Processes and Job Control

Foreground and Background (1)

- Unix is a multi-tasking operating system
  - some of these tasks are being done by other users logged in
  - some are being done by you in the background
    - e.g. watching for incoming mail
- When you run a task (a Unix command, like ls or vi) it executes in the foreground of your shell
  - it has the “control" of your screen and keyboard

Foreground and Background (2)

- If you still want to use the current shell
  - compute[1] > a_heavy_task &
  - [1] 13607
  - compute[2] >
- When you put a task in background
  - task keeps running, but you continue to work at the shell in the foreground
  - if any output is done, it appears on your screen immediately (can be confusing)
  - if input is required, process prints a message and stops
  - when it is done, a message will be printed

Foreground and Background (3)

- Explicit background jobs are needed less often with windowing systems
  - Just go to another window and run the command
- But explicit background jobs are still used often in Unix
  - A command needs a long time, you do not want to close that window by accident
  - Run a job at the background and logout
  - xterm& will open a new window, but leave the current shell window still available to use
    - (need ssh with X11 forwarding)

A Simple Script

- We use the following shell script to illustrate job control
- Edit a file make_noise

```
compute[2] > cat > make_noise
#!/bin/sh
while [ 1 ]
do
   date
   sleep 1
done
compute[3] > chmod u+x make_noise
```

- make_noise then is a shell script repeats to print the time for every second, until you terminate it using Ctrl-c.

Job Control – Suspending Jobs

- bash, csh, and tcsh allow you to manage the running of different processes
- Suspending jobs
  - the Ctrl-z special character stops the job

```
compute[4] > make_noise
Fri May 16 14:14:43 EDT 2003
……
^Z
[1] Stopped        make_noise
compute[5] > vi readme
^Z
[2] Stopped        vi readme
```

Job Control - Monitoring Jobs

- The "jobs" command shows which of your jobs are running and/or stopped.

```
compute[6] > jobs
[1]   Stopped                 make_noise
[2]   Stopped                 vi readme
```

- Here there are two suspended processes, the make_noise and a vi process.

Job Control – Resuming Jobs

- Putting jobs back into the foreground:
  - Use the "fg" command to move a job into the foreground.

    compute[7] > fg %2
  - Puts job number 2 into the foreground.
  - Works with either a background or stopped job.

- Putting jobs into the background:

    compute[8] > bg %1

Job Control – Killing Jobs

- Jobs can also be killed
  - Use the Unix "kill" command

    compute[9] > kill %1

    or if it won't die ...

    compute[10] > kill –9 %1 or kill –KILL %1
- Jobs can be stopped and continued

  compute[11] > a_heavy_task &

  [1] 26253

  compute[12] > kill –STOP %1 (or fg %1 then ^Z)

  compute[13] > bg %1

Using ps (1)

- Jobs are really just a special case of Unix processes that started by Unix shells
- ps can list the current processes

```
compute[14] > ps
PID   TTY     TIME        CMD
2312 pts/0     00:00:00   vi
2296 pts/0     00:00:00   bash
2313 pts/0     00:00:00   ps
```

- ps can take many options, depending on which version of ps you are using (/usr/bin/ps vs. /usr/ucb/ps)

Using ps (2)

- The ps command takes a number of options
  - –l gives you a long listing of what is going on
  - –u loginid tells you about loginid's processes
  - use man ps to see more options
- kill pid kills the process pid
  - TERM signal will be sent to the process pid
  - kill –9 or kill –KILL will send the KILL signal
  - Use man kill to find out more signals

Another useful command: ulimit

- The **ulimit** utility sets or reports the file-size writing limit imposed on files written by the shell and its child processes (files of any size may be read). User can decrease limit. Only a process with appropriate privileges can increase the limit.
  - –a prints all limits
  - –n maximum number of open file descriptors
  - –u maximum number of processes available
- Let us illustrate the interest of ulimit (not for every shell)

```
compute[15]> ulimit –u 15
compute[16]> more foo
 echo FOO; ./bar
compute[17]> more bar
 echo BAR; ./foo
compute[18]> ./foo
```