**NAME: DHANRAJ SUBHASH KORE**

**DIV: TY B** **BATCH: B3**

**ROLL NO: 60**

**SUB: SS LAB ASSIGNMENT**

**SYNCHRONIZATION PROBLEMS**

**Producer Consumer problem using semaphore**

#include<stdio.h>

#include<string.h>

// pthread support library

#include<pthread.h>

// semaphore support library

#include<semaphore.h>

// buffer size count

#define N 5

// buffer

int buffer[N];

int in = 0, out = 0;

// semaphores for producer consumer problem

sem\_t mutex, full, empty;

// initialize semaphore varilables

void initSemaphores() {

// sem\_init(semaphore variable reference, 0 to share semaphore between all threads, inital value)

// mutex (mutual exclusion) variable

sem\_init(&mutex, 0, 1);

// full check semaphore

sem\_init(&full, 0, 1);

// empty check semaphore

sem\_init(&empty, 0, N);

}

// Procuder program with semaphores

void \*producer() {

// run finite length

int i;

for (i = 0; i < 10; i++) {

// check for full buffer

// if buffer is full then put this process in sleep queue

// else add a item by using mutex semaphore

sem\_wait(&empty);

// check whether any other process is accessing buffer

// if yes put this process to sleep

// else add the element to the buffer

sem\_wait(&mutex);

// produced item

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

// add element to the buffer

buffer[in] = i;

in = (in + 1) % N;

sleep(1);

// free the mutex semaphore and and wake up a process in the sleep queue

sem\_post(&mutex);

// increment the full semaphore indicating the addition of new element to the buffer is complete

sem\_post(&full);

}

}

// Consumer program with semaphores

void \*consumer() {

int item;

// run finite length

int i;

for (i = 0; i < 10; i++) {

// check for empty buffer

// if buffer is empty then put this process to sleep

// else remove a item by using mutex semaphore

sem\_wait(&full);

// check whether any other process is accessing buffer

// if yes put this process to sleep

// else add the element to the buffer

sem\_wait(&mutex);

// remove element from the buffer

item = buffer[out];

// consumed item

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

out = (out + 1) % N;

sleep(1);

// free the mutex semaphore and and wake up a process in the sleep queue

sem\_post(&mutex);

// increment the empty semaphore indicating the removal of new element to the buffer is complete

sem\_post(&empty);

}

}

int main() {

initSemaphores();

// declare pthread (POSIX threads) ids

// A thread is a single sequence stream within in a process

pthread\_t produce, consume;

// create threads

pthread\_create(&produce ,NULL, producer, NULL);

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

// join thread

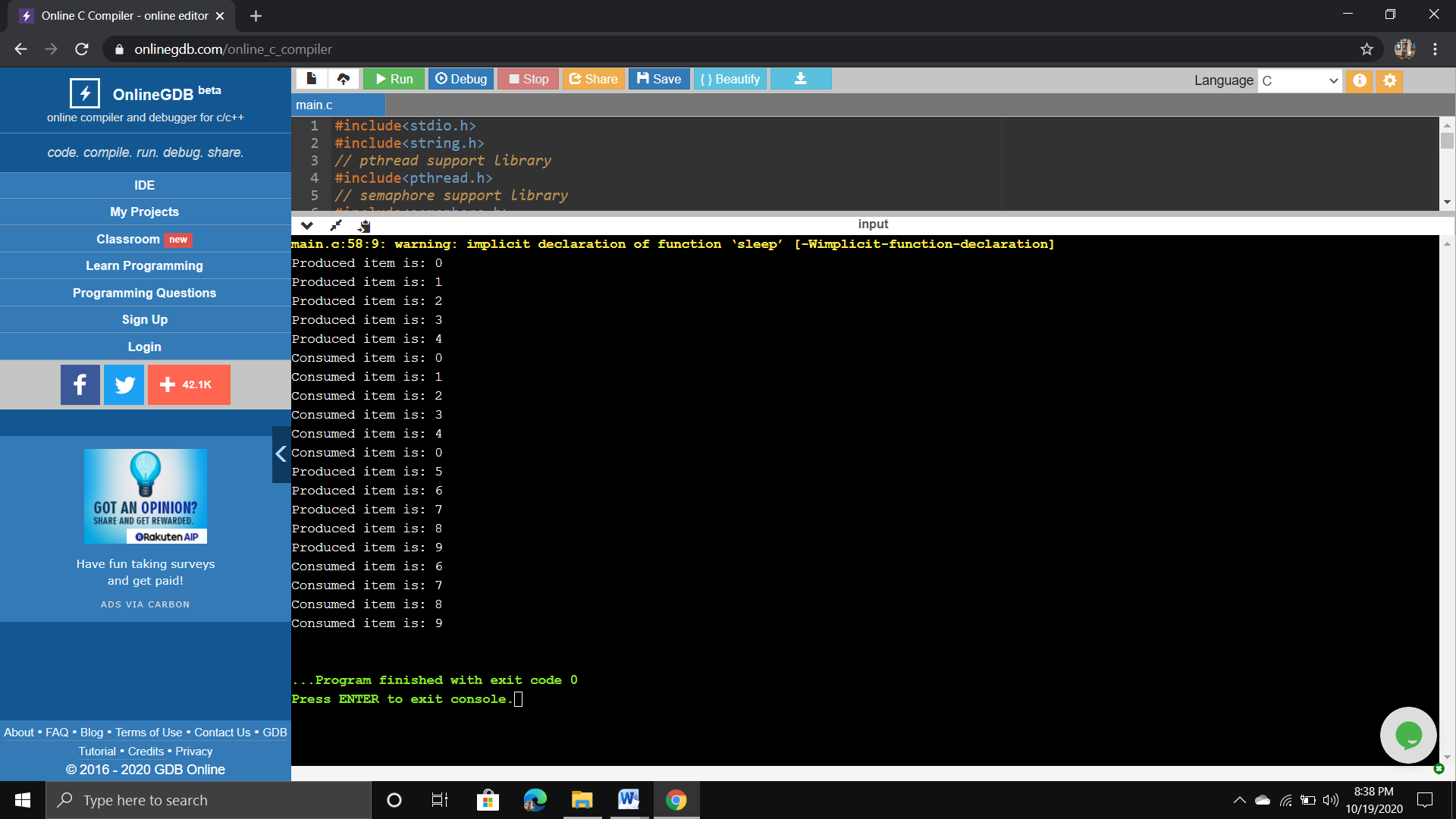
pthread\_join(produce, NULL);

pthread\_join(consume, NULL);

return 0;

}

**Output**



**Reader writer problem using mutex**

#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 = cnt\*2;

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

sem\_post(&wrt);

}

void \*reader(void \*rno)

{

// Reader acquire the lock before modifying numreader

pthread\_mutex\_lock(&mutex);

numreader++;

if(numreader == 1) {

sem\_wait(&wrt); // If this id the first reader, then it will block the writer

}

pthread\_mutex\_unlock(&mutex);

// Reading Section

printf("Reader %d: read count 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};

for(int i = 0; i < 10; i++) {

pthread\_create(&read[i], NULL, (void \*)reader, (void \*)&a[i]);

}

for(int i = 0; i < 5; i++) {

pthread\_create(&write[i], NULL, (void \*)writer, (void \*)&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**

