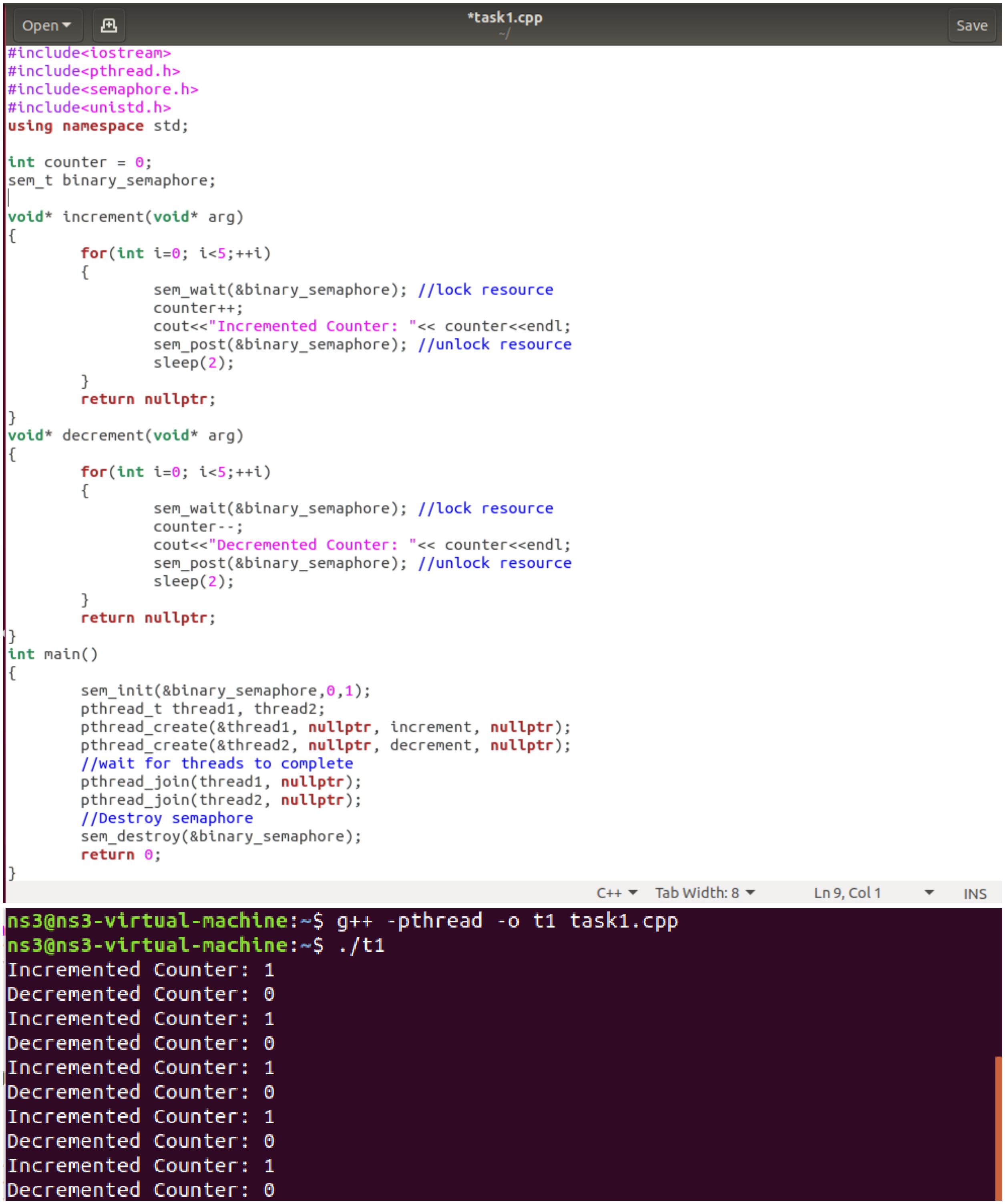
Operating Systems Lab

Mariam Fatima
22F-3168
Lab 10

Task 1:

```
#include<iostream>
#include<pthread.h>
#include<semaphore.h>
#include<unistd.h>

using namespace std;

int counter = 0;
sem_t binary_semaphore;


void* increment(void* arg)
{
	for(int i=0; i<5;++i)
	{
		sem_wait(&binary_semaphore); //lock resource
		counter++;
		cout<<"Incremented Counter: "<<	counter<<endl;
		sem_post(&binary_semaphore); //unlock resource
		sleep(2);
	}
	return nullptr;
}
void* decrement(void* arg)
{
	for(int i=0; i<5;++i)
	{
		sem_wait(&binary_semaphore); //lock resource
		counter--;
		cout<<"Decremented Counter: "<<	counter<<endl;
```

```
        sem_post(&binary_semaphore); //unlock resource
        sleep(2);
    }
    return nullptr;
}

int main()
{
    sem_init(&binary_semaphore,0,1);
    pthread_t thread1, thread2;
    pthread_create(&thread1, nullptr, increment, nullptr);
    pthread_create(&thread2, nullptr, decrement, nullptr);
    //wait for threads to complete
    pthread_join(thread1, nullptr);
    pthread_join(thread2, nullptr);

    //Destroy semaphore
    sem_destroy(&binary_semaphore);

    return 0;
}
```



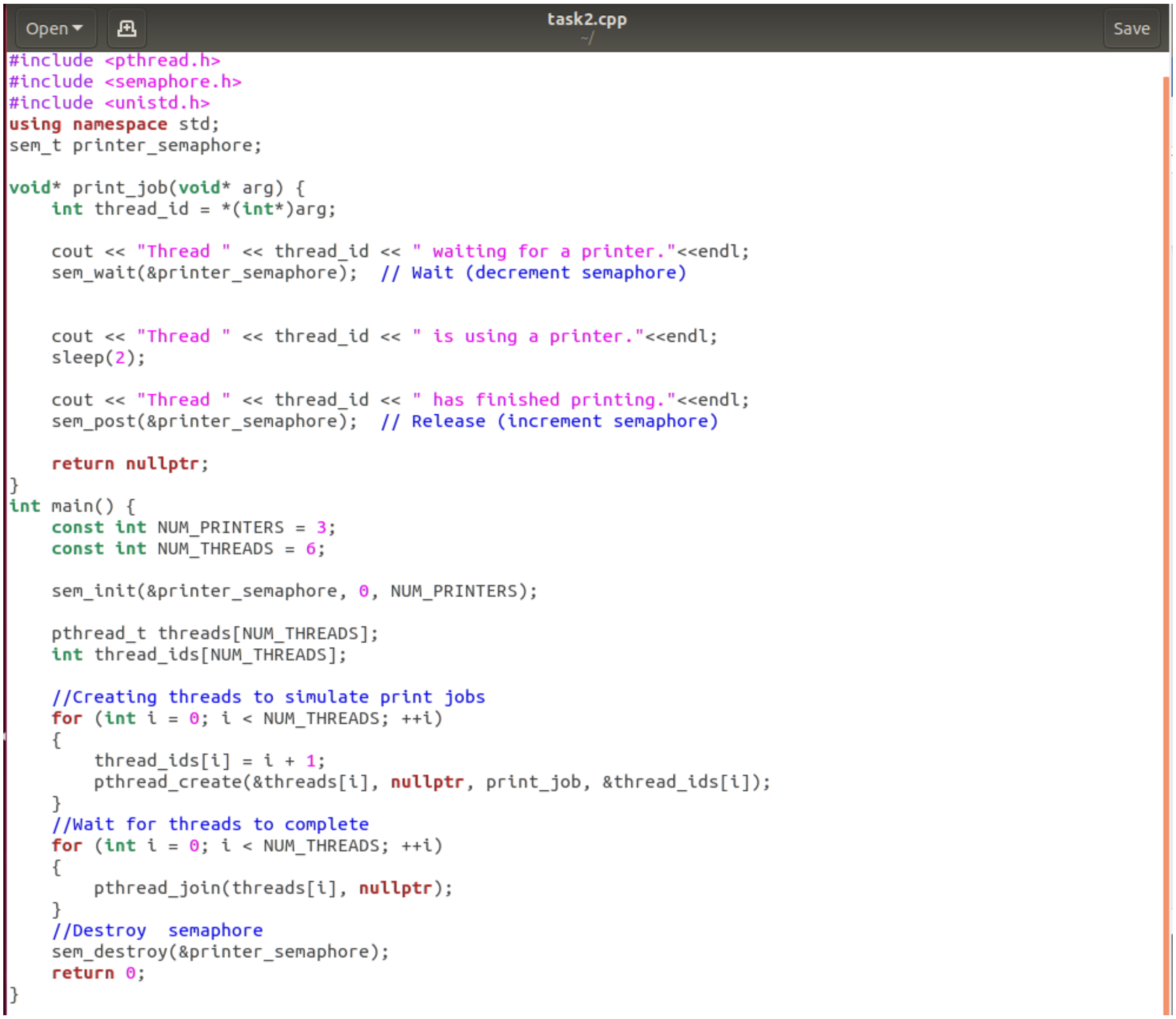
```cpp
#include<iostream>
#include<pthread.h>
#include<semaphore.h>
#include<unistd.h>
using namespace std;

int counter = 0;
sem_t binary_semaphore;

void* increment(void* arg)
{
        for(int i=0; i<5;++i)
        {
                sem_wait(&binary_semaphore); //lock resource
                counter++;
                cout<<"Incremented Counter: "<< counter<<endl;
                sem_post(&binary_semaphore); //unlock resource
                sleep(2);
        }
        return nullptr;
}
void* decrement(void* arg)
{
        for(int i=0; i<5;++i)
        {
                sem_wait(&binary_semaphore); //lock resource
                counter--;
                cout<<"Decremented Counter: "<< counter<<endl;
                sem_post(&binary_semaphore); //unlock resource
                sleep(2);
        }
        return nullptr;
}
int main()
{
        sem_init(&binary_semaphore,0,1);
        pthread_t thread1, thread2;
        pthread_create(&thread1, nullptr, increment, nullptr);
        pthread_create(&thread2, nullptr, decrement, nullptr);
        //wait for threads to complete
        pthread_join(thread1, nullptr);
        pthread_join(thread2, nullptr);
        //Destroy semaphore
        sem_destroy(&binary_semaphore);
        return 0;
}
```

```
ns3@ns3-virtual-machine:~$ g++ -pthread -o t1 task1.cpp
ns3@ns3-virtual-machine:~$ ./t1
Incremented Counter: 1
Decremented Counter: 0
Incremented Counter: 1
Decremented Counter: 0
Incremented Counter: 1
Decremented Counter: 0
Incremented Counter: 1
Decremented Counter: 0
Incremented Counter: 1
Decremented Counter: 0
```

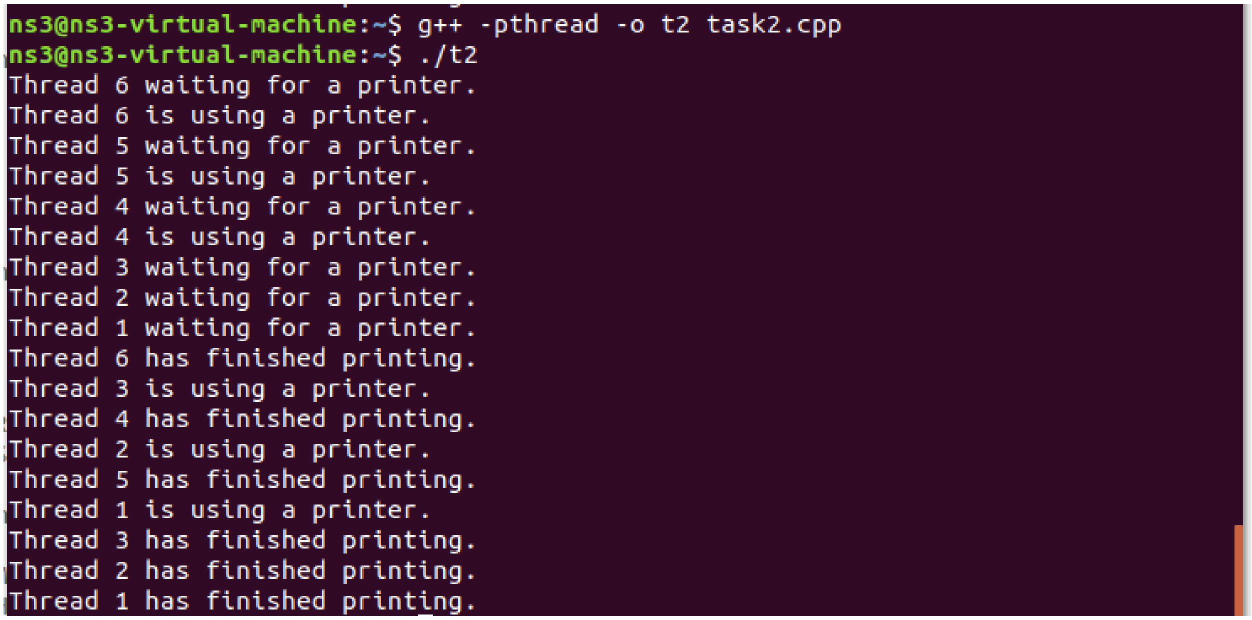
Task 2:

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

```
#include <semaphore.h>
#include <unistd.h>
using namespace std;
sem_t printer_semaphore;

void* print_job(void* arg) {
    int thread_id = *(int*)arg;

    cout << "Thread " << thread_id << " waiting for a printer."<<endl;
    sem_wait(&printer_semaphore);  // Wait (decrement semaphore)


    cout << "Thread " << thread_id << " is using a printer."<<endl;
    sleep(2);

    cout << "Thread " << thread_id << " has finished printing."<<endl;
    sem_post(&printer_semaphore);  // Release (increment semaphore)

    return nullptr;
}
int main() {
    const int NUM_PRINTERS = 3;
    const int NUM_THREADS = 6;

    sem_init(&printer_semaphore, 0, NUM_PRINTERS);

    pthread_t threads[NUM_THREADS];
    int thread_ids[NUM_THREADS];

    //Creating threads to simulate print jobs
    for (int i = 0; i < NUM_THREADS; ++i)
    {
        thread_ids[i] = i + 1;
        pthread_create(&threads[i], nullptr, print_job, &thread_ids[i]);
    }
    //Wait for threads to complete
    for (int i = 0; i < NUM_THREADS; ++i)
    {
        pthread_join(threads[i], nullptr);
    }
    //Destroy  semaphore
    sem_destroy(&printer_semaphore);
    return 0;
}
```



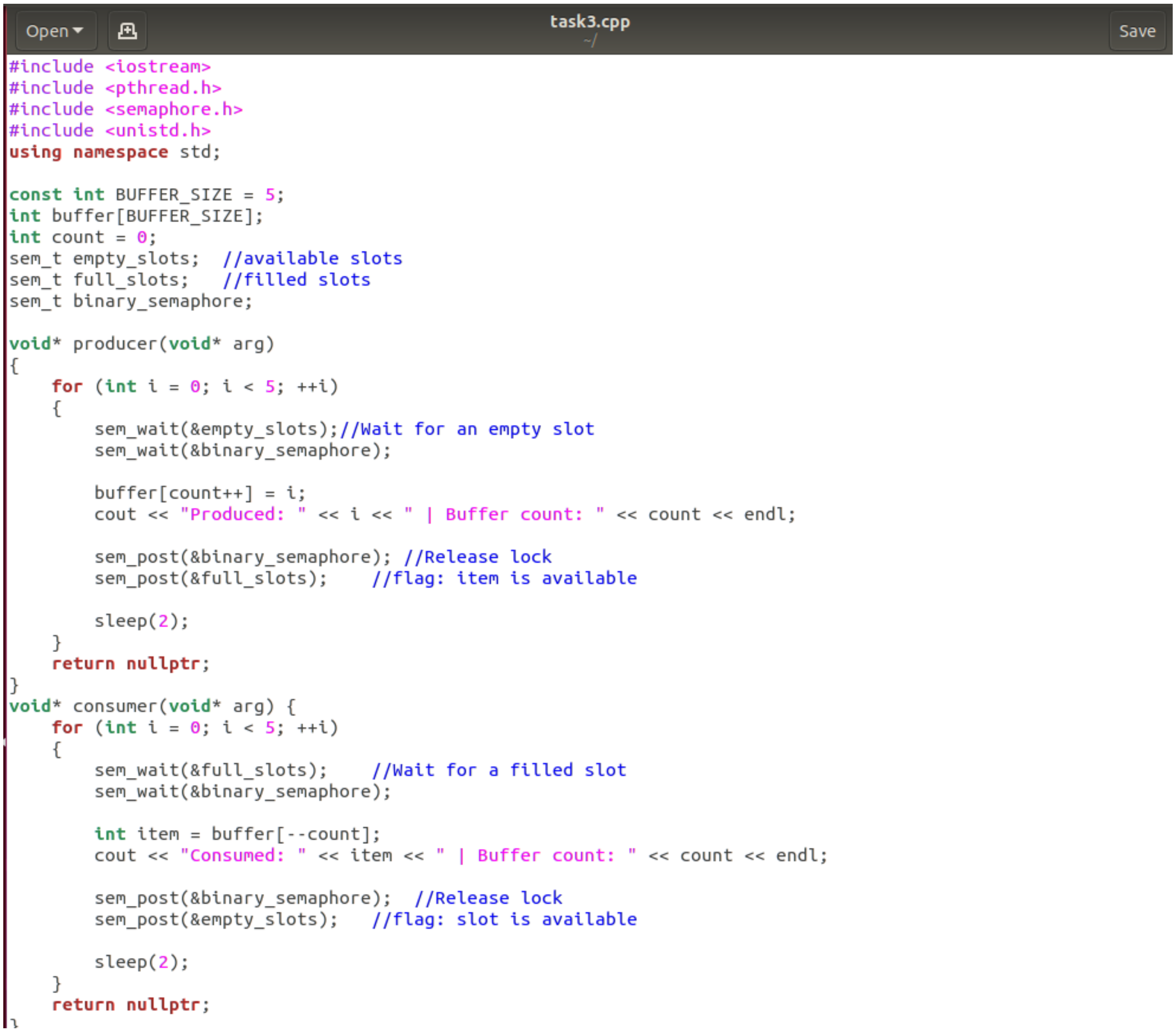
```
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
using namespace std;
sem_t printer_semaphore;

void* print_job(void* arg) {
    int thread_id = *(int*)arg;

    cout << "Thread " << thread_id << " waiting for a printer."<<endl;
    sem_wait(&printer_semaphore);  // Wait (decrement semaphore)


    cout << "Thread " << thread_id << " is using a printer."<<endl;
    sleep(2);

    cout << "Thread " << thread_id << " has finished printing."<<endl;
    sem_post(&printer_semaphore);  // Release (increment semaphore)

    return nullptr;
}
int main() {
    const int NUM_PRINTERS = 3;
    const int NUM_THREADS = 6;

    sem_init(&printer_semaphore, 0, NUM_PRINTERS);

    pthread_t threads[NUM_THREADS];
    int thread_ids[NUM_THREADS];

    //Creating threads to simulate print jobs
    for (int i = 0; i < NUM_THREADS; ++i)
    {
        thread_ids[i] = i + 1;
        pthread_create(&threads[i], nullptr, print_job, &thread_ids[i]);
    }
    //Wait for threads to complete
    for (int i = 0; i < NUM_THREADS; ++i)
    {
        pthread_join(threads[i], nullptr);
    }
    //Destroy  semaphore
    sem_destroy(&printer_semaphore);
    return 0;
}
```



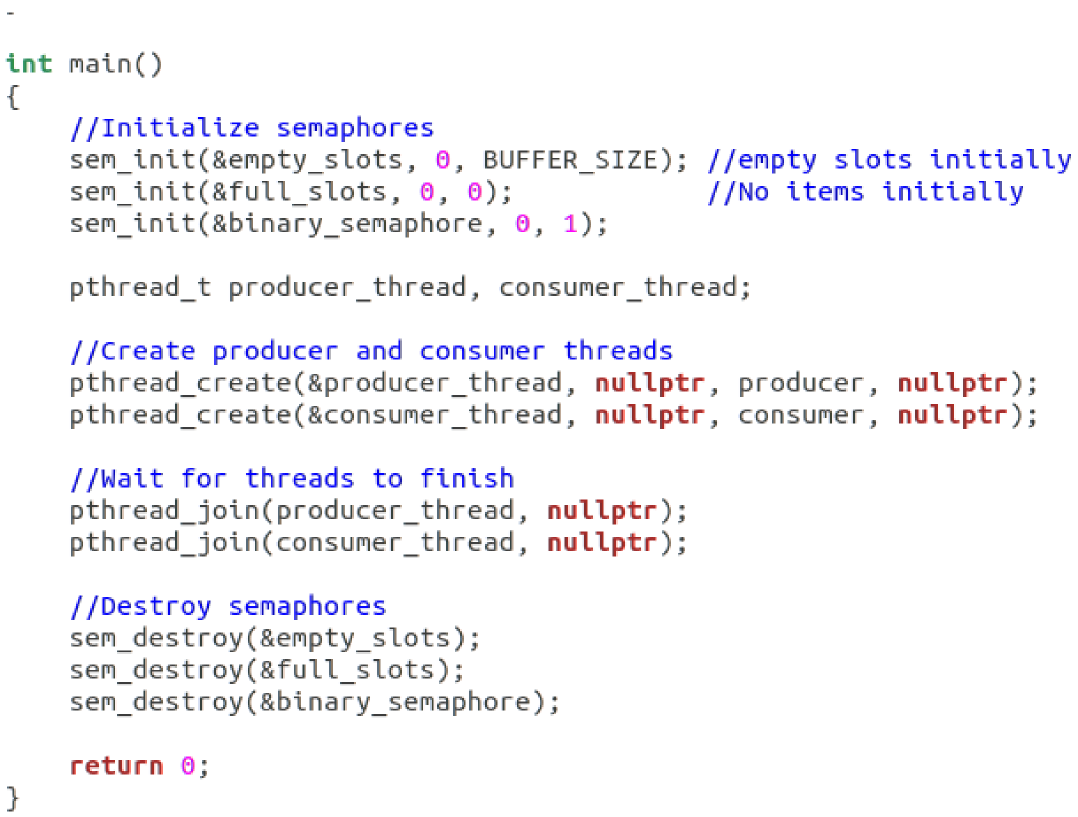
```
ns3@ns3-virtual-machine:~$ g++ -pthread -o t2 task2.cpp
ns3@ns3-virtual-machine:~$ ./t2
Thread 6 waiting for a printer.
Thread 6 is using a printer.
Thread 5 waiting for a printer.
Thread 5 is using a printer.
Thread 4 waiting for a printer.
Thread 4 is using a printer.
Thread 3 waiting for a printer.
Thread 2 waiting for a printer.
Thread 1 waiting for a printer.
Thread 6 has finished printing.
Thread 3 is using a printer.
Thread 4 has finished printing.
Thread 2 is using a printer.
Thread 5 has finished printing.
Thread 1 is using a printer.
Thread 3 has finished printing.
Thread 2 has finished printing.
Thread 1 has finished printing.
```

Task 3:

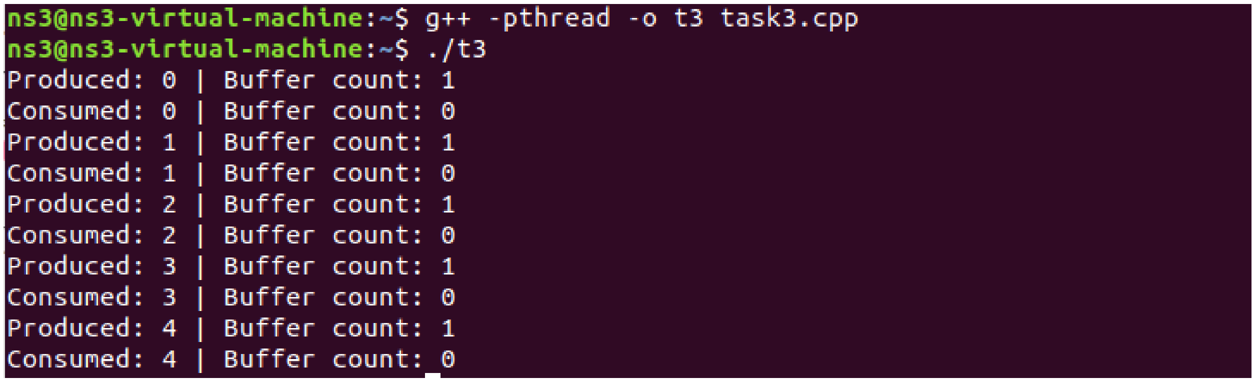
```
#include <iostream>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
using namespace std;

const int BUFFER_SIZE = 5;
int buffer[BUFFER_SIZE];
int count = 0;
sem_t empty_slots;  //available slots
sem_t full_slots;   //filled slots
sem_t binary_semaphore;

void* producer(void* arg)
{
    for (int i = 0; i < 5; ++i)
    {
        sem_wait(&empty_slots);//Wait for an empty slot
        sem_wait(&binary_semaphore);

        buffer[count++] = i;
        cout << "Produced: " << i << " | Buffer count: " << count << endl;
        sem_post(&binary_semaphore); //Release lock
        sem_post(&full_slots);    //flag: item is available

        sleep(2);
    }
    return nullptr;
}
void* consumer(void* arg) {
    for (int i = 0; i < 5; ++i)
    {
        sem_wait(&full_slots);    //Wait for a filled slot
        sem_wait(&binary_semaphore);

        int item = buffer[--count];
        cout << "Consumed: " << item << " | Buffer count: " << count << endl;

        sem_post(&binary_semaphore);  //Release lock
        sem_post(&empty_slots);   //flag: slot is available

        sleep(2);
    }
    return nullptr;
}

int main()
{
    //Initialize semaphores
    sem_init(&empty_slots, 0, BUFFER_SIZE); //empty slots initially
    sem_init(&full_slots, 0, 0);            //No items initially
    sem_init(&binary_semaphore, 0, 1);

    pthread_t producer_thread, consumer_thread;

    //Create producer and consumer threads
    pthread_create(&producer_thread, nullptr, producer, nullptr);
    pthread_create(&consumer_thread, nullptr, consumer, nullptr);

    //Wait for threads to finish
    pthread_join(producer_thread, nullptr);
    pthread_join(consumer_thread, nullptr);

    //Destroy semaphores
    sem_destroy(&empty_slots);
    sem_destroy(&full_slots);
    sem_destroy(&binary_semaphore);

    return 0;
}
```



```
#include <iostream>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
using namespace std;

const int BUFFER_SIZE = 5;
int buffer[BUFFER_SIZE];
int count = 0;
sem_t empty_slots;  //available slots
sem_t full_slots;   //filled slots
sem_t binary_semaphore;

void* producer(void* arg)
{
    for (int i = 0; i < 5; ++i)
    {
        sem_wait(&empty_slots);//Wait for an empty slot
        sem_wait(&binary_semaphore);

        buffer[count++] = i;
        cout << "Produced: " << i << " | Buffer count: " << count << endl;

        sem_post(&binary_semaphore); //Release lock
        sem_post(&full_slots);    //flag: item is available

        sleep(2);
    }
    return nullptr;
}
void* consumer(void* arg) {
    for (int i = 0; i < 5; ++i)
    {
        sem_wait(&full_slots);    //Wait for a filled slot
        sem_wait(&binary_semaphore);

        int item = buffer[--count];
        cout << "Consumed: " << item << " | Buffer count: " << count << endl;

        sem_post(&binary_semaphore);  //Release lock
        sem_post(&empty_slots);   //flag: slot is available

        sleep(2);
    }
    return nullptr;
}
```



```
-

int main()
{
    //Initialize semaphores
    sem_init(&empty_slots, 0, BUFFER_SIZE); //empty slots initially
    sem_init(&full_slots, 0, 0);            //No items initially
    sem_init(&binary_semaphore, 0, 1);

    pthread_t producer_thread, consumer_thread;

    //Create producer and consumer threads
    pthread_create(&producer_thread, nullptr, producer, nullptr);
    pthread_create(&consumer_thread, nullptr, consumer, nullptr);

    //Wait for threads to finish
    pthread_join(producer_thread, nullptr);
    pthread_join(consumer_thread, nullptr);

    //Destroy semaphores
    sem_destroy(&empty_slots);
    sem_destroy(&full_slots);
    sem_destroy(&binary_semaphore);

    return 0;
}
```



```
ns3@ns3-virtual-machine:~$ g++ -pthread -o t3 task3.cpp
ns3@ns3-virtual-machine:~$ ./t3
Produced: 0 | Buffer count: 1
Consumed: 0 | Buffer count: 0
Produced: 1 | Buffer count: 1
Consumed: 1 | Buffer count: 0
Produced: 2 | Buffer count: 1
Consumed: 2 | Buffer count: 0
Produced: 3 | Buffer count: 1
Consumed: 3 | Buffer count: 0
Produced: 4 | Buffer count: 1
Consumed: 4 | Buffer count: 0
```