Conceptual Questions

1. **fork()** creates a new process. This differs from creating a new thread in that **fork()** creates a copy of the original process and executes from the **fork()** statement. In multi-threading this doesn’t happen and a new thread executes another section of code. Another difference is that memory is not shared between processes, unlike between threads. Each process has its own copy of the memory.
2. Interprocess communication is how data can be shared between processes. A few methods are:

Files: processes read and write to the same file

Pipes: communicate via standard input and output

Semaphores: allow for synchronization of shared resources

Shared memory: Processes share the same block in the memory

1. Semaphores are a means of synchronization on shared resources. A semaphore is an integer variable with two atomic operations: wait() and signal(). If the semaphore is greater than 0, wait() will decrease the semaphore by 1. Otherwise it will wait for the semaphore to be greater than 0. Once wait() is successful, the process will execute the critical section. Once it is finished it will call signal() which increases the semaphore by 1, signaling other processes that they can use the resource.

A binary semaphore works in the same way as a mutex lock. However, semaphores can also be counting semaphores which can control a finite number of resources. That is the semaphore can be greater than 1, and represents the number of resources available.

1. wait: decreases the value of the semaphore and if it is greater than zero, allows the process to continue. If the value is negative, the process is blocked

signal: increases the value of the semaphore and wakes up a blocked process who is waiting

1. sem\_init() - initializes a semaphore to a given value

sem\_wait() - the wait function

sem\_post() - the signal function

sem\_getvalue() - gets the current value of a semaphore

sem\_destroy() - destroys a semaphore if there are no waiting semaphores