

Assignment #4

Naoaki Takatsu

Student ID: 015746144

CECS 326 Sec 05 5288 Operating Systems

Due 3, May 2018

Submitted 3, May 2018

Description:

Since the previous project had a race condition problem within the critical section of the consumer shared memory, the remaining number of seats often ended up with a negative value as well as the selling of seats being made simultaneously with other shared memory. To solve this problem, a named mutex semaphore will be created in the producer memory while the consumer memory will have access to it, both by the use of sem\_open(). Then, before entering the critical section, a sem\_wait() will be used to lock the semaphore and when leaving, a sem\_post() will be used to unlock the semaphore. Finally, the consumer memory will detach and close the shared memory before exiting the program.

shmp1.cpp:

This is the executable shared memory producer code. It will create 3 individual shared memory (child process) with each sharing an inventory of 15 seats. To resolve the race condition within the critical section of the shmc1, a named mutex semaphore is created by the use of sem\_open() upon execution. It will wait until the shared memory completes their tasks. When finished, it will detach and destroy the memory. Before exiting, it will also unlink and close the named mutex semaphore by the use of sem\_unlink() and sem\_close().

shmc1.cpp:

This is the shared memory consumer code. It will sell seats until the number of remaining seats become less than or equal to 0. Since sem\_wait() and sem\_post() is implemented, each shared memory can access the critical section if and only if no other shared memory exists inside that block at that time. To access the named mutex semaphore created in the shmp1, sem\_open() is used upon execution of this code.