Assignment 1: Producer and Consumer

Nolan Tuttle & Mathew Hobson

College of Engineering and Technology, Grand Canyon University

CST-315: Operating Systems Lecture and Lab

Professor Citro

January 26, 2025

Explanation of the Approach

The program implements a **producer-consumer problem** using multithreading in C. Below is a detailed breakdown of the **approach**, focusing on how the stack, threads, and synchronization work together.

---

1. Problem Scope

The **producer-consumer problem** involves:

- **Producer**: Continuously generates items and places them into a shared buffer (stack in this case).
- **Consumer**: Continuously removes items from the shared buffer for processing.
- The challenge lies in **synchronizing access** to the shared buffer to avoid:
    - **Race conditions**: Simultaneous access/modification leading to data corruption.
    - **Deadlocks**: Threads waiting indefinitely for resources.
    - **Starvation**: One thread unable to access the buffer because another monopolizes it.

---

2. Data Structures

- **Stack**: Acts as a shared buffer between the producer and consumer.
    - Implemented as a circular array with a fixed size (STACK_SIZE).
    - Indices (top and bottom) allow for efficient access:
        - **top**: Points to the next empty slot for the producer.

        - **bottom**: Points to the next item for the consumer.
    - **count**: Tracks the number of items in the stack.
- **Mutex**: Ensures **mutual exclusion**, so only one thread can modify the stack at a time.

---

3. Key Operations

1. **Push**:
    - Adds an item to the stack.
    - Checks if the stack is full; waits (busy-wait) if it is.
    - Locks the stack using a mutex to ensure thread safety during updates.
2. **Consume**:
    - Removes an item from the stack.
    - Checks if the stack is empty; waits (busy-wait) if it is.
    - Locks the stack using a mutex during removal.

---

4. Threads

- **Producer Thread**:
    - Continuously produces items using the produce() function.
    - Adds items to the stack using put() (internally calls push()).
- **Consumer Thread**:

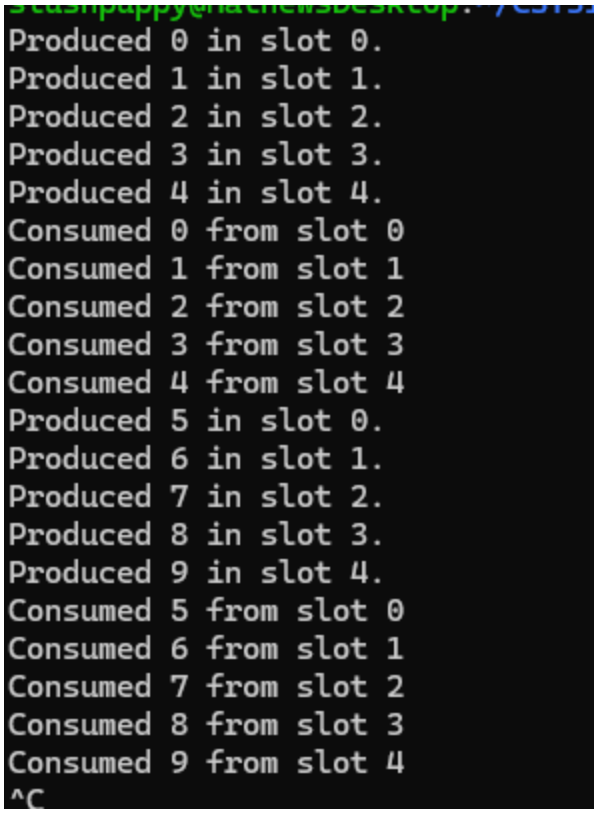
- Continuously retrieves items from the stack using get() (internally calls consume()).
- Processes retrieved items (e.g., prints them).

---

5. Synchronization

- A **mutex** is used to prevent concurrent access to the stack, ensuring:
  - Only one thread (producer or consumer) can modify the stack at a time.
  - Consistency
  - of shared data (top, bottom, stack).

Results:



Team Member	**Role**	**Tasks Completed**
Mathew Hobson	Group Member	Code, Documentation
Nolan Tuttle	Group member	Code, Documentation