Lecture 7

Introduction to Process Management

COP 3353 Introduction to UNIX

*

MultiTasking, MultiProcess System

- The UNIX Operating System provides an environment in which multiple “processes” can run concurrently
  - Multitasking or multiprogramming: the ability to run multiple programs on the same machine concurrently
  - Multiprocessing: the ability to use multiple processors on the same machine - sometimes multiprocessing also used to indicate that multiple concurrent processes can execute at the same time in a single processor environment
- UNIX supports both multiprogramming and multiprocessing (in both senses)
  - This is implemented through the process abstraction
    - More recently supporting Light Weight Processes and Threads has also become the norm

The Process Abstraction

- In traditional systems a *process* executes a single sequence of instructions in an *address space.*
  - The program counter (PC) is a special hardware register that tracks the current instruction that is to be execute
  - In UNIX, many processes are active at the same time and the OS provides some aspects of a *virtual machine*
    - *Processes have their own registers and memory, but rely on the OS for I/O, device control and interacting with other processes*
  - Processes:
    - Run in a virtual address space
    - Content for resources such as processor(s), memory, and peripheral devices
    - All of the above is managed by the OS the memory management system; the I/O system; the process management and scheduling system, and the Interprocess Communication system (IPC)

More about a *Process - Just FYI.*

- Processes are created by the OS, typically by the *fork* command.
  - *The process that calls fork is the parent and the new process is the child.*
  - *The child inherits a replica of the parent's address space and is essentially a clone. Both continue to execute the identical program. Fork returns the child's process id to the parent, and the value 0 to the child.*
  - *The* exec *system call loads another program and starts running this (typically in the child process).*
- States of a Process
  - Initial, ready to run, running (in user mode or kernel mode), asleep, stopped, zombie (upon exit), orphaned (no parent). Finally, when all resources are freed by the parent, the process is terminated or no longer exists.

Basic job control commands

ps: displays information about processes
Some common options:*[-e] or [-A]: all processes*
[-l] longer version
[-f] full format listing
[-aux] more complete listing
&: running a process (job) in the background ex: sleep 30 &
jobs: shows you all your background processes
fg <job #>: puts a background job into the foreground
CTRL-z: pauses (suspends) a process
bg <job #>: puts a job into the background
CTRL-c: kill the foreground job
kill: kill a specific job (-9 typically kills most processes)

More job control commands

sleep: causes the current process to sleep for the time indicated.

example: Try these and see why they differ.

```
sleep 15; ls
sleep 10; ls &
sleep 10 &; ls
```

stop: can be used to stop a specific job running in the background

Running more than one shell

- By typing the name of a shell, you can begin running another shell. You can change directories etc in this shell, and go back to your original shell by using the suspend/exit command.

    type shell name (try csh, or bash) *get a new shell, do some work, say change directory)*

    suspend *(go back to your original shell (does not close)) (work in your original shell)*

    jobs *(get the job number of the other shell, say 1)*

    fg 1 *(bring the csh shell into the foreground)*

    exit *(terminate the other shell and go back to original)*