CECS 326-01 Assignment 6 (10 points)
Due: 12/7/2021 by class time on BeachBoard

As you have come to understand, the *shmp2.cpp* and *shmc2.cpp* (or *shmp2.c* and *shmc2.c*) you compiled and ran in Assignment 4 have serious deficiency due to race condition. In Assignment 5 you used the POSIX named semaphore to provide access control to the shared data, thereby correcting the problem by enforcing mutual exclusion.

With named semaphores, however, you would need to generate a unique name for every semaphore and associate each name with the shared data you want to control. This approach will become difficult to manage when the number of shared data structures is large. For example, if you have 100 bus tours to manage, you will have to create 100 semaphores, each with a unique name. An alternative is to use the POSIX unnamed semaphores, which can be declared in the same shared memory segment where the tour data are stored. In this assignment, you will have to use a POSIX unnamed semaphore to achieve the necessary access control so that the seat selling results will not show errors. To include the semaphore in the bus tour data structure, the previous booking.h header file has been revised to booking3.h as shown in this document. The two C++ programs have also been revised as shmp3.cpp and shmc3.cpp to match the changes made in booking3.h. Add code as needed to these two programs so that the erroneous outcomes are corrected.

The POSIX implementation supports named and unnamed semaphores, both of which are defined in <**semaphore.h**>. The unamed semaphore mechanism includes **sem_wait()**, **sem_post()**, **sem_init()**, and **sem_destroy()**, and should be used in this assignment. Details on the definition of these system calls and their use may be found on Linux man pages. The programs should be compiled using g++ and link with –lpthread.

The program must run successfully on Linux.

Do the following for this assignment:

1. Add necessary synchronization code in the following C++ programs to correct problems due to race condition, and compile them into executables *shmp3* and *shmc3*, respectively. Make sure that sufficient and proper comments are included on the added code as well as the existing code.
2. Run your corrected version of *shmp3* (with *shmc3*) to make sure that the output is correct.
3. Submit on BeachBoard the two corrected programs, along with the *booking3.h* file, a screenshot that shows successful compile 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.
4. 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.

```
/* booking3.h
 * Header file to be used with
 * shmp3.cpp and shmc3.cpp
 */

#include <semaphore.h>  // header file needed for POSIX semaphore

struct TOUR {
   char bus_number[6];
   char date[9];
   char title[50];
   int seats_left;
};
struct BUS {
   sem_t sem1;   // semaphore to control access to tour data below
   TOUR tour1 = { "4321", "11262021", "Grand Canyon Tour", 20 };
} mybus;
```

```
/* shmp3.cpp */

#include "booking3.h"
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <iostream>
#include <stdio.h>
#include <memory.h>

using namespace std;

#define NCHILD  3

int     shm_init( void * );
void    wait_and_wrap_up( int [], void *, int );
void    rpterror( char *, char * );

int main(int argc, char *argv[])
{
   int     child[NCHILD], i, shmid;
   void    *shm_ptr;
   char    ascshmid[10], pname[14];

   shmid = shm_init(shm_ptr);
   sprintf (ascshmid, "%d", shmid);

   cout << "Bus " << mybus.tour1.bus_number << " for "
        << mybus.tour1.title << " on " << mybus.tour1.date << ", "
        << mybus.tour1.seats_left << " seats available. " << endl;
   cout << "Booking begins: " << endl << endl;

   for (i = 0; i < NCHILD; i++) {
      child[i] = fork();
      switch (child[i]) {
         case -1:
            sprintf (pname, "child%d", i+1);
            rpterror ((char *)"fork failed", pname);
            exit(1);
         case 0:
            sprintf (pname, "shmc%d", i+1);
            execl("shmc3", pname, ascshmid, (char *)0);
            rpterror ((char *)"execl failed", pname);
            exit (2);
      }
   }
   wait_and_wrap_up (child, shm_ptr, shmid);
}
```

```
int shm_init(void *shm_ptr)
{
   int     shmid;

   shmid = shmget(ftok(".",'u'), sizeof(BUS), 0600 | IPC_CREAT);
   if (shmid == -1) {
      perror ("shmget failed");
      exit(3);
   }
   shm_ptr = shmat(shmid, (void * ) 0, 0);
   if (shm_ptr == (void *) -1) {
      perror ("shmat failed");
      exit(4);
   }
   memcpy (shm_ptr, (void *) &mybus, sizeof(BUS) );
   return (shmid);
}

void wait_and_wrap_up(int child[], void *shm_ptr, int shmid)
{
   int wait_rtn, w, ch_active = NCHILD;

   while (ch_active > 0) {
      wait_rtn = wait( (int *)0 );
      for (w = 0; w < NCHILD; w++)
         if (child[w] == wait_rtn) {
            ch_active--;
            break;
         }
   }

   cout << "Parent removing shm" << endl;
   shmdt (shm_ptr);
   shmctl (shmid, IPC_RMID, (struct shmid_ds *) 0);
   exit (0);
}

void rpterror(char *string, char *pname)
{
   char errline[50];

   sprintf (errline, "%s %s", string, pname);
   perror (errline);
}
```

```
/* shmc3.cpp */

#include "booking3.h"
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <iostream>
#include <stdio.h>
#include <memory.h>

using namespace std;

BUS   *bus_ptr;
void  *memptr;
char  *pname;
int   shmid, ret;
void  rpterror(char *), srand(), perror(), sleep();
void  sell_seats();

int main(int argc, char* argv[])
{
   if (argc < 2) {
      fprintf (stderr, "Usage:, %s shmid\n", argv[0]);
      exit(1);
   }

   pname = argv[0];
   sscanf (argv[1], "%d", &shmid);
   memptr = shmat (shmid, (void *)0, 0);
   if (memptr == (char *)-1 ) {
      rpterror ((char *)"shmat failed");
      exit(2);
   }

   bus_ptr = (struct BUS *)memptr;

   sell_seats();

   ret = shmdt(memptr);
   exit(0);
}

void sell_seats()
{
   int all_out = 0;

   srand ( (unsigned) getpid() );
   while ( !all_out) {   /* loop to sell all seats */
      if (bus_ptr->tour1.seats_left > 0) {
```

```
         sleep ( (unsigned)rand()%2 + 1);
         bus_ptr->tour1.seats_left--;
         sleep ( (unsigned)rand()%5 + 1);
         cout << pname << " SOLD SEAT -- "
              << bus_ptr->tour1.seats_left << " left" << endl;
      }
      else {
         all_out++;
         cout << pname << " sees no seats left" << endl;
      }
      sleep ( (unsigned)rand()%5 + 1);
   }
}

void rpterror(char* string)
{
   char errline[50];

   sprintf (errline, "%s %s", string, pname);
   perror (errline);
}
```