CS 32 Notes: Intro to Operating Systems
(refer to CS 16 and CS 24 Notes for first part of class)

Unix Acronyms & Commands:

PPID: Parent Process ID

PID: Process ID

ps	Process Status: provides list of active processes on terminal
ps -l	Process Status with additional details
top	Shows resource consumption of current processes
jobs	Lists all processes running in the background
kill <PID>	terminates process associated with the PID

Operating Systems:

Application: terminal, vim, etc... + APIs (Application Programming Interface) which contains code for OS functionality and language libraries

OS Kernel: memory space + file/process/memory management

Hardware: physical components + software operating the physical components

Process: program in execution; instance of program being run; contains memory and space of program

- Foreground processes must be completed before the terminal takes additional commands
    - Foreground processes can be terminated with CTRL + C and suspended with CTRL + Z
    - Can resume a process by bringing it back into the foreground
- Background processes allow the terminal to accept additional commands; add & to executable to run it in the background

Thread: program unit executed independently of other units

- processes can create threads, which are managed by OS
- threads share same memory as main process

Unix Processes:

fork( ): copies the parent process; child process id = 0, child parent id = parent process ID

- the PPID of bash is the parent process which many things are copied from
- order of processes run in parallel; order of process output during intermediate execution may be unclear

exec( ): replaces the parent process with another process

- when process finishes, the parent process is no longer valid; everything terminates

Example: fork ls -l

- process makes a copy of the existing process
- the copy runs exec to replace the copy with 'ls -l'
- 'ls -l' finishes, the terminal is no longer valid, OS cleans this process from memory

Threads

- Each processor (core) runs instructions in parallel
    - For single-core architectures, it mimics concurrency but the processor is shared among all programs
    - A single thread can only be executed on a single core at any given time
- Concurrency increases efficiency when multiple parts are run at the same time (independently)
- Embarrassingly Parallel (obviously trivial that the problem can be split into independent parts) problems are a good fit for use of multi-threads
- Race Conditions: if threads share memory with each other and run concurrently, read and write conflicts may occur
    - Atomic Transaction: everything is executed or nothing is
- Mutex: mutual exclusion multi-threaded locking mechanism that only allows a single thread to execute code
    - only one thread can acquire the lock at a time; thread can only execute when the lock is acquired
    - provides atomic transactions while preventing race conditions
    - Deadlock: when two or more threads are waiting for each other to release a resource and execution is stalled