Working with processes

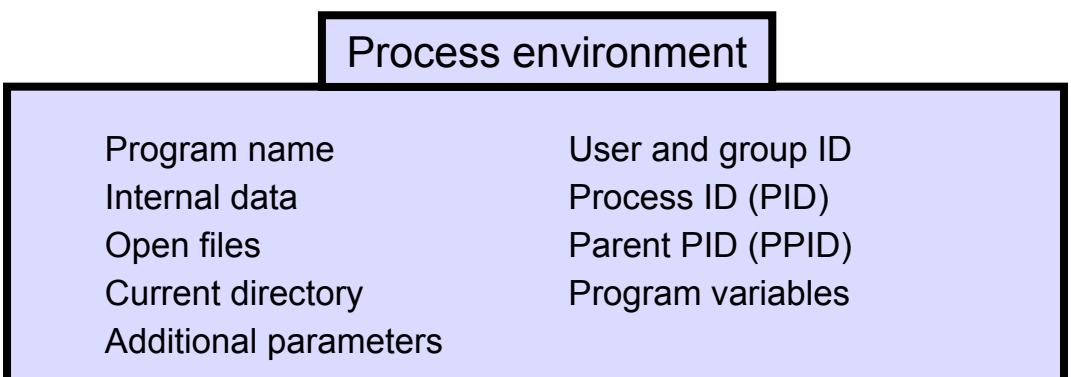
Unit objectives

After completing this unit, you should be able to:

- Define a Linux process
- Describe the relationship between parent and child processes
- Explain the purpose of a shell
- Start foreground and background processes
- Explain the concept of signals and use them to terminate processes
- Explain the concept of priorities and manage them

What is a process?

- A program is an executable file.
- A process is a program that is being executed.
- Each process has its own environment.



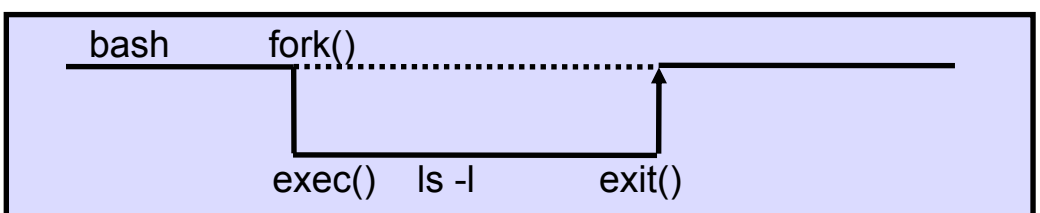
- To see the PID of your current shell process, type:

```
$ echo $$
```

Starting and stopping a process

- All processes are started by other processes.
  - This is called a parent/child relationship.

```
$ ls -l
```



- A process can be terminated for two reasons.
  - The process terminates itself when done.
  - The process is terminated by a signal from another process.

Login process environment

Linux system

Login:
Password:
$

/sbin/mingetty
...forks /bin/bash
-bash (login shell)

Environment

Program	-bash
UID	503 (tux3)
GID	100 (users)
Open files	/dev/tty1
PID	201

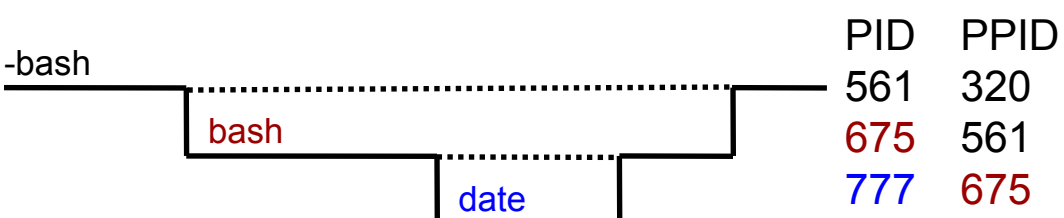
Parents and children


```
$ echo $$
561

$ bash
$ echo $$
675

$ date
Mon Jan 1 22:28:21 UTC 2007
$ <ctrl-d>

$ echo $$
561
```



Monitoring processes

- The **ps** command displays process status information.

```
$ ps aux
USER   PID %CPU %MEM  VSZ RSS   TTY   STAT START TIME COMMAND
root     1  0.0  0.0 1336 436     ?   S    Jan1  0:05 init
root     2  0.0  0.0    0   0     ?   SW   Jan1  0:00 [keventd]
root     3  0.0  0.0    0   0     ?   SW   Jan1  0:05 [kapmd]
root     4  0.0  0.0    0   0     ?   SW   Jan1  0:05 [kswapd]
...
root 10248  0.0  0.1 2852 884 pts/2  R    13:47 0:00 ps aux
```

- **ps** supports a large number of options; you typically use `ps aux`.
  - **a:** All processes attached to a terminal
  - **x:** All other processes
  - **u:** Provides more columns

Viewing process hierarchy

- **pstree** shows process hierarchy.

```
$ pstree
init-+-apmd
     |-atd
     |-crond
     |-gpm
     |-httpd---10*[httpd]
     |-inetd
     |-kattraction.kss
     |-kdm-+-X
     |     `-kdm---kwm-+-kbgndwm
     |                 |-kfm
     |                 |-kpanel
     |                 |-krootwm
     |                 |-kvt---bash---man---sh-+-gunzip
     |                 |                       `-less
     |                 `-startkde---autorun
     |-kflushd
```

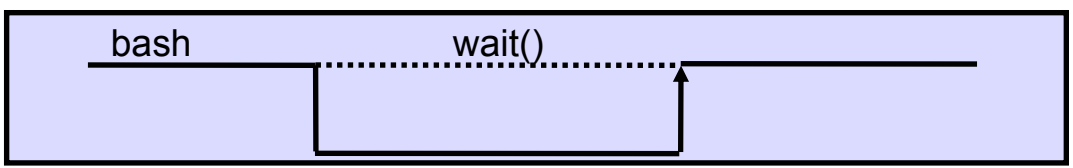
Controlling processes

- Processes can be controlled in two ways.
  - From the shell that started it, using its job number
  - From anywhere on the system, using its PID
- The following actions can be performed on a running process:
  - Terminate
  - Kill
  - Stop/continue
- These actions are performed by sending signals.

Starting processes

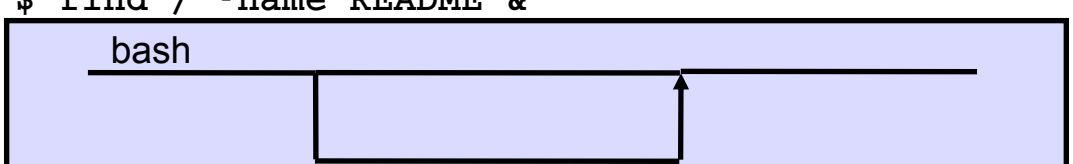
- Foreground processes
  - Foreground processes are invoked by simply typing a command at the command line.

```
$ find / -name README
```



- Background processes
  - Background processes are invoked by putting an ampersand (&) at the end of the command line.

```
$ find / -name README &
```



Job control in the bash shell

- **<Ctrl-z>:** Suspends foreground task
- **jobs:** Lists background or suspended jobs
- **fg:** Resumes suspended task in the foreground
- **bg:** Resumes suspended task in the background
- Specify a job number for bg, fg, and kill using `%job`

Job control example


```
$ find / -name README &
[1]  2863
$ jobs
[1]+     running            find / -name README &
$ fg %1
/usr/share/doc/packages/grub/README
....
<Ctrl-z>
[1]+   stopped              find / -name README
$ bg 2863
$ jobs
[1]+     running            find / -name README &
$ kill %find
```

Kill signals

- Several signals can be sent to a process.
  - Using keyboard interrupts (if foreground process)
  - Using the kill command
    - Synopsis: `kill -signal PID`
  - Using the killall command to kill all named apps
    - Synopsis: `killall -signal application`
- The following table lists the most important signals.

Signal	Keyboard	Meaning	Default action
01		Hangup	End process
02	Ctrl-C	Interrupt	End process
03	Ctrl-\	Quit	End process and core dump
09		Kill	End process - cannot be redefined - handled by kernel
15		Terminate	End process

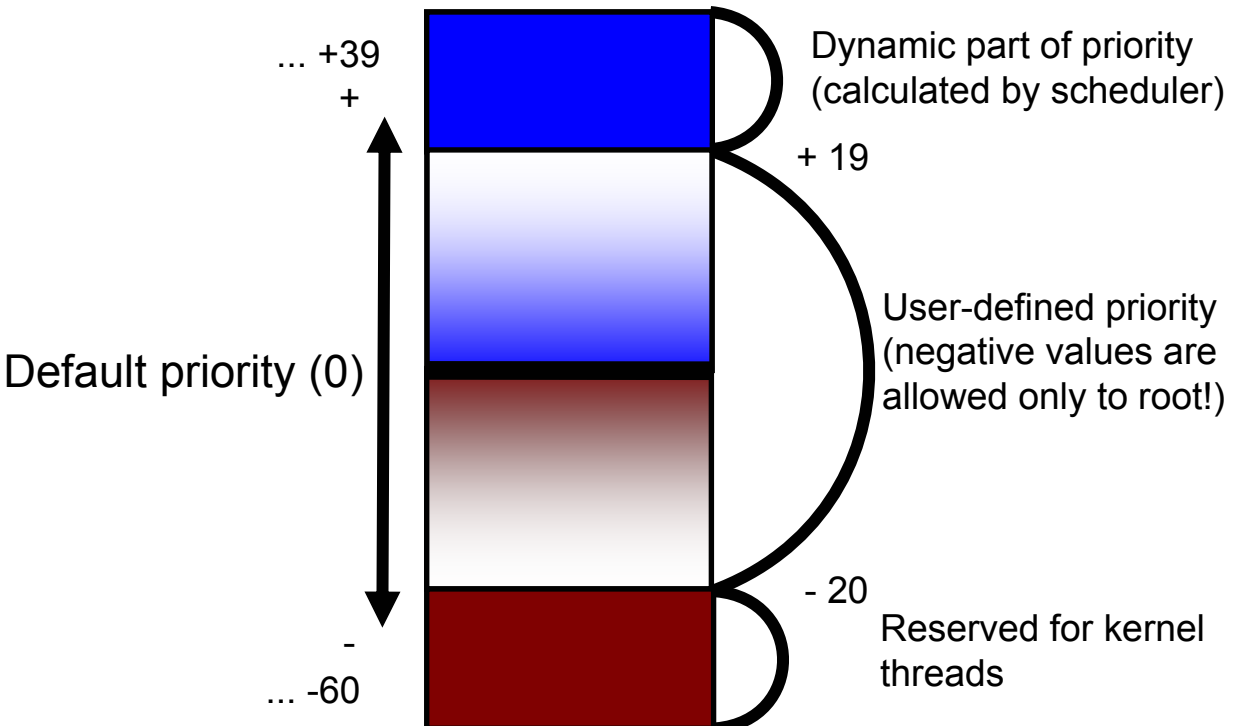
Running long processes

- The **nohup** command stops a process from being killed if you log off from the system before it completes by intercepting and ignoring the SIGHUP and SIGINTR (hangup and interrupt) signals and redirecting STDOUT and STDERR to a file.

```
$ nohup find / -name README &
nohup: appending output to `nohup.out´
$ logout
```

Managing process priorities

- Processes are scheduled according to priority.



Scheduling

- **Scheduling:** The algorithms by which the Linux kernel determines which processes should be active on the system.
  - Scheduling is a complex technology area.
- A process has an associated static priority (-100-39) as well as a nice value (-20 to 19) and a dynamic priority.
- The static priority and nice value, as well as factors like how much CPU time a process has used recently and how often a process has been prevented from running, are used to calculate the dynamic priority.
- Processes with lower dynamic priorities are scheduled to run more often.
- The dynamics of scheduling are all handled by the kernel.
- However, a user can influence how much processing time a process is allocated over time relative to other similar processes by setting or changing the nice value for a process.

The nice command

- The **nice** command is used to start a process with a user-defined priority.

```
nice [-n <value>] <original command>
```

```
$ nice -n 10 my_program &
[1] 4862
$ ps l
F   UID   PID    PRI    NI   VSZ   ...    COMMAND
0   500  4372     20     0   4860  ...     -bash
0   500  4862     30    10   3612  ...     my_program
0   00   4863     20     0   1556  ...     ps l
```

The renice command

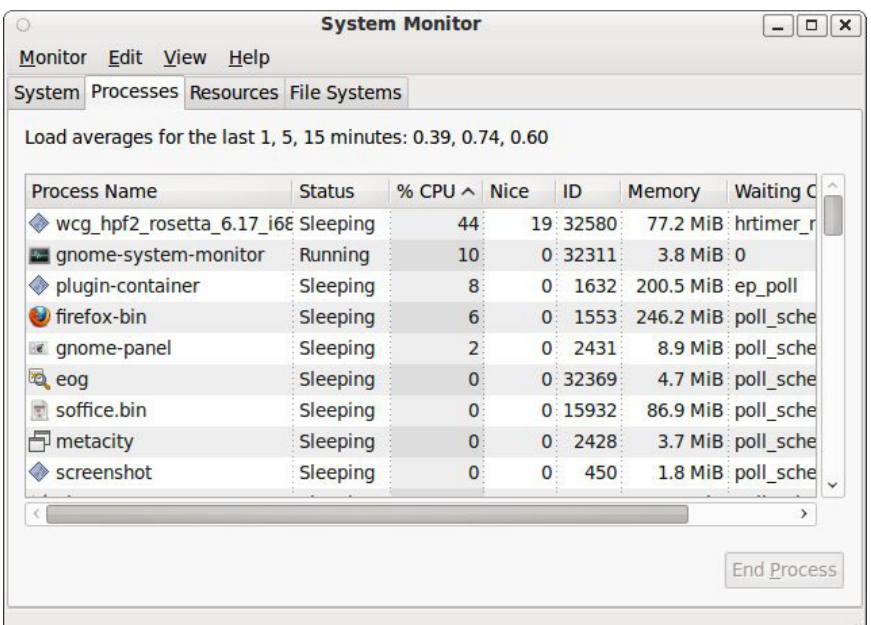
- The **renice** command is used to change the priority of a currently running process.

```
renice <new_priority> <PID>
```

```
$ renice 15 4862
4862: old priority 10, new priority 15
$ ps l
F   UID PID            PRI    NI          VSZ   ...   COMMAND
0   500 4372    20     0    4860 ...    -bash
0   500 4862    35    15    3612 ...    my_program
0   500 4868    20     0    1556 ...    ps l
```

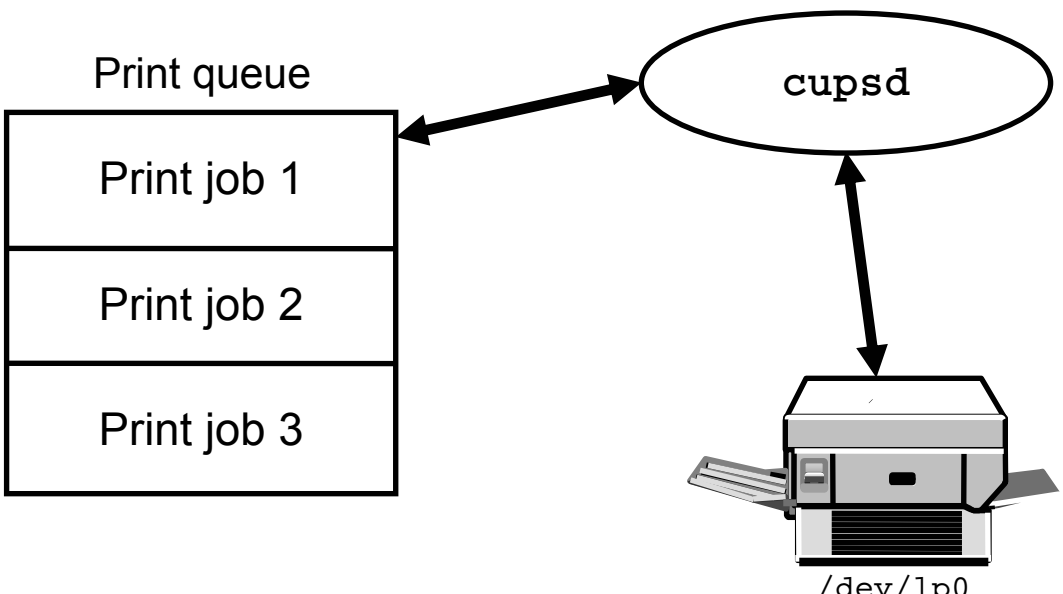
Integrated process management

- Various integrated tools exist for process management.
  - top, gnome-system-monitor, ksysguard
- Their availability depends on distribution.



Daemons

- The word *daemon* refers to a never-ending process, usually a system process, that controls a system resource, such as the printer queue, or performs a network service.



Unit review

- All processes are started by a parent process (except for init, which is started by the kernel).
- Every process is identified with a process identifier (PID).
- A special process is the shell, which can interpret user commands.
- Processes can terminate by themselves or upon reception of a signal.
- Signals can be sent by the shell, using a keyboard sequence, or by the **kill** and **killall** commands.
- Processes are started with equal priority, but this can be changed using the **nice** and **renice** commands.
- A daemon is a background process that typically controls a system resource or offers a network service.

Checkpoint

1. True or False: Any user can send a signal to a process of another user and cause that process to halt.

2. If a process is hanging, the proper order of trying to terminate it with the lowest chance of data corruption is:
   a. kill -1, <Ctrl-C>, kill, <Ctrl-\>
   b. <Ctrl-Z>, kill, kill -9, kill -15, <Ctrl-C>
   c. kill -9, kill -15, <Ctrl-C>, <Ctrl-Z>
   d. <Ctrl-C>, <Ctrl-\>, kill, kill -9

3. What is a daemon?

Checkpoint solutions

1. True or False: Any user can send a signal to a process of another user and cause that process to halt.

   The answer is false. A normal user will not have the appropriate permission to send signals to another user's processes. The root user does have permission to do this.

2. If a process is hanging, the proper order of trying to terminate it with the lowest chance of data corruption is:
   a. kill -1, <Ctrl-C>, kill, <Ctrl-\>
   b. <Ctrl-Z>, kill, kill -9, kill -15, <Ctrl-C>
   c. kill -9, kill -15, <Ctrl-C>, <Ctrl-Z>
   d. <Ctrl-C>, <Ctrl-\>, kill, kill -9

   The answer is <Ctrl-C>, <Ctrl-\>, kill, kill -9. kill -9 is always the last resort.

3. What is a daemon?

   The answer is a daemon is a never-ending process which provides a system service.

Exercise: Working with processes

What you will do in this exercise:

- Monitor processes
- Change and understand the process environment
- Control jobs
- Terminate processes

Unit summary

Having completed this unit, you should be able to:

- Define a Linux process
- Describe the relationship between parent and child processes
- Explain the purpose of a shell
- Start foreground and background processes
- Explain the concept of signals and use them to terminate processes
- Explain the concept of priorities and manage them