Pratibha Biswas

16BCA0032

OPERATING SYSTEMS

Assessment 3

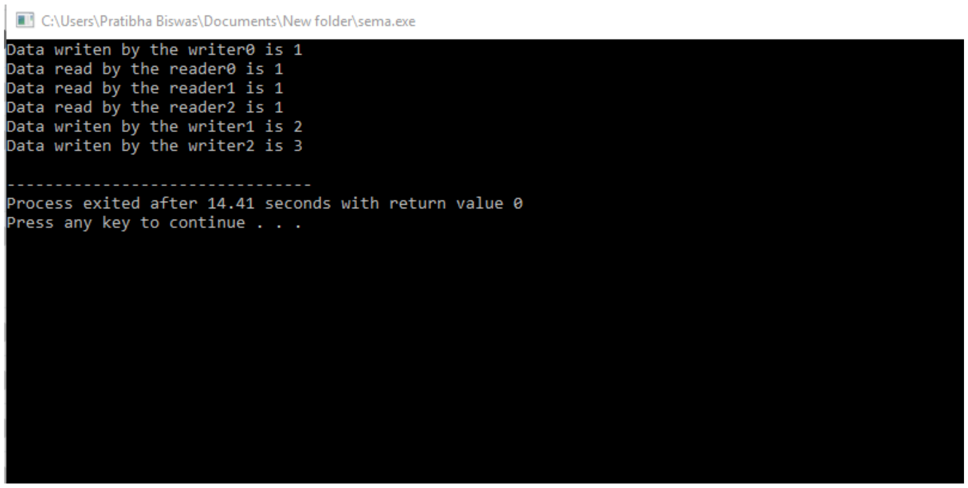
1. Consider a situation where we have a file shared between many people. If one of the people tries editing the file, no other person should be reading or writing at the same time, otherwise changes will not be visible to him/her. However if some person is reading the file, then others may read it at the same time. Implement the solution for the above problem using Semaphores.

1. CODE:

```
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>

sem_t mutex,writeblock;
int data = 0,rcount = 0;

void *reader(void *arg)
{
  int f;
  f = ((int)arg);
  sem_wait(&mutex);
  rcount = rcount + 1;
  if(rcount==1)
   sem_wait(&writeblock);
  sem_post(&mutex);
  printf("Data read by the reader%d is %d\n",f,data);
  sleep(1);
  sem_wait(&mutex);
  rcount = rcount - 1;
  if(rcount==0)
   sem_post(&writeblock);
  sem_post(&mutex);
}

void *writer(void *arg)
{
  int f;
  f = ((int) arg);
  sem_wait(&writeblock);
  data++;
  printf("Data writen by the writer%d is %d\n",f,data);
  sleep(1);
  sem_post(&writeblock);
}

int main()
{
  int i,b;
  pthread_t rtid[5],wtid[5];
  sem_init(&mutex,0,1);
  sem_init(&writeblock,0,1);
  for(i=0;i<=2;i++)
  {
```

```
  pthread_create(&wtid[i],NULL,writer,(void *)i);
  pthread_create(&rtid[i],NULL,reader,(void *)i);
}
for(i=0;i<=2;i++)
{
  pthread_join(wtid[i],NULL);
  pthread_join(rtid[i],NULL);
}
return 0;
            }
```

OUTPUT:



```
C:\Users\Pratibha Biswas\Documents\New folder\sema.exe
Data writen by the writer0 is 1
Data read by the reader0 is 1
Data read by the reader1 is 1
Data read by the reader2 is 1
Data writen by the writer1 is 2
Data writen by the writer2 is 3

--------------------------------
Process exited after 14.41 seconds with return value 0
Press any key to continue . . .
```

2. We have a buffer of fixed size. A producer can produce an item and can place in the buffer. A consumer can pick items and can consume them. We need to ensure that when a producer is placing an item in the buffer, then at the same time consumer should not consume any item. In this problem, buffer is the critical section. For implementing the solution of this problem, we need two counting semaphores – Full and Empty. “Full” keeps track of number of items in the buffer at any given time and “Empty” keeps track of number of unoccupied

Slots.

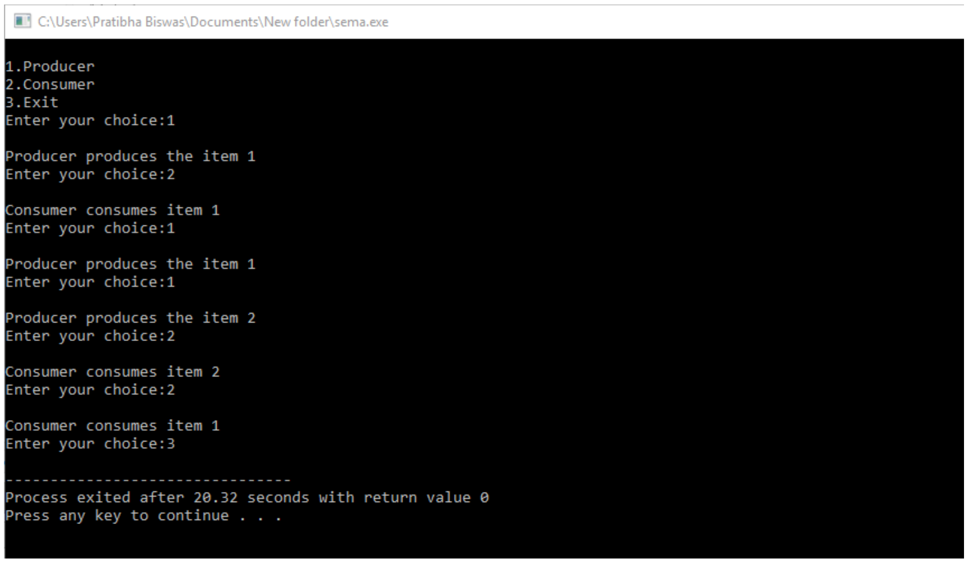
2. CODE:

```
#include<stdio.h>
#include<stdlib.h>

int mutex=1,full=0,empty=3,x=0;

int main()
{
    int n;
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    printf("\n1.Producer\n2.Consumer\n3.Exit");
    while(1)
    {
        printf("\nEnter your choice:");
        scanf("%d",&n);
        switch(n)
        {
            case 1:    if((mutex==1)&&(empty!=0))
                            producer();
                        else
                            printf("Buffer is full!!");
                        break;
            case 2:    if((mutex==1)&&(full!=0))
```

```
                    consumer();
                 else
                    printf("Buffer is empty!!");
                 break;
           case 3:
                 exit(0);
                 break;
        }
    }

    return 0;
}

int wait(int s)
{
    return (--s);
}

int signal(int s)
{
    return(++s);
}

void producer()
{
    mutex=wait(mutex);
    full=signal(full);
```

```
   empty=wait(empty);

   x++;

   printf("\nProducer produces the item %d",x);

   mutex=signal(mutex);

}


void consumer()

{

   mutex=wait(mutex);

   full=wait(full);

   empty=signal(empty);

   printf("\nConsumer consumes item %d",x);

   x--;

   mutex=signal(mutex);

         }
```

OUTPUT:



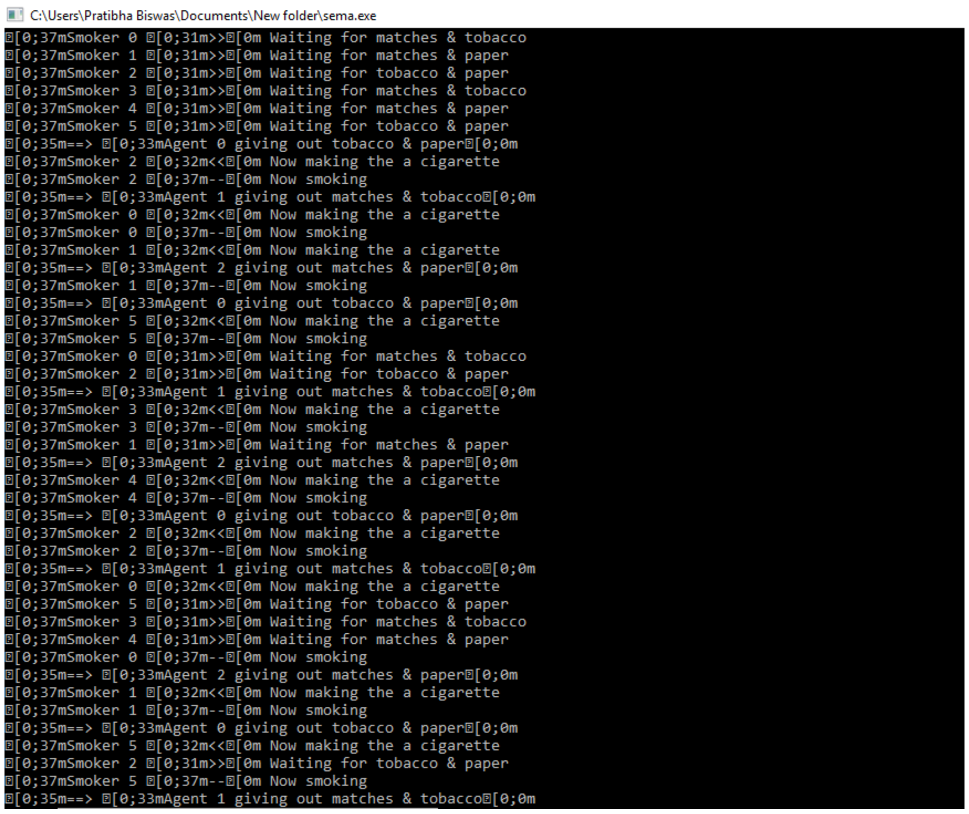
```
C:\Users\Pratibha Biswas\Documents\New folder\sema.exe

1.Producer
2.Consumer
3.Exit
Enter your choice:1

Producer produces the item 1
Enter your choice:2

Consumer consumes item 1
Enter your choice:1

Producer produces the item 1
Enter your choice:1

Producer produces the item 2
Enter your choice:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:3

--------------------------------
Process exited after 20.32 seconds with return value 0
Press any key to continue . . .
```

3. Consider a system with three smoker processes and one agent process. Each smoker continuously rolls a cigarette and then smokes it. But to roll and smoke a cigarette, the smoker needs three ingredients: tobacco, paper, and matches. One of the smoker processes has paper, another has tobacco, and the third has matches. The agent has an infinite supply of all three materials. The agent places two of the ingredients on the table. The smoker who has the remaining ingredient then makes and smokes a cigarette, signalling the agent on completion. The agent then puts out another two of the three ingredients, and the cycle repeats. Write an algorithm to synchronize the agent and the smokers using semaphore.

CODE:

```
#include <pthread.h>
#include <semaphore.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>

// An agent semaphore represents items on the table
sem_t agent_ready;

// Each smoker semaphore represents when a smoker has the items they need
sem_t smoker_semaphors[3];

// This is an array of strings describing what each smoker type needs
char* smoker_types[3] = { "matches & tobacco", "matches & paper", "tobacco & paper" };

// This list represents item types that are on the table. This should corrispond
// with the smoker_types, such that each item is the one the smoker has. So the
// first item would be paper, then tobacco, then matches.
bool items_on_table[3] = { false, false, false };

// Each pusher pushes a certian type item, manage these with this semaphore
sem_t pusher_semaphores[3];

/**
* Smoker function, handles waiting for the item's that they need, and then
* smoking. Repeat this three times
*/
void* smoker(void* arg)
{
    int smoker_id = *(int*) arg;
    int type_id   = smoker_id % 3;

    // Smoke 3 times
    for (int i = 0; i < 3; ++i)
    {
        printf("\033[0;37mSmoker %d \033[0;31m>>\033[0m Waiting for %s\n",
            smoker_id, smoker_types[type_id]);

        // Wait for the proper combination of items to be on the table
        sem_wait(&smoker_semaphors[type_id]);

        // Make the cigarette before releasing the agent
        printf("\033[0;37mSmoker %d \033[0;32m<<\033[0m Now making the a cigarette\n",
smoker_id);
        usleep(rand() % 50000);
        sem_post(&agent_ready);
        // We're smoking now
        printf("\033[0;37mSmoker %d \033[0;37m--\033[0m Now smoking\n", smoker_id);
        usleep(rand() % 50000);
    }

    return NULL;
}

// This semaphore gives the pusher exclusive access to the items on the table
sem_t pusher_lock;

/**
* The pusher is responsible for releasing the proper smoker semaphore when the
* right item's are on the table.
*/
void* pusher(void* arg)
{
    int pusher_id = *(int*) arg;

    for (int i = 0; i < 12; ++i)
    {
        // Wait for this pusher to be needed
        sem_wait(&pusher_semaphores[pusher_id]);
        sem_wait(&pusher_lock);

        // Check if the other item we need is on the table
        if (items_on_table[(pusher_id + 1) % 3])
        {
            items_on_table[(pusher_id + 1) % 3] = false;
            sem_post(&smoker_semaphors[(pusher_id + 2) % 3]);
        }
        else if (items_on_table[(pusher_id + 2) % 3])
        {
            items_on_table[(pusher_id + 2) % 3] = false;
            sem_post(&smoker_semaphors[(pusher_id + 1) % 3]);
        }
        else
        {
            // The other item's aren't on the table yet
            items_on_table[pusher_id] = true;
        }

        sem_post(&pusher_lock);
    }

    return NULL;
}

/**
* The agent puts items on the table
*/
void* agent(void* arg)
{
    int agent_id = *(int*) arg;
    for (int i = 0; i < 6; ++i)
    {
        usleep(rand() % 200000);

        // Wait for a lock on the agent
        sem_wait(&agent_ready);

        // Release the items this agent gives out
        sem_post(&pusher_semaphores[agent_id]);
        sem_post(&pusher_semaphores[(agent_id + 1) % 3]);

        // Say what type of items we just put on the table
        printf("\033[0;35m==> \033[0;33mAgent %d giving out %s\033[0;0m\n",
            agent_id, smoker_types[(agent_id + 2) % 3]);
    }

    return NULL;
}

/**
* The main thread handles the agent's arbitration of items.
*/
int main(int argc, char* arvg[])
{
    // Seed our random number since we will be using random numbers
    srand(time(NULL));

    // There is only one agent semaphore since only one set of items may be on
    // the table at any given time. A values of 1 = nothing on the table
    sem_init(&agent_ready, 0, 1);

    // Initalize the pusher lock semaphore
    sem_init(&pusher_lock, 0, 1);

    // Initialize the semaphores for the smokers and pusher
    for (int i = 0; i < 3; ++i)
    {
        sem_init(&smoker_semaphors[i], 0, 0);
        sem_init(&pusher_semaphores[i], 0, 0);
    }



    // Smoker ID's will be passed to the threads. Allocate the ID's on the stack
    int smoker_ids[6];

    pthread_t smoker_threads[6];

    // Create the 6 smoker threads with IDs
    for (int i = 0; i < 6; ++i)
    {
        smoker_ids[i] = i;

        if (pthread_create(&smoker_threads[i], NULL, smoker, &smoker_ids[i]) == EAGAIN)
        {
            perror("Insufficient resources to create thread");
            return 0;
        }
    }

    // Pusher ID's will be passed to the threads. Allocate the ID's on the stack
    int pusher_ids[6];

    pthread_t pusher_threads[6];

    for (int i = 0; i < 3; ++i)
    {
        pusher_ids[i] = i;

        if (pthread_create(&pusher_threads[i], NULL, pusher, &pusher_ids[i]) == EAGAIN)
        {
            perror("Insufficient resources to create thread");
            return 0;
        }
    }

    // Agent ID's will be passed to the threads. Allocate the ID's on the stack
    int agent_ids[6];

    pthread_t agent_threads[6];

    for (int i = 0; i < 3; ++i)
    {
        agent_ids[i] =i;
        if (pthread_create(&agent_threads[i], NULL, agent, &agent_ids[i]) == EAGAIN)
        {
            perror("Insufficient resources to create thread");
            return 0;
        }
    }

    // Make sure all the smokers are done smoking
    for (int i = 0; i < 6; ++i)
    {
        pthread_join(smoker_threads[i], NULL);
    }

    return 0;
                        }
```

OUTPUT:



```
C:\Users\Pratibha Biswas\Documents\New folder\sema.exe
▯[0;37mSmoker 0 ▯[0;31m>>▯[0m Waiting for matches & tobacco
▯[0;37mSmoker 1 ▯[0;31m>>▯[0m Waiting for matches & paper
▯[0;37mSmoker 2 ▯[0;31m>>▯[0m Waiting for tobacco & paper
▯[0;37mSmoker 3 ▯[0;31m>>▯[0m Waiting for matches & tobacco
▯[0;37mSmoker 4 ▯[0;31m>>▯[0m Waiting for matches & paper
▯[0;37mSmoker 5 ▯[0;31m>>▯[0m Waiting for tobacco & paper
▯[0;35m==> ▯[0;33mAgent 0 giving out tobacco & paper▯[0;0m
▯[0;37mSmoker 2 ▯[0;32m<<▯[0m Now making the a cigarette
▯[0;37mSmoker 2 ▯[0;37m--▯[0m Now smoking
▯[0;35m==> ▯[0;33mAgent 1 giving out matches & tobacco▯[0;0m
▯[0;37mSmoker 0 ▯[0;32m<<▯[0m Now making the a cigarette
▯[0;37mSmoker 0 ▯[0;37m--▯[0m Now smoking
▯[0;37mSmoker 1 ▯[0;32m<<▯[0m Now making the a cigarette
▯[0;35m==> ▯[0;33mAgent 2 giving out matches & paper▯[0;0m
▯[0;37mSmoker 1 ▯[0;37m--▯[0m Now smoking
▯[0;35m==> ▯[0;33mAgent 0 giving out tobacco & paper▯[0;0m
▯[0;37mSmoker 5 ▯[0;32m<<▯[0m Now making the a cigarette
▯[0;37mSmoker 5 ▯[0;37m--▯[0m Now smoking
▯[0;37mSmoker 0 ▯[0;31m>>▯[0m Waiting for matches & tobacco
▯[0;37mSmoker 2 ▯[0;31m>>▯[0m Waiting for tobacco & paper
▯[0;35m==> ▯[0;33mAgent 1 giving out matches & tobacco▯[0;0m
▯[0;37mSmoker 3 ▯[0;32m<<▯[0m Now making the a cigarette
▯[0;37mSmoker 3 ▯[0;37m--▯[0m Now smoking
▯[0;37mSmoker 1 ▯[0;31m>>▯[0m Waiting for matches & paper
▯[0;35m==> ▯[0;33mAgent 2 giving out matches & paper▯[0;0m
▯[0;37mSmoker 4 ▯[0;32m<<▯[0m Now making the a cigarette
▯[0;37mSmoker 4 ▯[0;37m--▯[0m Now smoking
▯[0;35m==> ▯[0;33mAgent 0 giving out tobacco & paper▯[0;0m
▯[0;37mSmoker 2 ▯[0;32m<<▯[0m Now making the a cigarette
▯[0;37mSmoker 2 ▯[0;37m--▯[0m Now smoking
▯[0;35m==> ▯[0;33mAgent 1 giving out matches & tobacco▯[0;0m
▯[0;37mSmoker 0 ▯[0;32m<<▯[0m Now making the a cigarette
▯[0;37mSmoker 5 ▯[0;31m>>▯[0m Waiting for tobacco & paper
▯[0;37mSmoker 3 ▯[0;31m>>▯[0m Waiting for matches & tobacco
▯[0;37mSmoker 4 ▯[0;31m>>▯[0m Waiting for matches & paper
▯[0;37mSmoker 0 ▯[0;37m--▯[0m Now smoking
▯[0;35m==> ▯[0;33mAgent 2 giving out matches & paper▯[0;0m
▯[0;37mSmoker 1 ▯[0;32m<<▯[0m Now making the a cigarette
▯[0;37mSmoker 1 ▯[0;37m--▯[0m Now smoking
▯[0;35m==> ▯[0;33mAgent 0 giving out tobacco & paper▯[0;0m
▯[0;37mSmoker 5 ▯[0;32m<<▯[0m Now making the a cigarette
▯[0;37mSmoker 2 ▯[0;31m>>▯[0m Waiting for tobacco & paper
▯[0;37mSmoker 5 ▯[0;37m--▯[0m Now smoking
▯[0;35m==> ▯[0;33mAgent 1 giving out matches & tobacco▯[0;0m
```

4. Sleeping barbers problem

CODE:

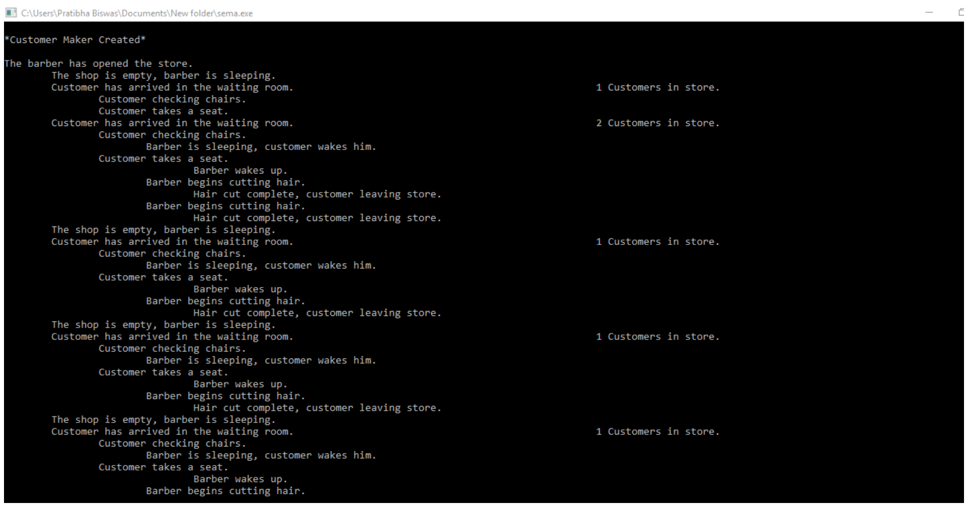
```
#include <stdio.h>


#include<pthread.h>


#include <stdlib.h>


#define seats 6 /*chairs for waiting customers*/


void *customer();
void *barberShop();
void *waiting_Room();
void checkQueue();
```

```
pthread_mutex_t queue = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t wait = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t sleepa = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t barberSleep = PTHREAD_COND_INITIALIZER;
pthread_cond_t barberWorking = PTHREAD_COND_INITIALIZER;

int returnTime=5,current=0,urakam =0, asdfgh;

int main(int argc, char *argv[])
{
        asdfgh=time(NULL);
        srand(asdfgh);
        //declare barber thread;
        pthread_t barber,customerM,timer_thread;
        pthread_attr_t barberAttr, timerAttr;
        pthread_attr_t customerMAttr;

        //define barber, and cutomerMaker default attributes
        pthread_attr_init(&timerAttr);
        pthread_attr_init(&barberAttr);
        pthread_attr_init(&customerMAttr);

        printf("\n");

        //create cutomerMaker
        pthread_create(&customerM,&customerMAttr,customer,NULL);
```

```
    //create barber
    pthread_create(&barber,&barberAttr,barberShop,NULL);

    pthread_join(barber,NULL);
    pthread_join(customerM,NULL);

    return 0;
}

void *customer()
{
    int i=0;
    printf("*Customer Maker Created*\n\n");
    fflush(stdout);
    pthread_t customer[seats+1];
    pthread_attr_t customerAttr[seats+1];
    while(i<(seats+1)) /*if there are no free chairs, leave*/
    {
        i++;/*increment count of waiting customer*/
        pthread_attr_init(&customerAttr[i]);
        while(rand()%2!=1)
        {
            sleep(1);/*go to sleep*/
        }
        pthread_create(&customer[i],&customerAttr[i],waiting_Room,NULL);
    }
    pthread_exit(0);/*shop is full,do not wait*/
}
```

```
void *waiting_Room()
{
        //take seat
        pthread_mutex_lock(&queue);
        checkQueue();

        sleep(returnTime);
        waiting_Room();
}

void *barberShop()
{
        int loop=0;
        printf("The barber has opened the store.\n");
        fflush(stdout);
        while(loop==0)
        {
                if(current==0)
                {
                printf("\tThe shop is empty, barber is sleeping.\n");
                fflush(stdout);
                pthread_mutex_lock(&sleepa);
                urakam=1;
                pthread_cond_wait(&barberSleep,&sleepa);
                urakam=0;
                pthread_mutex_unlock(&sleepa);
                printf("\t\t\t\tBarber wakes up.\n");
                fflush(stdout);
                }
                else
                {
                    printf("\t\t\tBarber begins cutting hair.\n");
                    fflush(stdout);
                    sleep((rand()%20)/5);
                    current--;
                    printf("\t\t\t\tHair cut complete, customer leaving store.\n");
                    pthread_cond_signal(&barberWorking);
                }
        }
        pthread_exit(0);
}

void checkQueue()
{
        current++;
        printf("\tCustomer has arrived in the waiting room.\t\t\t\t\t\t\t%d Customers in
store.\n",current);
        fflush(stdout);
        printf("\t\tCustomer checking chairs.\n");
        fflush(stdout);
        if(current<seats)
        {
                if(urakam==1)
                {
                    printf("\t\t\tBarber is sleeping, customer wakes him.\n");
                    fflush(stdout);
                    pthread_cond_signal(&barberSleep);
                }
                printf("\t\tCustomer takes a seat.\n");
                fflush(stdout);
                pthread_mutex_unlock(&queue);
                pthread_mutex_lock(&wait);
                pthread_cond_wait(&barberWorking,&wait);
                pthread_mutex_unlock(&wait);
                return;
        }
        if(current>=seats)
        {
                printf("\t\tAll chairs full, leaving store.\n");
                fflush(stdout);
                current--;
                pthread_mutex_unlock(&queue);
                return;
        }
}
```

OUTPUT:



C:\Users\Pratibha Biswas\Documents\New folder\sema.exe

```
*Customer Maker Created*

The barber has opened the store.
        The shop is empty, barber is sleeping.
        Customer has arrived in the waiting room.                                                   1 Customers in store.
                Customer checking chairs.
                Customer takes a seat.
        Customer has arrived in the waiting room.                                                   2 Customers in store.
                Customer checking chairs.
                        Barber is sleeping, customer wakes him.
                Customer takes a seat.
                                Barber wakes up.
                        Barber begins cutting hair.
                                Hair cut complete, customer leaving store.
                        Barber begins cutting hair.
                                Hair cut complete, customer leaving store.
        The shop is empty, barber is sleeping.
        Customer has arrived in the waiting room.                                                   1 Customers in store.
                Customer checking chairs.
                        Barber is sleeping, customer wakes him.
                Customer takes a seat.
                                Barber wakes up.
                        Barber begins cutting hair.
                                Hair cut complete, customer leaving store.
        The shop is empty, barber is sleeping.
        Customer has arrived in the waiting room.                                                   1 Customers in store.
                Customer checking chairs.
                        Barber is sleeping, customer wakes him.
                Customer takes a seat.
                                Barber wakes up.
                        Barber begins cutting hair.
                                Hair cut complete, customer leaving store.
        The shop is empty, barber is sleeping.
        Customer has arrived in the waiting room.                                                   1 Customers in store.
                Customer checking chairs.
                        Barber is sleeping, customer wakes him.
                Customer takes a seat.
                                Barber wakes up.
                        Barber begins cutting hair.
```