11/12/2013: Lecture 20

"ps" command in terminal gives a list of running processes
    Larger PID tends to indicate more recently started processes
    "ps aux" will print all processes by all users, as well as more info about them

Fork
- Each fork() call "splits" the process into 2, with each one continuing to run the rest of the code (for instance fork2.c in l20 directory prints total of 4 times)
- There is no definitive rule to the order that the child processes run
- Fork duplicates all the memory/processes of the parent process, but the parent and children share the same virtual I/O systems (see l20/fork2b.c with output directed to a file)
    - This means any values stored in the buffer will be duplicated
    - Can avoid this happening by calling "fflush(stdout)", for instance, to flush the buffer before forking

OS generally ensures that system calls have atomic effects
- Atomic: an operation that appears to have indivisible effects; an operation that appears to happen completely at one instant in time
    - In other words, an atomic process either occurs uninterrupted or not at all; it cannot only partially complete before stopping or being interrupted
- In forkmix.c, we see that "fputs" is not atomic, as we can get outputs of the form "CHmom" or "ILD", meaning one call was interrupted by another in the middle of writing
- If we replace the library call "fputs" with the system call "write" (which is atomic), we can avoid these interruption effects

Zombie processes are processes that have exited, but whose exit status has not yet been collected
- <defunct> processes seen after running "ps" are zombie processes
- "fork" creates new processes, "exit" kills current, and "waitpid" waits for child process to change status
- these zombie processes linger if the parent process never collects their exit status, as the OS doesn't know this and will keep around the children waiting for the exit status to be collected
- manyfork.c originally has no waitpid call, so its children's statuses are never collected, and all become zombies
    - this causes us to fill up all available process spaces for large amounts of forks

Waitpid(pid, status, flags) system call
- pid is the process to wait for (0 means any child), status is set to the exit status
- flag WNOHANG means: if no zombies, return 0
- kills the process pid after collecting the exit status

- zombie processes exist because we may want to see how a process has exited, and so processes are kept around until then

Other communication channels between processes besides the exit status exist

Pipes are one such communication channel between processes

- a stream, not a file, and not seekable
- FIFO (first in first out)
- used to connect the output of one process with the input of another
- pipelines are multiple processes connected by pipes on stdin/stdout

The pipe cannot be stored in process-shared memory, since the processes are isolated from each other

Pipe buffer is stored somewhere in kernel memory instead

- we use the same system calls "read" and "write" to read/write to the pipe buffer
- this is so that programs don't have to be modified for different types of input/output
- implemented in Unix as a pair of file descriptors: one for write and one for read

pipe(pipefd) call creates a pipe

- pipefd[0] contains the read file descriptor; pipefd[1] contains the write file descriptor

A pipe closes only when every reference to its "write" end is closed

- when forking processes all referencing one pipe, must be careful to close every reference to the pipe's write end

Question that came up during lecture:

Modern systems use strong modularity boundaries so that failures in separate processes do not affect each other

- Kernel isolation has kernel with complete control that enforces boundaries between the processes running on the system