

# 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

```
1 import threading
2 import time
3 import random
4
5 # Initialize shared buffer
6 buffer = []
7 buffer_size = 10
8
9 # Function producer
10 def producer():
11     item = 0
12     while True:
13         if len(buffer) < buffer_size:
14             buffer.append(item)
15             print(f"Produced: {item}")
16             item += 1
17         else:
18             print("Producer is waiting...")
19
20 # Function consumer
21 def consumer():
22     while True:
23         if buffer:
24             item = buffer.pop(0)
25             print(f"Consumed: {item}")
26         else:
27             print("Consumer is waiting...")
28
```

```

29 # Creating threads
30 producer_thread = threading.Thread(target=producer)
31 consumer_thread = threading.Thread(target=consumer)
32
33 # Starting threads
34 producer_thread.start()
35 consumer_thread.start()
36
37 # Joining threads
38 producer_thread.join()
39 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

```

39     print("Consumer is waiting...")
40     time.sleep(random.uniform(0.1, 0.5)) # Simulate consumption time
41
42 # Create threads
43 producer_thread = threading.Thread(target=producer)
44 consumer_thread = threading.Thread(target=consumer)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python Debug Console

```

Produced: 43
Consumed: 42
Produced: 44
Consumed: 43
Produced: 45
Consumed: 44
Produced: 46
Produced: 47
Consumed: 45
Produced: 48
Consumed: 46
Consumed: 47
Produced: 49
Consumed: 48
Consumed: 49
Total Produced: 50
Total Consumed: 50

```

## 5 Code With Synchronization

```
1 import threading
2 import time
3 import random
4
5 # Shared buffer
6 buffer = []
7 buffer_size = 10
8
9 # Total produced and consumed counters
10 total_produced = 0
11 total_consumed = 0
12
13 # Number of items to produce before stopping
14 num_items_to_produce = 50
15
16 # Lock and condition variable
17 lock = threading.Lock()
18 condition = threading.Condition(lock)
19
20 # Producer function
21 def producer():
22     global total_produced
23     item = 0
24     while total_produced < num_items_to_produce:
25         with condition:
26             while len(buffer) >= buffer_size: # Wait if buffer is
27                 full
28                     print("Producer is waiting...")
29                     condition.wait()
30
31             buffer.append(item)
32             print(f"Produced: {item}")
33             item += 1
34             total_produced += 1
35
36             condition.notify_all() # Notify the consumer
37             time.sleep(random.uniform(0.1, 0.5)) # Simulate production
38             time
39
40 # Consumer function
41 def consumer():
42     global total_consumed
43     while total_consumed < num_items_to_produce:
44         with condition:
45             while not buffer: # Wait if buffer is empty
46                 print("Consumer is waiting...")
47                 condition.wait()
48
49             item = buffer.pop(0)
50             print(f"Consumed: {item}")
51             total_consumed += 1
52
53             condition.notify_all() # Notify the producer
54             time.sleep(random.uniform(0.1, 0.5)) # Simulate
55             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.

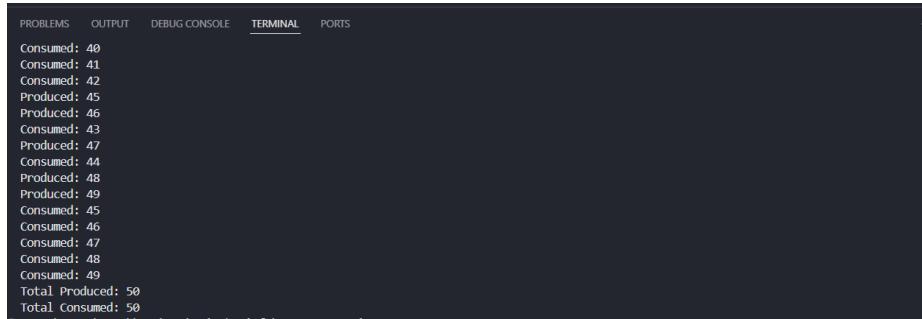
### 6.3 Consumer Function

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



The screenshot shows a terminal window with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS. The terminal displays a sequence of messages indicating the state of a shared buffer. The messages show alternating produced and consumed values, starting from 40 and ending at 50. The last message indicates that both the total produced and consumed counts are 50.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Consumed: 40
Consumed: 41
Consumed: 42
Produced: 45
Produced: 46
Consumed: 43
Produced: 47
Consumed: 44
Produced: 48
Produced: 49
Consumed: 45
Consumed: 46
Consumed: 47
Consumed: 48
Consumed: 49
Total Produced: 50
Total Consumed: 50
```

## 8 Differences Between Synchronized and Unsynchonized 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.