Processes (see chapters 23 and 24)

- Executables can be executed as processes
- Keyboard control of jobs
- `ps`, `top`, `pstree`
- `kill` doesn't kill, it sends signals
- `cron`, `anacron`

Processes come from executables

- A process has an entry in the process table, and is initially loaded from a file in the filesystem
- An executable is a file in the filesystem which
  - Has the appropriate “x” flag(s) set
  - Either begins with a line of the form `#!/SOME/OTHER/EXECUTABLE` or is in a binary format such as ELF or COFF

“Foreground” versus “Background”

- A process that is in the “foreground” of a shell means that the shell is waiting for the process to finish before accepting more input.
- A process that is in the “background” of a shell means that the shell will accept other commands while the process is executing. Generally, a “background” process can be brought to the “foreground”.

Shell communication with processes

- If a process is in the foreground, then by default when a ctrl-c is pressed and then mapped by stty to send a signal SIGINT, that SIGINT will be propagated to the foreground process. By default when a ctrl-z is pressed and then mapped by stty to send a signal SIGSTOP to the foreground process suspending the process. From there, you can either terminate it, put it in the background, or unsuspend it back to the foreground.
- If a process is in the background, you can use `kill` to explicitly send signals.

Shell job control

- You can place many processes simultaneously in the background; most shells will keep track of these and allow you to also access them via logical pids.
- You can either use ctrl-z / `bg` for a process that is in the foreground, or use a terminal “&” when you start the process.

Shell job control continued

- You can use `jobs` to keep up with which jobs you have running.
- You can use `fg %N` to bring job N back to the foreground.

kill

- Sending signals:
  - `kill -KILL pid` → "unstoppable" kill (aka `kill -9 pid`)
  - `kill -TERM pid` → terminate, usually much cleaner
  - `kill -HUP pid` → either reload or terminate, usually clean if termination
  - `kill -STOP pid` → suspend a process
  - `kill -CONT pid` → restart a suspended process
- `kill` is generally a built-in, but there is also usually a kill program. The program version will not usually work with logical pids (unless your shell happens to translate logical pids to real pids before invoking kill, or the kill program is written such that it reparses the command line. For example, try `/usr/bin/kill -STOP %1`).

`ps`

- You can also use `ps` to look at various portions of the process table.
- My favorites are `ps alxwww` and `ps -elf`.
- You pick and choose whatever format you like for output with the `ps -o -sort` option. For example, `ps -e -opid,uid,cmd -sort=uid`
- You can also show threads with the `ps -m` option.

top

- The program `top` gives you a dynamic view of the process table.
- You can make it run faster with the “s” command.
- You can do “snapshots” with the `-b` (batch) option and the `-i iterations` option.

pstree

- Shows processes as a tree. Some options are:
  - `-c` → Disable compaction.
  - `-G` → Try to make graphical line drawing rather than just character
  - `-Hpid` → Try to highlight a particular process and its ancestors
  - `-p` → Show pids
- You can limit output to a user (specified by a user name) or to pid (specified by pid number)

cron

- You can run programs at arbitrary times with `cron`
    - Use `crontab -e` to edit your crontab (you can set EDITOR to specify an editor)
    - The five time fields are `minute, hour, dayOfMonth, month, dayOfWeek` where Sunday=0 for dayOfWeek

- Example crontab:

```
COP4342$ crontab -l
#
10 0,3,6,18,21 * * * echo "test at `date +\%F-\%T`" |
                        Mail -s "`date +\%F-\%T`" langley@cs.fsu.edu
10 10 * * 0,6        echo "test at `date +\%F-\%T`" |
                        Mail -s "Weekend test `date +\%F-\%T`"
                        langley@cs.fsu.edu
```