SOFE 3950U: Tutorial 5

Group 1

Anthea Ariyajeyam 100556294
Justin Kaipada 100590167

Conceptual Question's Answers

1. `pthread_create` - Will create a new thread based in the *POSIX* standards using the given function as an entry point.

   `pthread_join` - Ask the calling thread to wait until the given thread terminates, essentially joining the current thread with the child/sub thread.

   `pthread_exit` - This function terminates the calling thread, it makes the passed pointer available to any successful join functions with the terminating thread

2. While threads have there on stack memory to store independent information such as local variables, all the threads created by a process share the same heap memory, data segment, code segment and so on that belongs to the same process.This means that all the global variables and input-output streams are shared by every thread but each thread will have there on copy of local variables.

   Although other thread's stack memory is not available by default, in theory you can still hold a pointer to another threads stack frame which can be accessed by any thread.

3. In **multithreading** many threads are created by a process to do work concurrently and/or parallel. In **multiprocessing** instead of creating multiple threads, the program created many processes to the work.

   Both have pros and cons of there own. Since threads share the same data and variables when in the same process, communication b/w threads are easy, we only need to synchronize them carefully. But for processes the data and variables are never shared so we need to use some sort of shared memory system or messaging system which will make them slower. If we use a cluster system to run our programs and we do not use shared memory, multiprocessing is the way to go because, each process can be run on different nodes in a cluster but multithreading does not enable us to do that. One of main problem when using multiprocessing to do simple work is the overhead of creating a process, which is high in both CPU usage and memory.

4. **Mutual exclusion** is the act of preventing simultaneous access of the shared data/streams between threads, by using a mutex or some other mechanisms like a mutex. In fact, MUTEX is an abbreviation for MUTual EXclusion.

When using multiple threads that share the same data, there will be the some section of code which is where we modify the data or access input or output streams, these sections of code are called critical sections which should be done atomically. We will use a mutex to do this, by locking the mutex before the critical section and unlocking it after we finish accessing/modifying the data in the critical section.

5. The methods used to perform mutual exclusion when using pthreads are

   `pthread_mutex_init (mutex,attr)` - This method is used to dynamically initialize the `mutex` with the attributes specified as `attr`.

   `pthread_mutex_destroy (mutex)` - This method is used to destroy/free the `mutex` object when it is no longer needed.

   `pthread_mutexattr_init (attr)` - This method is used to create a mutex `attr` which is needed for the creation of a new mutex.

   `pthread_mutexattr_destroy (attr)` - This method is used to destroy/free the `attr` object when we no longer need it.

   `pthread_mutex_lock (mutex)` - This method is used to acquire lock the specified `mutex` so the critical data can be accessed without interference from other threads

   `pthread_mutex_trylock (mutex)` - This will try to lock a `mutex` and return and error code if the mutex is already locked by another thread instead of waiting it to be unlocked preventing deadlocks.

   `pthread_mutex_unlock (mutex)` - This will unlock the `mutex` if called by the owning thread. This will enable other threads to lock and use the critical data.