COMP 3413 Operating Systems Lab 3 Hand–in

Moustafa	Mahmoud	Mark: 20
Family Name	Given Name	
3648276		
Student Number		

[2] 1a. Initializing a pthread mutex variable: There are 2 ways to initialize a mutex variable: through a method and statically. The method way will be done using the pthred_mutex_init() method. Statically by assigning PTHREAD_MUTEX_INITIALIZER to a variable of pthread_mutex_t type.

[2] 1b. `pthread_mutex_destroy()`: destroys the mutex object that is referenced by the mutex. This makes the object unintialized. If the destroyed object is referenced, its result is undefined. However, a mutex object that has been destroyed can be reinitialized using pthread_mutex_init();

[3] 1c. `pthread_mutex_trylock()`: this function locks a mutex object. It would be useful if the object is locked, and we want it to return immediately.

[3] 2a. `mutex_recursive.c`: it initializes a mutex attribute, then it sets the mutex type attribute, and it initializes the mutex itself. It, then, gets into the first set of for loops. In the first iteration, it skips the nested for loop, then, prints “locking”, and it locks the mutex. In the next iteration, it goes in the nested for loop, then prints “locking”, however, it waits for the mutex that was locked to be unlocked. So, it won’t go into the third iteration, until it is unlocked.

[3] 2b. modified `mutex_recursive.c`: we can use the mutex recursive type. It will do what this program attempts to do. The change that we will need to make in the program is (&attributes, PTHREAD_MUTEX_NORMAL) → (&attributes, PTHREAD_MUTEX_RECURSIVE). The result obtained is

```
Testing recursive mutex.
locking
 locking
  locking
   locking
    locking
     locking
      locking
       locking
        locking
         locking
         unlocking
        unlocking
       unlocking
      unlocking
     unlocking
    unlocking
   unlocking
  unlocking
 unlocking
unlocking

End of processing
```

An application in which we need synchronized calls would be one in which this functionality is needed.

[1] 3a. expected value of shared variable: The for loop iterates 50 times and there are 20 threads. Therefore 50*20 = 1000

[1] 3b. results of 10 runs before changes: 209, 211, 186, 173, 213,181, 229, 183, 182, 169. Because of the random number of iterations, it is expected to have threads that have different runtimes.

[5] 3c. results of 10 runs after changes: 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000. Those values do agree with the answer in 3a.