OS END SEMESTER EXAMINATION

PART C

COE18B064
MIHIR SHRI

1. LCS of two strings

Code:

“lcs.c” -
```
#include <pthread.h>
#include "util.h"

void *computeBlock(void * myid);

pthread_barrier_t barr;
int NUM_THREADS;
int SUBY_SIZE;
int SUBMAT_SIZE;
int **fTab;
int xSize,ySize,num_blocksX,num_blocksY;
char **subX;
char **subY;

int traceback(char *lineY, char *lineX, int **fTab, int i, int j, int count);

int main(int argc, char *argv[]) {
   if (argc != 4) {
         printf("Missing arguments: lcs input1 input2 numThreads \n");
         exit(0);
   }
   NUM_THREADS = atoi(argv[3]);
   pthread_t threads[NUM_THREADS];
   printf("PThreads Num Threads = %d\n",NUM_THREADS);
   double timer_1;
   int i,j,worker,b,k;
```

```
size_t lnlen1,lnlen2;
char *lineX=NULL; char *lineY=NULL;
FILE *inputFile1; FILE *inputFile2;

inputFile1 = fopen(argv[1],"r");
inputFile2 = fopen(argv[2],"r");
xSize = getline(&lineX,&lnlen1,inputFile1);
ySize = getline(&lineY,&lnlen2,inputFile2);
if (lineX[xSize-1] == '\n')
        xSize--;
if (lineY[ySize-1] == '\n')
        ySize--;
fclose(inputFile1); fclose(inputFile2);

fTab = (int **)malloc((ySize+1) * sizeof(int *));
for (i = 0; i <= ySize ; i++) {
        fTab[i] = (int *)malloc((xSize+1) * sizeof(int));
}

SUBY_SIZE = 100;
SUBMAT_SIZE = ceil((double)xSize / NUM_THREADS);
num_blocksY = ceil((double)ySize / SUBY_SIZE);
num_blocksX = ceil((double)xSize / SUBMAT_SIZE);

pthread_barrier_init(&barr,NULL,util_min(num_blocksY,NUM_THREADS));

subX = (char **)malloc(sizeof(char *) * num_blocksX);
subY = (char **)malloc(sizeof(char *) * num_blocksY);

for (b = 0; b < num_blocksX; b++) {
        subX[b] = (char *)malloc(sizeof(char) * (SUBMAT_SIZE+1));
        if (((b+1) * SUBMAT_SIZE) <= xSize) {
                strncpy(subX[b],lineX+(b*SUBMAT_SIZE),SUBMAT_SIZE);
        }
        else {
                strncpy(subX[b],lineX+(b*SUBMAT_SIZE),
                                xSize - b*SUBMAT_SIZE);
                subX[b][xSize - b*SUBMAT_SIZE] = '\0';
        }
}
```

```
for (b = 0; b < num_blocksY; b++) {
        subY[b] = (char *)malloc(sizeof(char) * (SUBY_SIZE+1));
        if (((b+1) * SUBY_SIZE) <= ySize) {
                strncpy(subY[b],lineY+(b*SUBY_SIZE),SUBY_SIZE);
                subY[b][SUBY_SIZE] = '\0';
        }
        else {
                strncpy(subY[b],lineY+(b*SUBY_SIZE), ySize - b*SUBY_SIZE);
                subY[b][ySize - b*SUBY_SIZE] = '\0';
        }
}


for ( i = 0; i <= ySize; i++) {
        fTab[i][0] = 0;
}
for ( j = 0; j <= xSize; j++) {
        fTab[0][j] = 0;
}

clear_timer(timer_1);
  start_timer(timer_1);

for (worker = 0; worker < NUM_THREADS; worker++) {
        pthread_create(&threads[worker],NULL,computeBlock,(void *)&worker);
}

for (worker = 0; worker < NUM_THREADS; worker++) {
        pthread_join(threads[worker],NULL);
}

  stop_timer(timer_1);
  printf("Time Taken: %.6lfs\n", get_timer(timer_1));

traceback(lineY,lineX,fTab,ySize,xSize,0);
printf("\n");

free(lineX); free(lineY);
for (i = 0; i < num_blocksX; i++)
```

```
            free(subX[i]);
    free(subX);
    for (i = 0; i < num_blocksY; i++)
            free(subY[i]);
    free(subY);
    for (i = 0; i <= ySize; i++)
            free(fTab[i]);
    free(fTab);
}

void *computeBlock(void * myid) {

    int id = *(int *)myid ;
    int i,d,j,r,c,rSize,cSize,addR,addC;

    d = 0;
            for (r = id; r < num_blocksY; r = r + NUM_THREADS) {
            addR = r * SUBY_SIZE;
            rSize = strlen(subY[r]);

            for (c = 0; c <= num_blocksX &&
                            d < (num_blocksY+num_blocksX-1) ; c++) {

                    if (c == num_blocksX && d < (num_blocksY-1)) {
                            break;
                    }
                    else if (c == num_blocksX && d >= (num_blocksY-1)) {
                            pthread_barrier_wait(&barr);
                            d++;
                            c--;
                            continue;
                    }
                    while (c > (d-r) ) {
                            pthread_barrier_wait(&barr);
                            d++;
                    }

                    cSize = strlen(subX[c]);
                    addC = c * SUBMAT_SIZE;
                    for (i = 0; i < rSize; i++) {

                              for (j = 0; j < cSize ; j++) {

                                        if (subX[c][j] == subY[r][i]) {
                                                  fTab[i+addR+1][j+addC+1] =
                                                                      fTab[i+addR][j+addC] + 1;
                                        }
                                        else {
                                                  fTab[i+addR+1][j+addC+1] =
                                                            util_max(fTab[i+addR][j+addC+1],
                                                                      fTab[i+addR+1][j+addC]);
                                        }
                              }
                    }
                    }
   }
          pthread_exit(NULL);
}

double wClockSeconds(void)
{
#ifdef __GNUC__
  struct timeval ctime;

  gettimeofday(&ctime, NULL);

  return (double)ctime.tv_sec + (double).000001*ctime.tv_usec;
#else
  return (double)time(NULL);
#endif
}

int traceback(char *lineY, char *lineX, int **fTab, int i, int j, int count) {
   if (i == 0 || j == 0) {
          printf("Length = %d\n",count);
          return 0;
   }
   if (lineX[j-1] == lineY[i-1]) {
          count++;
```

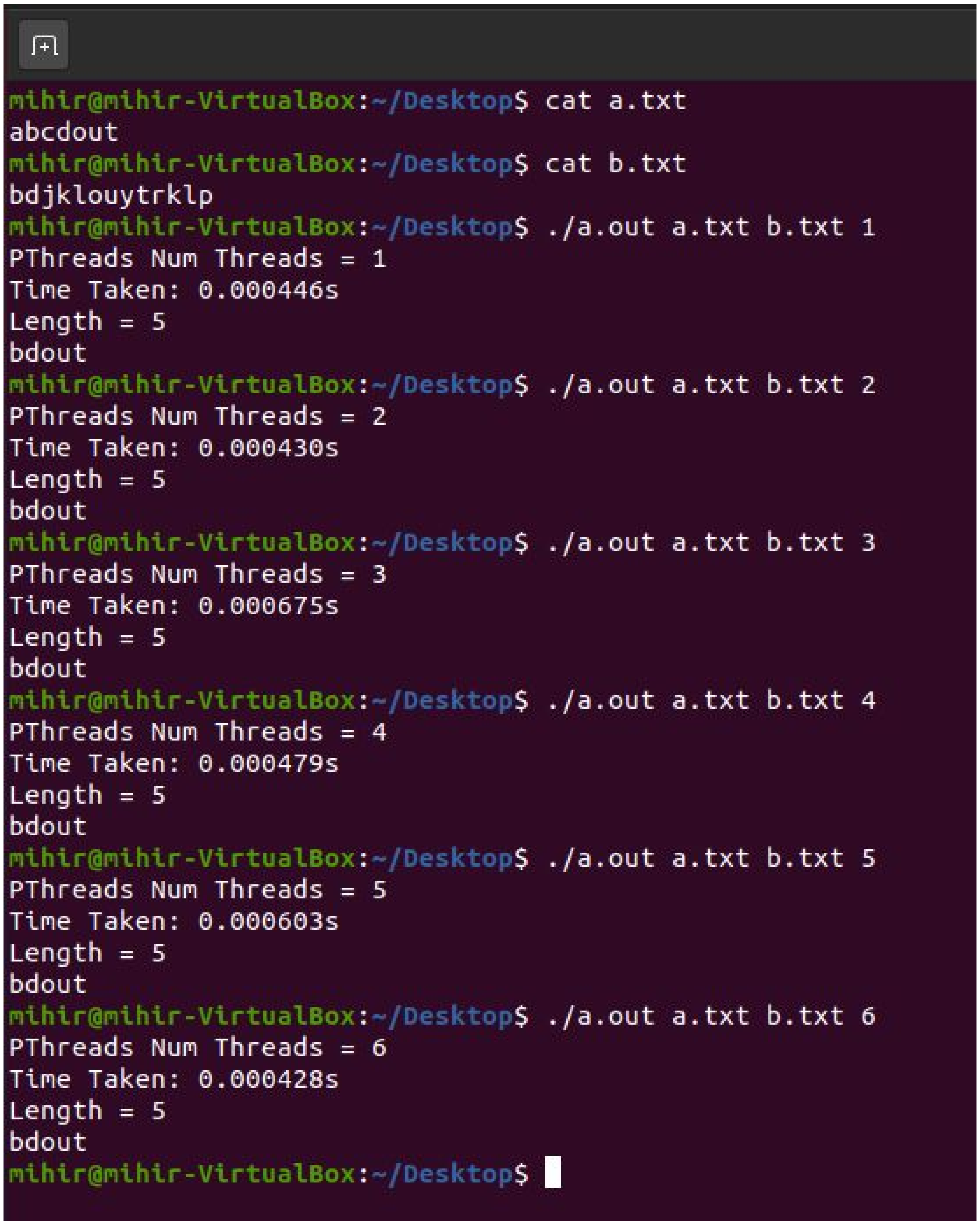
```
            traceback(lineY,lineX,fTab,i-1,j-1,count);
            printf("%c",lineX[j-1]);
    }
    else if (fTab[i-1][j] == fTab[i][j]) {
            traceback(lineY,lineX,fTab,i-1,j,count);
    }
    else {
            traceback(lineY,lineX,fTab,i,j-1,count);
    }
}
```

“utils.h” -

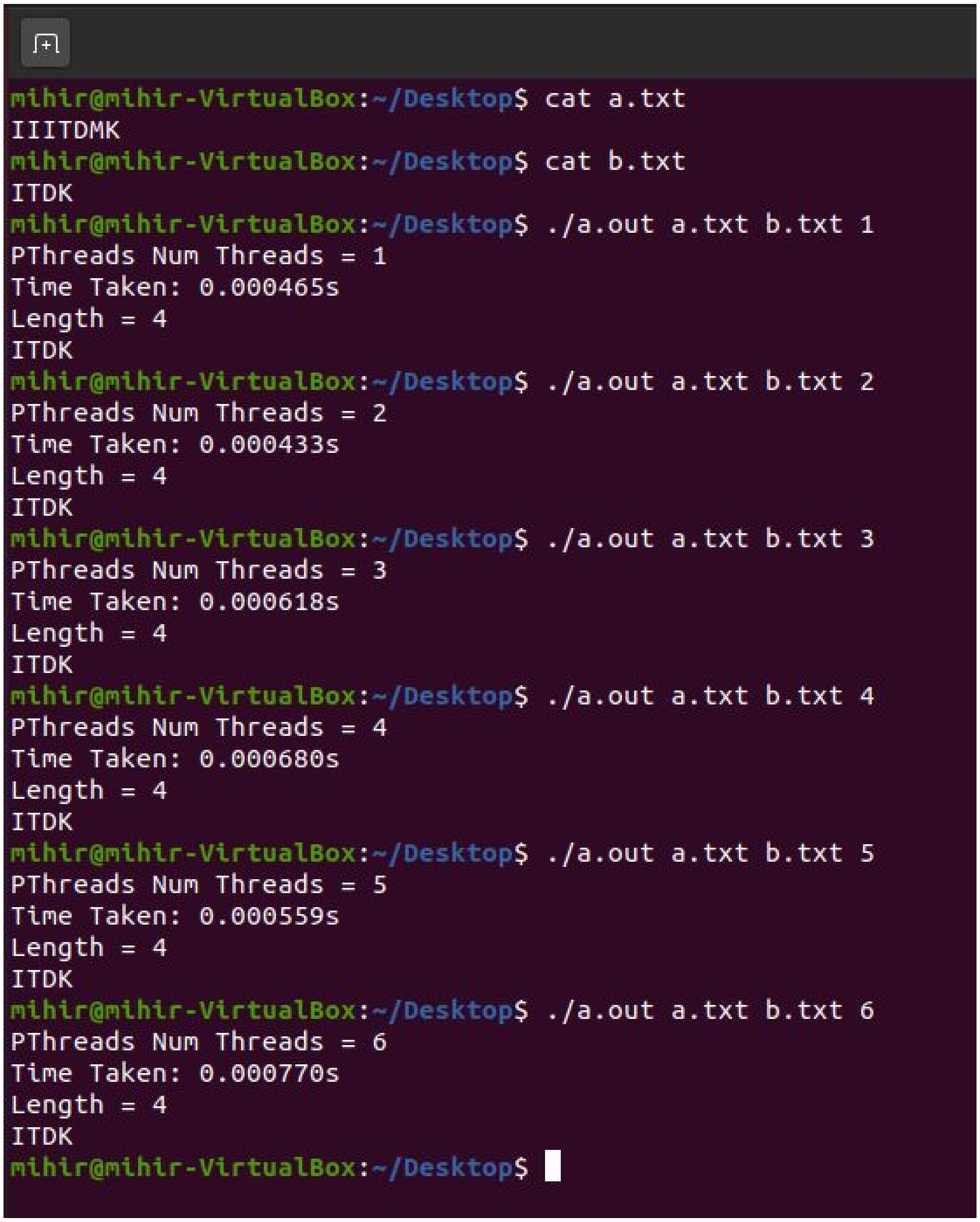
```
#include <stddef.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdarg.h>
#include <stdio.h>
#include <errno.h>
#include <ctype.h>
#include <math.h>
#include <float.h>
#include <time.h>
#include <string.h>
#include <limits.h>
#include <signal.h>
#include <setjmp.h>
#include <assert.h>
#include <inttypes.h>
#include <sys/resource.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

#define clear_timer(t) (t = 0.0)
#define start_timer(t) (t -= wClockSeconds())
#define stop_timer(t)  (t += wClockSeconds())
#define get_timer(t)   (t)
```

```
#define util_max(a, b) ((a) >= (b) ? (a) : (b))
#define util_min(a, b) ((a) >= (b) ? (b) : (a))

double wClockSeconds(void);
```

Output ss:



```
mihir@mihir-VirtualBox:~/Desktop$ cat a.txt
abcdout
mihir@mihir-VirtualBox:~/Desktop$ cat b.txt
bdjklouytrklp
mihir@mihir-VirtualBox:~/Desktop$ ./a.out a.txt b.txt 1
PThreads Num Threads = 1
Time Taken: 0.000446s
Length = 5
bdout
mihir@mihir-VirtualBox:~/Desktop$ ./a.out a.txt b.txt 2
PThreads Num Threads = 2
Time Taken: 0.000430s
Length = 5
bdout
mihir@mihir-VirtualBox:~/Desktop$ ./a.out a.txt b.txt 3
PThreads Num Threads = 3
Time Taken: 0.000675s
Length = 5
bdout
mihir@mihir-VirtualBox:~/Desktop$ ./a.out a.txt b.txt 4
PThreads Num Threads = 4
Time Taken: 0.000479s
Length = 5
bdout
mihir@mihir-VirtualBox:~/Desktop$ ./a.out a.txt b.txt 5
PThreads Num Threads = 5
Time Taken: 0.000603s
Length = 5
bdout
mihir@mihir-VirtualBox:~/Desktop$ ./a.out a.txt b.txt 6
PThreads Num Threads = 6
Time Taken: 0.000428s
Length = 5
bdout
mihir@mihir-VirtualBox:~/Desktop$
```



```
mihir@mihir-VirtualBox:~/Desktop$ cat a.txt
IIITDMK
mihir@mihir-VirtualBox:~/Desktop$ cat b.txt
ITDK
mihir@mihir-VirtualBox:~/Desktop$ ./a.out a.txt b.txt 1
PThreads Num Threads = 1
Time Taken: 0.000465s
Length = 4
ITDK
mihir@mihir-VirtualBox:~/Desktop$ ./a.out a.txt b.txt 2
PThreads Num Threads = 2
Time Taken: 0.000433s
Length = 4
ITDK
mihir@mihir-VirtualBox:~/Desktop$ ./a.out a.txt b.txt 3
PThreads Num Threads = 3
Time Taken: 0.000618s
Length = 4
ITDK
mihir@mihir-VirtualBox:~/Desktop$ ./a.out a.txt b.txt 4
PThreads Num Threads = 4
Time Taken: 0.000680s
Length = 4
ITDK
mihir@mihir-VirtualBox:~/Desktop$ ./a.out a.txt b.txt 5
PThreads Num Threads = 5
Time Taken: 0.000559s
Length = 4
ITDK
mihir@mihir-VirtualBox:~/Desktop$ ./a.out a.txt b.txt 6
PThreads Num Threads = 6
Time Taken: 0.000770s
Length = 4
ITDK
mihir@mihir-VirtualBox:~/Desktop$
```

Logic:
The code requires 2 aspects for generation of the longest common subsequence. The first aspect being initialization i.e. initialize the matrix for the base values of the substructure table. The next aspect is implementing the approach.

As mentioned in the question, we have to create an optimal number of threads for this task. But, in LCS problem, the optimal number of threads is not static or fixed. It depends on the length of the input strings. More number of threads doesn’t necessarily mean that the time of execution will be less. This is because after a certain number threads have been created, in relation to the purpose of the application, threads will spend most of its time communicating with each other rather than processing data. With the creation of each processing thread, comes a cost in terms of an overhead. When the total cost of the overhead becomes a greater factor than the extra resources it provides, the parallel slowdown occurs.

So, the number of threads has to be passed as a command line argument. There are 3 command line arguments, 1st is for the file containing the input string “a”, 2nd is for the file containing input string “b”, and 3rd is the number of threads the user desires to create.

Firstly, the strings are read from the file and stored in variables. Then, the dynamic programming approach has been used to find the longest common subsequence (explained at the end). The number of threads created will be passed as a command line argument and accordingly the compute blocks function is called.

The compute block function divides the strings into n parts (n being the number of threads) and then finds the LCS using DP approach.

The DP approach to finding LCS

Let the input sequences be X[0..m-1] and Y[0..n-1] of lengths m and n respectively. And let L(X[0..m-1], Y[0..n-1]) be the length of LCS of the two sequences X and Y. Following is the recursive definition of L(X[0..m-1], Y[0..n-1]).

If last characters of both sequences match (or X[m-1] == Y[n-1]) then

L(X[0..m-1], Y[0..n-1]) = 1 + L(X[0..m-2], Y[0..n-2])

If last characters of both sequences do not match (or X[m-1] != Y[n-1]) then

L(X[0..m-1], Y[0..n-1]) = MAX (L(X[0..m-2], Y[0..n-1]), L(X[0..m-1], Y[0..n-2]))

2. Our barbershop has three chairs, three barbers, and a waiting area that can accommodate four customers on a sofa and that has standing room for additional customers. Fire codes limit the total number of customers in the shop to 20.
A customer will not enter the shop if it is filled to capacity with other customers. Once inside, the customer takes a seat on the sofa or stands if the sofa is filled. When a barber is free, the customer that has been on the sofa the longest is served and, if there are any standing customers, the one that has been in the shop the longest takes a seat on the sofa. When a customer's haircut is finished, any barber can accept payment, but because there is only one cash register, payment is accepted for one customer at a time. The barbers divide their time among cutting hair, accepting payment, and sleeping in their chair waiting for a customer.
In other words, the following synchronization constraints apply:
• Customers invoke the following functions in order: enterShop, sitOnSofa, sitInBarberChair, pay, exitShop.
• Barbers invoke cutHair and acceptPayment.
• Customers cannot invoke enterShop if the shop is at capacity.
• If the sofa is full, an arriving customer cannot invoke sitOnSofa until one of the customers on the sofa invokes sitInBarberChair.
• If all three barber chairs are busy, an arriving customer cannot invoke sitInBarberChair until one of the customers in a chair invokes pay.
• The customer has to pay before the barber can acceptPayment.
• The barber must acceptPayment before the customer can exitShop.

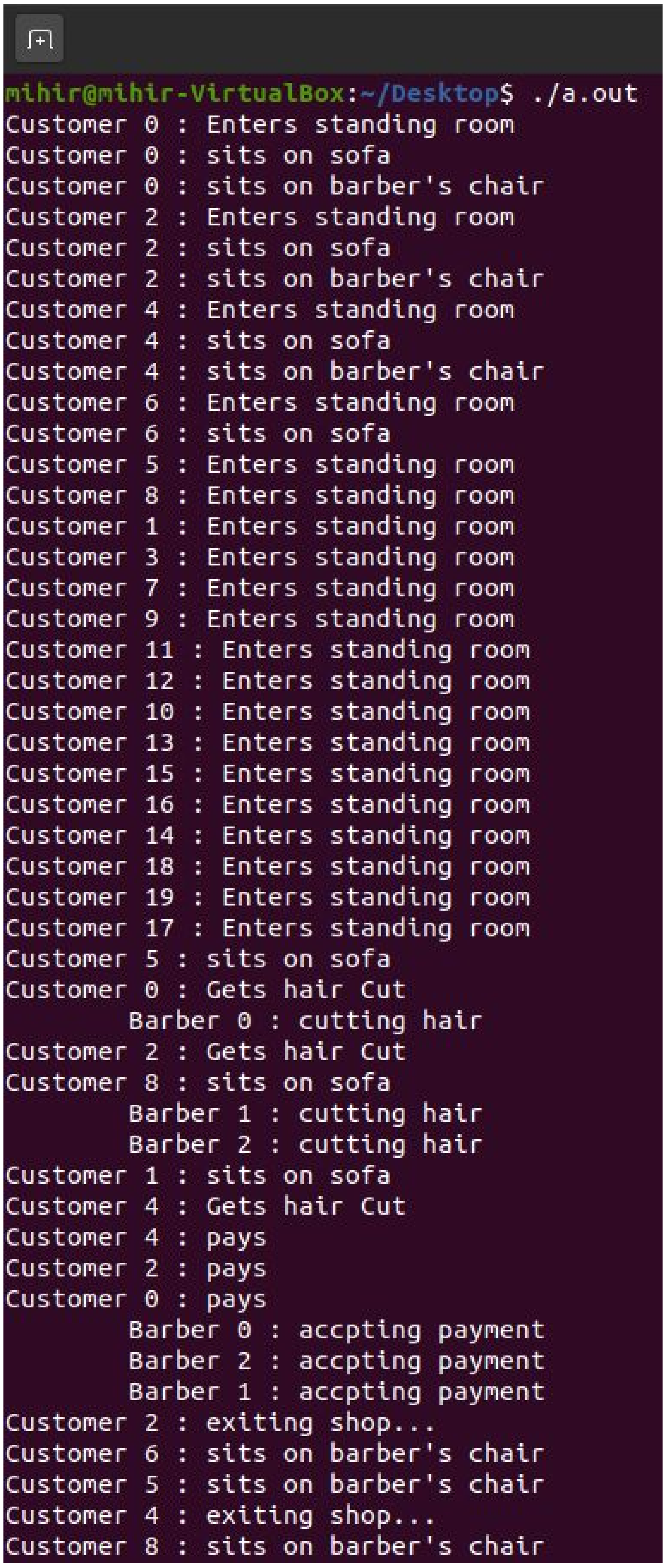
Code:

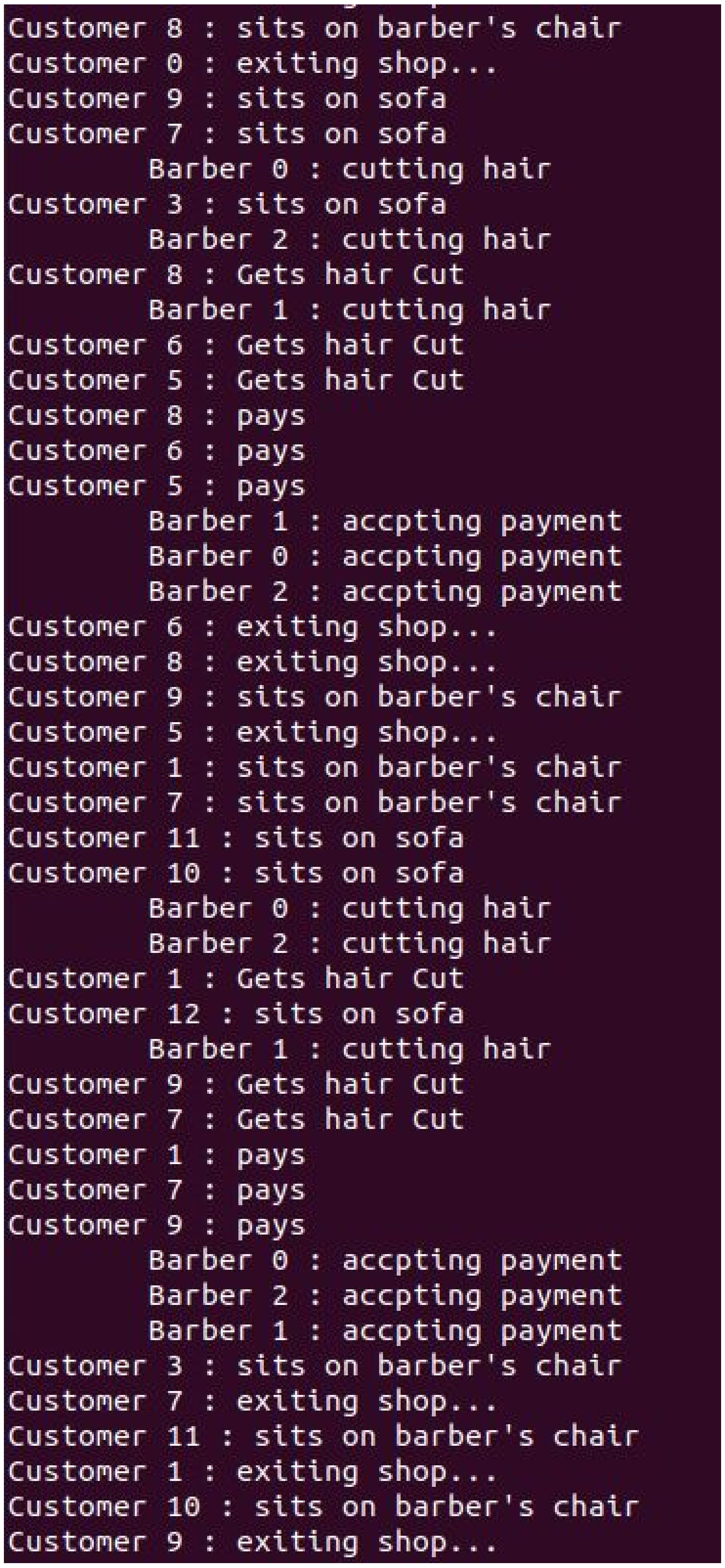
```
#include<pthread.h>
#include<semaphore.h>
#include<stdlib.h>
#include<stdio.h>
#define MAX 20
#define NO_OF_CUST 20

int customers=0;
pthread_t cust[NO_OF_CUST];
pthread_t barb[3];
sem_t mutex, chair, barber, customer, cash, reciept;
typedef struct{
```

```
sem_t leader;
sem_t follower;
}Fifo;
Fifo *standingRoom,*sofa;
Fifo* make_Fifo(int n)
{
Fifo* F=(Fifo*)malloc(sizeof(Fifo));
sem_init(&(F->leader),0,0);
sem_init(&(F->follower),0,n);
return(F);
}
void wait_Fifo(Fifo* F,int n)
{
sem_wait(&(F->follower));
sem_post(&(F->leader));
}
void signal_Fifo(Fifo* F)
{
sem_wait(&(F->leader));
sem_post(&(F->follower));
}
void* barbershop(void *arg)
{
int n=*(int *)arg;
//while(1)
//{
sem_wait(&mutex);
if(customers>=20)
{
sem_post(&mutex);
printf("Customer %d : exiting shop... \n",n);       //exit_shop();
}
customers+=1;
sem_post(&mutex);
wait_Fifo(standingRoom,n);          //standing_room_wait();
printf("Customer %d : Enters standing room \n",n);      //enter_shop();
wait_Fifo(sofa,n);     //sofa_wait();


printf("Customer %d : sits on sofa \n",n);           //sitOnSofa();
//sleep(1);
signal_Fifo(standingRoom);          //standing_room_signal();
sem_wait(&chair);
printf("Customer %d : sits on barber's chair \n",n);        //SitOnBarberChair();
sleep(3);
signal_Fifo(sofa);
sem_post(&customer);
sem_wait(&barber);
printf("Customer %d : Gets hair Cut \n",n);         //getHairCut();
sleep(2);
printf("Customer %d : pays \n",n);            //pay();
sem_post(&cash);
sem_wait(&reciept);
sem_wait(&mutex);
customers-=1;
sem_post(&mutex);
printf("Customer %d : exiting shop... \n",n);
//sleep(6);
//exit_shop);
//}
}
void* cutting(void* arg)
{
int n=*(int *)arg;
while(1)
{
sem_wait(&customer);
sem_post(&barber);
printf("\tBarber %d : cutting hair\n",n);
sleep(3);
sem_wait(&cash);
printf("\tBarber %d : accpting payment\n",n);
sleep(1);
sem_post(&reciept);
sem_post(&chair);
}
}
main()
{
int i,cust_id[NO_OF_CUST];
```

```
sem_init(&mutex,0,1);
sem_init(&chair,0,3);
sem_init(&barber,0,0);
sem_init(&customer,0,0);
sem_init(&cash,0,0);
sem_init(&reciept,0,0);
standingRoom=make_Fifo(16);
sofa=make_Fifo(4);
for(i=0;i<NO_OF_CUST;i++){
cust_id[i]=i;


pthread_create(&cust[i],0,barbershop,&cust_id[i]);
}
for(i=0;i<3;i++)
pthread_create(&barb[i],0,cutting,&cust_id[i]);
while(1);
}
```

Output ss:
(Output shown only upto a certain point)


```
mihir@mihir-VirtualBox:~/Desktop$ ./a.out
Customer 0 : Enters standing room
Customer 0 : sits on sofa
Customer 0 : sits on barber's chair
Customer 2 : Enters standing room
Customer 2 : sits on sofa
Customer 2 : sits on barber's chair
Customer 4 : Enters standing room
Customer 4 : sits on sofa
Customer 4 : sits on barber's chair
Customer 6 : Enters standing room
Customer 6 : sits on sofa
Customer 5 : Enters standing room
Customer 8 : Enters standing room
Customer 1 : Enters standing room
Customer 3 : Enters standing room
Customer 7 : Enters standing room
Customer 9 : Enters standing room
Customer 11 : Enters standing room
Customer 12 : Enters standing room
Customer 10 : Enters standing room
Customer 13 : Enters standing room
Customer 15 : Enters standing room
Customer 16 : Enters standing room
Customer 14 : Enters standing room
Customer 18 : Enters standing room
Customer 19 : Enters standing room
Customer 17 : Enters standing room
Customer 5 : sits on sofa
Customer 0 : Gets hair Cut
        Barber 0 : cutting hair
Customer 2 : Gets hair Cut
Customer 8 : sits on sofa
        Barber 1 : cutting hair
        Barber 2 : cutting hair
Customer 1 : sits on sofa
Customer 4 : Gets hair Cut
Customer 4 : pays
Customer 2 : pays
Customer 0 : pays
        Barber 0 : accpting payment
        Barber 2 : accpting payment
        Barber 1 : accpting payment
Customer 2 : exiting shop...
Customer 6 : sits on barber's chair
Customer 5 : sits on barber's chair
Customer 4 : exiting shop...
Customer 8 : sits on barber's chair
```



```
Customer 8 : sits on barber's chair
Customer 0 : exiting shop...
Customer 9 : sits on sofa
Customer 7 : sits on sofa
        Barber 0 : cutting hair
Customer 3 : sits on sofa
        Barber 2 : cutting hair
Customer 8 : Gets hair Cut
        Barber 1 : cutting hair
Customer 6 : Gets hair Cut
Customer 5 : Gets hair Cut
Customer 8 : pays
Customer 6 : pays
Customer 5 : pays
        Barber 1 : accpting payment
        Barber 0 : accpting payment
        Barber 2 : accpting payment
Customer 6 : exiting shop...
Customer 8 : exiting shop...
Customer 9 : sits on barber's chair
Customer 5 : exiting shop...
Customer 1 : sits on barber's chair
Customer 7 : sits on barber's chair
Customer 11 : sits on sofa
Customer 10 : sits on sofa
        Barber 0 : cutting hair
        Barber 2 : cutting hair
Customer 1 : Gets hair Cut
Customer 12 : sits on sofa
        Barber 1 : cutting hair
Customer 9 : Gets hair Cut
Customer 7 : Gets hair Cut
Customer 1 : pays
Customer 7 : pays
Customer 9 : pays
        Barber 0 : accpting payment
        Barber 2 : accpting payment
        Barber 1 : accpting payment
Customer 3 : sits on barber's chair
Customer 7 : exiting shop...
Customer 11 : sits on barber's chair
Customer 1 : exiting shop...
Customer 10 : sits on barber's chair
Customer 9 : exiting shop...
```

Logic:
Note that in each waiting area (the sofa and the standing room), customers have to be served in first-in-first-out (FIFO) order i.e. the one sitting on the sofa for the longest time (earliest arrival) shall get a seat on the barber's chair first (FIFO). Hence our implementation of semaphores must enforce FIFO queueing, then we can use nested multiplexes to create the waiting areas.

The semaphores used in this code : mutex, chair, barber, customer, cash, reciept. mutex protects customers, which keeps track of the number of customers in the shop so that if a thread arrives when the shop is full, it can exit. standingRoom and sofa are Fifos that represent the waiting areas. chair is a multiplex that limits the number of customers in the seating area.

The other semaphores are used for the rendezvous between the barber and the customers. The customer signals the barber and then waits on the customer. Then the customer signals cash and waits on the receipt.

Firstly, any customer that enters the shop has to follow the order of

Standing room → Sofa → Barber's chair → get haircut → pay → exit

We have maintained two queues. One queue keeps track of the order of customers going into the standing room and the other queue keeps track of the customers sitting on the sofa. The one waiting for the longest time in the standing room gets to sit on the sofa a soon as there is a vacant place on the sofa and the one waiting for the longest time on the sofa gets to sit on the barber's chair as soon as there is a vacant place.

In order to maintain FIFO order for the whole system, threads cascade from standingRoom to sofa to chair; as each thread moves to the next stage, it signals the previous stage.