

Danny Hido

10/22/2025

CS 33211 Operating Systems

Assignment One Documentation

### **Overview:**

Producer-Consumer Problem: A problem in operating problem where two processes (the producer and consumer) share a common, finite buffer. The producer generates data and puts it on the buffer. The consumer takes the data from the buffer and processes it. When the buffer is full, the producer will not add data to it. When the buffer is empty, the consumer will not take from it. To handle this, semaphores or mutex locks for synchronization are used to control access. Only one process is allowed to access the shared resource at the same time.

This project is an implementation of the consumer-producer problem in C++. POSIX semaphores are used to share memory between two processes in two files (producer.cpp and consumer.cpp). As these processes access shared memory, they output what they are doing to the console.

### **Usage Instructions**

This program must be run in a Linux environment with a g++ compiler.

For non-Linux users, Docker is a recommended virtual environment to use. Download Docker first and follow these commands.

#### Docker setup for non-linux users:

```
docker run -it -v %cd%:/workspace ubuntu bash
```

```
apt update
```

```
cd /workspace
```

Once program is local, enter these commands to run.

g++ installation:

```
apt install -y g++ make
```

Program compilation:

```
g++ producer.cpp -o producer -lpthread
```

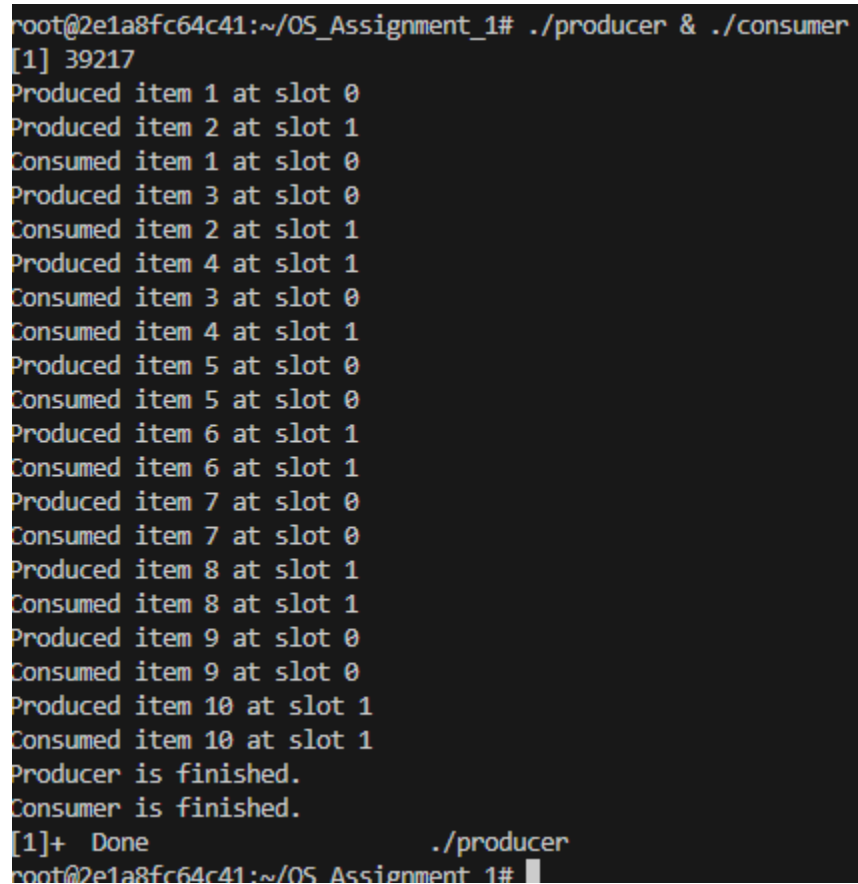
```
g++ consumer.cpp -o consumer -lpthread
```

Run Program:

```
./producer & ./consumer
```

**Expected Output:**

Below is a screenshot of expected output to the terminal.



```
root@2e1a8fc64c41:~/OS_Assignment_1# ./producer & ./consumer
[1] 39217
Produced item 1 at slot 0
Produced item 2 at slot 1
Consumed item 1 at slot 0
Produced item 3 at slot 0
Consumed item 2 at slot 1
Produced item 4 at slot 1
Consumed item 3 at slot 0
Consumed item 4 at slot 1
Produced item 5 at slot 0
Consumed item 5 at slot 0
Produced item 6 at slot 1
Consumed item 6 at slot 1
Produced item 7 at slot 0
Consumed item 7 at slot 0
Produced item 8 at slot 1
Consumed item 8 at slot 1
Produced item 9 at slot 0
Consumed item 9 at slot 0
Produced item 10 at slot 1
Consumed item 10 at slot 1
Producer is finished.
Consumer is finished.
[1]+  Done                  ./producer
root@2e1a8fc64c41:~/OS_Assignment_1#
```

## Key Components:

### 1. Shared Memory

- The shared memory portion (`shm_open`) is accessed by name.
- Producer calls `shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666)` to create or open the shared memory portion.
- `ftruncate` sets the object size to `sizeof(SharedBuffer)`.
- Consumer opens the name and calls `mmap` to access.

### 2. Semaphores

- Coordinate access to the shared buffer and count available spots.
- Three semaphores: empty, full, and mutex.
- Empty: number of free slots in the buffer. Initialized to `BUFFER_SIZE (2)`. Producer `sem_wait(empty)` before writing, and consumer posts after reading.
- Full: number of filled slots on the buffer. Initialized to 0. Consumer `sem_wait(full)` before reading, and producer posts after writing.
- Mutex: ensures mutual exclusion. Initialized to 1. Both producer and consumer `sem_wait(mutex)` before changing buffer state and `sem_post(mutex)` after.

### 3. Mutual Exclusion:

- Implemented using mutex similar to a lock. Only one process may hold it at a time.
- Prevents race conditions on shared state.

### 4. Random Mechanics:

- Stopping condition: after the producer finishes a loop of `MAX_ITEMS` iterations, `items` is changed to -1. Consumer checks for this each time, and upon completion, the condition hits, exiting the consumer loop.
- Sleep: Consumer sleeps for 2 seconds at the beginning of execution, allowing the producer to produce first.
- Initial Cleanup: Producer cleans up semaphores before instructions to ensure a clean start.