**Lab 7: IPC-3: Deadlock, Locking, Synchronization**

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 <pthread.h>

#include <semaphore.h>

int buf[5], 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);

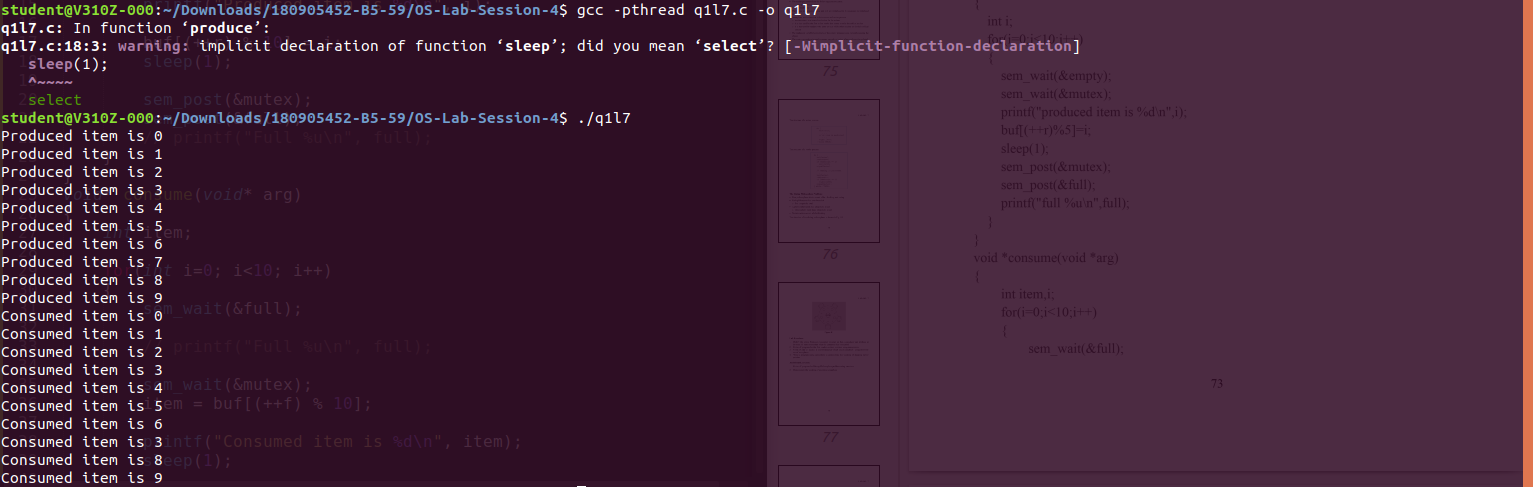
pthread\_create(&t2, NULL, consume, NULL);

pthread\_join(t1, NULL);

pthread\_join(t2, NULL);

}

Output



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 cnt = 1;

int numreader = 0;

void \*writer(void \*wno)

{

sem\_wait(&wrt);cnt \*= 2;

printf("Writer %d modified 'cnt' to %d\n", (\*((int \*)wno)), cnt);

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 'cnt' as %d\n",\*((int \*)rno),cnt);

// 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);

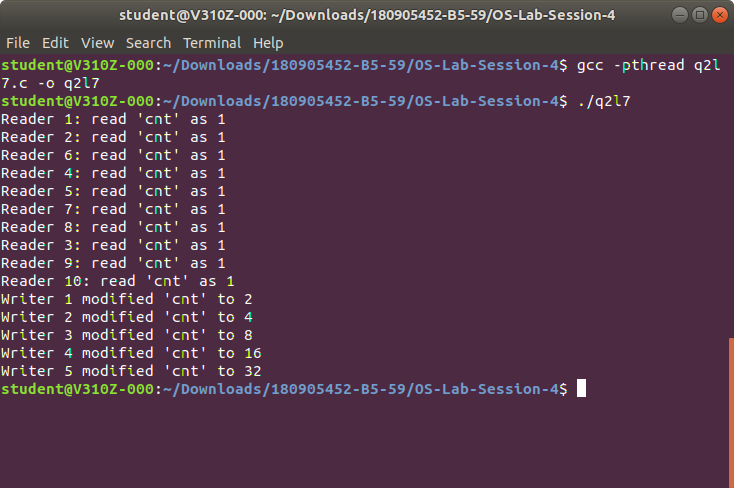
pthread\_mutex\_destroy(&mutex);

sem\_destroy(&wrt);

return 0;

}

Output



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)

{

sem\_wait(&s1);

sem\_wait(&s2);

printf("Thread 1\n");

sem\_post(&s1);

}

void \*func2(void \*p)

{

sem\_wait(&s2);

sem\_wait(&s1);

printf("Thread 2\n");

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);

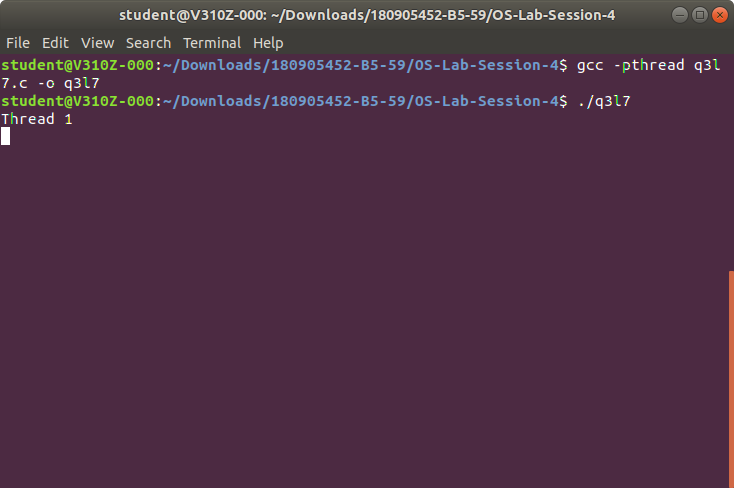
pthread\_join(threads[0],0);

pthread\_join(threads[1],0);sem\_destroy(&s1);

sem\_destroy(&s2);

}

Output



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 \*br(void \*args)

{

while(1)

{

sem\_wait(&customer);

pthread\_mutex\_lock(&seat);

if(free1<10)

free1++;

sleep(2);

printf("Cutting completed : free seats : %d\n",free1);

sem\_post(&barber);

pthread\_mutex\_unlock(&seat);

}

}

void \*cr(void \*args)

{

while(1)

{

pthread\_mutex\_lock(&seat);

if(free1 > 0)

{

free1--;

printf("Customer waiting : free seats : %d\n",free1);

sem\_post(&customer);

pthread\_mutex\_unlock(&seat);

sem\_wait(&barber);

}

else

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,br,NULL);

pthread\_create(&threads[1],NULL,cr,NULL);

pthread\_join(threads[0],NULL);

pthread\_join(threads[1],NULL);

sem\_destroy(&barber);

sem\_destroy(&customer);

pthread\_mutex\_destroy(&seat);

}

Output

