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. Also reading queue.

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 (safeSemWait(&run) == -1)
       pthread_exit(NULL);
   if (sem_post(&run) == -1)
       errExit("sem_post");
   if (exitSignal)
       pthread_exit(NULL);
   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 (safeSemWait(&empty) == -1)
              pthread_exit(NULL);
          if (exitSignal)
              pthread_exit(NULL);
          if (safeSemWait(&mutex) == -1)
              pthread_exit(NULL);
          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 (safeSemWait(&full) == -1)
          cleanAndExit(hiredStds,tids,N,stdQueue);
      if (exitSignal)
          cleanAndExit(hiredStds,tids,N,stdQueue);
      if (safeSemWait(&mutex) == -1)
          cleanAndExit(hiredStds,tids,N,stdQueue);
      if (!isQueueEmpty(stdQueue)){
          index = findStudent(hiredStds, getFront(stdQueue));
          if (hiredStds[index].price <= money)
          {
              if (safeSemWait(&dataMutex) == -1)
                  cleanAndExit(hiredStds,tids,N,stdQueue);

              //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 (safeSemWait(hiredStds[index].startSolve) == -1)
                  cleanAndExit(hiredStds,tids,N,stdQueue);
              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 (safeSemWait(&busy) == -1)
          cleanAndExit(hiredStds,tids,N,stdQueue);
  }
  if (safeSemWait(&busy) == -1)
      cleanAndExit(hiredStds,tids,N,stdQueue);
  index = findStudent(hiredStds, getFront(stdQueue));
}
if (hiredStds[index].price <= money)
{
  if (safeSemWait(&dataMutex) == -1)
      cleanAndExit(hiredStds,tids,N,stdQueue);

  //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 (safeSemWait(hiredStds[index].startSolve) == -1)
      cleanAndExit(hiredStds,tids,N,stdQueue);
}
if (exitSignal)
    cleanAndExit(hiredStds,tids,N,stdQueue);
```

Hired Student

```
while (!isFinished || this->isNotified)
{
  stdWaitMsg(this->name);
  //wait notificition
  if (safeSemWait(this->notify) == -1)
      pthread_exit(NULL);

  if (this->isNotified){

    if (safeSemWait(&dataMutex) == -1)
          pthread_exit(NULL);

    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 solving
    sleep(6 - (int)this->speed);

     if (safeSemWait(&dataMutex) == -1)
          pthread_exit(NULL);

    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;
}
```

2.1.4 CTRL-C Handling

In order to terminate on CTRL-C interrupt, I used **sigaction** function from **signal.h** library.But in order to do that we need to wait each thread return and then give resources back. Also, while in sem_wait(), we need to do this anyway threfore I wrote **my safeSemWait() function.** See my solution;

Handler and Usage

```
    //handler function
  volatile __sig_atomic_t exitSignal = 0;
  void exitHandler(int signal)
  {
      if (signal == SIGINT)
      {
          exitSignal = 1;
      }
  }
    //my sem_wait
  int safeSemWait(sem_t *sem){
      if (sem_wait(sem) == -1){
          if (errno == EINTR && exitSignal == 1){
              return -1;
          } else{
              errExit("sem_wait interrupt");
          }
      }
      return 0;
  }
  //usage in main thread
  if (exitSignal)
      cleanAndExit(hiredStds,tids,N,stdQueue);
    //or
    if (safeSemWait(&full) == -1)
       cleanAndExit(hiredStds,tids,N,stdQueue);

    //usage in other threads
    if (exitSignal)
      pthread_exit(NULL);
    //or
    if (safeSemWait(&dataMutex) == -1)
      pthread_exit(NULL);
```

Cleaner function after a exit signal

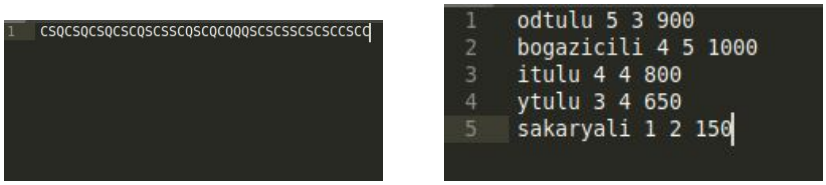
```
    //cleaner function
void cleanAndExit(student stds[],pthread_t ids[],int n,queue *head){
    printf("Termination signal received, closing.\n");
    if (sem_post(&dataMutex) == -1)
        errExit("sem_post");
    if (sem_post(&mutex) == -1)
        errExit("sem_post");
    if (sem_post(&empty) == -1)
        errExit("sem_post");
    for (int i = 0; i < n; ++i) {
        if (sem_post(stds[i].notify) == -1)
            errExit("sem_post");
    }
    for (int i = 0; i < n; i++)
    {
        if (!pthread_equal(pthread_self(), ids[i]))
            pthread_join(ids[i], NULL);
    }
    freeQueue(head);
    sem_destroy(&run);
    sem_destroy(&mutex);
    sem_destroy(&full);
    sem_destroy(&dataMutex);
    sem_destroy(&busy);
    sem_destroy(&empty);
    destroy(stds,n);
    exit(EXIT_SUCCESS);
}
```

3 References that was used

- Advanced Linux Programming Book Chapter 4 Threads
- Week-8 slides synchronization: Bounded buffer case in producer consumer problem.

4 Test Result

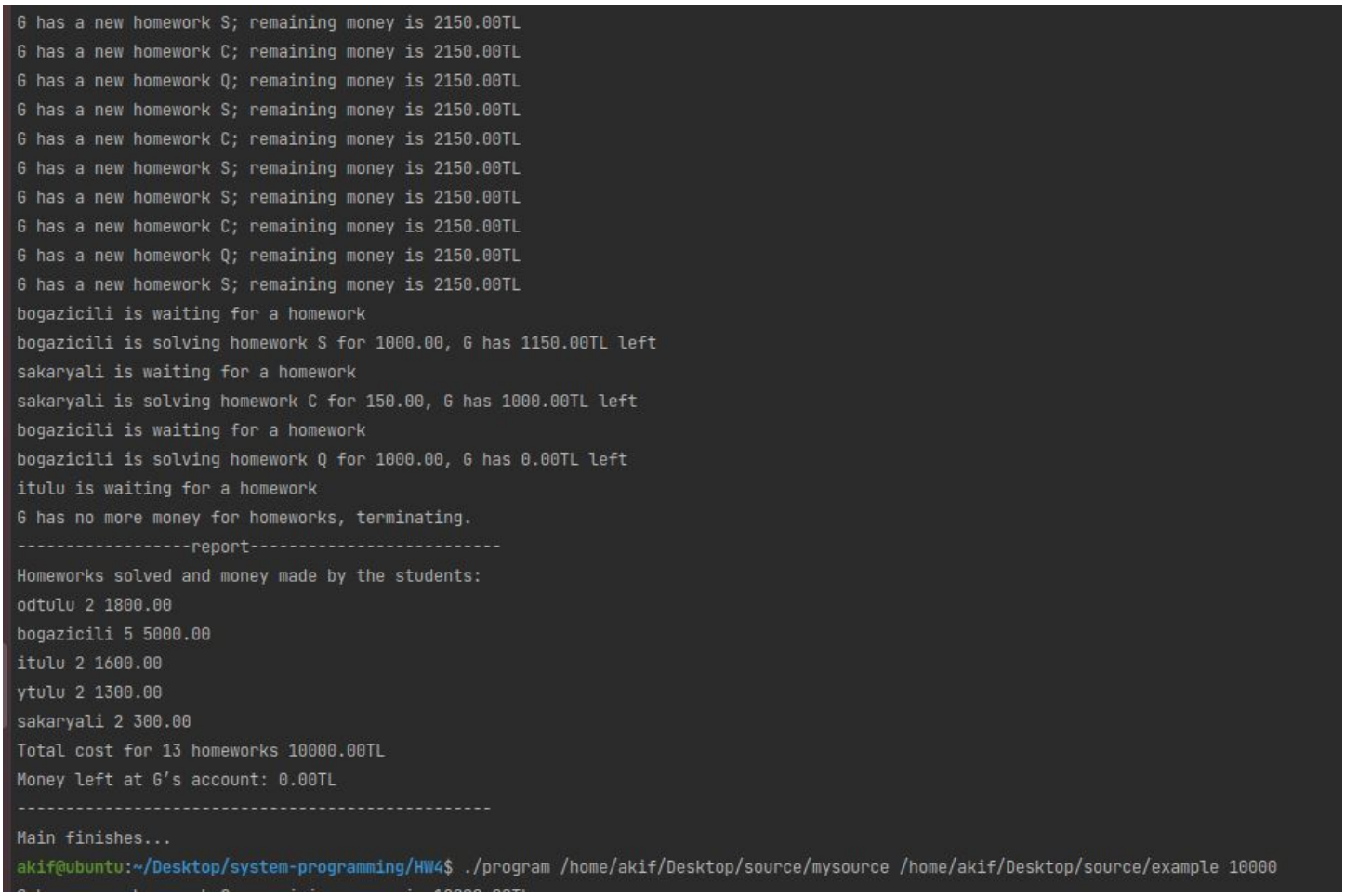
I tried to obey output example in midterm pdf. In following output all decisions belongs to CPU.
A simple test result is following;



(a) Homework File (b) Hired Students

Figure 1: Files Before Test

```
akif@ubuntu:~/Desktop/system-programming/HW4$ ./program /home/akif/Desktop/source/mysource /home/akif/Desktop/source/example 10000
G has a new homework C; remaining money is 10000.00TL
G has a new homework S; remaining money is 10000.00TL
G has a new homework Q; remaining money is 10000.00TL
G has a new homework C; remaining money is 10000.00TL
G has a new homework S; remaining money is 10000.00TL
G has a new homework Q; remaining money is 10000.00TL
G has a new homework C; remaining money is 10000.00TL
G has a new homework S; remaining money is 10000.00TL
G has a new homework Q; remaining money is 10000.00TL
G has a new homework C; remaining money is 10000.00TL
-----------------------------------------------
5 students-for-hire threads have been created.
Name    Q       S       C
odtulu 5.00 3.00 900.00
bogazicili 4.00 5.00 1000.00
itulu 4.00 4.00 800.00
ytulu 3.00 4.00 650.00
sakaryali 1.00 2.00 150.00
-----------------------------------------------
odtulu is waiting for a homework
ytulu is waiting for a homework
sakaryali is waiting for a homework
sakaryali is solving homework C for 150.00, G has 9850.00TL left
bogazicili is waiting for a homework
bogazicili is solving homework S for 1000.00, G has 8850.00TL left
odtulu is solving homework Q for 900.00, G has 7950.00TL left
ytulu is solving homework C for 650.00, G has 7300.00TL left
itulu is waiting for a homework
itulu is solving homework S for 800.00, G has 6500.00TL left
bogazicili is waiting for a homework
bogazicili is solving homework Q for 1000.00, G has 5500.00TL left
bogazicili is waiting for a homework
bogazicili is solving homework C for 1000.00, G has 4500.00TL left
itulu is waiting for a homework
ytulu is waiting for a homework
itulu is solving homework S for 800.00, G has 3700.00TL left
ytulu is solving homework Q for 650.00, G has 3050.00TL left
odtulu is waiting for a homework
odtulu is solving homework C for 900.00, G has 2150.00TL left
```


G has a new homework S; remaining money is 2150.00TL
G has a new homework C; remaining money is 2150.00TL
G has a new homework Q; remaining money is 2150.00TL
G has a new homework S; remaining money is 2150.00TL
G has a new homework C; remaining money is 2150.00TL
G has a new homework S; remaining money is 2150.00TL
G has a new homework S; remaining money is 2150.00TL
G has a new homework C; remaining money is 2150.00TL
G has a new homework Q; remaining money is 2150.00TL
G has a new homework S; remaining money is 2150.00TL
bogazicili is waiting for a homework
bogazicili is solving homework S for 1000.00, G has 1150.00TL left
sakaryali is waiting for a homework
sakaryali is solving homework C for 150.00, G has 1000.00TL left
bogazicili is waiting for a homework
bogazicili is solving homework Q for 1000.00, G has 0.00TL left
itulu is waiting for a homework
G has no more money for homeworks, terminating.
------------------report-------------------------
Homeworks solved and money made by the students:
odtulu 2 1800.00
bogazicili 5 5000.00
itulu 2 1600.00
ytulu 2 1300.00
sakaryali 2 300.00
Total cost for 13 homeworks 10000.00TL
Money left at G's account: 0.00TL

Main finishes...
akif@ubuntu:~/Desktop/system-programming/HW4$ ./program /home/akif/Desktop/source/mysource /home/akif/Desktop/source/example 10000

Figure 2