CSE 265: System and Network Administration

- Controlling Processes
  - Components of a process
  - Life cycle of a process
  - Signals
  - Send signals using kill and killall
  - Process states
  - Influence scheduling priority with nice and renice
  - Monitoring processes with ps and top
  - Runaway processes
  - Periodic processes
Components of a process

- A process is the instantiation of a program
- From the kernel's perspective, a process is:
  - An address space (the set of memory pages with code, libraries, and data)
  - Set of data structures (within the kernel)
    - The process's address space map
    - Current status
    - Execution priority
    - Resources used
    - Signal mask (which signals are blocked)
    - The owner
    - Which instructions are currently being executed
Process attributes

- Process ID – PID
  - Unique identifier, wraps around
- Parent PID – PPID
  - When a process is cloned, there is a parent and a child
- Real and effective user ID – UID and EUID
  - EUID is used to determine what permissions the process has
  - Also records original EUID (saved UID)
    - Can be re-accessed later in program (even after changing EUID)
- Real and effective group ID – GID and EGID
- Niceness
  - The CPU time available depends on its scheduling priority
  - Users can make their processes 'nicer' to the rest of the system
- Control terminal – where stdin, stdout, stderr are attached
Process life cycle

- An existing process calls fork(2)
  - Parent is told PID of child
  - Child process is told 0
- Child can use exec (or similar) to start a new program
- When ready to die, process calls _exit(2) with exit code
  - Process becomes a zombie
- Parent must wait(2) to collect status of dead children
  - Resource usage, why killed
- Orphans are re-mapped to init
Signals

- Signals are process-level interrupt requests

- Uses
  - Inter-process communication
  - Terminal driver can kill, interrupt or suspend processes (Ctrl-C, Ctrl-Z)
  - Can be sent by admin (with kill) for various purposes
  - Can be sent by kernel when process breaks a rule
    - e.g., division by zero
  - Can be sent by kernel for i/o available, death of child
Handling signals

- Process can designate a signal handler for a particular signal
- If no handler, kernel takes some default action
- When handler is finished catching signal, execution continues where the signal was received
- Process can request that particular signals be ignored, or blocked
- If signal is received while blocked, one instance of that signal is buffered until it is unblocked
# Important signals

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Description</th>
<th>Default</th>
<th>Catch?</th>
<th>Block?</th>
<th>Dump?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HUP</td>
<td>Hangup</td>
<td>Terminate</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Reset request; clean up process on terminal (modem hangup)</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>csh processes ignore HUP; bash users need nohup command</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>INT</td>
<td>Interrupt</td>
<td>Terminate</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Control-C, can catch and clean up before quitting.</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>QUIT</td>
<td>Quit</td>
<td>Terminate</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Similar to TERM, but generates a core dump</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>KILL</td>
<td>Kill</td>
<td>Terminate</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Never received by process; OS terminates process.</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>BUS</td>
<td>Bus error</td>
<td>Terminate</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Error signal. Typically a memory alignment problem.</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>SEGV</td>
<td>Segmentation Fault</td>
<td>Terminate</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Error signal. Typically a memory access to protected space.</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## More signals

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Description</th>
<th>Default</th>
<th>Catch?</th>
<th>Block?</th>
<th>Dump?</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>TERM</td>
<td>Software termination</td>
<td>Terminate</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Request to terminate execution. Process can clean up, exit.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STOP</td>
<td>Stop</td>
<td>Stop</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* OS suspends execution of process until CONT received.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TSTP</td>
<td>Keyboard stop</td>
<td>Stop</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Keyboard Ctrl-Z request to stop. Catchable.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CONT</td>
<td>Continue after stop</td>
<td>Ignore</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Continue after STOP or TSTP.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WINCH</td>
<td>Window changed</td>
<td>Ignore</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Sent by terminal emulator when config changes (resize)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>USR1</td>
<td>User-defined</td>
<td>Terminate</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* User defined. Apache restarts gracefully.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>USR2</td>
<td>User-defined</td>
<td>Terminate</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* User defined. Apache restarts gracefully.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Sending signals

```bash
# kill [-signal] pid
# kill sends TERM signal by default
# kill -9 pid === kill -KILL pid
  "Guarantees" that the process will die
# kill -USR1 910 3044
# sudo killall -USR1 httpd
  killall removes need for pid
```
Process states

- Process exist in one of four states
  - Runnable – can be executed
  - Sleeping – waiting for some resources
    - Gets no CPU time until resource is available
  - Zombie – trying to die (parent hasn't waited)
  - Stopped – process is suspended (i.e., not permitted to run)
    - Like sleeping, but can't wake until CONT received
Scheduling priority

- “Niceness” is hint to kernel about how often to schedule the process
- Linux ranges from -20 (high priority, not nice) to +19 (low priority, very nice), 0 is default
- User/process can raise, but not lower niceness
  - Root can lower
- Examples
  - % nice +5 ~/bin/longtask
  - % renice -5 8829
  - % sudo renice 5 -u boggs
Monitoring processes: `ps`

- `/bin/ps` primary tool
- Shows
  - PID, UID, priority, control terminal
  - Memory usage, CPU time, status
- Multiple variations of `ps`
  - `ps -aux` (BSD, Linux)
  - `ps -Af` (Solaris)
### Example ps output

<table>
<thead>
<tr>
<th>USER</th>
<th>PID</th>
<th>%CPU</th>
<th>%MEM</th>
<th>VSZ</th>
<th>RSS</th>
<th>TTY</th>
<th>STAT</th>
<th>START</th>
<th>TIME</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>1</td>
<td>0.0</td>
<td>0.0</td>
<td>1364</td>
<td>64</td>
<td>?</td>
<td>S</td>
<td>2003</td>
<td>3:03</td>
<td>init [5] --init</td>
</tr>
<tr>
<td>root</td>
<td>2</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>SW</td>
<td>2003</td>
<td>1:35</td>
<td>[keventd]</td>
</tr>
<tr>
<td>root</td>
<td>3</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>SWN</td>
<td>2003</td>
<td>0:27</td>
<td>[ksoftirqd_CPU0]</td>
</tr>
<tr>
<td>root</td>
<td>5</td>
<td>0.1</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>SW</td>
<td>2003</td>
<td>465:05</td>
<td>[kswapd]</td>
</tr>
<tr>
<td>root</td>
<td>6</td>
<td>3.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>SW</td>
<td>2003</td>
<td>7754:49</td>
<td>[kscand]</td>
</tr>
<tr>
<td>root</td>
<td>7</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>SW</td>
<td>2003</td>
<td>1:16</td>
<td>[bdflush]</td>
</tr>
<tr>
<td>root</td>
<td>8</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>SW</td>
<td>2003</td>
<td>0:06</td>
<td>[kupdated]</td>
</tr>
<tr>
<td>root</td>
<td>9</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>SW&lt;</td>
<td>2003</td>
<td>0:00</td>
<td>[mdrecoveryd]</td>
</tr>
<tr>
<td>root</td>
<td>13</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>SW</td>
<td>2003</td>
<td>16:12</td>
<td>[kjournald]</td>
</tr>
<tr>
<td>root</td>
<td>92</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>SW</td>
<td>2003</td>
<td>0:00</td>
<td>[khubd]</td>
</tr>
<tr>
<td>root</td>
<td>589</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>SW</td>
<td>2003</td>
<td>0:01</td>
<td>[eth0]</td>
</tr>
<tr>
<td>root</td>
<td>761</td>
<td>0.0</td>
<td>0.0</td>
<td>1424</td>
<td>340</td>
<td>?</td>
<td>S</td>
<td>2003</td>
<td>0:48</td>
<td>syslogd -m 0</td>
</tr>
<tr>
<td>root</td>
<td>766</td>
<td>0.0</td>
<td>0.0</td>
<td>1364</td>
<td>244</td>
<td>?</td>
<td>S</td>
<td>2003</td>
<td>0:00</td>
<td>klogd -x</td>
</tr>
<tr>
<td>rpc</td>
<td>786</td>
<td>0.0</td>
<td>0.0</td>
<td>1524</td>
<td>360</td>
<td>?</td>
<td>S</td>
<td>2003</td>
<td>0:22</td>
<td>portmap</td>
</tr>
<tr>
<td>rpcuser</td>
<td>814</td>
<td>0.0</td>
<td>0.0</td>
<td>1660</td>
<td>484</td>
<td>?</td>
<td>S</td>
<td>2003</td>
<td>1:27</td>
<td>rpc.statd</td>
</tr>
<tr>
<td>ntp</td>
<td>933</td>
<td>0.0</td>
<td>0.0</td>
<td>1884</td>
<td>1880</td>
<td>?</td>
<td>S</td>
<td>2003</td>
<td>11:18</td>
<td>ntpd -U ntp -g</td>
</tr>
<tr>
<td>root</td>
<td>1045</td>
<td>0.0</td>
<td>0.0</td>
<td>2140</td>
<td>164</td>
<td>?</td>
<td>S</td>
<td>2003</td>
<td>0:00</td>
<td>xinetd -stayalive</td>
</tr>
<tr>
<td>root</td>
<td>1092</td>
<td>0.0</td>
<td>0.0</td>
<td>1796</td>
<td>176</td>
<td>?</td>
<td>S</td>
<td>2003</td>
<td>0:00</td>
<td>rpc.rquotad</td>
</tr>
<tr>
<td>root</td>
<td>1097</td>
<td>0.1</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>SW</td>
<td>2003</td>
<td>267:24</td>
<td>[nfsd]</td>
</tr>
<tr>
<td>root</td>
<td>1105</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>SW</td>
<td>2003</td>
<td>0:05</td>
<td>[lockd]</td>
</tr>
<tr>
<td>root</td>
<td>1113</td>
<td>0.0</td>
<td>0.0</td>
<td>1960</td>
<td>608</td>
<td>?</td>
<td>S</td>
<td>2003</td>
<td>0:02</td>
<td>rpc.mountd</td>
</tr>
<tr>
<td>root</td>
<td>1209</td>
<td>0.0</td>
<td>0.0</td>
<td>1560</td>
<td>288</td>
<td>?</td>
<td>S</td>
<td>2003</td>
<td>1:14</td>
<td>crond</td>
</tr>
<tr>
<td>daemon</td>
<td>1383</td>
<td>0.0</td>
<td>0.0</td>
<td>1408</td>
<td>200</td>
<td>?</td>
<td>S</td>
<td>2003</td>
<td>0:00</td>
<td>/usr/sbin/atd</td>
</tr>
<tr>
<td>root</td>
<td>1456</td>
<td>0.0</td>
<td>0.0</td>
<td>1348</td>
<td>116</td>
<td>tty2</td>
<td>S</td>
<td>2003</td>
<td>0:00</td>
<td>/sbin/mingetty tt</td>
</tr>
</tbody>
</table>
Monitoring processes: top

- `/usr/bin/top` is optional in some OSes
- Shows top-n CPU-using processes
  - Plus other stats, like memory usage and availability, system load
  - Can renice within top
  - Automatically refreshes screen every 5 seconds
  - Can focus on a particular user
Sample top output

```
top - 20:30:57 up 1 day, 22:48, 15 users,  load average: 0.04, 0.07, 0.05
Tasks: 163 total,  1 running, 162 sleeping,  0 stopped,  0 zombie
Cpu(s):   4.7%us,  1.5%sy,  0.0%ni,  93.5%id,  0.0%wa,  0.2%hi,  0.2%si,  0.0%st
Mem:  2073964k total, 1525460k used,  548504k free,  200188k buffers
Swap:  4194296k total,     0k used, 4194296k free,   798200k cached

    PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
10001 brian     15   0  104m  27m  15m S    0  0.1   0:52.50 rhythmbox
 5792 brian     15   0  362m 196m  27m S    5  9.7 172:39.93 firefox-bin
 5540 brian     15   0 17984 9112  65m S    3  0.4   0:49.05 metacity
 5406 root      15   0 136m 107m  11m S    3  5.3  44:58.77 Xorg
 5406 root      15   0 136m 107m  11m S    3  5.3  44:58.77 Xorg
 5406 root      15   0 136m 107m  11m S    3  5.3  44:58.77 Xorg
 5406 root      15   0 136m 107m  11m S    3  5.3  44:58.77 Xorg
```

Runaway processes

- What can you do about processes using an unusual amount of resources (memory, CPU, disk space)?
  - Identify resource hogs using `top` and/or `ps`
  - Contact owner and ask about resource usage
  - Suspend using STOP signal (might break job)
    - Contact owner, restart or kill later
  - Renice CPU hog
Creating periodic processes

- Automation, as you've heard, is key to efficiency
- Instead of manually performing tasks daily, weekly, or monthly, you can schedule them
  - cron
  - anacron
- Includes tasks like:
  - monitoring, log rotation, backups, file distribution
cron
cron daemon performs tasks at scheduled times
- crontab files are examined by cron for schedule
  - /etc/crontab, /etc/cron.d/*, /var/spool/cron/*
- cron wakes up each minute and checks to see if anything needs to be executed
- cron is susceptible to changes in time
  - doesn't compensate for when machine is down, or time changes (clock adjustments or daylight savings time) that are sufficiently large (3 hours, at least for some implementations)
- anacron works daily
  - records when task last performed, and will catch up with missing time
crontab files

- Filename provides username in /var/spool/cron/
- Example crontab entries:

  # run make at 2:30 each Monday morning
  30 2 * * 1 (cd /home/joe4/project; make)

[Diagram showing crontab values and meanings]

Managing crontabs

- Use `crontab -e` to edit
  - Checks out a copy
  - Uses EDITOR environment variable
  - Resubmits it to the `/var/spool/cron/` directory
- `crontab -l` will list the contents to stdout
- `/etc/cron.allow` and `/etc/cron.deny` can control access to cron facilities
Using cron

- Distributions set up crontab entries to automatically run scripts in
  - /etc/cron.monthly/
  - /etc/cron.weekly/
  - /etc/cron.daily/
  - /etc/cron.hourly/

- Typical tasks:
  - Cleaning the filesystem (editor files, core files) using find
  - Distributing files (mail aliases, sendmail config, etc.) using rsync, rdist, or expect
  - Log rotation