**Hoi Kiu Haydn Pang**

**017463730**

**CECS 326 Sec 05 5288**

**Assignment 3**

**Due Date: 04/12/18**

**Submission Date: 04//11/18**

**registration.h**

Header file for the structure representing a class. This structure contains the variable seats\_left that is manipulated by the shmc1.cpp program. The other variables represent the class number, the date, and the title of the class.

**shmc1.cpp**

Represents a process that sell seats in a class. Requires at least two arguments passed into the program, representing the process name and the shared memory segment identifier respectively.

sell\_seats() function decrements the seats\_left variable in a CLASS structure. Once the seats\_left variable is decremented to 0 or less, the function completes execution as indicated by all\_out = 1. A pseudorandom generator is used to provide delays after checking there are seats available (seats\_left > 0), after selling a seat, and after printing out that a seat is sold.

rpterror(char\* string) is used to output a custom message alongside the default error message.

**shmp1.cpp**

Creates NCHILD number of child processes which then replaces the program code with that of the shmc1 program. NCHILD is defined as the constant number 3 to indicate the number of child processes. The CLASS structure is initialized in this class, which all three processes has shared access to.

shm\_init(void \*shm\_ptr) function is used to initialize the shared memory segment, assigns it to the pointer \*shm\_ptr, and returns the shared memory segment identifier.

wait\_and\_wrap(int child[], void \*shm\_ptr, int shmid) function is used to wait for all the child processes to terminate before detaching the shared memory segment from the pointer \*shm\_ptr and removing the shared memory segment identified by the identifier shmid.

rpterror(char \*string, char \*pname) is used to output a custom message and process name alongside the default error message.

This program contains a critical section problem as the three child processes all require access to a shared variable, seats\_left. Even though delays are implemented by using the sleep() method, it does not guarantee that only one process has access to its critical section. Thus, mutual exclusion is not guaranteed. This poses a problem when the number of seats left is less than the number of processes running the sell\_seats() function. Even though sell\_seats first checks the flag variable all\_out to indicate if seats are left, mutual exclusion is not guaranteed and thus, multiple processes can decrement seats\_left at the same time. This results in erroneous output as it can result in negative seats remaining. This is also possible due to sleep() being called immediately before manipulating seats\_left for a random duration between 1 and 5 seconds, allowing another process to manipulate seats\_left while it remains suspended.