Code:

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

#define MAX_SIZE 10
sem_t semEmpty;
sem_t semFull;

pthread_mutex_t mutexBuffer;

class Queue
{
    long long qFront;
    long long qRear;
    long long queue_array[MAX_SIZE];

public:
    Queue() : qFront(-1), qRear(-1) {}
    bool isEmpty()
    {
        return qFront == qRear && qFront == -1;
    }
    bool IsFull()
    {
        return (qRear + 1) % MAX_SIZE == qFront ? true : false;
    }
    void enque(long long x)
    {
        if (IsFull())
        {
            cout << "Queue Full!!\n";
            return;
        }
        else if (isEmpty())
        {
            qFront = qRear = 0;
        }
        else
        {
            qRear = (qRear + 1) % MAX_SIZE;
        }
        queue_array[qRear] = x;
    }
    void deque_element()
    {
        if (isEmpty())
        {
            cout << "Error!! Cannot deque empty queue!!\n";
        }
        else if (qFront == qRear)
        {
            queue_array[qFront] = 0;
            qFront = qRear = -1;
```

```
        }
        else
        {
            queue_array[qFront] = 0;
            qFront = (qFront + 1) % MAX_SIZE;
        }
    }
    long long get_front()
    {
        if (isEmpty())
        {
            cout << "Error!! Cannot return front from empty queue!\n";
            return -1;
        }
        else
        {
            return queue_array[qFront];
        }
    }
    void printStatus()
    {
        if (isEmpty())
        {
            cout << "Buffer Holds Nothing\n\n";
        }
        else
        {
            cout << "\nCurrent State of the buffer :  ";
            for (int i = 0; i < MAX_SIZE; i++)
            {
                cout << queue_array[i] << " ";
            }
            cout << "\n\n";
        }
    }
};
Queue buffer;
int item = 0;
void *producer(void *args)
{
    long tid  = reinterpret_cast<std::uintptr_t>(args);
    while (1)
    {
        if (buffer.IsFull())
        {
            cout <<"\nProducer "<<tid<<" is Waiting for a consumer to
consume\n";
        }
        // Add to the buffer
        sem_wait(&semEmpty);
        cout << "\n----> Producer "<<tid<<" is waiting for the mutex\n";
        pthread_mutex_lock(&mutexBuffer);
        cout << "\n----> Producer "<<tid<<" received the mutex\n";
        buffer.enque(++item);
        cout << "\nproducer "<<tid<<" produced : " <<item<< endl;
        buffer.printStatus();
        pthread_mutex_unlock(&mutexBuffer);
```

```
        sem_post(&semFull);
    }
}

void *consumer(void *args)
{

    long tid  = reinterpret_cast<std::uintptr_t>(args);
    while (1)
    {
        int y;
        if (buffer.isEmpty())
        {
            cout <<"\nConsumer " << tid << " is Waiting for a producer to
produce\n";
        }
        // Consuming (Removing) from the buffer
        sem_wait(&semFull);
        cout << "\n---->Consumer " << tid << " is waiting for the mutex \n";
        pthread_mutex_lock(&mutexBuffer);
        cout << "\n---->Consumer " << tid << " received the mutex \n";
        y = buffer.get_front();
        buffer.deque_element();
        cout << "\nConsumer " << tid << " Consumed " << y << endl;
        buffer.printStatus();
        pthread_mutex_unlock(&mutexBuffer);
        sem_post(&semEmpty);
        // Consumer
    }
}

int main()
{
    int producerCount, consumerCount;
    /*cout << "Enter The Number of Producers : ";
    cin >> producerCount;
    cout << "Enter The Number of Consumers : ";
    cin >> consumerCount;*/

    producerCount = consumerCount =5;
    pthread_t th[producerCount + consumerCount];
    pthread_mutex_init(&mutexBuffer, NULL);
    sem_init(&semEmpty, 0, 10);
    sem_init(&semFull, 0, 0);
    long i;
    for (i = 0; i < producerCount; i++)
    {
        if (pthread_create(&th[i], NULL, &producer, (void *)(i + 1)) != 0)
        {
            cerr << "Failed to create thread";
        }
    }
    for (int j = 0; j < consumerCount; j++)
    {

        if (pthread_create(&th[j + i], NULL, &consumer, (void *)(i + 1)) !=
0)
```

```
        {
            cerr << "Failed to create thread";
        }
    }
    for (i = 0; i < producerCount + consumerCount; i++)
    {
        if (pthread_join(th[i], NULL) != 0)
        {
            cerr << "Failed to join thread";
        }
    }
    sem_destroy(&semEmpty);
    sem_destroy(&semFull);
    pthread_mutex_destroy(&mutexBuffer);
    return 0;
}
```

Output :

```
----> Producer
----> Producer 4 is waiting for the mutex

----> Producer 5 is waiting for the mutex
1 is waiting for the mutex

----> Producer 3 is waiting for the mutex

----> Producer 2 is waiting for the mutex

Consumer 6 is Waiting for a producer to produce

Consumer 6 is Waiting for a producer to produce

Consumer 6 is Waiting for a producer to produce

Consumer 6 is Waiting for a producer to produce

Consumer 6 is Waiting for a producer to produce

----> Producer 4 received the mutex

producer 4 produced : 1

Current State of the buffer :  1 0 0 0 0 0 0 0 0 0
```

```
 ----> Producer 5 received the mutex

 producer 5 produced : 2

 Current State of the buffer :  1 2 0 0 0 0 0 0 0 0


 ----> Producer 5 is waiting for the mutex

 ----> Producer 1 received the mutex

 producer 1 produced : 3

 Current State of the buffer :  1 2 3 0 0 0 0 0 0 0


 ----> Producer 1 is waiting for the mutex

 ---->Consumer 6 is waiting for the mutex

 ---->Consumer 6 is waiting for the mutex

 ---->Consumer 6 is waiting for the mutex

 ----> Producer 4 is waiting for the mutex

 ----> Producer 3 received the mutex
```

```
producer 3 produced : 4

Current State of the buffer :  1 2 3 4 0 0 0 0 0 0


----> Producer 3 is waiting for the mutex

----> Producer 2 received the mutex

producer 2 produced : 5

Current State of the buffer :  1
---->Consumer 6 is waiting for the mutex
 2 3 4 5 0 0 0 0 0


----> Producer 5 received the mutex

producer 5 produced : 6

Current State of the buffer :  1 2 3 4 5 6 0 0 0 0


---->Consumer 6 is waiting for the mutex

----> Producer 2 is waiting for the mutex
```

```
----> Producer 1 received the mutex

producer 1 produced : 7

Current State of the buffer :  1 2 3 4 5 6 7 0 0 0


---->Consumer 6 received the mutex

Consumer 6 Consumed 1

Current State of the buffer :  0 2 3 4 5 6 7 0 0 0


---->Consumer 6 received the mutex

Consumer 6 Consumed 2

Current State of the buffer :  0 0 3 4 5 6 7 0 0 0


----> Producer 5 is waiting for the mutex

---->Consumer 6 received the mutex

Consumer 6 Consumed 3
```

```
Current State of the buffer :  0 0 0 4 5 6 7 0 0 0


---->Consumer 6 is waiting for the mutex

----> Producer 4 received the mutex

producer 4 produced : 8

Current State of the buffer :  0 0 0 4 5 6 7 8 0 0


---->Consumer 6 is waiting for the mutex

----> Producer 3 received the mutex

producer 3 produced : 9

Current State of the buffer :  0 0 0 4 5 6 7 8 9 0


----> Producer 4 is waiting for the mutex

---->Consumer 6 received the mutex

Consumer 6 Consumed 4
```