CECS 326-01 Assignment 5 (10 points)
Due: 12/1/2022 on BeachBoard

Recall in Assignment 4, the *master* process and its child processes executing the *slave* executable communicate through a shared memory segment that is structured as struct CLASS. *master* creates a shared memory segment, structure it as struct CLASS, and initialize the index variable there to zero. *master* then creates a number of child processes to execute the *slave* executable, in which each child writes in shared memory its child number in the slot of response array pointed to by the current value of index and then increment index. A named semaphore is used to guard against race condition in accessing the index variable.

Assignment 4, however, has neglected another resource that is shared by all processes: the display device for output. In this assignment you are to plug this hole by adding one more semaphore in the solution. The requirement for this assignment is that you use an unnamed semaphore for controlling access to the shared index variable, and a named semaphore for outputting to the monitor. The POSIX implementation supports named and unnamed semaphores, both of which are defined in <**semaphore.h**>. The named semaphore mechanism includes system calls **sem_wait()**, **sem_post()**, **sem_open()**, **sem_close()** & **sem_unlink()**, while the unnamed semaphore mechanism includes **sem_wait()**, **sem_post()**, **sem_init()**, and **sem_destroy()**. Details of the definition of these system calls and their use may be found on Linux man pages. A sample C program named observer-2.c showing the use of both types of semaphores for the enforcement of mutual exclusion on access to shared memory and output device can be found in the Content/Assignments folder on BeachBoard.

Your programs must run successfully on Linux.

Do the following for this assignment:

1. Modify your Assignment-3 C programs to use an unnamed semaphore to guard against race condition for the access to shared data in shared memory, and a named semaphore to guard against race condition for output to the monitor. A critical section should include only the code requiring mutual exclusion, possibly with a few lines of code that do not use the shared resource but are incidental to the processing. Program execution efficiency will suffer if you define unnecessarily large critical sections in your program.
2. Compile your programs into executables *master* and *slave*, respectively. Make sure that sufficient and proper comments are included on the added code as well as the existing code.
3. Run your revised version of *master* (with *slave*) to make sure that the programs behave as required and the outputs are correct. Command for execution should take the form of *./master n shm_name sem_name*, where *n* is the number of child processes, *shm_name* and *sem_name* are shared memory name and semaphore name, respectively.
4. Submit on BeachBoard the two C programs master.c and slave.c, along with the *myShm.h* file, a screenshot that shows successful compilation of both programs as well as a successful run, and a cover page that provides your name, your student ID, course # and section, assignment #, due date, submission date, and a clear program description detailing what you have done for the correction. Format of the cover page should follow the cover page template on BeachBoard.
5. The programs must be **properly formatted and adequately commented** to enhance readability and understanding. Detailed documentation on <u>all</u> system calls are especially needed.