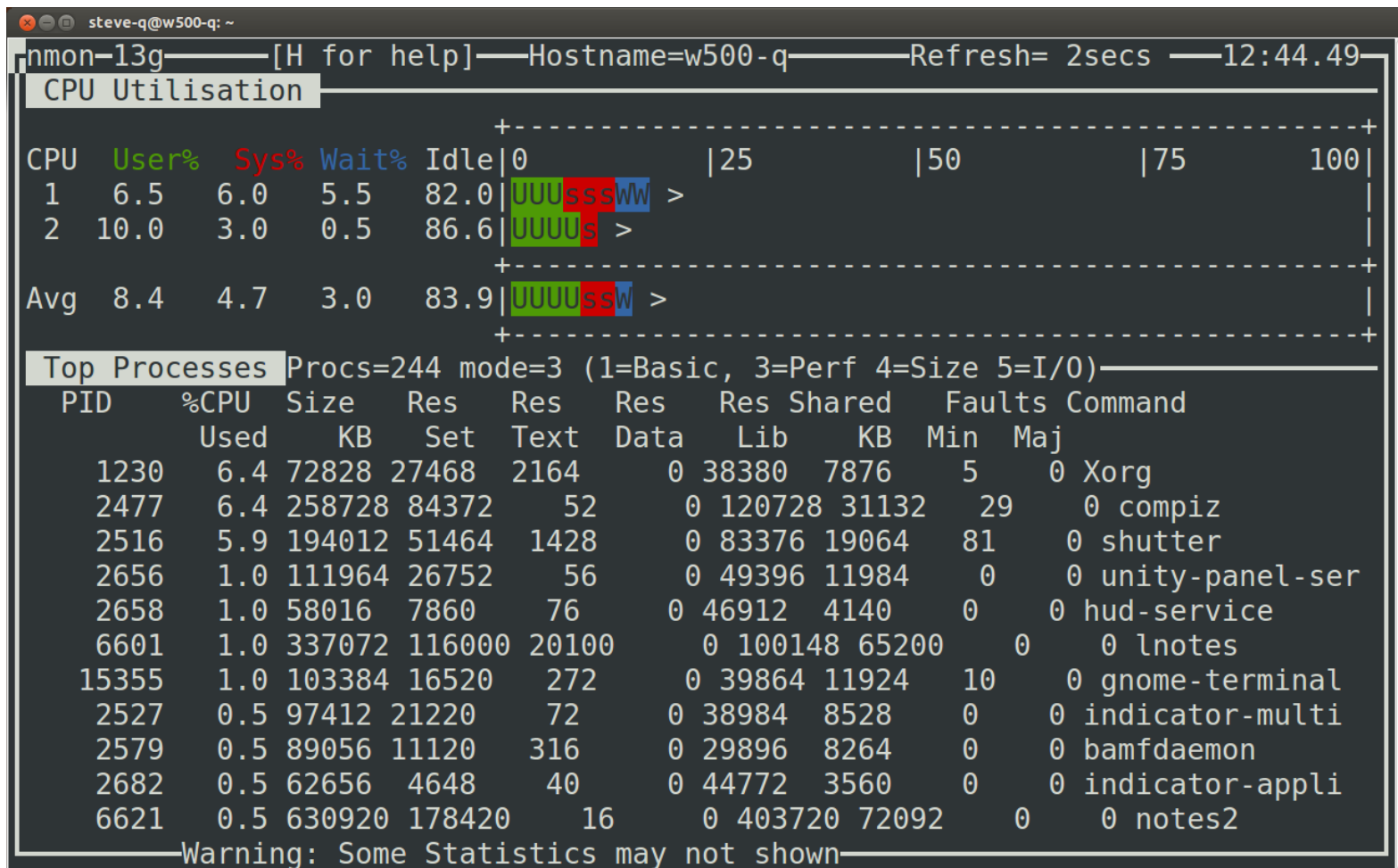
%MEM: The percentage of physical RAM used by the process.

TIME+: How much processor time the process has used.

COMMAND: The name of the command that started the process.

Another favorite → nmon

```
$ sudo apt-get install nmon
```



tu5

```
topas_nmon #=PURR Stats Host=sassrv1 Refresh=2 secs 12:50.21
Top-Processes-(1618) Mode=5 [1=Basic 2=CPU 3=Perf 4=Size 5=I/O 6=Cmds]
  PID      %CPU  ResSize  Char  Command
    Used      KB  T/O
39387532  97.6  2111776  126M /usr/sas/sas82base/sas /afs/btv/data/sg_dtap/source/WFT_chiple
48485802  23.7  21128 36487 /usr/sas/sas82base/sas /afs/btv.ibm.com/u1/v2telys/wasa/sp1/so
31130156  24.5  16804 28354 /usr/sas/sas82base/sas /afs/btv.ibm.com/u1/v2telys/wasa/sp1/fl
 7996010  26.2  26068 27651 /usr/sas/sas82base/sas /afs/btv.ibm.com/u1/v2telys/2s/elec_tim
23593100  77.8  16552 24593 /usr/sas/sas82 -set fab btv -set toolbox spcview -set family d
20251568  13.1  17668 12762 /usr/sas/sas82base/sas /afs/btv.ibm.com/u1/v2telys/2s/elec_tim
46334564  12.7  23848 8553 /usr/sas/sas82base/sas /afs/btv.ibm.com/u1/v2telys/2s/elec_tim
32965616  66.7  23412 5535 /usr/sas/sas82base/sas -autoexec /afs/btv.ibm.com/data/datavie
38929092  97.6  16348 5503 /usr/sas/sas82 -set fab btv -set toolbox phasein -set family b
26673658  12.1  21524 3447 /usr/sas/sas82base/sas /afs/btv.ibm.com/u/22soi/bt/automated_s
18612528  96.5  43452 1983 /usr/sas/sas82base/sas -autoexec /afs/btv.ibm.com/data/datavie
41157490  98.6  30828 1727 /usr/sas/sas82 -set fab btv -set toolbox spcview -set family C
32112680  0.9  31972 22 Xvnc :10 -desktop Steve Ruegsegger VNC -httpd /afs/btv.ibm.com
35652136  0.5  1740 8 /usr/dt/bin/dtterm
27852990  0.1  7708 2 Xvnc :8 -desktop sassrv1:8 (barthold) -auth /afs/btv.ibm.com/u
23986808  0.7  18660 1 Xvnc :35 -desktop sassrv1:35 (vedrine) -auth /afs/btv.ibm.com/
15532362  3.5  19784 1 /usr/bin/topas_nmon
34931198  0.0  5844 0 Xvnc :11 -desktop sassrv1:11 (jcrafts) -auth /afs/btv.ibm.com/
40895322  0.2  772 0 autocutsel
1311712  18.2  608 0 /usr/sbin/syncd 60
```

Signaling Processes

- A process can be sent a signal by the kernel or by another process
- Each signal is a very simple message. There are two ways to identify the message:
 - A small whole number
 - With a mnemonic name
- Signal names are all-caps, like INT
 - They are often written with SIG as part of the name: SIGINT
- Some signals are treated specially by the kernel; others have a conventional meaning
- There are about 30 signals available, not all of which are very useful

Common Signals for Interactive Use

- The command `$ kill -l` lists all signals
- The following are the most commonly used:

Name	Number	Meaning
INT	2	Interrupt — stop running. Sent by the kernel when you press <code>Ctrl+C</code> in a terminal.
TERM	15	“Please terminate.” Used to ask a process to exit gracefully.
KILL	9	“Die!” Forces the process to stop running; it is given no opportunity to clean up after itself.
TSTP	18	Requests the process to stop itself temporarily. Sent by the kernel when you press <code>Ctrl+Z</code> in a terminal.
HUP	1	Hang up. Sent by the kernel when you log out, or disconnect a modem. Conventionally used by many daemons as an instruction to re-read a configuration file.

Sending Signals: `kill`

- The `kill` command is used to send a signal to a process
 - Usually, I do `$ kill -9 pid`
 - However, `kill` is not just to terminate a running process!
- It is a normal executable command, but many shells also provide it as a built-in
- Use `kill -HUP pid` or `kill -s HUP pid` to send a `SIGHUP` to the process with that `pid`
- If you miss out the signal name, `kill` will send a `SIGTERM`
- You can specify more than one `pid` to signal all those processes

Typical scenario: Dæmons

- On Unix systems, long-lived processes that provide some service are often referred to as dæmons
- Dæmons typically have a configuration file (usually under /etc) which affects their behaviour
- Many dæmons read their configuration file only at startup
- If the configuration *changes*, you have to explicitly tell the dæmon by sending it a SIGHUP signal
- Typical use:

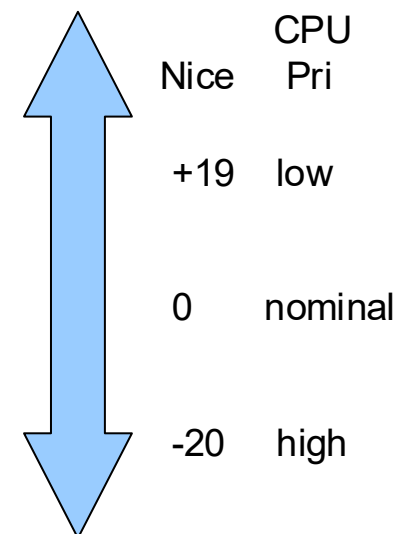
```
$ sudo kill -HUP $(pidof sshd)
```

Share!

- Not all tasks require the same amount of execution time
- Linux has the concept of execution priority to deal with this
- The kernel dynamically alters process priority
- You can view the current priority by looking at `top` or `ps -l` and looking at the PRI column
- The priority can be biased using `nice`
 - The current bias can be seen in the NI column in `top`

niceness

- “niceness” is the inverse of CPU priority
 - The more “nice”, the less demanding of CPU attention you request
- 40 niceness values:
 - integers from +19 (very nice) to -20 (not very nice (nasty?)).
 - Default is 0.
- Non-root users can only specify new nice values ≥ 0
 - more nice
 - from 1 to 19
- Only root can specify the full range of values
 - less nice, more demanding
 - negative niceness



nice command

- Just put “nice” before the normal command

```
$ nice tar -cvf etc.backup.tar /etc
```

```
$ nice gzip /tester/*.TEDS
```

- Default niceness for `nice` is 10
 - (recall positive niceness is 0-19)
- To run a command at increased niceness give a positive number after the dash:

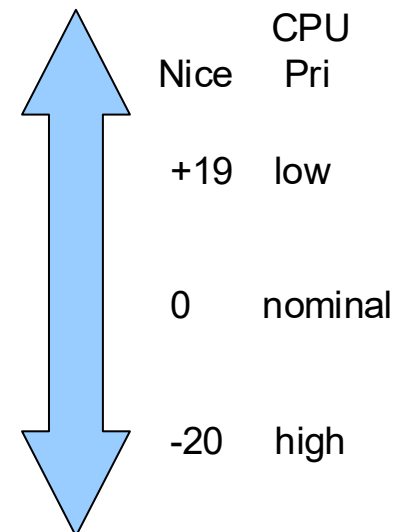
```
$ nice -9 long-running-command &
```

```
$ nice -n 5 long-running-command &
```

- Only root can *decrease* the niceness with a negative number:

```
$ sudo nice --15 important-command &
```

```
$ sudo nice -n -15 important-command &
```



renice

- `renice` changes the niceness of existing processes
- Typical format: `$ renice pri pid`
- Rules for `renice`:
 - Non-root users are only permitted to increase a process's niceness
 - And they can only change a process they own
 - The niceness priority is the absolute number: Not an increment
 - Root, of course, can change the prio of any process to any value
- Examples:
 - To set the process with pid 2984 to the maximum niceness:

```
$ renice 20 2984
```

- To set the process with pid 3598 to a lower niceness:

```
$ sudo renice -15 3598
```

- Root can also change the niceness of all a user's processes:

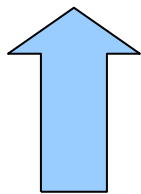
```
$ sudo renice 15 -u mikeb
```

```
ps -elf
```

- To see a list of 'nice' values, use -l 'long' option

```
steve@cis2230a:~$ ps -elf | head -1 && ps -elf | grep steve
```

F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	STIME	TTY	TIME	CMD
0	S	steve	9361	1	0	80	0	-	510	wait	22:05	?	00:00:00	/bin/sh /usr/
0	S	steve	9362	9361	0	80	0	-	8358	futex_	22:05	?	00:00:00	/usr/lib/lib
0	S	steve	9374	9362	2	80	0	-	59225	poll_s	22:05	?	00:00:21	/usr/lib/lib
0	R	steve	9447	4651	97	90	10	-	510	-	22:20	pts/3	00:01:22	/bin/sh bin/
0	R	steve	9470	4651	94	90	10	-	24776	-	22:21	pts/3	00:00:02	/bin/bash bi
0	R	steve	9471	2595	0	80	0	-	1253	-	22:21	pts/0	00:00:00	ps -efl
0	S	steve	9472	2595	0	80	0	-	1112	pipe_w	22:21	pts/0	00:00:00	grep --color

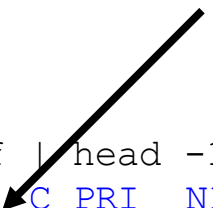


Default is '0', but 2 are 'extra nice' at +10.

orphans

- What is ppid of 1?
- What is pid #1?
- A process with ppid #1 is often an “orphan”
- You can kill the orphan! (Ack)

```
steve@cis2230a:~$ ps -elf | head -1 && ps -elf | grep steve
```



F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	STIME	TTY	TIME	CMD
0	S	steve	9361	1	0	80	0	-	510	wait	22:05	?	00:00:00	/bin/sh /usr/
0	S	steve	9362	9361	0	80	0	-	8358	futex_	22:05	?	00:00:00	/usr/lib/lib
0	S	steve	9374	9362	2	80	0	-	59225	poll_s	22:05	?	00:00:21	/usr/lib/lib
0	R	steve	9447	4651	97	90	10	-	510	-	22:20	pts/3	00:01:22	/bin/sh bin/
0	R	steve	9470	4651	94	90	10	-	24776	-	22:21	pts/3	00:00:02	/bin/bash bi
0	R	steve	9471	2595	0	80	0	-	1253	-	22:21	pts/0	00:00:00	ps -efl
0	S	steve	9472	2595	0	80	0	-	1112	pipe_w	22:21	pts/0	00:00:00	grep --color