CECS 326-01 Assignment 5 (10 points)
Due: 12/5/2024 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 created by *master* and structured as a predefined struct. *master* initializes the index variable in the struct to zero and each child process first acquires the current index value and saves it in a local variable i, increments index, then writes in the shared struct its child number in report[i]. An unnamed 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 monitor 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, in addition to the **unnamed semaphore** you had used for controlling access to the shared index variable, use a **named semaphore** for I/O control. To ensure that the output by the child processes are cleanly separated, each child will send all its output to a file; and when all the output is done, acquires control of the monitor using the named semaphore and sends its output file to display there. 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. Two sample C++ programs named observer-1.cpp and outputToFile.cpp, respectively, showing the use of named semaphore and file I/O 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 a named semaphore and file I/O.

Note:
1. Header files required for using POSIX semaphores and file I/O and how to compile these programs may be found in sample codes observer-1.cpp and outputToFile.cpp.
2. Each child process writes in 1 slot of the report array.
3. The file name each child process outputs to must be distinct.
4. Suppose 3 child processes are created, the child numbers 1, 2, 3 written to the report array may not appear in this order.
5. 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.
6. Final content of the shared memory displayed by master must include the index value and the part of the report array filled by the child processes.

Your programs must run successfully on Linux.

Submit on Convas the following:
1. Two source programs *master.cpp* and *slave.cpp* and the header file myShm.h;
2. A screenshot that shows successful compilation of *slave.cpp* to *slave*, and *master.cpp* to *master*;
3. A screenshot that shows successful run of *master*, which will create child processes to execute *slave*; and the *master* and *slave* processes produce outputs as specified;
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 <u>all</u> system calls are especially needed.