SOFE 3950U: Tutorial 3 Activity
Group 1

Justin Kaipada 100590167
Anthea Ariyajeyam 100556294

Conceptual Questions

Question 1

The function fork() is used to create a new process (i.e. child process). It differs from multi-threading by copying all of the resources (i.e. variables, arrays) in the parent process to the child process whereas multithreading share these resources.

Question 2

Inter-process communication (IPC) allow different processes to communicate via shared memory or message passing. Shared memory enables processes to communicate using the same area of memory while message passing allows processes to use system calls to exchange messages.

Question 3

A semaphore is an integer that represents the number resources the program can access. This ensures processes and threads to synchronize their activities during execution.

Semaphores are accompanied by two methods, wait() and signal(), that will manipulate the value of the semaphore. They are both placed placed between the critical section of the code (i.e. the shared variables that affect results of all threads/processes). If there are no resources available, the thread/process must wait for the resource to become available to continue execution.

This differs from mutual exclusion by allowing more than one threads/process to access their critical section at the same time.

Question 4

Wait and Signal are the main modifier methods used to control and use semaphores when syncing many processes in a multi-processing system/program. Wait will increase the value of the semaphore while signal will decrease the value of the semaphore. When the value of the semaphore is less than 0 another threads/processes are asked to sleep instead of executing the critical section.

Question 5

Main functions used from <semaphore.h> are as follows:-

- `int sem_init(sem_t*, int, unsigned)` - This will create an unnamed semaphore. Upon successful creation this function will return 0 or -1 if the initialization failed.
- `int sem_destroy(sem_t * sem)` - This will destroy the semaphore pointed by `sem`. Only a semaphore created using `sem_init()` can be destroyed this way. Upon successful destruction this function will return 0 or -1 if the destruction failed.
- `sem_t *sem_open(const char *, int, ...)` - This will create a connection between a semaphore and a process. The semaphore must be named for successful use of this method. Upon successful execution the method will return the address of the semaphore, if failed it will return the value `SEM_FAILED`.
- `int sem_post(sem_t *sem)` - This will unlock the semaphore pointed by `sem` and allows execution of the critical section to happen. The semaphore value will be just incremented, if it is positive other processes can follow the execution after it. Upon successful execution this function will return 0 or -1 if the execution failed.
- `int sem_wait(sem_t *)` - This simply decrements the semaphore pointed by `sem` locking it. If the semaphore's value is greater than zero then the function returns at once. If the value is 0 it waits until it is possible for documenting until it is interrupted by something else.