

## Assignment - 3

The producer-consumer problem is a synchronization challenge in the field of operating systems, particularly in scenarios involving concurrent programming and multi-threading. It revolves around two types of processes.

- Producers: These processes are responsible for generating data or items and placing them in a shared buffer.
- Consumers: These processes retrieve and consume items from the buffer.

The primary goal is to ensure the following conditions are met:

Producers should refrain from producing items if the buffer is full. Consumers should avoid consuming items if the buffer is empty. The central objective is to maintain synchronization between producers and consumers to prevent issues such as data corruption, race conditions, and deadlocks.

- Solution Approach:

A practical solution involves the use of bounded buffers and semaphores, which can be explained as follows:

1. Shared Buffer: A fixed-size buffer is utilized, acting as a common storage space for both producers and consumers.
2. Semaphore for Empty Slots (empty): Initialized to the size of the buffer. Represents the count of empty slots in the buffer. Decreases by producers when they add an item. Decreases by consumers when they remove an item.
3. Semaphore for Full Slots (full): Initialized to 0. Represents the count of filled slots in the buffer. Increased by producers when they add an item. Decreases by consumers when they remove an item.

Example:

Let's consider a scenario in a restaurant where there are chefs (producers) preparing dishes and waiters (consumers) serving these dishes to customers. The shared buffer is the kitchen counter where dishes are temporarily placed before being served.

1. Shared Buffer (Kitchen Counter): Represents the kitchen counter where the prepared dishes are temporarily stored.
2. Semaphore for Empty Serving Plates (empty): Initialized to the maximum capacity of the counter. Indicates the count of empty serving plates on the kitchen counter. Decreases as chefs place prepared dishes on empty plates. Decreases as waiters take dishes from the counter to serve customers.
3. Semaphore for Full Serving Plates (full): Initialized to 0. Represents the count of plates with prepared dishes on the counter. Increases as chefs place dishes on empty plates. Decreases as waiters take dishes from the counter to serve customers. In this analogy, the restaurant ensures that chefs don't prepare more dishes if there are no empty plates, and waiters don't serve if there are no prepared dishes on the counter, effectively managing the flow of food production and service.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import threading
import time
```

```
# Shared Memory variables
CAPACITY = 10
buffer = [-1 for i in range(CAPACITY)]
in_index = 0
out_index = 0
```

```
# Declaring Semaphores
mutex = threading.Semaphore()
empty = threading.Semaphore(CAPACITY)
full = threading.Semaphore(0)
```

```
# Producer Thread Class
class Producer(threading.Thread):
    def run(self):
        global CAPACITY, buffer, in_index, out_index
        global mutex, empty, full
        items_produced = 0
        counter = 0
        while items_produced < 20:
            empty.acquire()
            mutex.acquire()
            counter += 1
            buffer[in_index] = counter
            in_index = (in_index + 1)%CAPACITY
            print("Producer produced : ", counter)
            mutex.release()
            full.release()
            time.sleep(0)
            items_produced += 1
```

```
# Consumer Thread Class
class Consumer(threading.Thread):
    def run(self):
        global CAPACITY, buffer, in_index, out_index, counter
        global mutex, empty, full
        items_consumed = 0
        while items_consumed < 20:
            full.acquire()
            mutex.acquire()
            item = buffer[out_index]
            out_index = (out_index + 1)%CAPACITY
            print("Consumer consumed item : ", item)
            mutex.release()
            empty.release()
            time.sleep(0.5)
            items_consumed += 1
```

```
producer = Producer()
consumer = Consumer()
consumer.start()
producer.start()
producer.join()
consumer.join()
```

```
Producer produced : 1
Consumer consumed item : 1
Producer produced : 2
Producer produced : 3
Producer produced : 4
Producer produced : 5
Producer produced : 6
Producer produced : 7
Producer produced : 8
Producer produced : 9
Producer produced : 10
Producer produced : 11
Consumer consumed item : 2
Producer produced : 12
Consumer consumed item : 3
Producer produced : 13
Consumer consumed item : 4
Producer produced : 14
Consumer consumed item : 5
Producer produced : 15
Consumer consumed item : 6
Producer produced : 16
Consumer consumed item : 7
Producer produced : 17
Consumer consumed item : 8
Producer produced : 18
Consumer consumed item : 9
Producer produced : 19
Consumer consumed item : 10
Producer produced : 20
Consumer consumed item : 11
Consumer consumed item : 12
Consumer consumed item : 13
Consumer consumed item : 14
Consumer consumed item : 15
Consumer consumed item : 16
Consumer consumed item : 17
Consumer consumed item : 18
Consumer consumed item : 19
Consumer consumed item : 20
```

✓ Implementantation using if-else and for loop

```
CAPACITY = 10
buffer = [-1 for i in range(CAPACITY)]
```

```
in_index = 0
out_index = 0
```

```
mutex = threading.Semaphore()
empty = threading.Semaphore(CAPACITY)
full = threading.Semaphore(0)
```

```
class Producer(threading.Thread):
    def run(self):
        global CAPACITY, buffer, in_index
        global mutex, empty, full
        for counter in range(1, 21):
            empty.acquire()
            mutex.acquire()
            buffer[in_index] = counter
            in_index = (in_index + 1) % CAPACITY
            print("Producer produced:", counter)
            mutex.release()
            full.release()
            time.sleep(0)
```

```
class Consumer(threading.Thread):
    def run(self):
        global CAPACITY, buffer, out_index
        global mutex, empty, full
        for _ in range(20):
            full.acquire()
            mutex.acquire()
            item = buffer[out_index]
            out_index = (out_index + 1) % CAPACITY
            print("Consumer consumed item:", item)
            mutex.release()
            empty.release()
            time.sleep(0.5)
```

```
producer = Producer()
consumer = Consumer()
consumer.start()
producer.start()
producer.join()
consumer.join()
```

```
Producer produced: 1
Producer produced: 2
Producer produced: 3
Producer produced: 4
Producer produced: 5
Producer produced: 6
Producer produced: 7
Producer produced: 8
Producer produced: 9
Producer produced: 10
Consumer consumed item: 1
Producer produced: 11
Consumer consumed item: 2
Producer produced: 12
Consumer consumed item: 3
Producer produced: 13
Consumer consumed item: 4
Producer produced: 14
Consumer consumed item: 5
Producer produced: 15
Consumer consumed item: 6
Producer produced: 16
Consumer consumed item: 7
Producer produced: 17
Consumer consumed item: 8
Producer produced: 18
Consumer consumed item: 9
Producer produced: 19
Consumer consumed item: 10
Producer produced: 20
Consumer consumed item: 11
Consumer consumed item: 12
Consumer consumed item: 13
Consumer consumed item: 14
Consumer consumed item: 15
Consumer consumed item: 16
Consumer consumed item: 17
Consumer consumed item: 18
Consumer consumed item: 19
Consumer consumed item: 20
```

Example : Restaurant Order Fulfillment System - Producer-Consumer Problem

- Producers (Chefs): Represented by the Chef thread, they generate items by preparing dishes and place them in a shared buffer (kitchen counter).
- Buffer (Kitchen Counter): Simulates the shared buffer in the producer-consumer problem. It's a queue (kitchen\_counter) where Chefs (producers) deposit prepared dishes, and Waiters (consumers) retrieve and serve them.
- Consumers (Waiters): Represented by the Waiter thread, they consume items from the shared buffer (kitchen counter) by taking and

```
import queue
```

```
# Shared resources
kitchen_counter = queue.Queue(maxsize=5) # Shared buffer (kitchen counter)
empty_plates_semaphore = threading.Semaphore(5) # Semaphore for empty serving plates
full_plates_semaphore = threading.Semaphore(0) # Semaphore for full serving plates
```


```
# Producer (Chef) function
class Chef(threading.Thread):
    def run(self):
        for _ in range(10):
            time.sleep(1) # Simulate time taken to prepare a dish
            empty_plates_semaphore.acquire() # Decrease count of empty plates
            dish = f'Dish {time.time()}'
            kitchen_counter.put(dish) # Place the prepared dish on the kitchen counter
            print(f'Chef prepared {dish}')
            full_plates_semaphore.release() # Increase count of full plates
```

```
# Consumer (Waiter) function
class Waiter(threading.Thread):
    def run(self):
        for _ in range(10):
            full_plates_semaphore.acquire() # Decrease count of full plates
            dish = kitchen_counter.get() # Take a dish from the kitchen counter
            print(f'Waiter served {dish}')
            empty_plates_semaphore.release() # Increase count of empty plates
```

```
# Create threads for chef and waiter
chef_thread = Chef()
waiter_thread = Waiter()
```

```
# Start the threads
chef_thread.start()
waiter_thread.start()
```

```
# Allow the threads to run for a while (you can interrupt the program manually)
chef_thread.join()
waiter_thread.join()
```

```
 Chef prepared Dish 1706694540.4464412
Waiter served Dish 1706694540.4464412
Chef prepared Dish 1706694541.4482815
Waiter served Dish 1706694541.4482815
Chef prepared Dish 1706694542.4495764
Waiter served Dish 1706694542.4495764
Chef prepared Dish 1706694543.4508727
Waiter served Dish 1706694543.4508727
Chef prepared Dish 1706694544.4521263
Waiter served Dish 1706694544.4521263
Chef prepared Dish 1706694545.453453
Waiter served Dish 1706694545.453453
Chef prepared Dish 1706694546.4547088
Waiter served Dish 1706694546.4547088
Chef prepared Dish 1706694547.4556842
Waiter served Dish 1706694547.4556842
Chef prepared Dish 1706694548.4571862
Waiter served Dish 1706694548.4571862
Chef prepared Dish 1706694549.4584432
Waiter served Dish 1706694549.4584432
```