ITF22519 - Introduction to Operating System

Lab7: Thread Synchronization

Note: This is the mandatory assignment. The deadline is 14 Oct, 2021 23:59. However, if you need more time to complete the assignment, please contact me. In addition, please follow submission instructions stated in this document. Your git account has to be included in your report. Your `.c` files and your output have to follow the requirements of the specific exercise.

---

1. (30 points) **Exercise 1 (Thread Programming)**
   Let T_{serial} be the execution time of a serial program. Suppose that you want to parallelize that serial program by using `p` threads so that the program can be executed faster. Let $T_{concurrent}$ be the execution time of the code with `p` threads. Without doing any programming to measure execution time, answer the following questions:
   (a) Is this possible to achieve $T_{concurrent} = \frac{T_{serial}}{p}$?
   (b) Explain why (or why not).

2. (30 points) **Exercise 2 (Mutex)**
   In the C program in *`mutex.c`* file, two threads try to access the critical section and modify the global variable `count`.
   (a) Run the program several times and explain how it works.
   (b) The global variable `count` is expected to be 0 when the program finishes its execution. However, the output may be different. Use `mutex` to lock and unlock the critical section so that the output is 0.

3. (30 points) **Exercise 3 (Conditional Variables)**
   Write a C program that uses Thread 1 to print out 'Pfizer' and Thread 2 to print out 'Moderna' five times. The main Thread waits for the two threads to finish and then prints out 'Astrazeneca!'. Use **condition variable(s)** to synchronize two threads so that the output is the following:

```
Moderna
Pfizer
Moderna
Pfizer
Moderna
Pfizer
Moderna
Pfizer
Moderna
Pfizer
Astrazeneca!
```

Functions you may need include:

```
int pthread_mutex_init(pthread_mutex_t *restrict mutex, const pthread_mutexattr_t *restrict attr);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_cond_init(pthread_cond_t *restrict cond, const pthread_condattr_t *restrict attr);
int pthread_cond_destroy(pthread_cond_t *cond);
```

```
int pthread_cond_wait(pthread_cond_t *restrict cond, pthread_mutex_t *restrict mutex);
int pthread_cond_signal(pthread_cond_t *cond);
int pthread_join(pthread_t thread, void **retval);
```

The relevant portions of a main() function are provided in the box below.

- Fill in the code for `print_Pfizer()`, `print_Moderna()` and `main()`.
- Make the output file *`Cond.c`* for your code and put it under lab7 repository in your GitHub.

```
...
//Your code for global variables (if any)
void* print_Pfizer() {
        //Your code for Thread 1
}
void* print_Moderna() {
        //Your code for Thread 2
}
int main(int argc, char** argv) {

        //Your code for initializing any local or global variables

        pthread_t t1, t2;

        pthread_create(&t1, NULL, print_Pfizer, NULL);
        pthread_create(&t2, NULL, print_Moderna, NULL);

       //Your code for the rest of the program

return 0;

}
```

4. (30 points) **Exercise 4 (Semaphores)**

Write a C program that uses Thread 1 to print out 'Department of Computer Science and Communication' and Thread 2 to print 'Faculty of Computer Science, Engineering and Economics' five times. The main Thread first printes out 'Ostfold University College!' and then waits for the two Threads to finish. Use **semaphore(s)** to synchronize the threads so that the output is:

```
Ostfold University College!

Department of Computer Science and Communication
Faculty of Computer Science, Engineering and Economics

Department of Computer Science and Communication
Faculty of Computer Science, Engineering and Economics

Department of Computer Science and Communication
Faculty of Computer Science, Engineering and Economics

Department of Computer Science and Communication
Faculty of Computer Science, Engineering and Economics
```

```
14
15 Department of Computer Science and Communication
16 Faculty of Computer Science, Engineering and Economics
```

Functions you may need include:

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_destroy(sem_t *sem);
int sem_wait(sem_t *sem);
int sem_post(sem_t *sem);
int pthread_join(pthread_t thread, void **retval);
```

The relevant portions of a main() function are provided in the box below.

- Fill in the code for `print_Department()`, `print_Faculty()` and `main()`.
- Make the output file *`Sem.c`* for your code and put it under lab7 repository in your GitHub.

```
1  ...
2  //Your code for global variables (if any)
3
4  void* print_Department() {
5  //Your code for Thread 1
6  }
7
8  void* print_Faculty() {
9  //Your code for Thread 2
10 }
11
12 int main(int argc, char** argv) {
13
14      //Your code for initializing any local or global variables and other
15
16      pthread_t t1, t2;
17
18      pthread_create(&t1, NULL, print_Department, NULL);
19      pthread_create(&t2, NULL, print_Faculty, NULL);
20
21      //Your code for the rest of the program
22
23      return 0;
24 }
```