

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

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

More signals

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

Sending signals

```
# 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

```
USER      PID  %CPU  %MEM    VSZ   RSS TTY      STAT   START       TIME COMMAND
root         1   0.0   0.0   1364    64 ?        S      2003       3:03  init [5]    --init
root         2   0.0   0.0     0     0 ?        SW     2003       1:35  [keventd]
root         3   0.0   0.0     0     0 ?        SWN    2003       0:27  [ksoftirqd_CPU0]
root         5   0.1   0.0     0     0 ?        SW     2003    465:05  [kswapd]
root         6   3.0   0.0     0     0 ?        SW     2003    7754:49 [kscand]
root         7   0.0   0.0     0     0 ?        SW     2003       1:16  [bdflush]
root         8   0.0   0.0     0     0 ?        SW     2003       4:06  [kupdated]
root         9   0.0   0.0     0     0 ?        SW<    2003       0:00  [mdrecoveryd]
root        13   0.0   0.0     0     0 ?        SW     2003      16:12  [kjournald]
root        92   0.0   0.0     0     0 ?        SW     2003       0:00  [khubd]
root       589   0.0   0.0     0     0 ?        SW     2003       0:01  [eth0]
root       761   0.0   0.0   1424   340 ?        S      2003       0:48  syslogd -m 0
root       766   0.0   0.0   1364   244 ?        S      2003       0:00  klogd -x
rpc        786   0.0   0.0   1524   360 ?        S      2003       0:22  portmap
rpcuser    814   0.0   0.0   1660   484 ?        S      2003       1:27  rpc.statd
ntp        933   0.0   0.0   1884  1880 ?        SL     2003      11:18  ntpd -U ntp -g
root     1045   0.0   0.0   2140   164 ?        S      2003       0:00  xinetd -stayalive
root     1092   0.0   0.0   1796   176 ?        S      2003       0:00  rpc.rquotad
root     1097   0.1   0.0     0     0 ?        SW     2003    267:24  [nfsd]
root     1105   0.0   0.0     0     0 ?        SW     2003       0:05  [lockd]
root     1113   0.0   0.0   1960   608 ?        S      2003       0:02  rpc.mountd
root     1209   0.0   0.0   1560   288 ?        S      2003       1:14  crond
daemon    1383   0.0   0.0   1408   200 ?        S      2003       0:00  /usr/sbin/atd
root     1456   0.0   0.0   1348   116 tty2    S      2003       0:00  /sbin/mingetty tt
```

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
5792	brian	15	0	362m	196m	27m	S	5	9.7	172:39.93	firefox-bin
5540	brian	15	0	17984	9112	6532	S	3	0.4	0:49.05	metacity
5406	root	15	0	136m	107m	11m	S	3	5.3	44:58.77	Xorg
10001	brian	15	0	104m	27m	15m	S	0	1.4	0:52.50	rhythmbox
17511	brian	15	0	2168	1040	792	R	0	0.1	0:00.01	top
25759	root	5	-10	508m	158m	154m	S	0	2.0	74:54.98	vmware-vmx
17124	hadoop	21	0	1207m	15m	2716	S	0	0.2	7:46.71	java
17231	hadoop	15	0	1204m	12m	1304	S	0	0.2	1:55.97	java
25370	root	15	0	382m	4976	2428	S	0	0.1	7:50.96	vmplayer
2513	ntp	15	0	19116	4808	3716	S	0	0.1	0:04.37	ntpd
23138	root	15	0	84980	3184	2492	S	0	0.0	0:00.03	sshd
3184	root	12	-3	120m	1764	1196	S	0	0.0	0:01.83	python
1	root	15	0	2044	640	552	S	0	0.0	0:02.74	init
2	root	RT	0	0	0	0	S	0	0.0	0:00.00	migration/0
3	root	34	19	0	0	0	S	0	0.0	0:00.00	ksoftirqd/0
4	root	RT	0	0	0	0	S	0	0.0	0:00.00	watchdog/0
5	root	RT	0	0	0	0	S	0	0.0	0:00.00	migration/1

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)
```

EXECUTE BACKUP.SH SCRIPT EVERY SUNDAY AT 2:36 AM
36 2 * * 7 root /usr/local/sbin/backup.sh

VALUE RANGE	VALUE RANGE	VALUE RANGE	VALUE RANGE	VALUE RANGE		
0-59	0-23	1-31	1-12	0-7	root	/usr/local/sbin/backup.sh
						- COMMAND TO EXECUTE
						- EXECUTE COMMAND AS A USER ROOT
						- DAY OF WEEK: Sunday =0, Monday =1, Tuesday=2, Wednesday=3 Thursday=4, Friday=5, Saturday=6, Sunday=7
						- MONTH: January =1, February=2, March=3, April=4, May=5, June=6 July=7, August=8, September=9, October=10, November=11, December=12
						- DAY OF MONTH
						- HOUR
						- MINUTE

<http://www.notesbit.com/index.php/scripts-unix/crontab-quick-complete-reference-setting-up-cronjobs-in-unix-and-linux/>

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

