CECS 326-01 Assignment 5 (10 points)

Due: 12/5/2023 on Canvas

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 first acquires the current index value and saves it in a local variable i, increments index by 2, then requests user for a lucky number then writes in shared memory its child number & lucky number in response[i] & response[i+1]. A named semaphore is used to guard against race condition in accessing the index variable.

Assignment 4, however, has neglected another set of resources that is shared by all processes: the keyboard & monitor for I/O. 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 I/O control. 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 I/O device can be found on Canvas.

Specifications on how *master* and *slave* should behave are similar to Assignment 4, with necessary modifications due to the addition of an unnamed semaphore and the change of role of the named semaphore.

Note:

1. Header files required for using POSIX semaphores and how to compile these programs may be found in sample codes observer.c and observer-2.c.
2. Each child process writes in 2 slots of the response array.
3. Unnamed semaphore has no name that processes can referred to for access. It must be placed in a location where all cooperating processes can access, namely, the shared memory. Hence your struct CLASS would need to be modified to include a place for the unnamed semaphore.
4. Use an unnamed semaphore to guard against race condition for the access to shared index variable, and a named semaphore to guard against race condition for the I/O operations. 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.

Your programs must run successfully on Linux.

**Submit on Convas** the following:

1. Two source programs *master.c* and *slave.c* and the header file myShm.h.
2. A screenshot that shows successful compilation of *slave.c* to *slave*, and *master.c* to *master*;
3. A screenshot that shows successful run of *master*, which will create child processes to execute *slave*; producing outputs as described above;
4. 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 the programs are about. Format of the cover page should follow the cover page template posted on Canvas.
5. The programs must be properly formatted and adequately commented to enhance readability and understanding. Detailed documentation on all system calls are especially needed.