**Project 2: Producer and Consumer**

Duc Vo & Ty Gehrke

Grand Canyon University

CST-315: Operating Systems Lecture and Lab

Professor Citro

1/28/2024

# 

# Team Members

* Ty Gehrke
* Duc Vo

# Project Description

This project aims to implement a multi-threaded solution for the classic Producer-Consumer problem using C and the pthread library. The primary goal is to create Producer and Consumer processes that coordinate efficiently without utilizing explicit synchronization mechanisms like semaphores. Instead, the implementation relies on thread management, sleep, and wakeup commands to ensure seamless interaction between the producer and consumer threads.

Theoretical Background

The Producer-Consumer problem is a fundamental challenge in programming, involving the synchronized sharing of resources between a producer and a consumer. This project delves into multi-threading principles by designing threads for the Producer and Consumer. The circular buffer, managed by the put() and get() functions, facilitates item exchange without traditional synchronization constructs. By introducing delays through sleep() and wakeup(), the project provides a practical exploration of thread coordination, offering students insights into concurrent programming and enhancing their skills in C programming and thread management.

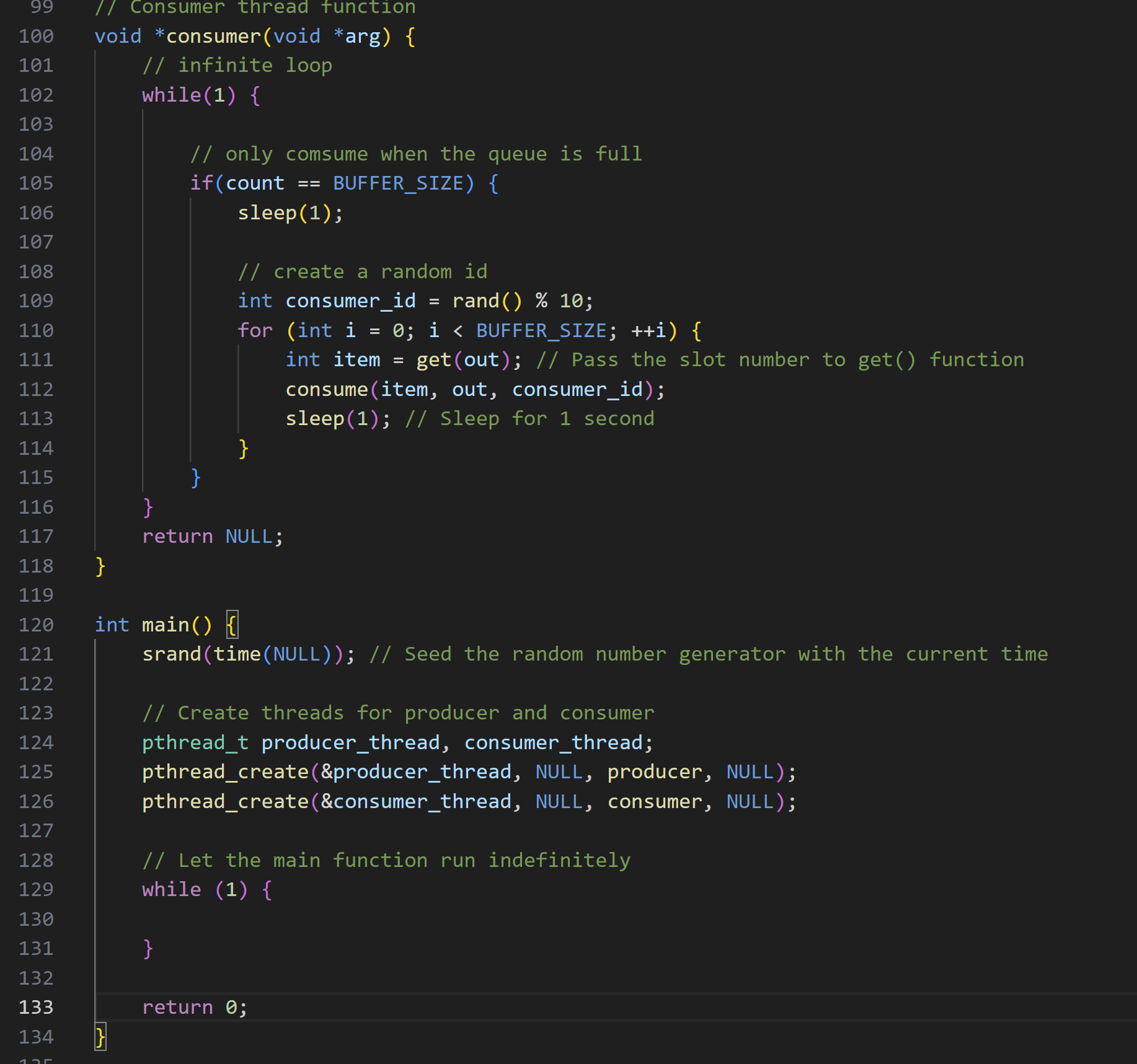
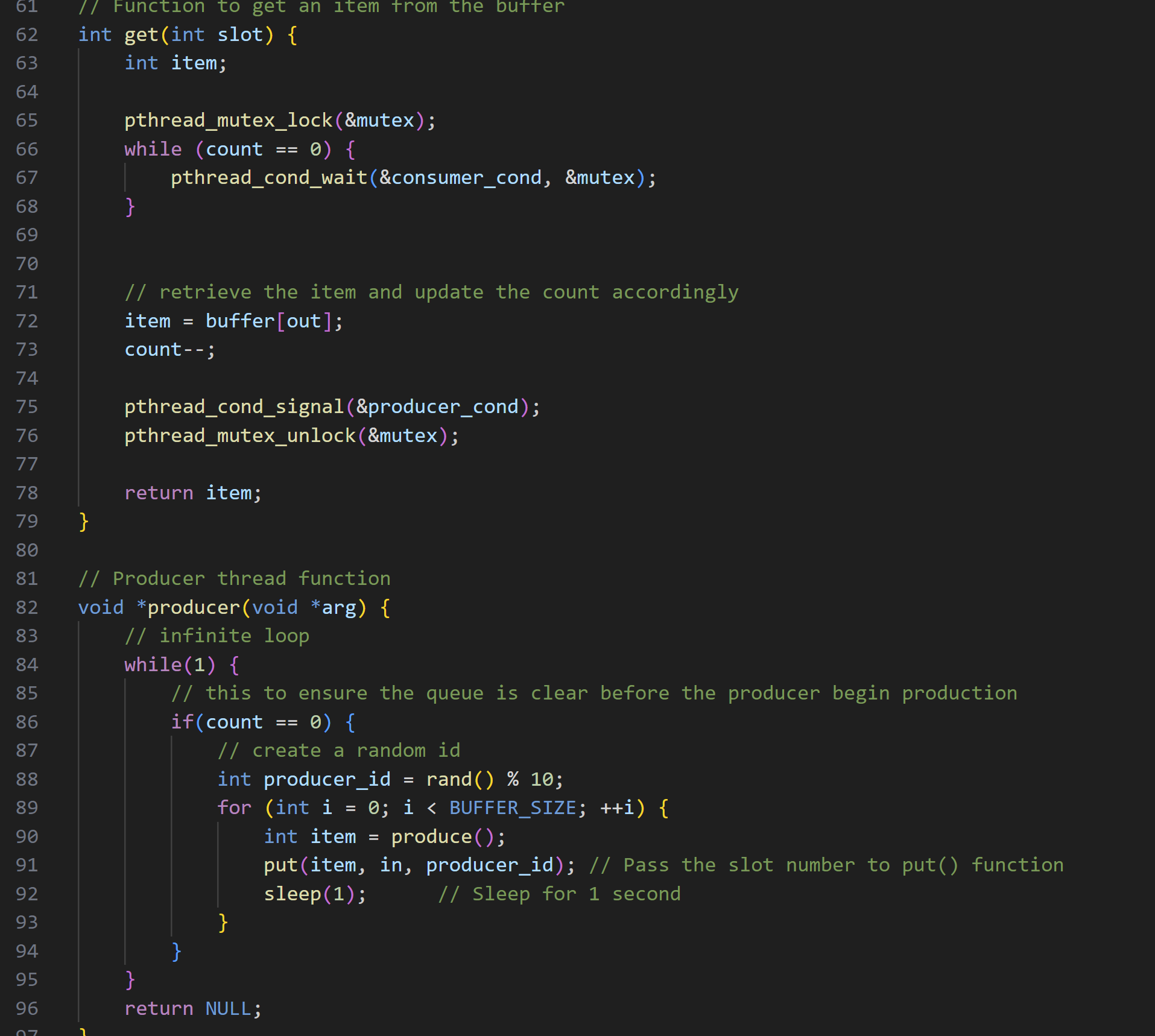
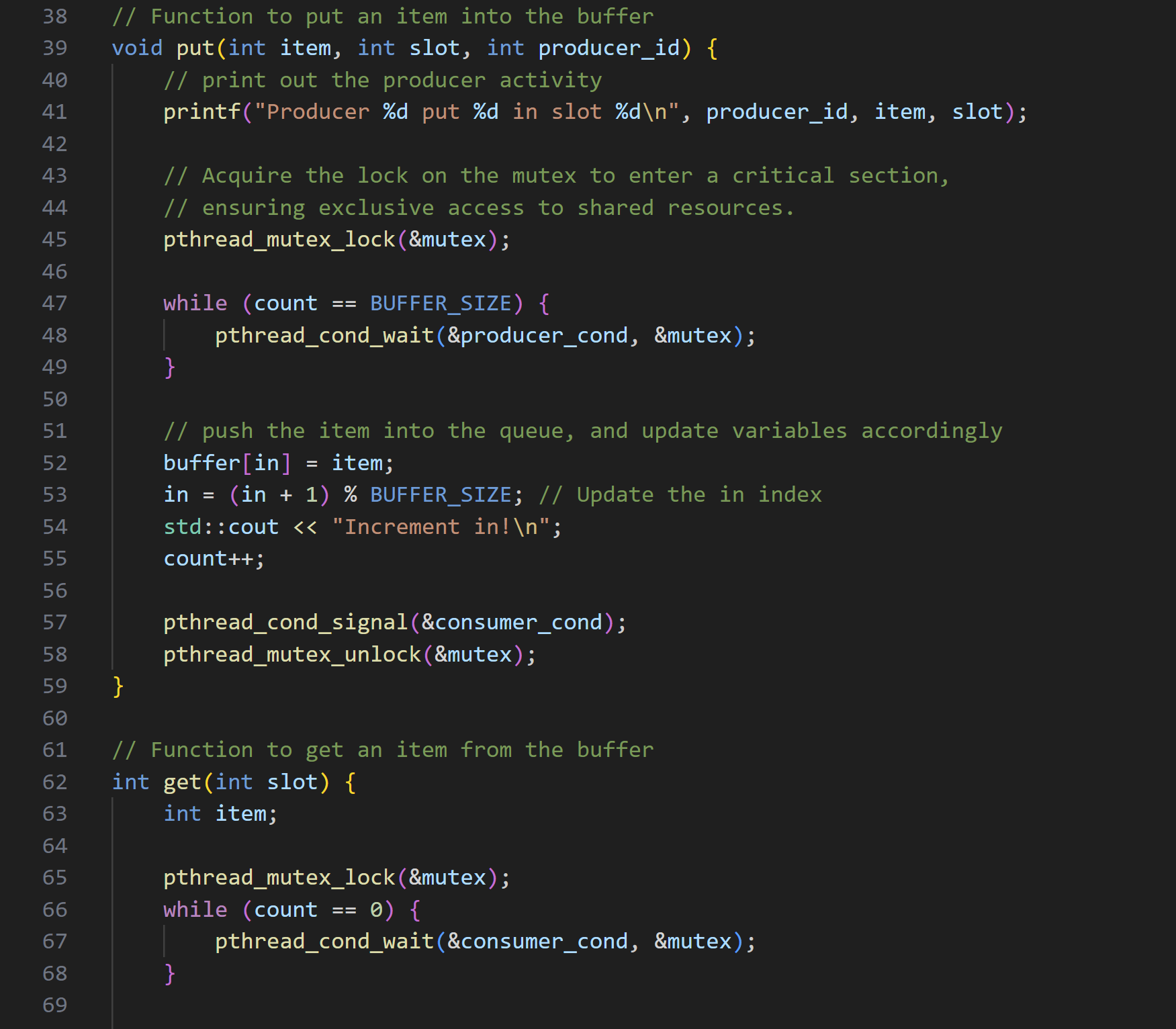
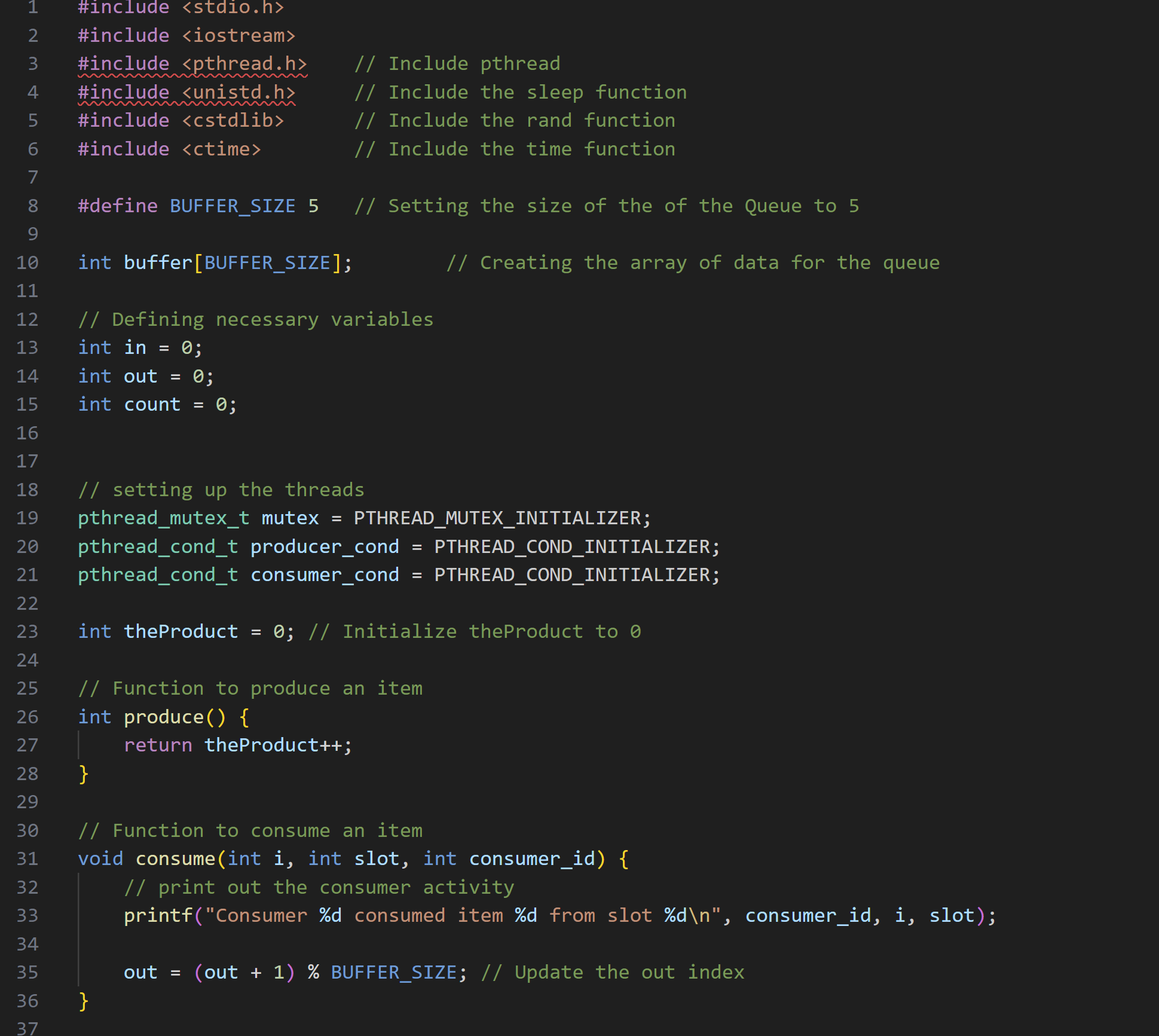
# Approach and Implementation

* **Buffer and Pointers:**
  + The shared buffer is represented as an array (buffer) with a specified size (BUFFER\_SIZE).
  + Two pointers (in and out) keep track of the position in the buffer, and count maintains the number of items currently in the buffer.
* **Mutex and Condition Variables:**
  + The code utilizes the pthread\_mutex\_t type for a mutex (mutex) and two condition variables (producer\_cond and consumer\_cond) for synchronization.
  + The mutex ensures exclusive access to shared data structures, preventing race conditions.
  + Condition variables are used to signal and wait for specific conditions to be met, facilitating communication between the producer and consumer threads.
* **Producer and Consumer Functions:**
  + The produce() function generates a unique item (incrementing theProduct).
  + The consume() function prints a message indicating that a consumer has consumed an item from a specific slot.
* **put() and get() Functions:**
  + The put() function adds an item to the buffer, considering the buffer size and waiting if it is full.
  + The get() function retrieves an item from the buffer, considering the buffer size and waiting if it is empty.
* **Producer and Consumer Threads:**
  + The producer and consumer functions represent the main execution logic of their respective threads.
  + These threads run indefinitely, producing and consuming items in a loop. The loop is triggered when the buffer is either empty (count == 0) for the producer or full (count == BUFFER\_SIZE) for the consumer.
* **Main Function:**
  + The main function creates threads for the producer and consumer, and it waits for their completion using pthread\_join.
* **Sleep and Randomization:**
  + The sleep(1) statements are introduced to simulate the production and consumption times.
  + rand() % 10 is used to generate random producer and consumer IDs.
* **Thread Safety:**
  + The use of mutexes and condition variables ensures that the producer and consumer threads interact safely, avoiding data races and ensuring proper synchronization.

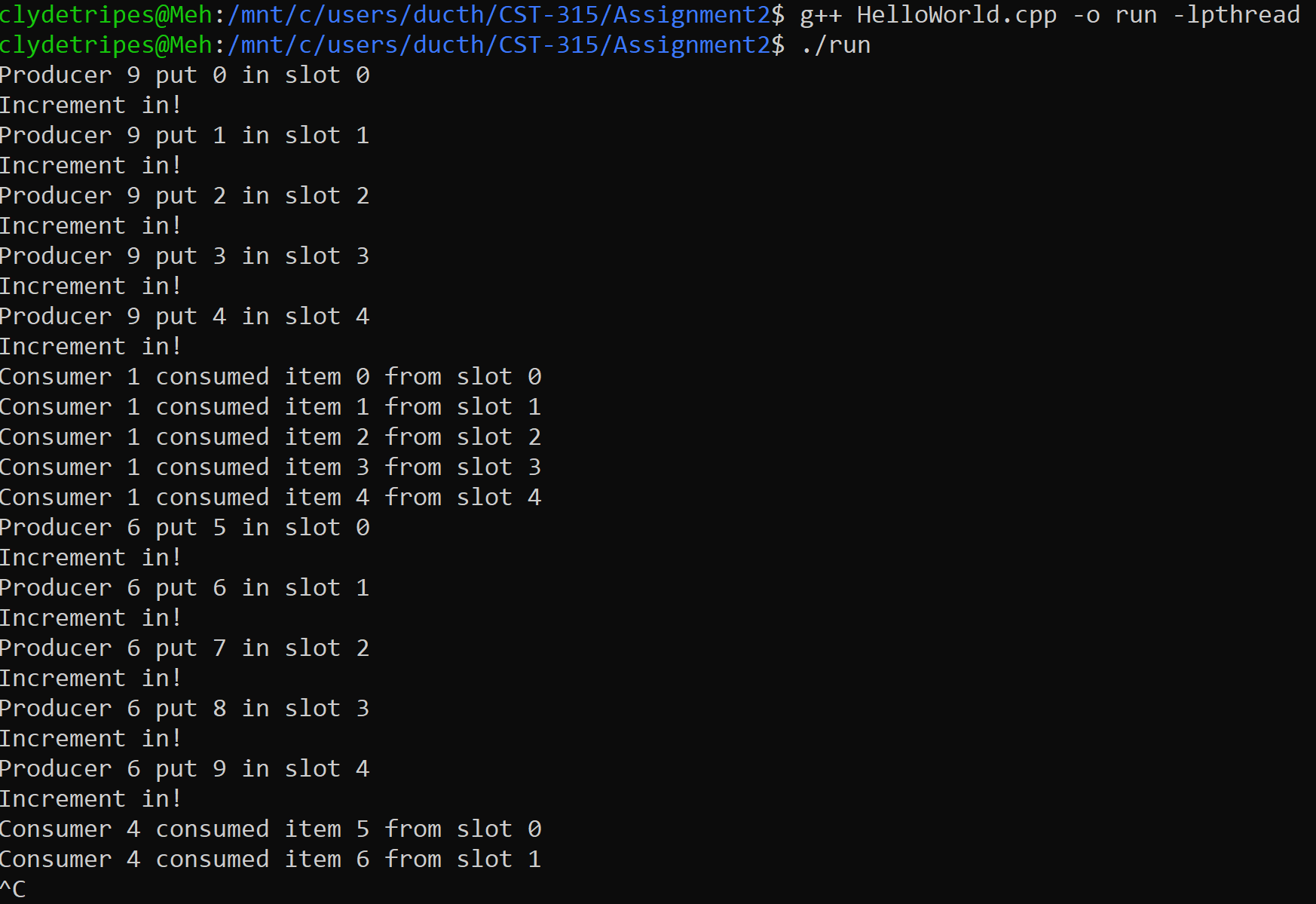
This implementation effectively demonstrates the coordination between producer and consumer threads in a multi-threaded environment, addressing issues related to buffer overflow and underflow through synchronization mechanisms.

Screenshots

Code:



Output:



# References

GeeksforGeeks. (2023, May 9). *Thread functions in C/C++*. GeeksforGeeks. <https://www.geeksforgeeks.org/thread-functions-in-c-c/>

Ippolito, G. (n.d.). *POSIX thread (pthread) libraries*. Linux Tutorial: POSIX Threads. <https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html>