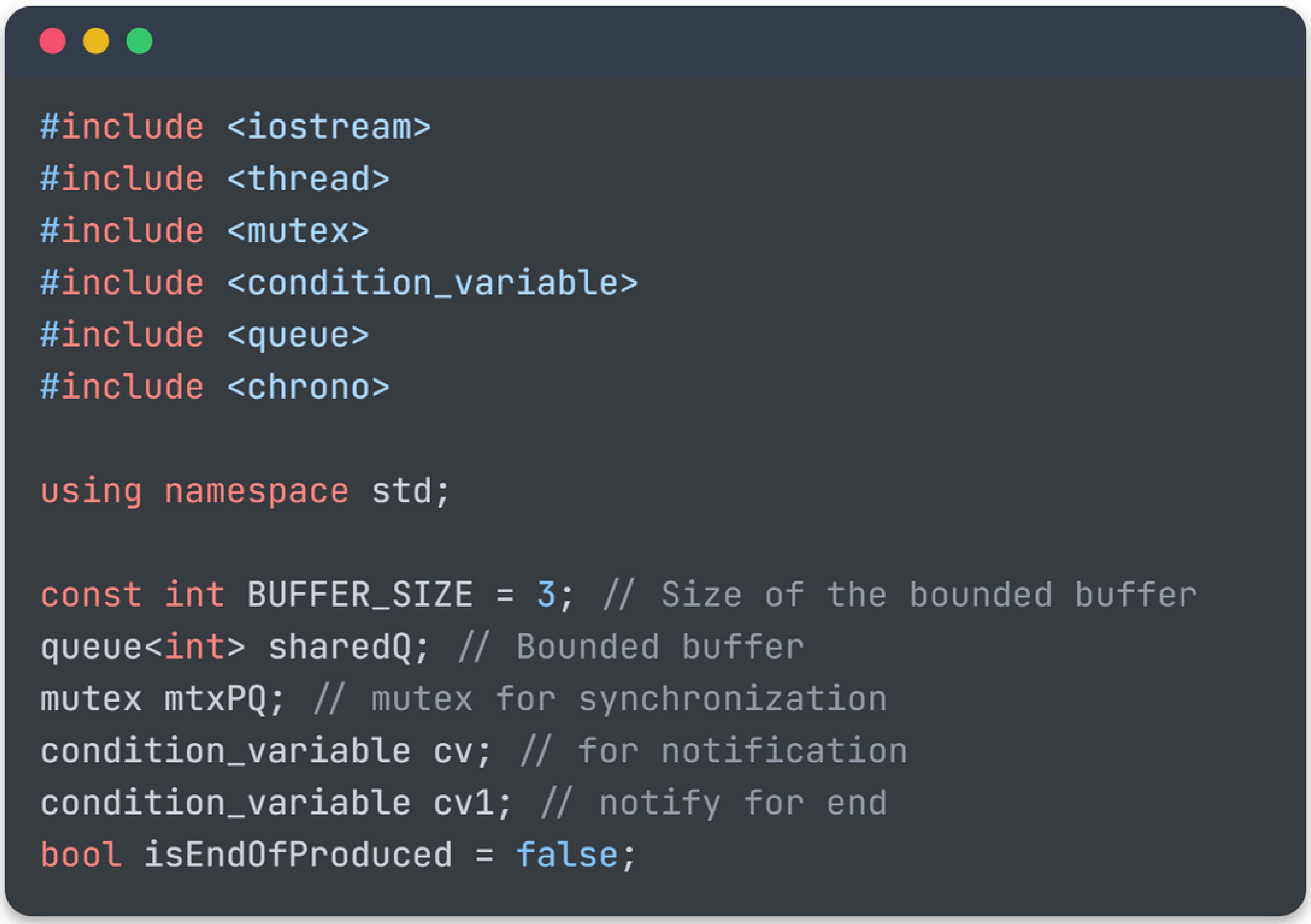
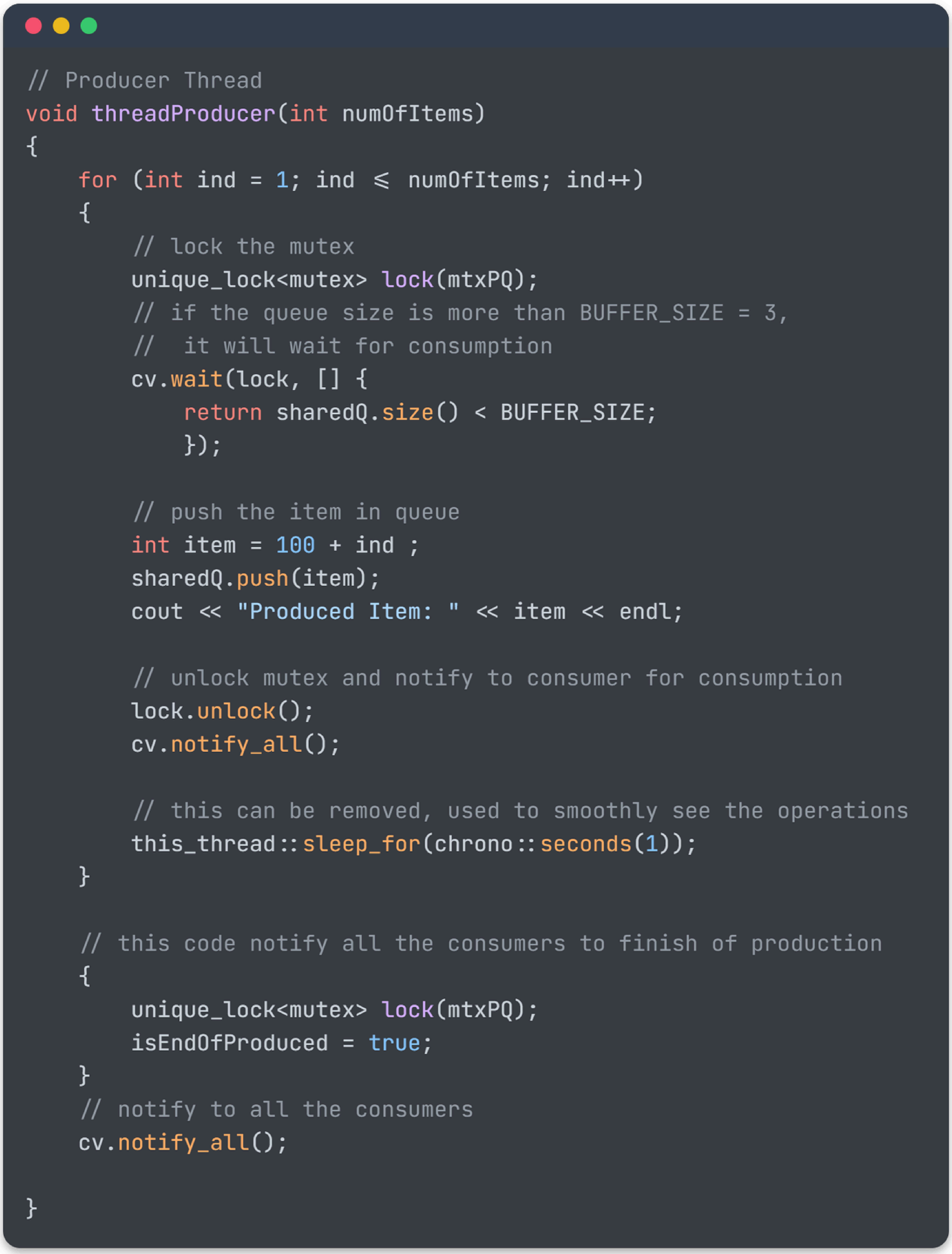
Producer Consumer Problem in Multithreading



```cpp
#include <iostream>
#include <thread>
#include <mutex>
#include <condition_variable>
#include <queue>
#include <chrono>

using namespace std;

const int BUFFER_SIZE = 3; // Size of the bounded buffer
queue<int> sharedQ; // Bounded buffer
mutex mtxPQ; // mutex for synchronization
condition_variable cv; // for notification
condition_variable cv1; // notify for end
bool isEndOfProduced = false;
```

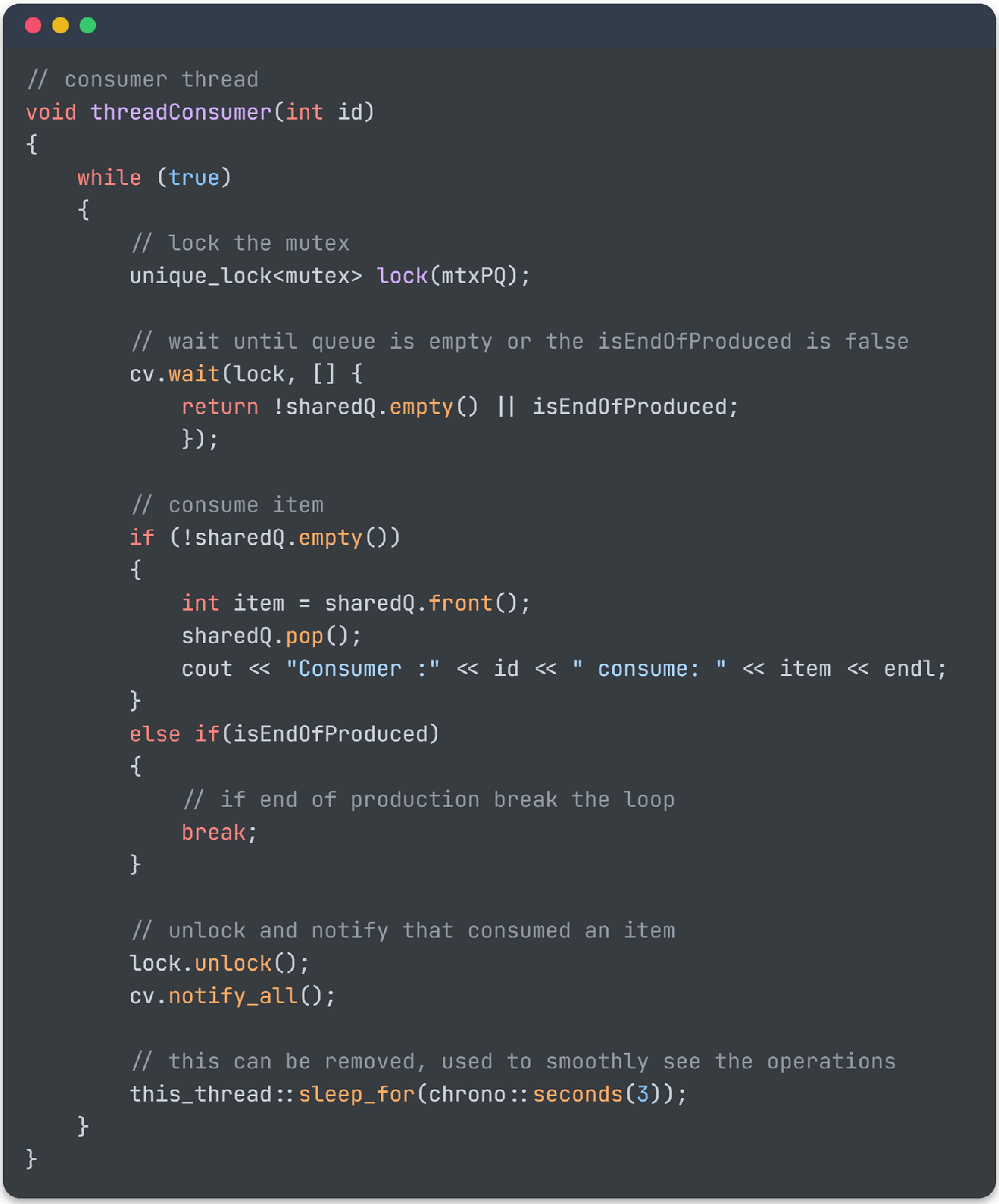
Smitesh Tamboli

NEXT



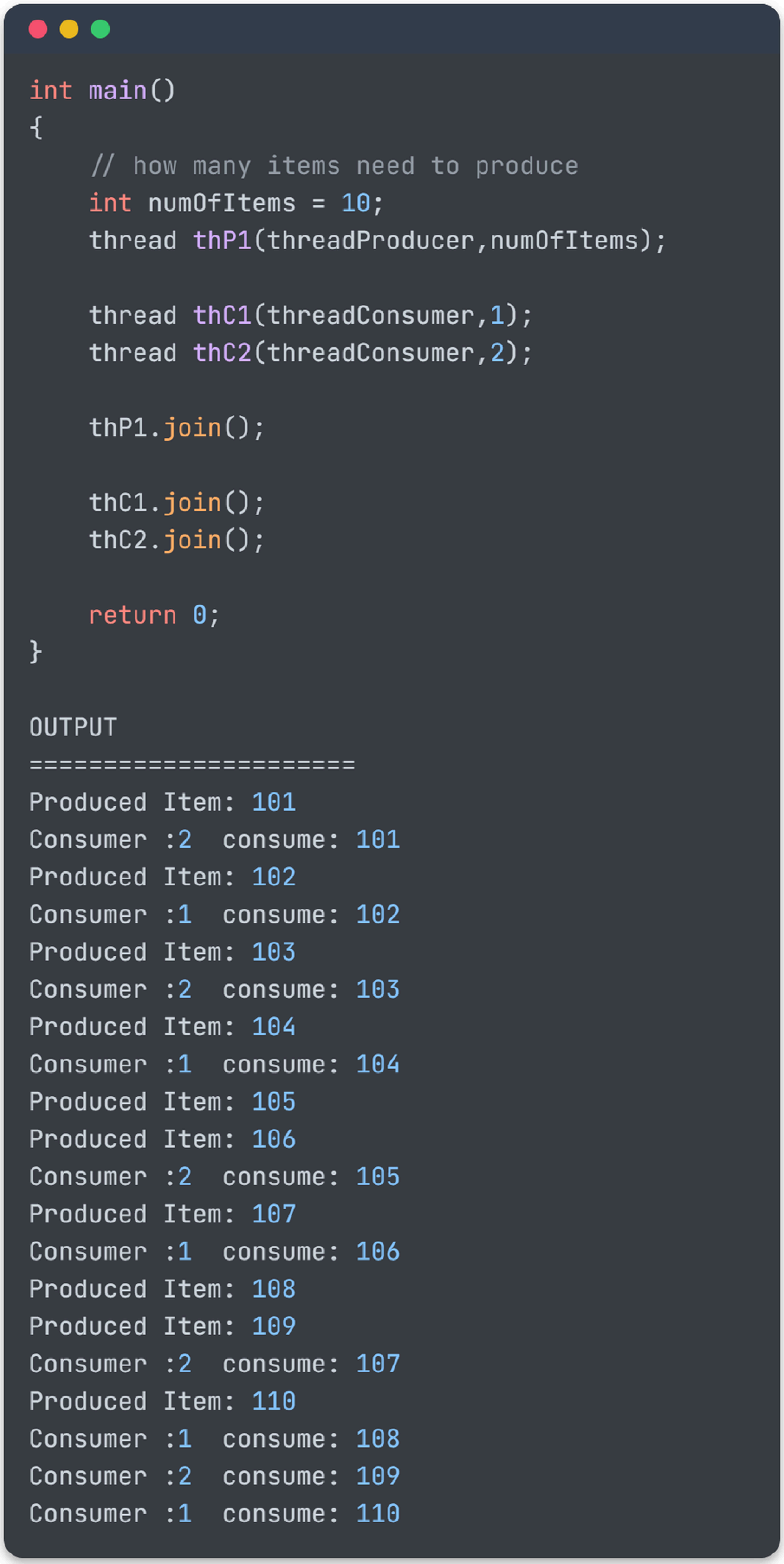
```cpp
// Producer Thread
void threadProducer(int numOfItems)
{
    for (int ind = 1; ind <= numOfItems; ind++)
    {
        // lock the mutex
        unique_lock<mutex> lock(mtxPQ);
        // if the queue size is more than BUFFER_SIZE = 3,
        //  it will wait for consumption
        cv.wait(lock, [] {
            return sharedQ.size() < BUFFER_SIZE;
            });

        // push the item in queue
        int item = 100 + ind ;
        sharedQ.push(item);
        cout << "Produced Item: " << item << endl;

        // unlock mutex and notify to consumer for consumption
        lock.unlock();
        cv.notify_all();

        // this can be removed, used to smoothly see the operations
        this_thread::sleep_for(chrono::seconds(1));
    }

    // this code notify all the consumers to finish of production
    {
        unique_lock<mutex> lock(mtxPQ);
        isEndOfProduced = true;
    }
    // notify to all the consumers
    cv.notify_all();

}
```

NEXT



```cpp
// consumer thread
void threadConsumer(int id)
{
    while (true)
    {
        // lock the mutex
        unique_lock<mutex> lock(mtxPQ);

        // wait until queue is empty or the isEndOfProduced is false
        cv.wait(lock, [] {
            return !sharedQ.empty() || isEndOfProduced;
            });

        // consume item
        if (!sharedQ.empty())
        {
            int item = sharedQ.front();
            sharedQ.pop();
            cout << "Consumer :" << id << " consume: " << item << endl;
        }
        else if(isEndOfProduced)
        {
            // if end of production break the loop
            break;
        }

        // unlock and notify that consumed an item
        lock.unlock();
        cv.notify_all();

        // this can be removed, used to smoothly see the operations
        this_thread::sleep_for(chrono::seconds(3));
    }
}
```

NEXT



```
int main()
{
    // how many items need to produce
    int numOfItems = 10;
    thread thP1(threadProducer,numOfItems);

    thread thC1(threadConsumer,1);
    thread thC2(threadConsumer,2);

    thP1.join();

    thC1.join();
    thC2.join();

    return 0;
}

OUTPUT
======================
Produced Item: 101
Consumer :2  consume: 101
Produced Item: 102
Consumer :1  consume: 102
Produced Item: 103
Consumer :2  consume: 103
Produced Item: 104
Consumer :1  consume: 104
Produced Item: 105
Produced Item: 106
Consumer :2  consume: 105
Produced Item: 107
Consumer :1  consume: 106
Produced Item: 108
Produced Item: 109
Consumer :2  consume: 107
Produced Item: 110
Consumer :1  consume: 108
Consumer :2  consume: 109
Consumer :1  consume: 110
```