CS180 Notes - 2/22/16

Interprocess Communication

- Shared Memory
  - Processes can access a common block of memory
  - Can read/write to this memory like any other type of memory
  - Tends to be faster than message passing, but has potential synchronization issues

  - Functions used to obtain shared memory
    - *shmget* creates (or retrieves) memory.
      - Be sure to use a provided key for homework purposes
      - Common permissions for *shmget* is 600 or 666 (see *chmod* for examples)
    - *shmat* attaches memory to the process
    - *shmdt* detaches the memory from the process
    - *shmctl* deletes the memory block
      - You MUST make sure to delete the memory, as it is **not** retrieved after the process ends
  - If the program closes pre-maturely (likely because of crashes), you must reclaim this memory with ***ipcrm***. If you don't, you won't be able to create the shared memory again.

- Message passing
  - Information is passed by messages through the kernel.
  - Safer, but tends to be slower.
  - Functions needed to use message passing
    - *msgget* creates (or retrieves) a message queue
    - *msgsnd* posts a message to the queue
      - When we send a message, even though it looks like we're sending the message by reference in the parameter, it is still actually a copy. This is good as messages persist after a crash.
    - *msgrcv* retrieves a message from the queue (and removes it)
    - *msgctl* deletes the message queue
  - All messages are passed using structs, and anything can be placed in these structs
    - The first member of the struct **must** be a long int. (which is the type of message)
      - In *msgsnd* and *msgrcv*, we can denote what type of message to get, which is where the long int comes into play
        - If we choose 0, get whatever the next message is.
        - If we choose anything positive, get the next message with a type **equal** to that provided value
        - If we choose anything negative, get the first message on the queue whose type is less than or equal to the absolute value of the negative number
        - You'll mostly see 0 as the parameter, but the other options exist.
  - We never have to worry about incomplete read/writes of messages.
  - We can also have one server and multiple clients
    - It doesn't matter who starts the queue first
    - If we run out of space on the queue, any further message sending is blocked until room is made (by reading).

  - The size of the message is the size of the structure **without** the long int identifier.
  - Run the command *ipcs -q* to see the queue.

- POSIX pipes
  - A stream of communication between two processes
  - A process can read or write to a pipe
  - Processes can communicate by a pipe without being aware of it
  - It uses fd[0] (stdin) and fd[1] (stdout), which is assigned when *pipe* is called.
  - If we know that a child or parent process won't be reading/writing, be a good Samaritan and close that particular descriptor (as seen in the notes)
  - The message size read will always include the newline and NULL-terminator, giving us an extra two bytes.
  - Anonymous pipes cannot be accessed outside of the process that created it and no longer exist when the process ends.