Juan Villa

Ernie Argel

Juan.Villa@student.csulb.edu

Ernie.Argel@student.csulb.edu

8 May 2020

CECS 326

Programming with Semaphores and Shared Memory: This program initializes an inventory of items in shared memory. Then, 10 child processes are spawned that each place 100 random orders until a certain item goes out of stock. If an order is placed for an item that is out of stock, the order is locked until the item is restocked. Items are restocked periodically by another process that handles shipment arrivals.

Inventory.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <limits.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <time.h>
#include <iostream>
#include "semaphore.h"
using namespace std;

// header to be included with your implementation
// shipment is an array containing 5 integer elements

// return value contains the new shipment of:
//    overgrips, strings, shoes, balls, accessories,
//    respectively (to be added to current stock)
int *shipment_arrival(int shipment[]);

enum {MUTEX};
int main() {
    srand(time(0));

    int shmid;
      char *shmBUF;
    long childPID;
    key_t key = ftok(".", 69);

    SEMAPHORE sem(1); //initialize MUTEX

    //increment mutex to 1 to allow the child processes to use

    sem.V(MUTEX);
    shmid = shmget(key, sizeof(int), 0600 | IPC_CREAT); //shared item
    cout << "Shmid: " << shmid << endl;

    int *shipment;
    shipment = (int*)shmat(shmid, NULL, 0); //pointer to shared
memory and cast to int pointer

    shipment[0] = 1;
    shipment[1] = 1;
    shipment[2] = 1;
    shipment[3] = 1;

    for(int i = 0; i < 10; i++) {   //create 10 child processes for
each customer
        childPID = fork();
        if(childPID == 0) { //execute child process/ customer

            for(int j = 0; j < 100; j++) {  //iterate 100 times for
each order
                cout << "childPID: " << childPID << " PID: " <<
getpid() << " Parent PID: " << getppid() << endl;
                //place order for each item
                int og = 1;     //1 item for over grip
                int rs = rand() % 2;    //zero to one for racket
strings
                int ts = rand() % 2;    //zero to one for tennis
shoes
                int tb = rand() % 2; //zero to one for tennis balls
                int ta = rand() % 2; //zero to one for tennis
accessories
                cout << "Placing order \n";

                //sem.P(MUTEX);  //decrement the semaphore and block
any process trying to order an overgrip
                sem.P(MUTEX);    //halts iorder if inventory is not
stocked
```

```
                shipment[0] = shipment[0] - og; //decrements 1 order
of overgrip in shared inventory
                if(rs == 1) {    //if customer is trying to place an
order for any racket strings
                    //sem.P(Sem2);    //halts if there are none
                    shipment[1] = shipment[1] - rs; //decrement 1
order of racket string in shared inventory
                }
                if(ts == 1) {    //if customer is trying to place an
order for any tennis shoes
                    //sem.P(Sem3);    //halts if there are none
                    shipment[2] = shipment[2] - ts; //decrement 1
order of tennis shoes in shared inventory
                }
                if(tb == 1) {    //if customer is trying to place an
order for any tennis balls
                    //sem.P(Sem4);    //halts if there are none
                    shipment[3] = shipment[3] - tb; //decrement 1
order of tennis balls in shared inventory
                }
                if(ta == 1) {    //if customer is trying to place an
order for any tennis accessories
                    //sem.P(Sem5);    //halts if there are none
                    shipment[4] = shipment[4] - ta; //decrement 1
order of tennis accessories in shared inventory
                }
                cout << "Order Shipped! Shipped: " << og << " Over
grip, " << rs << " Racket String, " << ts << " Tennis Shoe, " << tb
<< " Tennis Ball, " << ta << " Tennis Accessory \n";
                //sem.V(MUTEX);
            }

        }

    }
    childPID = fork();  //create child process for shipment receiver
to restock inventory
    if(childPID == 0) {
        while(true) {
            cout << "Shipment Arriving... \n";
            sleep(5); //delay each shipment
            cout << "childPID: " << childPID << " PID: " << getpid()
<< " Parent: " << getppid() << endl;

            cout << "Restocking Inventory \n";
            shipment_arrival((int*)shipment);
            sem.V(MUTEX);   //resume a customers order since the
inventory has
        }
    }
    exit(0);
}

/**
*creates temporary array to stock each item
**/
int *shipment_arrival(int shipment[]) {
    int *temp = new int[5];
    temp[0] = shipment[0] + 1;
    temp[1] = shipment[1] + 1;
    temp[2] = shipment[2] + 1;
    temp[3] = shipment[3] + 1;
    return temp;
}
```

semaphore.cpp

```
// This software may be used in your CECS326 programs

// simple implementation of SEMAPHORE class with some error
// and signal handling

#include "semaphore.h"

SEMAPHORE::SEMAPHORE(int size) {
        this->size = size;
        semid = semget(IPC_PRIVATE, size, PERMS);
        init();
        }

int SEMAPHORE::remove() {
        semun dummy;
        return semctl(semid, 0 /*not used*/, IPC_RMID, dummy);
        }

int SEMAPHORE::P(int id){
        int retval;
        struct sembuf *p = &((pset+id)->sb);
        while(((retval=semop(semid,p,1))==-1)&&(errno==EINTR));
        return retval;
}

int SEMAPHORE::V(int id){
        int retval;
        struct sembuf *v = &((vset+id)->sb);
        while(((retval=semop(semid,v,1))==-1)&&(errno==EINTR));
        return retval;
}

void SEMAPHORE::set_sembuf_p(int k, int op, int flg){
        (pset+k)->sb.sem_num = (short) k;
        (pset+k)->sb.sem_op = op;
        (pset+k)->sb.sem_flg = flg;
}

void SEMAPHORE::set_sembuf_v(int k, int op, int flg){
        (vset+k)->sb.sem_num = (short) k;
        (vset+k)->sb.sem_op = op;
        (vset+k)->sb.sem_flg = flg;
}

int SEMAPHORE::init() {
	pset = new mysembuf[size];
	vset = new mysembuf[size];
	for(int k=0; k<size; k++) {
		set_sembuf_p(k, -1, 0 /*suspend*/);
		set_sembuf_v(k, 1, 0 /*suspend*/);
	}

	// initialize all to zero
	semun arg;
	ushort initv[size];
	for(int k=0; k<size; k++) {
		initv[k]=0;
	}
	arg.array = initv;
	return semctl(semid, size, SETALL, arg);
}

SEMAPHORE::~SEMAPHORE() {
	delete pset;
	delete vset;
}
```

semaphore.h

```
// This software may be used in your CECS326 programs

#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#define PERMS S_IWUSR|S_IRUSR

class mysembuf {
public:
	struct sembuf sb;
};

// need to be checking for existing definition
union semun {		// truncated definition
	int val;	// value used with SETVAL
	ushort *array;	// array of values: GETALL and SETALL
};

class SEMAPHORE {
private:
	int size;
	pid_t semid;
	mysembuf *pset, *vset;

	int init();
	void set_sembuf_p(int, int, int);
	void set_sembuf_v(int, int, int);
public:
	// create a number of semaphores denoted by size
	// precondition: size is larger than zero
	// postcondition: all semaphores are initialized to zero
	SEMAPHORE(int size);
	~SEMAPHORE();

	// deallocates all semaphores created by constructor
	// precondition: existence of SEMAPHORE object
	// postcondition: all semaphores are removed
	int remove();

	// semaphore P operation on semaphore semname
	// precondition: existence of SEMAPHORE object
	// postcondition: semaphore decrements and process may be
blocked
	// return value: -1 denotes an error
	int P(int semname);

	// semaphore V operation on semaphore semname
	// precondition: existence of SEMAPHORE object
	// postcondition: semaphore increments and process may be
resumed
	// return value: -1 denotes an error
	int V(int semname);
};
```