More Processes
Taking Care of the Undead

Benjamin Brewster

First two slides adapted from Jon Herlocker, OSU

Is your cpu too slow?

- Imagine you have a program that is taking a long time to complete - is it because of your CPU?
- If you have one processor, then you can only have one process in the "running" state at any time
- Processes in the "waiting" state are not consuming cpu resources
- However, we can look at the number of "runnable" processes
  - This is the number of processes waiting for CPU only

Diagnosing a slow CPU

- The *uptime* command shows the average number of runnable processes over several different periods of time

```
% uptime
1:23pm up 25 day(s), 5:59, 72 users, load average: 0.18, 0.19, 0.20
```

- This shows the average number of runnable processes over that last 1, 5 and 15 minutes

- If uptime is showing that your runnable queue is consistently longer than 2 processes, your CPU is a bottleneck

Bounding

- Memory Bound
  - The process you're running is limited by your available (or total) RAM
- CPU Bound
  - The process you're running is limited by the speed of your processor
  - This is the normal case

Running Processes

- How can we tell which processes are running?

- Use the `ps` command

ps

Note: no ‘-’

```
% ps x
PID   TTY      STAT   TIME COMMAND
13703 ?        S      0:00 sshd: brewstbe@pts/1
13704 pts/1    Ss     0:00 -csh
14613 pts/1    R+     0:00 ps x
```

- PID: process identification number
- TTY: controlling terminal of the process
- STAT: state of the job
- TIME: amount of CPU time the process has acquired so far
- COMMAND: name of the command that issued the process

ps - States

- Possible states for processes are:
  - O: The process is running
  - S: The process is blocked on I/O; sleeping
  - R: The process is ready to run
  - I: The process is idle; sleeping for > ~30 seconds
  - Z: The process is a *zombie*
  - T: The process is being traced/debugged by its parent; stopped via being placed in the background
  - X: The process is waiting for more memory

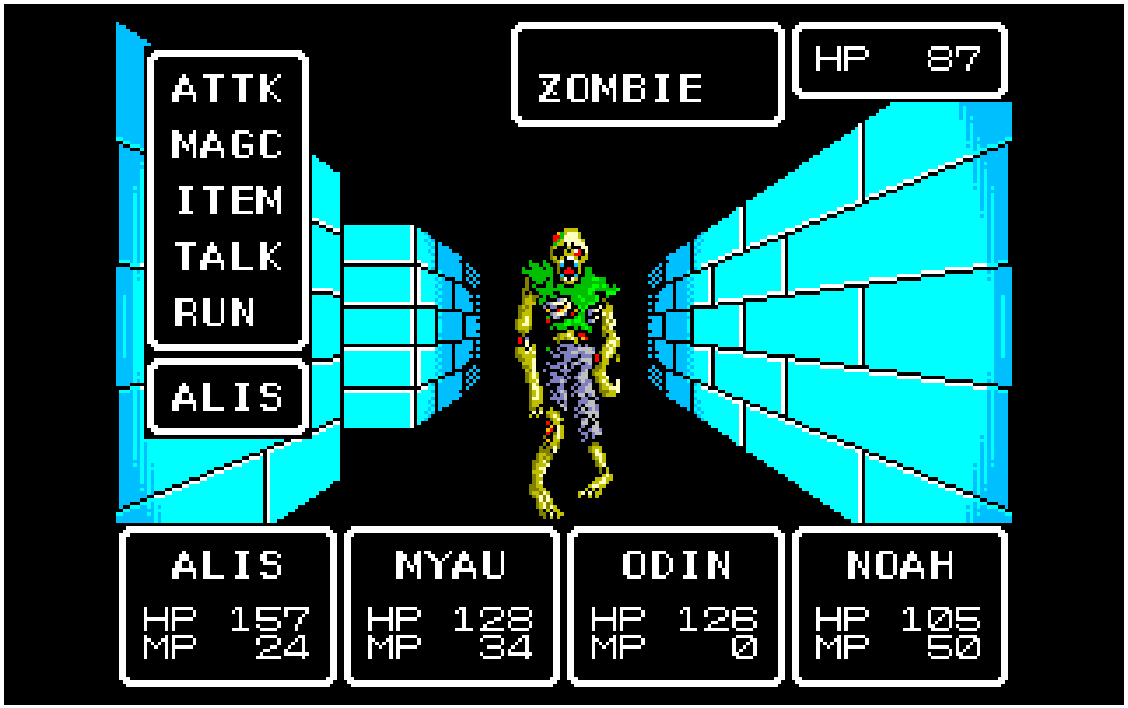
ps

```
% ps -u minoura
PID  TTY          TIME     CMD
5945 ?        00:00:00 sshd
5946 pts/0    00:00:00 bash
```

- `% ps aux`
  - Show me every single process being run – including those of other users (including root)

Zombie?

- When a process terminates, but its parent does not wait for it, it becomes a zombie



Zombies!?!

- Processes must report to their parents before they can terminate themselves

- If the parents aren't waiting for their children, the processes become the *living undead – forever enslaved to a non-life of waiting and watching.*

How to deal with Zombies

- Zombies stay in the system until they are waited for

How to deal with Zombies

- Zombies stay in the system until they are waited for

Orphan zombies!

- If the parent then terminates, without cleaning up its zombies, the zombies become orphans

Traditionally

- Orphans are adopted by the `init` process (pid = 1) which periodically waits for orphans

- Thus eventually, the orphan zombies die

`kill`

- Used to kill programs
  - another version is called `kfork`

- “kill” is really a misnomer – it really just sends signals

kill

- The first parameter is the signal to send
- The second parameter is the pid of the process being signaled
- The given pid affects who the signal is sent to:
  - If pid > 0, then the signal will be sent to the process with a process id of pid
  - If pid == 0, then the signal is sent to all processes in the same process group as the sender
  - more trickiness for pid < 0
- kill *–signal pid*
  - `kill -TERM 1234`
  - `kill -15 1234`
  - `kill -KILL 1234`
  - `kill -9 1234`

(See Readings for signal defs)

top

- `top` allows you to view the processes running on the machine in real time
- `top` demo
- `ps` demo

Job Control

- How do we start a program, and still retain access to the command line?
  - This ain’t no windowing system
- Can we run multiple processes at once?
- This is called Job Control in UNIX-speak

Foreground/Background

- There can be only one Foreground process – it's the one you're currently interacting with
  - If you're at the command prompt, then your foreground app is the shell itself
- Processes in the background can still be executing, but they can also be in any number of stopped states:
  - S, I, Z, T, X

Foreground

- Sending the TSTP signal stops (not terminates) a process, and puts it into the background
  - Control-z

```
% ping www.oregonstate.edu
PING www.orst.edu (128.193.4.112) (…)
64 bytes from www.orst.edu (…) ttl=62 time=0.319 ms
64 bytes from www.orst.edu (…) ttl=62 time=0.287 ms
64 bytes from www.orst.edu (…) ttl=62 time=0.295 ms

Suspended
%
```

- Our shell is again the foreground

jobs

- Use the `jobs` command to see what you're running:

```
% ping www.oregonstate.edu
PING www.orst.edu (128.193.4.112) (…)
64 bytes from www.orst.edu (128.193.4.112)(…) ttl=62 time=0.319 ms
64 bytes from www.orst.edu (128.193.4.112)(…) ttl=62 time=0.287 ms
64 bytes from www.orst.edu (128.193.4.112)(…) ttl=62 time=0.295 ms

Suspended
% jobs
[1]  + Suspended                    ping www.oregonstate.edu
% ps
PID   TTY      TIME     CMD
29916 pts/3    00:00:00 csh
31385 pts/3    00:00:00 ping
31584 pts/3    00:00:00 ps
```

`fg`

- Use the job numbers provided by `jobs` to manipulate processes
- Bring job 1 from the background to the foreground, and start it running
  - `fg %1`
- Bring most recent backgrounded job to the foreground, and start it running
  - `fg`

bg

- Start a specific, stopped program in the background (and keep it there)
  - `bg %1`
- Start the most recently stopped program in the background (and keep it there)
  - `bg`

You're suspended

- Suspend a program running in the background when you're at the shell

```
% ps
PID TTY          TIME CMD
29916 pts/3    00:00:00 csh
31385 pts/3    00:00:00 ping
32300 pts/3    00:00:00 ps
% kill -TSTP 31385
```

Start backgrounded

- Here's how to start a program in the background in the first place
  - `% ping www.oregonstate.edu &`
- The ampersand means to start in the background

Who's got control of stdout?

- Be advised – background processes can still write to any file – including stdout/stderr!
- Jobs demo:

1. `ping www.oregonstate.edu`
2. `CTRL-Z`
3. `jobs`
4. `fg %1`
5. `CTRL-Z`
6. `jobs`
7. `bg %1`
8. `ps`
9. `CTRL-Z`
10. `fg %1`
11. `CTRL-C`
12. `(disconnect)`
13. `ps x`
14. `kill -l`
15. `Kill -KILL` *`pid`*

history

- The history command provides a listing of your previous commands

```
% history 5
152  11:51   ping www.oregonstate.edu &
153  11:51   ps
154  11:52   kill -9 481
155  11:52   jobs
156  11:59   history 5
```

Execute a previous command

```
flip 157 % history 3
   155  11:52   jobs
   156  11:59   history 5
   157  12:02   history 3
flip 158 % !155
jobs
flip 159 % history 3
   157  12:02   history 3
   158  12:02   jobs
   159  12:03   history 3
flip 160 % !-2
jobs
flip 161% !!
jobs
flip 162 CS311% history 3
   160  12:03   jobs
   161  12:04   jobs
   162  12:06   history 3
```

Note no exclamation marks