CS 3305B: Operating Systems
Department of Computer Science
Western University
Assignment 3
Winter 2017
Due Date: April 9th 2017

Purpose
The goals of this assignment are the following:

- Get hands-on experience in developing mutual exclusion / semaphore / critical section techniques/algorithms.
- Gain more experience with the C programming language from an OS's mutual exclusion / semaphore / critical section perspective.

Assignment Description
Using C programming language, you will be developing a mutual exclusion algorithm for a given process synchronization problem. You need to make sure that your mutual exclusion algorithm ensures that only one process can access the critical section portion of your code at a given point in time. You are not allowed to use any mutual exclusion / semaphore related C systems calls.

Part I: Process Creation (20 points)
Create a total of ten processes: 1 parent process, and 9 children processes. Each of the process will eventually enter into the critical section part of your code.

Part II: Critical Section (20 points)
Implement the following function which you can consider as the critical section of your code:

```
int critical_section (int loop_limit, int who_called)
{
        for (loop limit)
                printf (who_called, loop index);
        printf ("returning to caller process x"); //x is the caller process
        return 0;
}
```

Conceptually, you can consider the above function critical_section(int, int) as the critical section portion of your code. In this function, loop_limit is the number of iterations and who_called represents the unique process number of the caller process.

Part III: Semaphore (60 points)
You need to implement a semaphore (i.e., flag / mutual exclusion) mechanism to implement the mutual exclusion that will be needed to allow no two processes access the critical section simultaneously. The value of consider a simple semaphore technique (0 or 1) that could be used as a flag so that the processes that are willing to enter into the critical section (i.e., executing the critical_section function) will either wait or execute (wait if flag is 1, and execute if flag is 0).

Hints:
You should make two versions of your code for testing purpose. Initial version and the final version. In the initial version, you should exclude critical section and semaphore. When you run the initial version, you will notice that the processes are often pre-empted before they finish their task (i.e., before completing the for() loop). This is normal and expected as CPU switches between processes (i.e., context switching). When you run the full version (with mutual exclusion implemented through semaphore and critical section) you should be able to see the clear difference between the initial version and the final version by looking at their output. In the final version, there should not be any process pre-emption because once a process enters into the critical section it will complete the for loop before any other process can access the critical section.

Assignment related technical resources
Please visit the course website for specific technical instructions and relevant materials. Also, consult TAs, and Instructor for any question you may have regarding this assignment.