May 8, 2021

GTU Department of Computer Engineering
CSE344 - Spring 2021
Homework 4 Report

Akif Kartal
171044098

1 Problem Definition

The problem is to implement **producer-consumer** problem by using POSIX threads and Semaphores as initiation to POSIX threads.

2 Solution

The homework was finished as expected in homework pdf file.

2.1 Some Problems and Solutions

2.1.1 Semaphore Usage

In order to provide synchronization between threads I used **6 POSIX unnamed semaphore** common between threads. Followings are my semaphores;

```
    sem_init(&run, 0, 0);
    sem_init(&mutex, 0, 1);
    sem_init(&empty, 0, 10);
    sem_init(&full, 0, 0);
    sem_init(&busy, 0, 0);
    sem_init(&dataMutex, 0, 1);
```

Usage purpose of each semaphore will be explained in detail coming pages.

2.1.2 Thread Creation

Main Thread

Main thread is main function of program and it is responsible to initialize important variables such as queue, semaphores, money etc.

Thread G

This thread is detached thread so its different than other. Also it is filling the queue therefore we need to **pass as parameter** the queue to the this thread. Note that queue is **not** global it is in main thread. See the code;

```
pthread_attr_t attr;
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

/*create G thread*/
queue *stdQueue = createQueue();
pthread_create(&idG, &attr, threadG, (void *)stdQueue);
pthread_attr_destroy(&attr);
```

Threads of student-for-hire

Here, we have threads as many as number of student but important thing is **they will use same thread function**. Therefore, we need to design a communication mechanism between each student thread and main thread. My solution is following;

Setting Up the Student Threads

Since, these threads are using same function somehow we need to distinguish each other. In order to do that we will pass an encapsulated data, which contains information about student for each thread while creating. Note that after creating **they need to wait** until all of them initialized with it's data. See the code;

```c
//data for each student to pass to the it's thread
typedef struct info
{
   char name[41];
   double price;
   double speed;
   double quality;
   double income;
   int solvedCount;
   int index;
   int isBusy;
   int isNotified;
   sem_t *notify;
   sem_t *startSolve;
   char currentHw;
   pthread_t id;
} student;
```

In this data semaphores will be used to communicate and synchronize. Other variables are clear.

Creating Students

```c
//N = number of student(itu,ytu etc.)
pthread_t tids[N];
student hiredStds[N];

/*create hired students threads and init them*/
for (int i = 0; i < N; i++)
{
   pthread_create(&tids[i], NULL, threadStd, &hiredStds[i]);
}
initStudents(hiredStds, fdStd, N, tids);
mainPrintStudents(hiredStds,N);
/*post that students can run now*/
if (sem_post(&run) == -1)
       errExit("sem_post");
/*student thread*/
void *threadStd(void *info)
{
   if (sem_wait(&run) == -1)
       errExit("sem_wait");
   if (sem_post(&run) == -1)
       errExit("sem_post");
   student *this = (student *)info;
   ...
}
```

2.1.3 Communication and Synchronization Problems

Main Thread and Thread G

I this two thread we have **producer-consumer** problem. In order to solve this problem I used 3 semaphore which are **full, empty, mutex**.

Solution;

Producer(Thread G)

```
do
  {
       nextHw = readOneChar(fdHw);
       if (nextHw != 'x' && !isFinished)
       {
            if (sem_wait(&empty) == -1)
                errExit("sem_wait");
            if (sem_wait(&mutex) == -1)
                errExit("sem_wait");
            if (isFinished)//money run out
                break;

           gNewHwMsg(nextHw,money);
           addRear(p, nextHw);

           if (sem_post(&mutex) == -1)
                errExit("sem_post");
           if (sem_post(&full) == -1)
                errExit("sem_post");
       }

  } while (nextHw != 'x' && !isFinished);
```

Consumer(Main Thread)

```
while (money > 0 && !isFinished)
{
       if (sem_wait(&full) == -1)
           errExit("sem_wait");
       if (sem_wait(&mutex) == -1)
           errExit("sem_wait");
       if (!isQueueEmpty(stdQueue)){
           index = findStudent(hiredStds, getFront(stdQueue));
           if (hiredStds[index].price <= money)
           {
                if (sem_wait(&dataMutex) == -1)
                    errExit("sem_wait");

                //remove hwk and update values
                hiredStds[index].currentHw = removeFront(stdQueue);
                hiredStds[index].isNotified += 1;
                money = money - hiredStds[index].price;
                hiredStds[index].income += hiredStds[index].price;
                hiredStds[index].solvedCount += 1;

                //notify the selected student to start solving it
                if (sem_post(hiredStds[index].notify) == -1)
                    errExit("sem_post");
                if (sem_post(&dataMutex) == -1)
                    errExit("sem_post");

                //wait first student to start
                if (sem_wait(hiredStds[index].startSolve) == -1)
                    errExit("sem_wait");

                if (sem_post(&mutex) == -1)
                  errExit("sem_post");
                if (sem_post(&empty) == -1)
                  errExit("sem_post");

           }
           else{
             break;
         }
       else{
           break;
       }
}
```

In this code piece some semaphores belongs to main thread and hired student communication and synchronization. Also, I determine Queue data structure **upper-bound limit is 10**.

Main Thread and Hired Students

I this two thread we have **communication and synchronization** problems. In order to solve this problems I used 3 semaphores which are **run, busy, dataMutex**.Also in info for each student we are using **notify and startSolve** seamphores.

Usage;

- run: When a student thread created, it will wait to initialize its data by main thread.
- busy: When all students are busy main thread will wait until one student post to busy thread.
- dataMutex: Both students and main thread are updating student info, therefore in order to avoid race condition we will use this semaphore.
- notify: notify a student when it was chosen.
- startSolve: After notification was send wait the student to start **not** completely finished solving.

The solution of these problems is following;

Main Thread

```
index = findStudent(hiredStds, getFront(stdQueue));
if (index == -1)
{
  //all of them are busy
  int counter;
  if (sem_getvalue(&busy, &counter) == -1)
    errExit("sem_get");
  if (counter > 0){
    if (sem_wait(&busy) == -1)
      errExit("sem_wait");
  }
  if (sem_wait(&busy) == -1)
    errExit("sem_wait");
  index = findStudent(hiredStds, getFront(stdQueue));
}
if (hiredStds[index].price <= money)
{
  if (sem_wait(&dataMutex) == -1)
    errExit("sem_wait");

  //remove hwk and update values
  hiredStds[index].currentHw = removeFront(stdQueue);
  hiredStds[index].isNotified += 1;
  money = money - hiredStds[index].price;
  hiredStds[index].income += hiredStds[index].price;
  hiredStds[index].solvedCount += 1;

  //notify the selected student to start solving it
  if (sem_post(hiredStds[index].notify) == -1)
    errExit("sem_post");
  if (sem_post(&dataMutex) == -1)
    errExit("sem_post");

  //wait first student to start
  if (sem_wait(hiredStds[index].startSolve) == -1)
    errExit("sem_wait");


}
```

Hired Student

```
while (!isFinished || this->isNotified)
{
  stdWaitMsg(this->name);
  //wait notificition
  if (sem_wait(this->notify) == -1)
    errExit("sem_wait");

  if (this->isNotified){
    if (sem_wait(&dataMutex) == -1)
      errExit("sem_wait");
    this->isBusy = 1;
    stdSolvingMsg(this->name,this->price,money,this->currentHw);

    //release start mutex
    if (sem_post(this->startSolve) == -1)
      errExit("sem_post");
    if (sem_post(&dataMutex) == -1)
      errExit("sem_post");

    //simulate solbing
    sleep(6 - (int)this->speed);

    if (sem_wait(&dataMutex) == -1)
      errExit("sem_wait");
    this->isNotified -= 1;
    this->isBusy = 0;

    if (sem_post(&dataMutex) == -1)
      errExit("sem_post");
    int counter;
    if (sem_getvalue(&busy, &counter) == -1)
      errExit("sem_get");
    if (counter == 0){
      if (sem_post(&busy) == -1)
        errExit("sem_post");
    }
  }
  else
    break;
}
```

In this solution 3 **binary** semaphore was used. These are **vac, cit, run**.

Note: In this solution by using semaphores, I provided print message on screen is happening like a queue. In other words, each citizen will print its message directly after getting an invite from a vaccinator and vaccinator message. I had to do this otherwise messages on the screen will be messed up.

2.1.4 Removing Resources and Exiting

In order to run a cleaner method after finish, I used a cleaner method like this;

```
    void removeAll()
    {
        if (sem_close(sem_full1) == -1)
            errExit("sem_close");
        if (sem_close(sem_empty) == -1)
            errExit("sem_close");
        if (sem_close(sem_full2) == -1)
            errExit("sem_close");
        if (sem_close(sem_run) == -1)
            errExit("sem_close");
        if (sem_close(sem_mutex) == -1)
            errExit("sem_close");
        if (sem_close(sem_vac) == -1)
            errExit("sem_close");
        if (sem_close(sem_cit) == -1)
            errExit("sem_close");
        sem_unlink("mutex344");
        sem_unlink("full344");
        sem_unlink("empty344");
```

```
20          sem_unlink("full3442");
21          sem_unlink("whorunfirst");
22          sem_unlink("wait_vaccinator");
23          sem_unlink("wait_citizen");
24          shm_unlink(memoryName); //generic data
25          shm_unlink(helperMemory); //current cit pid
26      }
```

Usage:

```
void cleanAndExit()
{

    if (processInfo.type == PARENT)
    {
        reapDeadChildren();
        if (close(biontech->fd) < 0)
        {
            errExit("close file error!");
        }
    }
    else
    {
        removeAll();
    }
    exit(EXIT_SUCCESS);
}
```

2.1.5 Reaping Dead Children

In order not to let zombie process, I used waitpid to wait dead children. Example;

```
void reapDeadChildren()
{
    pid_t childPid;
    int status;
    while ((childPid = waitpid(-1, &status, 0)) > 0);
    if (childPid == -1 && errno != ECHILD)
        errExit("waitpid");
}
```

2.1.6 CTRL-C Handling

In order to give a message on CTRL-C interrupt, I used **sigaction** function from **signal.h** library. Also, in midterm pdf there is no force to print a message on CTRL-C interrup, therefore I didn't give any message.

```
//handler function
void exitHandler(int signal)
{
    if (signal == SIGINT)
    {
        int savedErrno = errno;
        cleanAndExit();
        errno = savedErrno;
    }
}
```

There is no message, give resources back and exit elegantly.

3 References that was used

While doing this howemork following references was used;

- Course Textbook Listing 26-5: Reaping dead children via a handler for SIGCHLD
- Week-8 slides synchronization: Bounded buffer case in producer consumer problem.

4 Test Result

I tried to obey output example in midterm pdf. In following output all decisions belongs to CPU.

A simple test result is following;

```
akif@ubuntu:~/Desktop/system-programming/Midterm Project$ ./program -n 3 -v 2 -c 3 -b 11 -t 3 -i /home/akif/Desktop/source/mysource
Welcome to the GTU344 clinic. Number of citizens to vaccinate c=3 with t=3 doses.
Nurse 1 (pid=20123) has brought vaccine 1:the clinic has 1 vaccine1 and 0 vaccine2.
Nurse 1 (pid=20123) has brought vaccine 1:the clinic has 2 vaccine1 and 0 vaccine2.
Nurse 1 (pid=20123) has brought vaccine 1:the clinic has 3 vaccine1 and 0 vaccine2.
Nurse 1 (pid=20123) has brought vaccine 1:the clinic has 4 vaccine1 and 0 vaccine2.
Nurse 2 (pid=20124) has brought vaccine 1:the clinic has 5 vaccine1 and 0 vaccine2.
Nurse 2 (pid=20124) has brought vaccine 1:the clinic has 6 vaccine1 and 0 vaccine2.
Nurse 2 (pid=20124) has brought vaccine 1:the clinic has 7 vaccine1 and 0 vaccine2.
Nurse 2 (pid=20124) has brought vaccine 1:the clinic has 8 vaccine1 and 0 vaccine2.
Nurse 2 (pid=20124) has brought vaccine 1:the clinic has 9 vaccine1 and 0 vaccine2.
Nurse 3 (pid=20125) has brought vaccine 2:the clinic has 9 vaccine1 and 1 vaccine2.
Vaccinator 2 (pid=20122) is inviting citizen pid=20119 to the clinic
Citizen 2 (pid=20119) is vaccinated for the 1st time: the clinic has 8 vaccine1 and 0 vaccine2
Nurse 1 (pid=20123) has brought vaccine 2:the clinic has 8 vaccine1 and 1 vaccine2.
Nurse 1 (pid=20123) has brought vaccine 2:the clinic has 8 vaccine1 and 2 vaccine2.
Nurse 1 (pid=20123) has brought vaccine 2:the clinic has 8 vaccine1 and 3 vaccine2.
Vaccinator 2 (pid=20122) is inviting citizen pid=20118 to the clinic
Citizen 1 (pid=20118) is vaccinated for the 1st time: the clinic has 7 vaccine1 and 2 vaccine2
Vaccinator 2 (pid=20122) is inviting citizen pid=20120 to the clinic
Citizen 3 (pid=20120) is vaccinated for the 1st time: the clinic has 6 vaccine1 and 1 vaccine2
Nurse 3 (pid=20125) has brought vaccine 2:the clinic has 6 vaccine1 and 2 vaccine2.
Nurse 3 (pid=20125) has brought vaccine 2:the clinic has 6 vaccine1 and 3 vaccine2.
Vaccinator 2 (pid=20122) is inviting citizen pid=20119 to the clinic
Citizen 2 (pid=20119) is vaccinated for the 2nd time: the clinic has 5 vaccine1 and 2 vaccine2
Vaccinator 2 (pid=20122) is inviting citizen pid=20118 to the clinic
Citizen 1 (pid=20118) is vaccinated for the 2nd time: the clinic has 4 vaccine1 and 1 vaccine2
Nurse 3 (pid=20125) has brought vaccine 2:the clinic has 4 vaccine1 and 2 vaccine2.
Nurse 3 (pid=20125) has brought vaccine 2:the clinic has 4 vaccine1 and 3 vaccine2.
Nurse 3 (pid=20125) has brought vaccine 2:the clinic has 4 vaccine1 and 4 vaccine2.
Nurses have carried all vaccines to the buffer, terminating.
Vaccinator 1 (pid=20121) is inviting citizen pid=20120 to the clinic
Citizen 3 (pid=20120) is vaccinated for the 2nd time: the clinic has 3 vaccine1 and 3 vaccine2
Vaccinator 1 (pid=20121) is inviting citizen pid=20119 to the clinic
Citizen 2 (pid=20119) is vaccinated for the 3rd time: the clinic has 2 vaccine1 and 2 vaccine2
Citizen is leaving. Remaining citizens to vaccinate: 2
Vaccinator 1 (pid=20121) is inviting citizen pid=20118 to the clinic
Citizen 1 (pid=20118) is vaccinated for the 3rd time: the clinic has 1 vaccine1 and 1 vaccine2
Citizen is leaving. Remaining citizens to vaccinate: 1
Vaccinator 1 (pid=20121) is inviting citizen pid=20120 to the clinic
Citizen 3 (pid=20120) is vaccinated for the 3rd time: the clinic has 0 vaccine1 and 0 vaccine2
Citizen is leaving. Remaining citizens to vaccinate: 0
All citizens have been vaccinated.
Vaccinator 1 (pid=20121) vaccinated 4 doses.
Vaccinator 2 (pid=20122) vaccinated 5 doses.
The clinic is now closed. Stay healthy.
akif@ubuntu:~/Desktop/system-programming/Midterm Project$
```

Figure 1