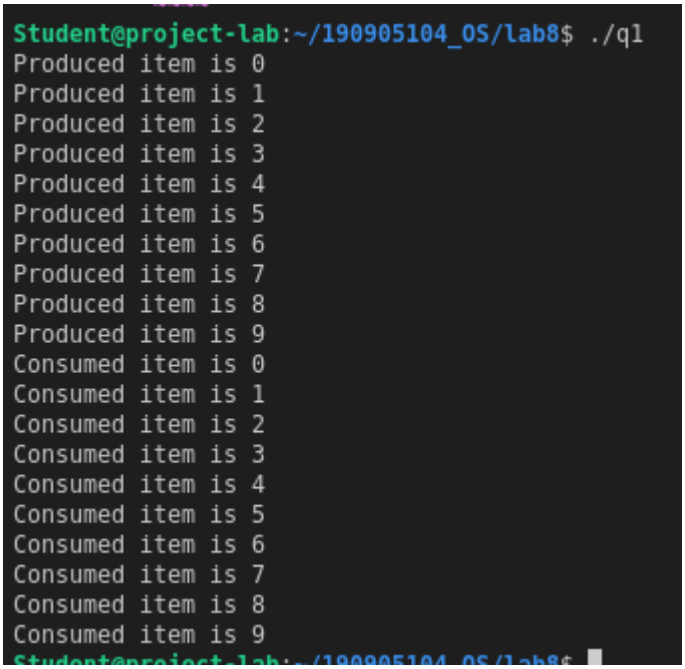
190905104
Parth Shukla
Lab 8

Deadlock, Locking, Synchronising

1) Modify the above Producer-Consumer program so that, a producer can produce at the most 10 items more than what the consumer has consumed.

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

int buf[10], f, r;
sem_t mutex, full, empty;

void* produce(void* arg){
   for(int i=0; i < 10; i++){
      sem_wait(&empty);
      sem_wait(&mutex);

      printf("Produced item is %d\n", i);

      buf[(++r) % 10] = i;
      sleep(1);

      sem_post(&mutex);
      sem_post(&full);
      // printf("full %u\n", full);
   }
}



void* consume(void* arg){
   int item;

   for(int i = 0; i<10; i++){
      sem_wait(&full);
      // printf("full %u\n", full);

      sem_wait(&mutex);
      item = buf[(++f) % 10];

      printf("Consumed item is %d\n", item);
      sleep(1);

      sem_post(&mutex);
      sem_post(&empty);
   }
}

int main(){
   pthread_t t1, t2;
   sem_init(&mutex, 0, 1);
   sem_init(&full, 0, 1);
   sem_init(&empty, 0, 10);
```

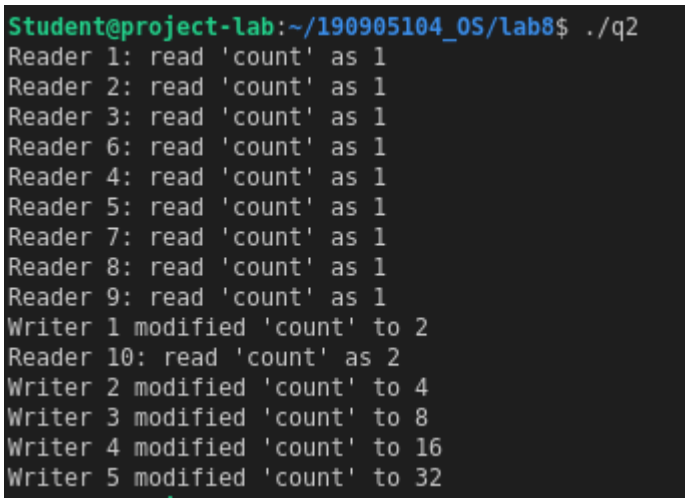
```
    pthread_create(&t1, NULL, produce, NULL);
    pthread_create(&t2, NULL, consume, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
}
```



2) Write a C program for the first readers-writers problem using semaphores.

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

sem_t wrt;
pthread_mutex_t mutex;
int count = 1;
int numreader = 0;

void *writer(void *wno)
{
    sem_wait(&wrt);
    count *= 2;
    printf("Writer %d modified 'count' to %d\n", (*((int *)wno)), count);
    sem_post(&wrt);
}

void *reader(void *rno)
{
    pthread_mutex_lock(&mutex);
    numreader++;

    if(numreader == 1)
        sem_wait(&wrt); // first reader will block the writer
    pthread_mutex_unlock(&mutex);

    // Reading Section, no locks

    printf("Reader %d: read 'count' as %d\n",*((int *)rno),count);
```

```
    // Reader acquire the lock before modifying numreader
    pthread_mutex_lock(&mutex);
    numreader--;

    if(numreader == 0)
        sem_post(&wrt); // If this is the last reader, it will wake up the writer.

    pthread_mutex_unlock(&mutex);
}

int main()
{
    pthread_t read[10],write[5];
    pthread_mutex_init(&mutex, NULL);

    sem_init(&wrt,0,1);

    int a[10] = {1,2,3,4,5,6,7,8,9,10}; //used for numbering the producer and consumer

    for(int i = 0; i < 10; i++)
        pthread_create(&read[i], NULL, reader, &a[i]);

    for(int i = 0; i < 5; i++)
        pthread_create(&write[i], NULL, writer, &a[i]);

    for(int i = 0; i < 10; i++)
        pthread_join(read[i], NULL);
    for(int i = 0; i < 5; i++)
        pthread_join(write[i], NULL);

    pthread_mutex_destroy(&mutex);
    sem_destroy(&wrt);
    return 0;
}
```



3) Write a Code to access a shared resource which causes deadlock using improper use of semaphore.

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

```
#include <semaphore.h>

sem_t s1,s2;

void *func1(void *p)
{
   // trying to decrement s1
   sem_wait(&s1);
   // trying to decrement s2
   sem_wait(&s2);
   printf("Thread 1\n");
   // increment s1
   sem_post(&s1);
}

void *func2(void *p)
{
   // trying to decrement s2, however would not be able to since it is already 0 and is not
incremented in func1()
   sem_wait(&s2);
   // trying to decrement s1
   sem_wait(&s1);
   printf("Thread 2\n");
   // increment s2
   sem_post(&s2);
}

int main()
{
   pthread_t threads[2];
   sem_init(&s1,0,1);
   sem_init(&s2,0,1);
   pthread_create(&threads[0],0,func1,0);
   pthread_create(&threads[1],0,func2,0);
   pthread_join(threads[0],0);
   pthread_join(threads[1],0);sem_destroy(&s1);
   sem_destroy(&s2);
}
```

4) Write a program using semaphore to demonstrate the working of sleeping barber problem.

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

sem_t customer,barber;
pthread_mutex_t seat;
int free1 = 10;

void *barber_(void *args)
```

```
{
   while(1)
   {
      // Sleeping
      sem_wait(&customer);
      // customer is here, so locking the seat
      pthread_mutex_lock(&seat);

      if(free1<10)
         free1++;
      sleep(2);

      printf("Cutting completed: free seats: %d\n",free1);
      // Barber is cutting hair
      sem_post(&barber);
      // release the chair
      pthread_mutex_unlock(&seat);
   }
}

void *customer_(void *args)
{
   while(1)
   {
      // Locking seat for customer
      pthread_mutex_lock(&seat);

      if(free1 > 0)
      {
         free1--;

         printf("Customer waiting: free seats: %d\n",free1);
         // Telling barber a seat is free
         sem_post(&customer);
         // Unlock seat
         pthread_mutex_unlock(&seat);
         // Waiting if barber is busy
         sem_wait(&barber);
      }
      else // no seat empty, customer leaves
         pthread_mutex_unlock(&seat);
   }
}

int main()
{
   pthread_t threads[2];
   sem_init(&barber,0,1);
   sem_init(&customer,0,1);
   pthread_mutex_init(&seat,0);
   pthread_create(&threads[0],NULL,barber_,NULL);
   pthread_create(&threads[1],NULL,customer_,NULL);
   pthread_join(threads[0],NULL);
   pthread_join(threads[1],NULL);
   sem_destroy(&barber);
   sem_destroy(&customer);
   pthread_mutex_destroy(&seat);
}
```

```
Student@project-lab:~/190905104_OS/lab8$ gcc q4.c -o q4 -lpthread
Student@project-lab:~/190905104_OS/lab8$ ./q4
Cutting completed: free seats: 10
Customer waiting: free seats: 9
Customer waiting: free seats: 8
Customer waiting: free seats: 7
Cutting completed: free seats: 8
Cutting completed: free seats: 9
Cutting completed: free seats: 10
Customer waiting: free seats: 9
Customer waiting: free seats: 8
Customer waiting: free seats: 7
Cutting completed: free seats: 8
Cutting completed: free seats: 9
Cutting completed: free seats: 10
Customer waiting: free seats: 9
Customer waiting: free seats: 8
Customer waiting: free seats: 7
Cutting completed: free seats: 8
```