

Chapter 9: Process management

Chapter 9 Process management



Chapter 9 Outline

- In this chapter we will learn about:
 - ✓ Processes and process concepts
 - ✓ Examining processes
 - ✓ Adjusting process priority and job control
 - ✓ Signals, orphans and zombies

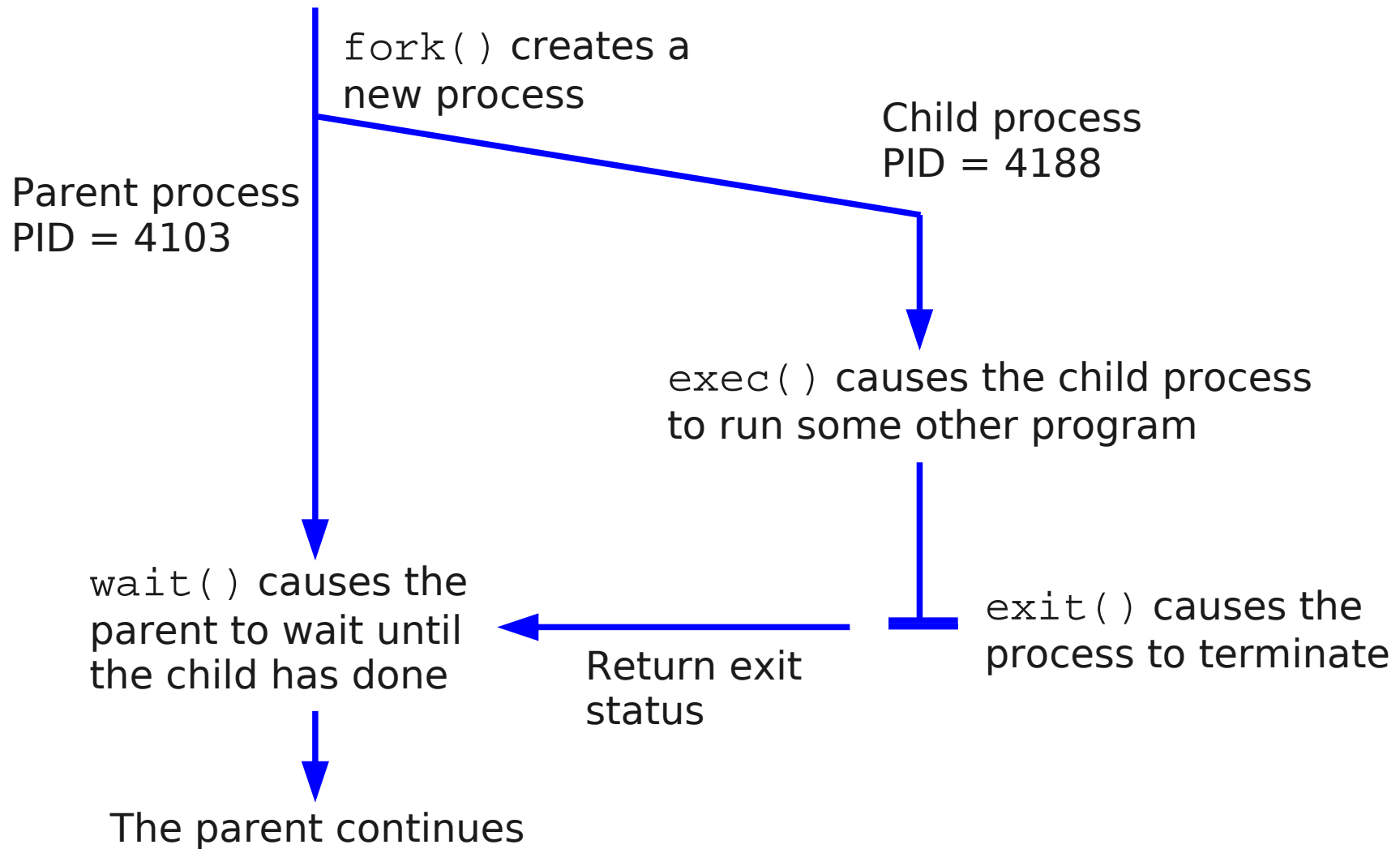
Process concepts

- Process concepts
 - Processes
 - Typical process life cycle

Processes

- A process represents an active instance of a program
- Some processes are started by users
 - Typing commands on a command line
 - Selecting applications from a desktop menu
- Some processes are started automatically, usually at boot time
 - Provide system services e.g. file servers, print services, web servers
 - Known as “daemons”
- At any one time, a linux system is likely to have between 50 and 500 processes
- Linux performs *pre-emptive multitasking* between processes allowing them to share resources (including CPU time) on the computer
- Each process has a numeric identifier, called its *process ID* (PID)

Typical process life cycle



Examining processes

- Examining processes
 - Listing processes with ps
 - Command line options for ps
 - Long process listings
 - Listing system processes
 - Listing processes with pstree
 - Showing process activity with top
 - KDE System Guard

Listing processes with `ps`

- The `ps` command lists the processes running on the system

```
$ ps
  PID TTY          TIME CMD
 3540 tty1        00:00:00 bash
 3632 tty1        00:00:00 ps
$
```

With no arguments, `ps` shows only the processes running on the current terminal

The command used to launch the process

The process ID. These are allocated sequentially in the range 0-32767, then wrap round to re-use available values

HH:MM:SS of CPU time used by process

The device name of the associated terminal

Command line options for `ps`

- A confusing set of option flags control which processes are shown and how much detail is shown about each process
 - Try `man ps` for details
- Process selection flags include:

| Flag | Meaning |
|---------------------|--------------------------------------------------|
| <code>x</code> | List processes that have no controlling terminal |
| <code>a</code> | Show processes belonging to other users |
| <code>U user</code> | Show processes owned by <i>user</i> |

- Process detail flags include:

| Flag | Meaning |
|----------------|---------------------------------------|
| <code>l</code> | Show long format (more detail) |
| <code>u</code> | Show detailed “user oriented” format |
| <code>e</code> | Show the process environment |
| <code>f</code> | Show processes as a hierarchical tree |

“Long” process listings

- The 'l' option provides additional detail for each process

```
$ ps l
F  UID      PID    PPID  PRI   NI   VSZ   RSS  WCHAN  STAT  TTY      TIME  COMMAND
0  500     3540   3539   15    0  2884  1700  wait4   S     pts/1    0:00  /bin/bash
0  500     3553   2186   15    0  2868  1648  schedu  S     tty1     0:00  -bash
0  500     4107   3540   19    0  3680  1708  -       R     pts/1    0:00  ps l
$
```

ID of the user running the process

ID of the parent process

ID of the process

Priority (lower values mean higher priority)

“NICE” value

Memory usage

What the process is waiting for

Process status:
R Runnable
S Sleeping
T Suspended
Z Zombie

“Long” process listings (continued)

- The 'u' option shows a slightly different set of fields:

```
$ ps u
USER      PID  %CPU  %MEM    VSZ   RSS TTY      STAT   START       TIME     COMMAND
chris    3540   0.0   0.3   2884  1700 pts/1    S       16:07       0:00    /bin/bash
chris    3553   0.0   0.3   2868  1648 tty1     S       16:07       0:00    -bash
chris    4172   0.0   0.1   2668   720 pts/1    R       20:39       0:00    ps u
$
```

Percentage of
CPU time this
process is using

Percentage of
memory this
process is using

Time (HH:MM)
when process
was started

Listing system processes

- The command `ps ax` shows all processes

```
$ ps ax
PID TTY          STAT       TIME COMMAND
   1 ?            S           0:04  init [3]
   2 ?            SW          0:00  [keventd]
   3 ?            SWN        0:00  [ksoftirqd_CPU0]
   4 ?            SW          0:00  [kswapd]
   ....
  724 ?            S           0:00  /sbin/syslogd
  727 ?            S           0:00  /sbin/klogd -c 1 -2
1081 ?            S           0:00  /sbin/portmap
1115 ?            S           0:00  /sbin/rpc.statd
1278 ?            S           0:00  /usr/sbin/acpid
   ....
8432 ?            S           0:00  login -- tux
8433 tty1         S           0:00  -bash
8453 tty1         R           0:00  ps ax
```

This list has been heavily edited

Kernel processes

Daemons

User processes

Listing processes with `pstree`

- The `pstree` command shows the parent/child relationships of processes as a tree using “ASCII art”

```
$ pstree -u
init-+-acpid
    |-bdflush
    |-cardmgr
    |-cpufreqd
    |-cron
    |-cupsd(lp)
    |-dhcpcd
    |-httpd2-prefork---5*[httpd2-prefork(wwrun)]
    |-kalarmd(chris)
    |-kamix(chris)
    |-kdeinit(chris)-+-artsd
        |-3*[kdeinit]
        |-kdeinit---bash---pstree
        `--soffice.bin---soffice.bin---4*[soffice.bin]
    |-10*[kdeinit(chris)]
    |-kdm-+-X
    `--kdm---kde(chris)---kwrapper
```

Showing process activity with `top`

- The `top` command shows a real-time display of process activity which automatically updates at regular intervals
 - Shows the 'top' screenful of processes sorted on a selected column
 - %CPU, %Memory, etc.
 - Interactive commands allow
 - selection of whose processes to show
 - selection of which field to sort on
 - selection of refresh interval
 - killing of processes
 - 'h' command shows help screen, 'q' quits

An example of top
























```
top - 19:39:49 up 9:07, 4 users, load average: 0.72, 0.28, 0.13
Tasks: 96 total, 2 running, 94 sleeping, 0 stopped, 0 zombie
Cpu(s): 98.4% user, 1.6% system, 0.0% nice, 0.0% idle
Mem: 514372k total, 472548k used, 41824k free, 82732k buffer
Swap: 1036184k total, 2688k used, 1033496k free, 145852k cached
```

| PID | USER | PR | NI | VIRT | RES | SHR | S | %CPU | %MEM | TIME+ | COMMAND |
|------|-------|-----|----|-------|------|------|---|------|------|---------|----------|
| 3942 | chris | 25 | 0 | 272 | 272 | 220 | R | 93.1 | 0.1 | 0:54.76 | soak |
| 2232 | chris | -51 | 0 | 6336 | 6260 | 4656 | S | 3.0 | 1.2 | 0:02.32 | artsd |
| 2003 | root | 15 | 0 | 44836 | 19m | 4608 | S | 1.3 | 3.9 | 2:48.67 | X |
| 3922 | chris | 16 | 0 | 952 | 952 | 748 | R | 1.3 | 0.2 | 0:01.69 | top |
| 2562 | chris | 15 | 0 | 75556 | 73m | 50m | S | 0.7 | 14.7 | 2:38.78 | soffice. |
| 3946 | chris | 18 | 0 | 1032 | 1032 | 816 | S | 0.7 | 0.2 | 0:00.02 | xwd |
| 2269 | chris | 15 | 0 | 15980 | 15m | 13m | S | 0.3 | 3.1 | 0:28.87 | kdeinit |
| 1 | root | 15 | 0 | 256 | 256 | 220 | S | 0.0 | 0.0 | 0:04.94 | init |
| 2 | root | 15 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.25 | keventd |
| 3 | root | 34 | 19 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.01 | ksoftirq |
| 4 | root | 15 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.26 | kswapd |
| 5 | root | 25 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 | bdflush |
| 6 | root | 15 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.27 | kupdated |
| 7 | root | 16 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.04 | kinoded |

KDE System Guard

- KDE system guard is a graphical application for displaying the process table and charting the system load
- *Process controller window* provides functionality similar to `top`
- *System load window* supports charting of a number of 'sensors'
 - Discussed in Chapter 10
- Able to monitor process activity on remote machines
 - Connects to `ksysguardd` daemon on target machine
 - Allows graphical monitoring of activity on (non-graphical) servers

KDE System Guard process controller

| Name | PID | User% | System% | Nice | VmSize | VmRss | Login | Command |
|------------------------------------------------------------------------------------------------|------|-------|---------|------|--------|-------|---------|-------------------------------------------------------|
|  kdesktop | 2242 | 0.00 | 0.00 | 0 | 25672 | 15612 | chris | kdeinit: kdesktop |
|  kicker | 2245 | 0.00 | 0.25 | 0 | 28076 | 17588 | chris | kdeinit: kicker |
|  klipper | 2251 | 0.00 | 0.00 | 0 | 25196 | 13904 | chris | kdeinit: klipper |
|  kamix | 2253 | 0.00 | 0.00 | 0 | 25144 | 13828 | chris | kamix |
|  suseplugger | 2258 | 0.00 | 0.25 | 0 | 26300 | 15048 | chris | suseplugger |
|  susewatcher | 2259 | 0.00 | 0.00 | 0 | 24128 | 13376 | chris | susewatcher |
|  knotes | 2268 | 0.00 | 0.00 | 0 | 25204 | 14436 | chris | knotes |
|  konsole | 2269 | 0.00 | 0.00 | 0 | 26832 | 16032 | chris | kdeinit: konsole -session 11c0a800010001084013 |
|  konqueror | 2270 | 0.00 | 0.00 | 0 | 30160 | 21064 | chris | kdeinit: konqueror --preload |
|  kalarmd | 2272 | 0.00 | 0.00 | 0 | 23160 | 12072 | chris | kalarmd |
|  bash | 2273 | 0.00 | 0.00 | 0 | 2884 | 1764 | chris | /bin/bash |
|  mingetty | 2757 | 0.00 | 0.00 | 0 | 1500 | 528 | root | /sbin/mingetty |
|  kio_uiserver | 2787 | 0.00 | 0.00 | 0 | 25164 | 14704 | chris | kdeinit: kio_uiserver |
|  konqueror | 3018 | 0.00 | 0.00 | 0 | 34632 | 23876 | chris | kdeinit: konqueror --silent |
|  kio_file | 3026 | 0.00 | 0.00 | 0 | 22188 | 9952 | chris | kdeinit: kio_file file /tmp/ksocket-chris/klauncherYK |
|  pickup | 3693 | 0.00 | 0.00 | 0 | 4204 | 1312 | postfix | pickup |
|  bash | 3923 | 0.00 | 0.00 | 0 | 2888 | 1764 | chris | /bin/bash |
|  gimp | 3943 | 1.00 | 0.50 | 0 | 18024 | 14912 | chris | gimp |
|  script-fu | 3944 | 0.00 | 0.00 | 0 | 6652 | 2960 | chris | /usr/lib/gimp/1.2/plugin-ins/script-fu |
|  su | 4009 | 0.00 | 0.00 | 0 | 2664 | 1160 | root | su |
|  bash | 4010 | 0.00 | 0.00 | 0 | 2808 | 1656 | root | -bash |
|  ksysguard | 4034 | 4.00 | 0.25 | 0 | 25848 | 16508 | chris | ksysguard |
|  ksysguardd | 4035 | 0.50 | 0.50 | 0 | 1972 | 848 | chris | ksysguardd |

Tree All Processes Refresh Kill

Exercise: Using `ps` and `top`

1. Use an appropriate `ps` command to find the process that is running the daemon 'syslogd'

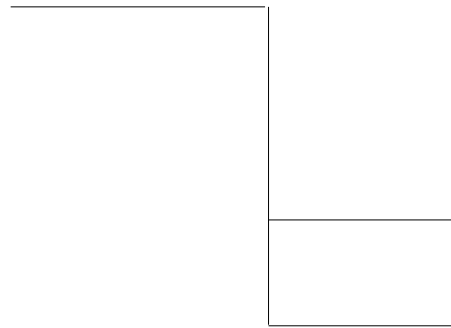
- What is the process ID? _____
- Who owns this process? _____
- What is the process ID of its parent? _____
- Hint: use `grep` to search the output of `ps` for the process you want

2. Run the `top` command

- How long has the system been up? _____
- What is the total amount of memory in the machine? _____
- How much memory is linux using? _____
- Using the help screen as a guide, sort the `top` listing on (a) the process ID and (b) the %CPU utilisation of the processes

Process priority and job control

- Process priority and job control



Adjusting process priority
Job control in the shell

Adjusting process priority

- Process priorities are adjusted dynamically and cannot be set explicitly
- The `nice` command adjusts the 'nice' level of a process which affects its dynamic priority
- Nice values range from -20 (highest priority) to +19 (lowest priority)

```
$ nice -n +5 long_running_CPU_bound_program
```

- Only root can start a process with negative niceness (nastiness?)
- The `renice` command adjusts the nice level of a running process

```
$ renice +1 -u tux
```



All processes owned by tux

```
$ renice +2 1712
```



Process 1712 only

The 'r' command in `top` can also be used to adjust nice levels

- Only root can adjust the nice level downwards

Job control in the shell

- The bash shell is able to manage multiple jobs
 - Jobs can be started in the foreground - shell waits for it to finish
 - Jobs can be started in the background by appending '&' to the command - shell prompts for another command immediately
- The following job control commands are available:
 - `jobs` Lists the current jobs
 - `^Z` Stop (pause) the current foreground job
 - `bg` Restart a foreground job in the background
 - `fg` Bring a background job into the foreground
 - `kill` Terminate a job

Job control in the shell (continued)

```
$ xclock -update 1 &
```

Job started in background

```
[1] 3197
```

Shell reports job number and PID

```
$ xeyes &
```

```
[2] 3198
```

```
$ jobs
```

```
[1]-  Running          xclock -update 1 &
```

```
[2]+  Running          xeyes &
```

'+' indicates the current
Bring current job
into foreground

```
$ fg  
xeyes
```

```
[2]+  Stopped          ^Z stops (pauses) the current job  
xeyes
```

```
$ kill %1
```

'%1' means job number 1

```
$ fg  
xeyes
```

```
[1]  Terminated      ^C terminates the foreground job  
te 1
```

Shell reports
termination of job 1

Exercise: Using job control

- The program `soak` (written especially for this exercise), simply soaks up CPU time in an infinite loop. If started with a command line argument, it will print that argument, the process ID, and a loop counter every pass through the loop

```
$ ./soak A
soak A 4487 0
soak A 4487 1
^C
$
```

- The program `sleep` (a standard tool) simply sleeps for a specified number of seconds, then exits

```
$ sleep 10
$
```

← 10 seconds later

- We will use these two programs to investigate background processes and the use of 'nice' values

Exercise: Using job control (continued)

1. Start 2 instances of `soak` (with no argument) and one instance of `sleep 20`, all in the background
2. List your background jobs with the `jobs` command
 - What are their job numbers and process ids? _____
3. Wait 20 seconds
4. Re-run the `jobs` command.
 - How many jobs are there now? _____
 - When does the shell notify you that a background job is done? _____
5. Run the command `ps u`
 - What are the PIDs of the two instances of `soak`? _____
 - What percentage of CPU time are these processes using? _____
6. Kill both background jobs using `kill` and the job numbers

Exercise: Adjusting the `nice` value

7. Start 2 more instances of `soak`, the first with a nice value of 0 (the default) and the second with a nice value of +10
8. Execute `ps u` and examine the %CPU use of both instances
 - Can you see any difference? _____
9. Adjust the nice level of the first `soak` to match that of the second (+10)
10. Re-examine the %CPU usage of both processes. Can you see a change? (It may take a few seconds to see a significant change)
11. Try to adjust the nice level of the first `soak` back down to zero.
 - What happens? _____
12. Kill both jobs
13. Finally, start 2 more instances of `soak` and examine them using `top`. Within `top`, adjust the nice values of the processes, observe their %CPU usage, and finally kill both processes

Signals, orphans and zombies

- Signals, orphans and zombies

Sending signals

Signal types

Signal handlers

Orphan processes

Zombie processes

Getting rid of zombies

Sending signals

- Signals are software interrupts delivered to a process by the kernel
 - Linux defines over 30 signal types, `kill -l` will give the full list
- Some signals can be generated by the terminal driver in response to specific key combinations
 - This only works for programs running in the foreground

`^C` Send SIGINT signal

`^\` Send SIGQUIT signal

- If a program is not running in the foreground, or has no attached terminal, signals can be delivered using the `kill` command:

```
$ kill -15 4321
```

```
$ kill -SIGTERM 4321
```

```
$ kill 4321
```

Signals can only be sent to a process by its owner (or by root)

SIGTERM is the default type sent by kill

- The `killall` command is similar but specifies processes by name

```
$ killall -SIGHUP xinetd
```

Signal types

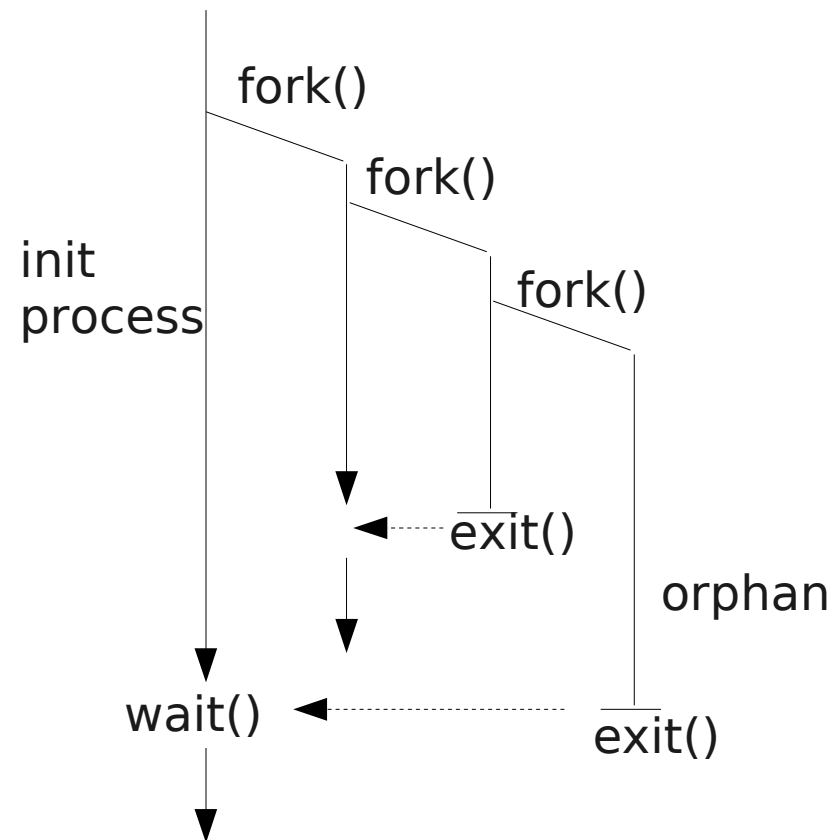
| Number | Signal Name | Description | Default Handling |
|--------|-------------|-------------------------------------------------------------------------------|----------------------------------|
| 1 | SIGHUP | Most shells send this signal to all child processes when they terminate | Kill process |
| 2 | SIGINT | Sent by terminal driver in response to ^C. Intended as a graceful termination | Kill process |
| 3 | SIGQUIT | Sent by terminal driver in response to ^\ | Kill process and write core dump |
| 9 | SIGKILL | This signal cannot be caught or ignored | Kill process |
| 15 | SIGTERM | This is the default signal sent by the kill command | Kill process |
| 18 | SIGCONT | Causes a stopped or traced process to continue | Continue process |

Signal handlers

- A process can elect how it wants to react on receipt of a signal by nominating a *handler* for that signal type
 - A function that will be called when the signal is delivered
 - Within a shell script, the `trap` command defines signal handlers
- If a handler is not specified, each signal type has a default behaviour
 - For most signals, the default is to kill the process
 - For some signals, the default is to ignore the signal
- Signals 9 (SIGKILL) and 19 (SIGSTOP) cannot be caught or ignored
 - Use SIGKILL as a last resort – the program has no opportunity to clean up
- Some programs respond to a signal by re-reading their config file
 - The SIGHUP signal is conventionally used for this
 - Allows services to be re-configured on the fly without stopping them

Orphan processes

- Sometimes, a parent process exits without waiting for its child(ren) to finish
 - Any children are automatically inherited by the `init` process



Zombie processes

- Occasionally, a parent process will not wait for its children but will block on some other event
- Processes that exit when no process is waiting for them become 'zombies'
 - Cannot be laid to rest as they want to pass back their exit status
 - Marked as <defunct> in a `ps` listing
 - In the worst case, zombies accumulate and fill the process table so that no more processes can be started
- The way to get rid of zombies is to kill their parent
 - The zombies will then be inherited by `init`
- Zombies are generally a result of bad program design

Getting rid of zombies

Killing the parent will usually eliminate the zombies

```
$ ps l
F  UID  PID  PPID  STAT  TTY  TIME  COMMAND
0  500  2359  2358  S     pts/1  0:00  /bin/bash
0  500  3705  3704  S     pts/2  0:00  /bin/bash
0  500  3766  2358  S     pts/3  0:00  /bin/bash
0  500  3876  2359  S     pts/1  0:00  ./myprog
0  500  3877  3876  Z     pts/1  0:00  [myprog] <defunct>
0  500  3878  3876  Z     pts/1  0:00  [myprog] <defunct>
0  500  3879  3876  Z     pts/1  0:00  [myprog] <defunct>
0  500  3882  3876  Z     pts/1  0:00  [myprog] <defunct>
0  500  3883  2359  R     pts/1  0:00  ps l
```

Notice that the zombies have a common parent

Exercise: Sending signals

1. You should be logged in as `tux` for this exercise
2. Start an instance of `soak` in the background like this:

```
$ soak "Annoying output" &
```
3. Start another terminal window and use `ps` to find the process ID of the `soak` program
4. Send `soak` a signal to terminate it
5. Return to the original window, verify that the process has stopped
6. Repeat the experiment, using `killall` instead of `kill` to kill the `soak` process by name
7. Try to kill the `init` process by sending it a `SIGKILL`. What happens?

Exercise: Killing zombies

8. Run the program `zombify` (in your home directory) in the background
 - This program is written specially for this exercise. It creates four child processes then pauses indefinitely. The child processes immediately exit and so become zombies
9. Examine the process table
 - How many zombie (defunct) processes do you see?
 - Write down the PIDs of the zombies: _____
 - ... and the PID of their parent _____
10. Try to kill one of the zombies by sending it a SIGKILL signal
 - Did this work?
11. Kill the parent process
 - Verify that the parent and the zombies are now all gone

Quiz

- Name four programs that allow you to display a list of processes
- Describe the circumstances that lead to the formation of zombies
- Which type of signal cannot be caught or ignored?
- Which type of signal is sent by the terminal driver in response to `^C`?
- Explain the difference between the commands `kill 1` and `kill %1`
- In the output from `ps 1`, what does an 'R' in the STAT column mean?
- True or false?
 - The `ps` command can display the processes running on a remote machine
 - A program's priority can be set to a fixed level using the bash shell
 - The command `kill 1234` sends a SIGTERM signal to process 1234
 - The `top` command can order processes based on %CPU utilisation