Operating Systems

Lab 6 (Process Control and Scheduling)

Overview

Every program that runs on Unix, Linux, and BSD systems is considered to be a process. Processes are assigned a unique identifier which it used to monitor and control the process. This process identifier (PID) can be used to control processes using the command line utilities discussed in this section.

Commands covered in this lesson:

Command	Purpose
`ps`	Display running processes.
`pgrep`	Find processes by name.
`pstree`	Display all running processes in a tree view.
`kill`	Terminate a process.
`killall`	Terminate all processes with the specified name.
`nice`	Execute programs at the specified CPU priority level.
`renice`	Alter the priority of a running process.
`&`	Start a process in the background.
`bg` `fg`	Move a process to the background/foreground.
`jobs`	Display background and suspended jobs.
`nohup`	Run a process immune to hang-up signals.
`batch`	Schedule programs to run during low CPU load.
`at`	Schedule programs to run at the specified time.
`atq`	Display queued `at` jobs.
`atrm`	Remove scheduled `at` jobs from the queue.
`crontab`	Schedule programs to run at the specified time(s).

ps

Purpose: Display running processes.

Usage syntax: ps [OPTIONS]

```
$ ps
  PID TTY          TIME CMD
 4958 pts/0    00:00:00 bash
 9596 pts/0    00:00:00 ps
```

Example output of the ps command

The ps command displays running process on the system. Executing the ps command with no options will display all processes owned by the current user as shown in the above example. For a complete listing of processes, use the -e option as demonstrated in the next example.

```
$ ps -e
  PID TTY          TIME CMD
    1 ?        00:00:01 init
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 migration/0
    4 ?        00:00:00 ksoftirqd/0
    5 ?        00:00:00 watchdog/0
...
```

Using the -e option with the ps command

The -ef option can be used to display detailed information about all processes on the system. This includes the user ID, process ID, parent process ID, and other helpful information, as shown in the next example.

```
$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Jun24 ?        00:00:02 /sbin/init
root         2     0  0 Jun24 ?        00:00:00 [kthreadd]
root         3     2  0 Jun24 ?        00:00:00 [migration/0]
root         4     2  0 Jun24 ?        00:00:00 [ksoftirqd/0]
root         5     2  0 Jun24 ?        00:00:00 [watchdog/0]
...
```

Output of the ps -ef command

pgrep

Purpose: Find processes by name.

Usage syntax: pgrep [OPTIONS] [NAME]

```
$ pgrep apache
3952
4075
4076
4077
4078
4079
```

Displaying PIDs with the name of apache using the pgrep command

The pgrep command displays all processes matching the specified name. In the above example the PIDs for the apache web server are displayed. This is the equivalent of typing ps -e|grep apache and is provided as a shortcut to produce the same results.

The -l option can be used with pgrep to display the full process name for each PID as demonstrated in the next example.

```
$ pgrep -l apache
3952 apache2
4075 apache2
4076 apache2
4077 apache2
4078 apache2
4079 apache2
```

Using the -l option with pgrep to display the full name of each process

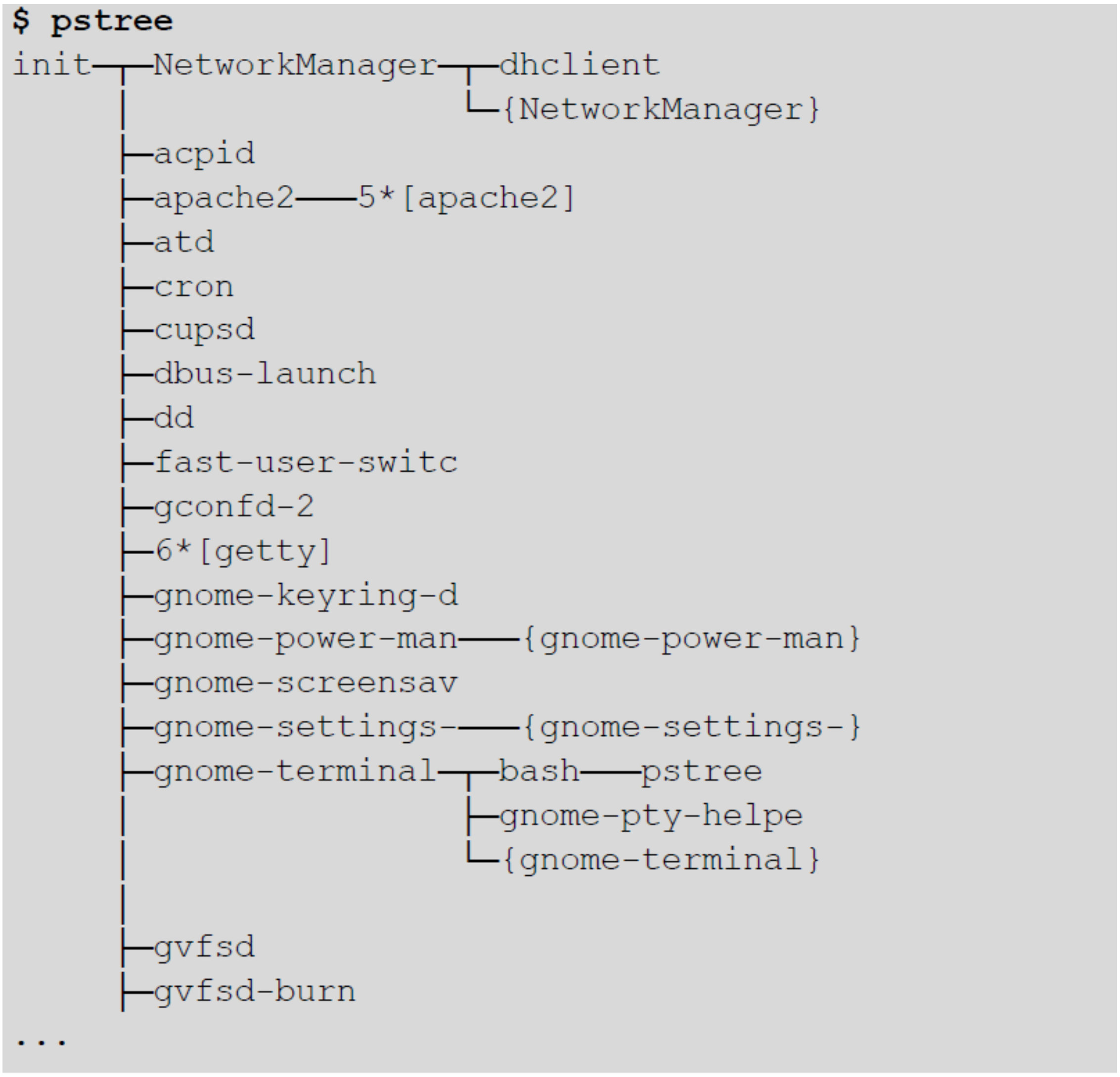
Common usage examples:

`pgrep [NAME]`	Display PIDs matching the specified name
`pgrep -l [NAME]`	Display the process name in addition to the PID
`pgrep -P [PPID]`	Display all child processes of the specified PPID
`pgrep -c [NAME]`	Display the total number of matching processes

pstree

Purpose: Display all running processes in a tree view.

Usage syntax: pstree [OPTIONS]



```
$ pstree
init─┬─NetworkManager─┬─dhclient
     │                └─{NetworkManager}
     ├─acpid
     ├─apache2───5*[apache2]
     ├─atd
     ├─cron
     ├─cupsd
     ├─dbus-launch
     ├─dd
     ├─fast-user-switc
     ├─gconfd-2
     ├─6*[getty]
     ├─gnome-keyring-d
     ├─gnome-power-man───{gnome-power-man}
     ├─gnome-screensav
     ├─gnome-settings-───{gnome-settings-}
     ├─gnome-terminal─┬─bash───pstree
     │                ├─gnome-pty-helpe
     │                └─{gnome-terminal}
     │
     ├─gvfsd
     ├─gvfsd-burn
...
```

Displaying a process tree using the pstree command

The pstree command draws a process tree for all processes currently running on the system, as shown in the above example. The output of pstree makes it easy to see the relationship between parent and child processes.

Common usage examples:

`pstree`	Display a basic process tree listing
`pstree -p`	Include PID numbers in the tree listing
`pstree -a`	Include command line options for each process
`pstree [USER]`	Display processes owned by the specified user
`pstree [PID]`	Display child processes of the specified PID

kill

Purpose: Terminate a process.

Usage syntax: kill [OPTIONS] [PID]

```
# pgrep -l mysqld
5540 mysqld
# kill 5540
```

Using the kill command to terminate a process

The kill command terminates the specified PID. In the above example, the mysqld process with a PID of 5540 is terminated using kill.

The default behavior of the kill command requests the process to gracefully exit. If a process fails to properly terminate, optional kill signals can be sent to force the process to end. The most commonly used kill signal in this situation is -9 which forcefully terminates the specified PID as displayed in the next example.

```
# pgrep -l mysqld
5545 mysqld
# kill -9 5545
```

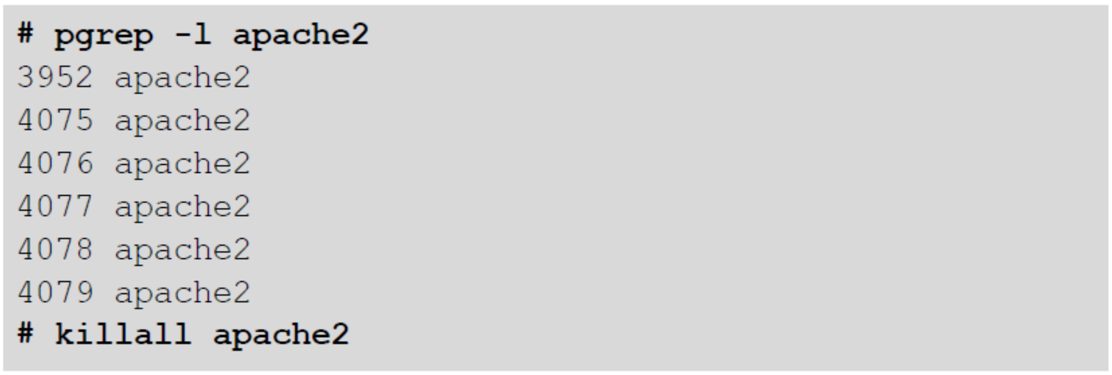
Using the -9 option to kill a hung process

Note	*It is recommended to only use the* **`kill -9`** *command in rare situations where a hung process refuses to gracefully exit. The* **`-9`** *signal is only used in these extreme situations, as it can cause undesired results such as system instability and "zombie" child processes.*

killall

Purpose: Terminate all processes with the specified name.

Usage syntax: killall [OPTIONS] [NAME]



```
# pgrep -l apache2
3952 apache2
4075 apache2
4076 apache2
4077 apache2
4078 apache2
4079 apache2
# killall apache2
```

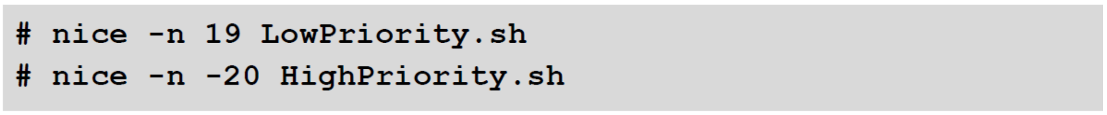
Terminating processes by name using the killall command

The killall command terminates all processes that match the specified name. This can be helpful if you have several processes with the same name that you need to kill. In the above example, multiple processes related to the apache2 service are terminated using killall.

nice

Purpose: Execute programs at the specified CPU priority level.

Usage syntax: nice [OPTIONS] [COMMAND]

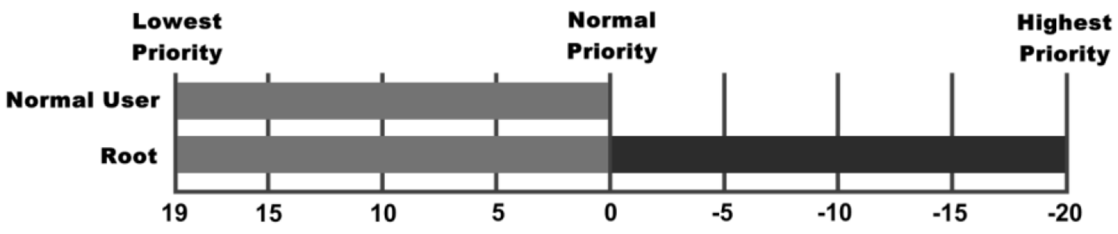


```
# nice -n 19 LowPriority.sh
# nice -n -20 HighPriority.sh
```

Changing the priority of a process using the nice command

The nice command allows programs to be started at a lower or higher than normal scheduling priority. This allows you to control which processes the kernel should favor when dividing processor time among running programs. Processes with the lowest nice number are executed with the highest priority and vice versa. The example above demonstrates using nice to start programs with modified scheduling priority.

On most systems, priority levels for normal users range from 0 to 19, with 0 being the highest priority and 19 being the lowest priority. The root user can create processes with a range of -20 (highest) to 19 (lowest).



Unix/Linux/BSD scheduling priority

Most programs start with a nice level of 0 by default. Critical system services usually have a negative nice level so that they are always given preference over user programs.

renice

Purpose: Alter the priority of a running process.

Usage syntax: renice [OPTIONS] [PID]

```
# renice +5 -p 7279
7279: old priority 0, new priority 5
```

Changing the priority of a process

The renice command adjusts the priority of a running process. In the above example PID 7279 is adjusted to have a +5 priority. This effectively lowers the priority of the process since the previous priority was 0.

&

Purpose: Start a process in the background.

Usage syntax: [COMMAND] &

```
$ ./SomeProgram.sh &
[1] 10717
$ ps
 PID TTY          TIME CMD
10696 pts/0    00:00:00 bash
10717 pts/0    00:00:03 SomeProgram.sh
10721 pts/0    00:00:00 ps
```

Executing a program in the background with the & command line operator

& is a command line operator that instructs the shell to start the specified program in the background. This allows you to have more than one program running at the same time without having to start multiple terminal sessions. In the above example, a shell script called SomeProgram.sh is started as a background process. Executing the ps command shows the requested program is now running in the background.

bg / fg

Purpose: Move a process to the background/foreground.

Usage syntax: bg [JOBID]

```
$ ./SomeProgram.sh
<CTRL + Z>
[1]+  Stopped                SomeProgram.sh
$ bg 1
$ jobs
[1]-  Running                 SomeProgram.sh
```

Suspending a running program and then resuming it in the background

The bg command sends a process to the background. This allows you to multitask several programs within a single terminal session. In the above example the shell script SomeProgram.sh is suspended (by pressing CTRL + Z on the keyboard) and then resumed as a background process using bg.

The fg command moves background processes to the foreground. In the next example the fg command is used to bring the specified job ID to the foreground.

Usage syntax: `fg [JOBID]`

```
$ jobs
[1]-  Running                    SomeProgram.sh
$ fg 1
SomeProgram.sh
...
```

Moving a background job to the foreground

jobs

Purpose: Display background and suspended jobs.

Usage syntax: jobs [OPTIONS]

```
$ jobs
[1]-  Running                    SomeProgram.sh
[2]+  Stopped                    AnotherProgram.sh
[3]   Stopped                    Test.sh
```

Displaying background programs using the jobs command

The jobs command displays the status of background programs and suspended processes. In the above example, three background jobs are displayed. The previously mentioned fg and bg commands can be used to move the processes to the foreground and background respectively.

The following table describes the output fields of the jobs command.

Job ID	Status	Program
[1]-	Running	SomeProgram.sh
[2]+	Stopped	AnotherProgram.sh
[3]	Stopped	Test.sh

Output fields of the jobs command

nohup

Purpose: Run a process immune to hang-up signals.

Usage syntax: nohup [COMMAND] &

```
$ nohup SomeProgram.sh &
nohup: ignoring input and appending output to 'nohup.out'
$ exit
```

Using the nohup command to launch a background program

The nohup command makes processes immune to hang-up signals. A hang-up signal is used to inform child processes of a shell that the parent process is terminating. This would normally cause all child processes to terminate. Using nohup allows a program to continue running after you log out, as demonstrated in the above example.

at / atq / atrm

Purpose: Schedule a program to run at the specified time.

Usage syntax: at [OPTIONS] [TIME|DATE]

```
$ at 1am
at> MyProgram.sh
at> <CTRL + D>
job 1 at Sat May 30 01:00:00 2009
```

Scheduling a process using the at command

The at command schedules programs to run at the specified time. In the above example, the at command is used to launch a shell script called MyProgram.sh. The launch time in this example is specified as 1am on the command line. You can also use more traditional Unix time specifications such as 01:00 or any other HH:MM combination.

The atq command displays information about queued at jobs as shown in the next example.

Usage syntax: atq

```
$ atq
1       Sat May 30 01:00:00 2009 MyProgram.sh nick
```

Using the atq command to display the job queue

The atrm command deletes a scheduled job as shown in the next example.

Usage syntax: atrm [JOBID]

```
$ atrm 1
```

Removing a scheduled job using the atrm command

batch

Purpose: Schedule programs to run during low CPU load.

Usage syntax: batch [OPTIONS]

```
$ batch
at> BigProgram.sh
at> <CTRL + D>
job 1 at Fri May 29 20:01:00 200
```

Scheduling a batch process

The batch command schedules a program to run when the system CPU load is low. This is useful for running resource-intensive programs that would normally interfere with system performance.

In this example the batch program is used to schedule the BigProgram.sh script to run when the system CPU load is low.

crontab

Purpose: Schedule programs to run at the specified time(s).

Usage syntax: crontab [OPTIONS]

```
# crontab -l
30 12 * * 1-5 /root/SomeProgram.sh
```

Displaying configured cron jobs

The crontab command manages a user's scheduled cron jobs. Cron is a subsystem found on most Unix, Linux, and BSD systems that schedules programs to run at a specific interval. It differs from the previously discussed at command because cron jobs run at reoccurring intervals (where at jobs run only once).

In the above example, the -l option is used to display the current user's crontab. The table below describes the fields found in the crontab file.

Minute	Hour	Day of Month	Month	Day of Week	Program
30	12	*	*	1-5	/root/SomeProgram.sh

Crontab fields

In this example SomeProgram.sh is run every Monday-Friday at 12:30 PM. The day of week is symbolized as 0=Sunday, 1=Monday, 2=Tuesday, etc. An asterisk (*) is a wild card that represents all valid values.

crontab -e is used to edit the current user's crontab as demonstrated in the next example.