

# 07 - Processes and Jobs

CS 2043: Unix Tools and Scripting, Spring 2016 [1]

---

Stephen McDowell

February 10th, 2016

Cornell University

# Table of contents

1. Processes Overview
2. Modifying Processes
3. Jobs
4. Job Control Demo

## Some Logistics

- HW1 due Friday, 2/12/2016 at 5pm

## Some Logistics

- HW1 due Friday, 2/12/2016 at 5pm
- Drop deadline is today.

# Some Logistics

- HW1 due Friday, 2/12/2016 at 5pm
- Drop deadline is **today**.
- **Lecture-demo solutions...thanks Joe!**

## Some Logistics

- HW1 due Friday, 2/12/2016 at 5pm
- Drop deadline is today.
- Lecture-demo solutions...thanks Joe!
- The nature of the material in this topic basically dictates not covering OSX. They may exist, they may not.

# Some Logistics

- HW1 due Friday, 2/12/2016 at 5pm
- Drop deadline is today.
- Lecture-demo solutions...thanks Joe!
- The nature of the material in this topic basically dictates not covering OSX. They may exist, they may not.
  - They may also give very different results.

## Processes Overview

---



# What is a Process?

- A process is just an instance of a running program.

# What is a Process?

- A process is just an instance of a running program.
- Not just a "program" - it is being *executed*.

# What is a Process?

- A process is just an instance of a running program.
- Not just a "program" - it is being *executed*.
- Not just a "running program", as you can execute the same program multiple times.

# What is a Process?

- A process is just an instance of a running program.
- Not just a "program" - it is being *executed*.
- Not just a "running program", as you can execute the same program multiple times.
  - These would be multiple processes running an instance of the same program.

# What is a Process?

- A process is just an instance of a running program.
- Not just a "program" - it is being *executed*.
- Not just a "running program", as you can execute the same program multiple times.
  - These would be multiple processes running an instance of the same program.
- Example: if you open more than one terminal (windows or tabs), you are running multiple processes of your shell.

# What is a Process?

- A process is just an instance of a running program.
- Not just a "program" - it is being *executed*.
- Not just a "running program", as you can execute the same program multiple times.
  - These would be multiple processes running an instance of the same program.
- Example: if you open more than one terminal (windows or tabs), you are running multiple processes of your shell.
  - You can execute **echo \$\$** to see the process of the current running shell.

- Processes have a unique "Process ID" (PID) when created.

# Identification

- Processes have a unique "Process ID" (PID) when created.
- The PID allows you to distinguish between multiple instances of the same program.



# Identification

- Processes have a unique "Process ID" (PID) when created.
- The PID allows you to distinguish between multiple instances of the same program.
- There are countless ways to discover the PID, as well as what processes are running.

# Identification

- Processes have a unique "Process ID" (PID) when created.
- The PID allows you to distinguish between multiple instances of the same program.
- There are countless ways to discover the PID, as well as what processes are running.
- These methods often depend on how much information you want, as well as what your user privileges are.

## Identification: **ps**

### Process Snapshot

**ps** [options]

- Reports a snapshot of the current running processes, including PIDs.
- By default, only the processes started by the user.
- Use **-e** to list every process currently running on the system.
- Use **-ely** to get more information than you can handle.
- Use **-u <username>** to list all processes for user **username**.
- **Note:** very different for BSD/OSX, read the man page...

# Identification: **ps**

## Process Snapshot

**ps** [options]

- Reports a snapshot of the current running processes, including PIDs.
- By default, only the processes started by the user.
- Use **-e** to list every process currently running on the system.
- Use **-ely** to get more information than you can handle.
- Use **-u <username>** to list all processes for user **username**.
- **Note:** very different for BSD/OSX, read the man page...

- To see more information about a process, pipe through **grep**.

# Identification: **ps**

## Process Snapshot

**ps** [options]

- Reports a snapshot of the current running processes, including PIDs.
  - By default, only the processes started by the user.
  - Use **-e** to list every process currently running on the system.
  - Use **-ely** to get more information than you can handle.
  - Use **-u <username>** to list all processes for user **username**.
  - **Note:** very different for BSD/OSX, read the man page...
- To see more information about a process, pipe through **grep**.
  - For example: **ps -e | grep firefox** shows us the results about **firefox** processes.

# Identification: **lsof**

## List of Open Files

### `lsof [options]`

- Very similar to **ps**, with more information by default.
- Frequently used for monitoring port connections...
- Use **-i** to list IP sockets.
  - E.g. **lsof -i tcp:843** shows all tcp processes on port **843**.
- Many options...read the man page if you are intrigued.

# Identification: **lsof**

## List of Open Files

### **lsof** [options]

- Very similar to **ps**, with more information by default.
- Frequently used for monitoring port connections...
- Use **-i** to list IP sockets.
  - E.g. **lsof -i tcp:843** shows all tcp processes on port **843**.
- Many options...read the man page if you are intrigued.

- As with **ps**, often best served with a side of **grep**.

# Identification: **lsnf**

## List of Open Files

### **lsnf** [options]

- Very similar to **ps**, with more information by default.
- Frequently used for monitoring port connections...
- Use **-i** to list IP sockets.
  - E.g. **lsnf -i tcp:843** shows all tcp processes on port **843**.
- Many options...read the man page if you are intrigued.

- As with **ps**, often best served with a side of **grep**.
- More useful for administration, especially when managing a networked environment.



## Display and Update **top** CPU Processes

**top** [**options**]

- Displays the amount of resources in percentages each process is using.
- Use **-d <seconds>** to control the update frequency.
  - The act of monitoring is an expensive process...
- Use **-u <user>** to show only the processes owned by **user**.
- Use **-p <PID>** to show only the statistics on process with id number **PID**.

# Resource Usage

## Display and Update **top** CPU Processes

**top** [**options**]

- Displays the amount of resources in percentages each process is using.
- Use **-d <seconds>** to control the update frequency.
  - The act of monitoring is an expensive process...
- Use **-u <user>** to show only the processes owned by **user**.
- Use **-p <PID>** to show only the statistics on process with id number **PID**.

- When used in conjunction with **ps** or **lsof**, can be a very powerful analysis tool.

# Resource Usage

## Display and Update **top** CPU Processes

**top** [options]

- Displays the amount of resources in percentages each process is using.
- Use **-d <seconds>** to control the update frequency.
  - The act of monitoring is an expensive process...
- Use **-u <user>** to show only the processes owned by **user**.
- Use **-p <PID>** to show only the statistics on process with id number **PID**.

- When used in conjunction with **ps** or **lsuf**, can be a very powerful analysis tool.
- Example sequence on the next page.

## Example: Resource Monitoring

```
>>> ps -e | grep firefox
12975 ?          00:01:45 firefox
>>> top -p 12795
top - 09:37:56 up 1 day, 13:52,  5 users,  load average: 0.19, 0.20, 0.19
Tasks:  1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.1 us,   0.5 sy,   0.0 ni, 98.4 id,   0.0 wa,   0.0 hi,   0.0 si,   0.0 st
KiB Mem : 16386660 total, 5990760 free, 3562320 used, 6833580 buff/cache
KiB Swap: 4194300 total, 4194300 free,      0 used. 12551476 avail Mem
   PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM     TIME+ COMMAND
 12975 sven      20   0 1437888 396868 105116 S   1.7   2.4   1:46.39 firefox
```

- You'll be best off reading through the man page to understand everything going on here.

## Example: Resource Monitoring

```
>>> ps -e | grep firefox
12975 ?          00:01:45 firefox
>>> top -p 12795
top - 09:37:56 up 1 day, 13:52,  5 users,  load average: 0.19, 0.20, 0.19
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.1 us,   0.5 sy,   0.0 ni, 98.4 id,   0.0 wa,   0.0 hi,   0.0 si,   0.0 st
KiB Mem : 16386660 total, 5990760 free, 3562320 used, 6833580 buff/cache
KiB Swap: 4194300 total, 4194300 free,      0 used. 12551476 avail Mem
  PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM    TIME+  COMMAND
 12975 sven       20   0 1437888 396868 105116 S   1.7   2.4   1:46.39 firefox
```

- You'll be best off reading through the man page to understand everything going on here.
- Some great examples in [3].

## Example: Resource Monitoring

```
>>> ps -e | grep firefox
12975 ?          00:01:45 firefox
>>> top -p 12975
top - 09:37:56 up 1 day, 13:52,  5 users,  load average: 0.19, 0.20, 0.19
Tasks:  1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.1 us,   0.5 sy,   0.0 ni, 98.4 id,   0.0 wa,   0.0 hi,   0.0 si,   0.0 st
KiB Mem : 16386660 total, 5990760 free, 3562320 used, 6833580 buff/cache
KiB Swap: 4194300 total, 4194300 free,      0 used. 12551476 avail Mem
  PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM    TIME+  COMMAND
 12975 sven      20   0 1437888 396868 105116 S   1.7   2.4   1:46.39 firefox
```

- You'll be best off reading through the man page to understand everything going on here.
- Some great examples in [3].
  - I've found myself on that website *many* times, he has a lot of excellent examples about a large quantity of topics.

## Example: Resource Monitoring

- Now I have opened about thirty tabs in firefox, and we get much different results:

## Example: Resource Monitoring

- Now I have opened about thirty tabs in firefox, and we get much different results:
- Look at the cpu usage!



## Example: Resource Monitoring

- Now I have opened about thirty tabs in firefox, and we get much different results:
- Look at the cpu usage!

```
>>> top -p 12795
top - 09:43:09 up 1 day, 13:57,  5 users,  load average: 1.33, 0.75, 0.41
Tasks:  1 total,   1 running,   0 sleeping,   0 stopped,   0 zombie
%Cpu(s): 13.4 us,  3.3 sy,   0.0 ni, 83.2 id,   0.0 wa,   0.0 hi,   0.0 si,   0.0 st
KiB Mem : 16386660 total, 3622768 free, 5679500 used, 7084392 buff/cache
KiB Swap: 4194300 total, 4194300 free,    0 used. 10300816 avail Mem

   PID USER      PR  NI   VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 12975 sven      20   0 3451396 1.372g 133688 R   75.7   8.8   5:00.96 firefox
```

## Example: Resource Monitoring

- Now I have opened about thirty tabs in firefox, and we get much different results:
- Look at the cpu usage!

```
>>> top -p 12795
top - 09:43:09 up 1 day, 13:57,  5 users,  load average: 1.33, 0.75, 0.41
Tasks:   1 total,   1 running,   0 sleeping,   0 stopped,   0 zombie
%Cpu(s): 13.4 us,  3.3 sy,   0.0 ni, 83.2 id,   0.0 wa,   0.0 hi,   0.0 si,   0.0 st
KiB Mem : 16386660 total, 3622768 free, 5679500 used, 7084392 buff/cache
KiB Swap: 4194300 total, 4194300 free,   0 used. 10300816 avail Mem

   PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM     TIME+ COMMAND
 12975 sven        20   0 3451396 1.372g 133688 R   75.7   8.8   5:00.96 firefox
```

- 75.7%?!!! Pretty common actually, this is why I always tell you to use your browser inside your Virtual Machine...

## Modifying Processes

---

- Suppose you want to run some long calculation that might take days, but would consume 100% of your CPU.

# Priority

- Suppose you want to run some long calculation that might take days, but would consume 100% of your CPU.
- Can we tell the server to give your process less priority in terms of CPU time?

# Priority

- Suppose you want to run some long calculation that might take days, but would consume 100% of your CPU.
- Can we tell the server to give your process less priority in terms of CPU time?
- Recall that although Unix seems to run tens or hundreds of processes at once, one CPE can only run one process at a time\*.

# Priority

- Suppose you want to run some long calculation that might take days, but would consume 100% of your CPU.
- Can we tell the server to give your process less priority in terms of CPU time?
- Recall that although Unix seems to run tens or hundreds of processes at once, one CPE can only run one process at a time\*.
- Quick switching back and forth between processes makes it seem as though they are all running simultaneously.

# Priority

- Suppose you want to run some long calculation that might take days, but would consume 100% of your CPU.
- Can we tell the server to give your process less priority in terms of CPU time?
- Recall that although Unix seems to run tens or hundreds of processes at once, one CPE can only run one process at a time\*.
- Quick switching back and forth between processes makes it seem as though they are all running simultaneously.
- The Unix masters anticipated this need, and each process was given a **priority** when it starts.



# Initial Priority

Start a process with a non-default priority:

## The **nice** command

**nice** [options] command

- Runs **command** with a specified "*nice*ness" value (default: 10).
- *Nice*ness values range from **-20** (highest priority) to **19** (lowest priority).
- Only **root** can give a process a *negative nice*ness value.
- Commands run without **nice** have priority **0**.

## Example

**nice -n 10 deluge**

- Keeps torrents from hogging the CPU.

# Adjusting Priority

The **renice** command

```
renice <priority> -p <PID>
```

- Changes the *nice*ness of the process with id **PID** to **<priority>**.
- Remember: only **root** can assign *negative* values.
- You can only **renice** a process you started.

Some Examples

```
renice 5 -p 10275
```

- Set the *nice*ness of the process with **PID 10275** to **5**
  - Slightly lower than normal *nice*ness

```
renice 19 -u sven
```

- Set the *nice*ness of **all** my processes to **19**

# Ending Processes: I

Sometimes you need to end a process.

## kill

```
kill [-signal] <PID>
```

- Sends the specified **signal** to the process with id **PID**.
- By default, it terminates execution.

## killall

```
killall [-signal] <name>
```

- Kills processes by name.
- E.g. `killall firefox`.

**Note:** These are dangerous commands, and should generally be last resorts.

# Useful Kill Signals

- Kill signals can be used by number or name.

# Useful Kill Signals

- Kill signals can be used by number or name.
- **TERM** or **15**: terminates execution (default).

# Useful Kill Signals

- Kill signals can be used by number or name.
- **TERM** or **15**: terminates execution (default).
- **HUP** or **1**: hang-up (restarts the program).

# Useful Kill Signals

- Kill signals can be used by number or name.
- **TERM** or **15**: terminates execution (default).
- **HUP** or **1**: hang-up (restarts the program).
- **KILL** or **9**: like bleach, can kill anything.

# Useful Kill Signals

- Kill signals can be used by number or name.
- **TERM** or **15**: terminates execution (default).
- **HUP** or **1**: hang-up (restarts the program).
- **KILL** or **9**: like bleach, can kill anything.
- Some examples:



# Useful Kill Signals

- Kill signals can be used by number or name.
- **TERM** or **15**: terminates execution (default).
- **HUP** or **1**: hang-up (restarts the program).
- **KILL** or **9**: like bleach, can kill anything.
- Some examples:

Killing 101

# Useful Kill Signals

- Kill signals can be used by number or name.
- **TERM** or **15**: terminates execution (default).
- **HUP** or **1**: hang-up (restarts the program).
- **KILL** or **9**: like bleach, can kill anything.
- Some examples:

## Killing 101

`kill 9009`: terminates process with **PID 9009**.

# Useful Kill Signals

- Kill signals can be used by number or name.
- **TERM** or **15**: terminates execution (default).
- **HUP** or **1**: hang-up (restarts the program).
- **KILL** or **9**: like bleach, can kill anything.
- Some examples:

## Killing 101

`kill 9009`: terminates process with **PID 9009**.

`kill -9 3223`: REALLY kills the process with **PID 3223**.

# Useful Kill Signals

- Kill signals can be used by number or name.
- **TERM** or **15**: terminates execution (default).
- **HUP** or **1**: hang-up (restarts the program).
- **KILL** or **9**: like bleach, can kill anything.
- Some examples:

## Killing 101

`kill 9009`: terminates process with **PID 9009**.

`kill -9 3223`: **REALLY** kills the process with **PID 3223**.

`kill -HUP 12221`: restarts the process with **PID 12221**.

# Useful Kill Signals

- Kill signals can be used by number or name.
- **TERM** or **15**: terminates execution (default).
- **HUP** or **1**: hang-up (restarts the program).
- **KILL** or **9**: like bleach, can kill anything.
- Some examples:

## Killing 101

`kill 9009`: terminates process with **PID 9009**.

`kill -9 3223`: **REALLY** kills the process with **PID 3223**.

`kill -HUP 12221`: restarts the process with **PID 12221**.

- very useful for servers and daemon processes.

# Useful Kill Signals

- Kill signals can be used by number or name.
- **TERM** or **15**: terminates execution (default).
- **HUP** or **1**: hang-up (restarts the program).
- **KILL** or **9**: like bleach, can kill anything.
- Some examples:

## Killing 101

`kill 9009`: terminates process with **PID 9009**.

`kill -9 3223`: **REALLY** kills the process with **PID 3223**.

`kill -HUP 12221`: restarts the process with **PID 12221**.

- very useful for servers and daemon processes.

- Remember **top**? You can both *renice* and *kill* processes from within it!

# Jobs



# What are Jobs?

## Jobs

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

- Job control is a built-in feature of most shells, allowing the user to pause and resume tasks.
- The user can also run them in the background.
- Not covered here: **crontab**. For the future sys admins, read the article in [2].



# Why do you want this?

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

## Ping

**ping** <server>

- Measures network response time (latency) to a remote server and back.
- Sends short bursts to the server, then measures time until they return.

## Example:

**ping google.com**

- Remember, **ctrl+c** kills the process.

## Why we Need Job Control

As long as **ping** runs, we lose control of our shell. This happens with many other applications.

## Why we Need Job Control

As long as **ping** runs, we lose control of our shell. This happens with many other applications.

- Moving large quantities of files.

# Why we Need Job Control

As long as **ping** runs, we lose control of our shell. This happens with many other applications.

- Moving large quantities of files.
- **Compiling source code.**

# Why we Need Job Control

As long as **ping** runs, we lose control of our shell. This happens with many other applications.

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

# Why we Need Job Control

As long as **ping** runs, we lose control of our shell. This happens with many other applications.

- Moving large quantities of files.
- Compiling source code.
- Playing multimedia.
- Scientific computing.

# Why we Need Job Control

As long as **ping** runs, we lose control of our shell. This happens with many other applications.

- Moving large quantities of files.
- Compiling source code.
- Playing multimedia.
- Scientific computing.
- etc.

# Why we Need Job Control

As long as **ping** runs, we lose control of our shell. This happens with many other applications.

- Moving large quantities of files.
- Compiling source code.
- Playing multimedia.
- Scientific computing.
- etc.

Example:

```
vlc
```



# Starting a Job in the Background

To run a job in the background, we will use a new operator:

**&**

`<command> [arguments] &`

- Runs the specified command as a background job.
- Unless told otherwise, will send output to the terminal!
- But at least we can type in our terminal again.

**Example:**

`vlc best_song_ever.flac &`

# Sending a Job to the Background

If you already started the job, but don't want to wait any more:

## Pausing a Job

Press **ctrl+z** to pause a running process!

# Sending a Job to the Background

If you already started the job, but don't want to wait any more:

## Pausing a Job

Press `ctrl+z` to pause a running process!

- Note this is still `ctrl` even on Mac...just like `ctrl+c`.

# Sending a Job to the Background

If you already started the job, but don't want to wait any more:

## Pausing a Job

Press `ctrl+z` to pause a running process!

- Note this is still `ctrl` even on Mac...just like `ctrl+c`.
- The shell will pause the jobs **JOB ID** (similar to **PID**).

# Sending a Job to the Background

If you already started the job, but don't want to wait any more:

## Pausing a Job

Press `ctrl+z` to pause a running process!

- Note this is still `ctrl` even on Mac...just like `ctrl+c`.
- The shell will pause the jobs **JOB ID** (similar to **PID**).
- We can bring it back.

# Revivals

## Background

`bg <JOB ID>`

- Resumes the job with id **JOB ID** in the *background*.
- Without **JOB ID**, resumes last job placed in background.

## Foreground

`fg <JOB ID>`

- Resumes the job with id **JOB ID** in the *foreground*.
- Without **JOB ID**, resumes last job placed in background.

## Discovering your **jobs**

`jobs`

- Prints the running, paused, or recently stopped jobs.
- Prints jobs with their **JOB IDs**.

## Dealing with Excess Output

- Many programs output continuously as they run. Try `vlc`. Pretty, but also annoying.

# Dealing with Excess Output

- Many programs output continuously as they run. Try `vlc`.  
Pretty, but also annoying.
- Redirect the output!



# Dealing with Excess Output

- Many programs output continuously as they run. Try `vlc`.  
Pretty, but also annoying.
- Redirect the output!
- Saving the output:

# Dealing with Excess Output

- Many programs output continuously as they run. Try `vlc`.  
Pretty, but also annoying.
- Redirect the output!
- Saving the output:

# Dealing with Excess Output

- Many programs output continuously as they run. Try `vlc`. Pretty, but also annoying.
- Redirect the output!
- Saving the output:

## Save `ping` results

```
ping google.com > testping.log &
```

- A `.log` file is common.

# Dealing with Excess Output

- Many programs output continuously as they run. Try `vlc`. Pretty, but also annoying.
- Redirect the output!
- Saving the output:

## Save **ping** results

```
ping google.com > testping.log &
```

- A `.log` file is common.
- Note you need to eventually end this **ping**!

# Dealing with Excess Output

- Many programs output continuously as they run. Try `vlc`. Pretty, but also annoying.
- Redirect the output!
- Saving the output:

## Save `ping` results

```
ping google.com > testping.log &
```

- A `.log` file is common.
- Note you need to eventually end this `ping`!

- Ignoring the output:

# Dealing with Excess Output

- Many programs output continuously as they run. Try `vlc`. Pretty, but also annoying.
- Redirect the output!
- Saving the output:

## Save `ping` results

```
ping google.com > testping.log &
```

- A `.log` file is common.
- Note you need to eventually end this `ping`!

- Ignoring the output:

```
# Should work in most Linux. Warning: non-POSIX compliant.  
>>> vlc best_song_ever.flac &> /dev/null &      # bash 4.0+  
# BSD/OSX/way out of date Linux:  
>>> vlc best_song_ever.flac > /dev/null 2>&1 & # before 4.0
```

# Detaching Jobs

When you launch jobs with an **&** and then close your terminal, those jobs will *end*.

## No Hangup

```
nohup <command> [args]
```

- Launches **command** so it will not end with **SIGHUP** signals.
- E.g. `nohup vlc best_song_ever.flac > /dev/null 2>&1 &`
  - Now we do not lose **vlc** when we close our terminal.

# Detaching Jobs

When you launch jobs with an `&` and then close your terminal, those jobs will *end*.

## No Hangup

```
nohup <command> [args]
```

- Launches **command** so it will not end with **SIGHUP** signals.
- E.g. `nohup vlc best_song_ever.flac > /dev/null 2>&1 &`
  - Now we do not lose **vlc** when we close our terminal.

If you have already launched the job, you can still save it.

## Disown a Job

```
disown [flags] jobspec
```

- The **-h** flag prevents **jobspec** from **SIGHUP** killing it.
- The **jobspec** is the job number (e.g. run **jobs**).
- E.g. if **jobID** 1 is **vlc**, then `disown -h %1` will work.



## Job Control Demo

---

I did a demo on-the-fly in class demonstrating job control, pausing, resuming, etc. I encourage you to follow the *ex post facto* demo here:

<https://github.com/cs2043-sp16/lecture-demos/tree/master/lec07>

## References

- [1] B. Abrahao, H. Abu-Libdeh, N. Savva, D. Slater, and others over the years.

Previous cornell cs 2043 course slides.

- [2] C. Hope.

Linux and unix crontab command help and examples.

`http:`

`//www.computerhope.com/unix/ucrontab.htm.`

- [3] R. Natarajan.

Can you top this? 15 practical linux top command examples.

`http://www.thegeekstuff.com/2010/01/`

`15-practical-unix-linux-top-command-examples/.`