Assignment 2: Producer-Consumer Problem

Karthikeyan G
Subject Code: 24CY613
Roll Number: CB.SC.P2.CYS24008

December 16, 2024

1 Introduction

This document explains the implementation of the producer-consumer problem using Python's threading module, both with and without synchronization. The producer generates items and adds them to a shared buffer, while the consumer takes items from the buffer.

2 Code Without Synchronization

```
import threading
import time
import random

# Initialize shared buffer
buffer = []
buffer_size = 10

# Function producer
def producer():
    item = 0
    while True:
        if len(buffer) < buffer_size:
            buffer.append(item)
            print(f"Produced: {item}")
            item += 1
        else:
            print("Producer is waiting...")

# Function consumer
def consumer():
    while True:
        if buffer:
            item = buffer.pop(0)
            print(f"Consumed: {item}")
        else:
            print("Consumer is waiting...")

```

```
# Creating threads
producer_thread = threading.Thread(target=producer)
consumer_thread = threading.Thread(target=consumer)

# Starting threads
producer_thread.start()
consumer_thread.start()

# Joining threads
producer_thread.join()
consumer_thread.join()
```

Listing 1: Producer-Consumer Code Without Synchronization

3 Explanation Without Synchronization

3.1 Shared Buffer

The shared buffer is initialized as an empty list with a maximum size of 10.

3.2 Producer Function

The producer function generates items and adds them to the buffer if the buffer is not full. If the buffer is full, the producer waits.

3.3 Consumer Function

The consumer function takes items from the buffer if the buffer is not empty. If the buffer is empty, the consumer waits.

3.4 Threads

Two threads are created: one for the producer and one for the consumer. These threads are started and then joined to ensure they complete execution.

4 Output Without Synchronization

5 Code With Synchronization

```
import threading
import time
import random

# Shared buffer
buffer = []
buffer_size = 10

# Total produced and consumed counters
total_produced = 0
total_consumed = 0

# Number of items to produce before stopping
num_items_to_produce = 50

# Lock and condition variable
lock = threading.Lock()
condition = threading.Condition(lock)

# Producer function
def producer():
    global total_produced
    item = 0
    while total_produced < num_items_to_produce:
        with condition:
            while len(buffer) >= buffer_size:  # Wait if buffer is
    full
                print("Producer is waiting...")
                condition.wait()

            buffer.append(item)
            print(f"Produced: {item}")
            item += 1
            total_produced += 1

            condition.notify_all()  # Notify the consumer
        time.sleep(random.uniform(0.1, 0.5))  # Simulate production
     time

# Consumer function
def consumer():
    global total_consumed
    while total_consumed < num_items_to_produce:
        with condition:
            while not buffer:  # Wait if buffer is empty
                print("Consumer is waiting...")
                condition.wait()

            item = buffer.pop(0)
            print(f"Consumed: {item}")
            total_consumed += 1

            condition.notify_all()  # Notify the producer
        time.sleep(random.uniform(0.1, 0.5))  # Simulate
    consumption time
```

```
53
54 # Create threads
55 producer_thread = threading.Thread(target=producer)
56 consumer_thread = threading.Thread(target=consumer)
57
58 # Start threads
59 producer_thread.start()
60 consumer_thread.start()
61
62 # Wait for threads to finish
63 producer_thread.join()
64 consumer_thread.join()
65
66 # Print total production and consumption stats
67 print(f"Total Produced: {total_produced}")
68 print(f"Total Consumed: {total_consumed}")
```

Listing 2: Producer-Consumer Code With Synchronization

6 Explanation With Synchronization

6.1 Shared Buffer

The shared buffer is initialized as an empty list with a maximum size of 10.

6.2 Producer Function

The producer function generates items and adds them to the buffer if the buffer is not full. If the buffer is full, the producer waits. The producer uses a condition variable to wait and notify the consumer.

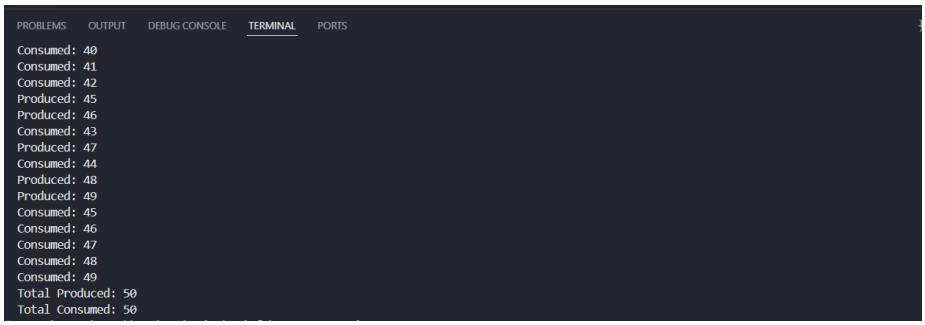
6.3 Consumer Function

The consumer function takes items from the buffer if the buffer is not empty. If the buffer is empty, the consumer waits. The consumer uses a condition variable to wait and notify the producer.

6.4 Threads

Two threads are created: one for the producer and one for the consumer. These threads are started and then joined to ensure they complete execution.

7 Output With Synchronization



8 Differences Between Synchronized and Unsynchronized Versions

8.1 Without Synchronization

- The producer and consumer threads do not use any synchronization mechanisms.
- This can lead to race conditions where both threads access the shared buffer simultaneously, causing data corruption or inconsistent states.
- The producer may overwrite items in the buffer, and the consumer may read invalid data.

8.2 With Synchronization

- The producer and consumer threads use a lock and a condition variable to synchronize access to the shared buffer.
- This ensures that only one thread can access the buffer at a time, preventing race conditions.
- The producer waits if the buffer is full, and the consumer waits if the buffer is empty, ensuring proper coordination between the threads.
- The condition variable is used to notify the waiting threads when the buffer state changes.