Lecture 5: Jobs and Processes

CS2042 - UNIX Tools

October 8, 2008

Lecture Outline

Intro to Processes

Definition:

A *process* is an instance of a running program.

- More specific than "a program" because it's being executed!
- More specific than "a running program" because the same program can be run multiple times simultaneously!

Example:

Many users could be simultaneously running **ssh** to connect to other servers. In this case, each *instance* of **ssh** is a separate process.

Process Identification

How do we tell one process from another?

- Each process is assigned a unique "Process ID" (or PID) when it is created.
- These PIDs are used to differentiate between separate instances of the same program.

How do we find out which processes are running, and with which PIDs?

The Process Snapshot (PS) Command

ps [options]

- Reports a snapshot of the current running processes, including PID

More PS Options

Default:

ps

- Lists processes started by the user from the current terminal

All Processes:

ps -e

- Lists every process currently running on the system

Verbose:

ps -ely

- Gives more info about your processes than you'll ever need

See manpages for details - options for the BSD version are different!

Lecture Outline

Priority

Even though UNIX seems to run tens or hundreds of processes at once, one CPU can only run one process at a time. Quick switching back and forth between processes makes it seem as though they are running simultaneously.

- What if we need a process to get more (or less) time on the CPU than others?
- The UNIX developers saw this coming - each process is given a *priority* value when it starts.

Initial Priority

Start a new process with a non-default priority:

The Nice Command

nice [options] [command]

- Runs [command] with a specified "niceness value" (default: 0)
- Niceness values range between -20 (highest priority) and 19 (lowest priority)
- Only root can give a process a negative niceness value!

Example:

nice -n 10 azureus

- Keeps torrent downloads from hogging all our CPU time!

Adjusting Priority

Adjust the niceness of a running process:

The Renice Command

renice <priority> -p <PID>

- Changes the niceness of the indicated process to <priority>
- Again, only root can go below 0!
- Can only renice processes YOU started!

Example:

renice 5 -p 10275

- Sets the niceness of the process with PID 10275 to 5 (slightly lower priority than default)

Lecture Outline

Rogue Processes

Ctrl+C ends programs that are running in the foreground.

- What about a background process that stops working?
- What is the UNIX version of CTRL+ALT+DELETE?

The Kill Command

Kill

kill [-signal] <PID>

- Sends the specified signal to the process
- By default, terminates execution

So, to terminate a process:

- Look up the process's PID with **ps**
- Use that PID to **kill** the process!

Useful Signals

Signals used with **kill** can either be specified by their names or numeric values.

- TERM or 15 : Terminates execution (default)
- HUP or 1 : Hang-up (restarts the program)
- KILL or 9 : Like bleach, can kill anything

Generally speaking, the default TERM will get the job done.

Example:

- **kill 9000** : Terminates process 9000
- **kill -9 3200** : REALLY kills PID 3200
- **kill -HUP 12118** : Restarts 12118 (handy for servers & daemons)

Lecture Outline

Intro to Job Control

Jobs

A *job* is a process running under the influence of a job control facility.

- What does this mean?!

- Job control is a built-in feature of most shells, allowing the user to pause and resume tasks, as well as run them in the background (so that the shell is usable while it executes!)

How Does This Help?

Let's use the **ping** command as an example.

Ping

ping <server>

- Measures the network response time (or network latency) to a remote server.
- Sends short bursts to the server, then measures the time until they are returned.
- Can be a good way to check your network connection.

Try pinging a reliable location, like **google.com**!

Example:

ping google.com

Why We Need Job Control

As long as **ping** runs, we lose control of our shell. This happens with many applications which run either indefinitely or for long periods:

- Moving large quantities of files
- Compiling source code
- Playing multimedia

Example:

play mixtape.mp3

- Takes over the shell until the mp3 finishes
- To listen to music while we work, we need job control!

Lecture Outline

Starting a Job in the Background

To run a job in the background, we will use a new command-line operator: the ampersand (&)

&

Usage: <**command**> **[arguments] &**

- Runs the specified command as a background job
- Unless told otherwise, will send output to the terminal!

Since **cat** runs indefinitely with no arguments, this will illustrate our point:

Example:

cat &

- Try it without the &!

Backgrounding a Running Job

What if we started a process normally and it's taking too long?

Pausing a Job

Press CTRL+Z to pause a running process!

- When we do this, the shell tells us the paused job's Job ID
- This Job ID is used like a process's PID
- Once we have a process paused, we can tell it to continue in the background

The Background Command

bg's Usage

bg <Job ID>

- Resumes a paused job in the background

How do we find these Job IDs?

The Job Table

jobs

- Prints currently running, paused, or recently stopped

jobs

- Prints jobs with their Job IDs

Foregrounding an Existing Job

What if we want to resume a job in the foreground?

fg's usage

fg <Job ID>

- Resumes a paused job in the foreground

To kill a job, either foreground it and then hit CTRL+C, or you can use the **kill** command with the Job ID.

Dealing with Excess Output

Certain programs output continuously as they run. For example, **ping** and **play** both clutter up the terminal with output even when they are backgrounded.

- The solution? Output redirection!

Example:

ping google.com > testping.log &

- When you care about a program's output, redirect it to a log file.

Example:

play a_song.mp3 > /dev/null &

- If the text output doesn't matter, redirect it to */dev/null*.

/dev/null

/dev/null is a special file which has the following properties:

- Any user can write to it.
- Anything written to it goes nowhere.
- It always reports a successful write.

It works like a black hole for data - you can output to it all day and it will never fill up. Anything you redirect to */dev/null* just disappears.