9.1.1 Multi-threaded C with PThreads

The following Google Colab includes the code and description of the assignment Threads and Synchronisation Colab link

1. PThreads

Question 1 2 / 2

TODO 0: Which header files need to be included?

- [x] stdio.h
- [ ] semaphore.h
- [x] pthread.h
- [ ] stdlib.h

Question 2 2 / 2

TODO 1: What is the correct return type?

- ( ) void
- ( ) void&
- (•) void*
- ( ) void**

Question 3 2 / 2

TODO 2: What is a correct way of exiting the child thread PrintHello function?

- [x] Calling pthread_exit with any argument (e.g. NULL) as the value is not used
- [ ] Calling pthread_exit with the thread name (string) as an argument
- [x] Just returning from the function is enough
- [ ] Using exit()

Question 4 2 / 2

TODO 3: What is the correct way of starting each thread?

- ( ) pthread_create(threads[t], NULL, PrintHello, (void *)t)
- (•) pthread_create(&threads[t], NULL, PrintHello, (void *)t)
- ( ) pthread_create(&threads[t], NULL, *PrintHello, (void *)t)
- ( ) pthread_create(threads[t], NULL, &PrintHello, (void *)t)

Question 5 2 / 2

TODO 4: What is the correct way of exiting the main thread?

- [x] Calling pthread_exit with any argument (e.g. NULL) as the value is not used
- [ ] Calling pthread_exit with the thread name (string) as an argument
- [ ] Just returning from the function is enough
- [ ] Using break

Question 6 2 / 2

In what order will the 5 launched threads print out their IDs to stdout?

- ( ) 1 > 2 > 3 > 4 > 5
- ( ) 5 > 4 > 3 > 2 > 1
- ( ) Only the first thread that gets a handle of stdio will print its ID out
- (•) The order is non-deterministic

9.1.2 More synchronisation

The following Google Colab includes the code and description of the assignment Threads and Synchronisation Colab link

1. More on PThreads

Question 1 2 / 2

How many critical sections are in the hello_mutex function?

- ○ 0
- ◉ 1
- ○ 2
- ○ 3

Question 2 2 / 2

What will happen if the values of the constants NUM_THREADS and NUM_ROUNDS are swapped?

- ☑ There will be more contention for the shared_variable
- ☑ More threads will take part in fewer rounds
- ☐ More rounds will take place
- ☐ The shared_variable will be updated with higher values

Question 3 1 / 1

TODO 5: Which pair of functions will guarantee the shared variable is not accessed by more than 1 thread at the same time?

- ○ lock(0) & unlock(0)
- ○ pthread_mutex_lock(PTHREAD_MUTEX_INITIALIZER) & pthread_mutex_unlock(PTHREAD_MUTEX_INITIALIZER)
- ◉ pthread_mutex_lock(mutex_object) & pthread_mutex_unlock(mutex_object)
- ○ pthread_lock() & pthread_release()

Question 4 2 / 2

TODO 6: What is true about waiting on the conditional variable?

- ☑ When the wait succeeds, the mutex will be acquired by the calling thread
- ☐ The wait is a non-blocking call
- ☑ The mutex object must already have been acquired by the calling thread when it enters the conditional wait
- ☐ The mutex object does not need to have been acquired by the calling thread when it enter the conditional wait, as it will be acquired after the wait succeeds

Question 5 2 / 2

TODO 7: What is true about signalling the conditional variable?

- ○ It will always unblock all other threads waiting on the variable
- ◉ It is a non-blocking call
- ○ It will block the calling thread if no other thread is waiting on the conditional variable

Question 6 1 / 1

TODO 8: Which pair of functions will guarantee semaphore-based coordination between threads calling hello_semaphore?

- ◉ sem_wait(semaphore) & sem_post(semaphore)
- ○ pthreads_mutex_lock(semaphore) & pthreads_mutex_release(semaphore)
- ○ semaphore.lock() & semaphore.unlock()

Question 7 2 / 2

TODO 9: What is true about initialising the semaphores and conditional variables?

- [x] It is sufficient to initialise the semaphores as only shared between threads of this process
- [x] To ensure all threads will wait until explicitly signalled, the initial values of the semaphores should be 0
- [x] Initialising the conditional variable successfully will return 0

Question 8 2 / 2

TODO 10: What is the correct way of starting the first thread waiting on the semaphore?

- ( ) sem_post(-1)
- ( ) sem_post(semaphores[0])
- (•) sem_post(&semaphores[0])
- ( ) sem_post(&semaphores[1])

Question 9 2 / 2

TODO 11: What is true about running the code with the mutex method of synchronisation compared to semaphore?

- [x] The printed output of the program will be the same in both cases, except for the Hello World messages
- [ ] In the semaphore case, threads will not be updating the shared_variable in a deterministic order
- [x] In the mutex case, the shared_variable will be updated up to the value of 49
- [x] In both cases, all threads will set the shared_variable in all rounds