CECS 326-01 Assignment 5 (10 points)
Due: 11/9/2021 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 this assignment you are to correct the problem using one of the semaphore mechanisms that Linux provides. Note that any corrections you make should not include the removal or changes of the *sleep*() calls in the *shmc1* program.

Two implementations of semaphore are commonly available on most distributions of UNIX and Linux operating systems: System V and POSIX. In this assignment you will use the POSIX implementation. The POSIX implementation supports named and unnamed semaphores, both of which are defined in <**semaphore.h**>. The named semaphore mechanism includes **sem_wait()**, **sem_post()**, **sem_open()**, **sem_close()** & **sem_unlink()**, 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 program must run successfully on Linux.

Do the following for this assignment:

1. Add necessary synchronization code in your Assignment-4 C/C++ programs to correct problems due to race condition, and compile them into executables *shmp2* and *shmc2*, 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 *shmp2* (with *shmc2*) to make sure that the output is correct.
3. Submit on BeachBoard the two corrected programs, along with the *booking.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.

```
/* booking.h */
/* Header file to be used with
 * shmp2.cpp and shmc2.cpp
 */

struct BUS {
     char bus_number[6];
     char date[9];
     char title[50];
     int seats_left;
};
```

```
/* shmp2.cpp */

#include "booking.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 mybus = { "4321", "11262021", "Grand Canyon Tour", 20 };

#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.bus_number << " for "
           << mybus.title << " on " << mybus.date << ", "
           << mybus.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("shmc2", 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);
}
```

```
/* shmc2.cpp */

#include "booking.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->seats_left > 0) {
                 sleep ( (unsigned)rand()%2 + 1);
                 bus_ptr->seats_left--;
                 sleep ( (unsigned)rand()%5 + 1);
                 cout << pname << " SOLD SEAT -- "
                      << bus_ptr->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);
}
```