DIVYANSHU MAKHARIA
221070036

ASSIGNMENT 4

AIM:

Write a Python program to simulate producer-consumer problem using semaphores.

THEORY:

The Producer-Consumer problem is a classical multi-process synchronization problem, that is we are trying to achieve synchronization between more than one process.There is one Producer in the producer-consumer problem, Producer is producing some items, whereas there is one Consumer that is consuming the items produced by the Producer. The same memory buffer is shared by both producers and consumers which is of fixed-size.The task of the Producer is to produce the item, put it into the memory buffer, and again start producing items. Whereas the task of the Consumer is to consume the item from the memory buffer.

Concepts:

- **Shared Buffer:** A common space where producers deposit data and consumers retrieve it.
- **Producer Process:** Generates data and writes it to the buffer.
- **Consumer Process:** Reads data from the buffer and consumes it. ● Critical Section: The code section where the buffer is accessed (adding or removing data).
- **Mutual Exclusion:** Only one process can access the critical section at a time, ensuring data consistency.

- **Semaphores:** Integer variables used for signaling and synchronization.
    - **Binary Semaphore:** Can have values 0 or 1. Used for mutual exclusion.
    - **Counting Semaphore:** Can have non-negative integer values. Tracks available resources (e.g., empty buffer slots).

The solution leveraging semaphores involves:

Empty Semaphore: Initialized to the buffer size, indicating the number of empty slots.

Full Semaphore: Initialized to 0, indicating an initially empty buffer.

Producer: Acquires the empty semaphore (waiting if full), adds data to the buffer, and releases the full semaphore.

Consumer: Acquires the full semaphore (waiting if empty), removes data from the buffer, and releases the empty semaphore.

Using semaphores offers several benefits, including ensuring mutual exclusion for buffer access, preventing race conditions and data corruption, and enabling flexible synchronization for multiple producers and consumers

DIVYANSHU MAKHARIA
221070036

CODE:

```
import threading
import time

buffer = [0, 0, 0, 0, 0, 0, 0, 0]
empty = (len(buffer))
full = (0)
mutex = 1 # binary semaphore
i =0
j= 0
def signal(param):
    param+=1
def wait(param):
    param-=1

def producer():
    global i
    while True:
        wait(empty)
        wait(mutex)
        print(f"item {i} is being produced")
        signal(mutex)
        signal(full)
        buffer[i] =1
        print(buffer,"\n")
        i= (i + 1) % len(buffer)
        time.sleep(1)  # Producer sleeps for 1 second

def consumer():
    global j
    while True:
        wait(full)
        wait(mutex)
        print(f"item {j} is being consumed")
        signal(mutex)
```

```
        signal(empty)
        buffer[j] = 0
        print(buffer,"\n")
        j=(j + 1) % len(buffer)
        time.sleep(2)  # Consumer sleeps for 2 seconds

t1=threading.Thread(target=producer)
t2=threading.Thread(target=consumer)
t1.start()
t2.start()

# Printing condition where access is blocked
while True:
    wait(mutex)
    if not full:
        print("Access is blocked: Buffer is empty")
    elif not empty:
        print("Access is blocked: Buffer is full")
    signal(mutex)
    time.sleep(2)  # Check every 2 seconds
```

DIVYANSHU MAKHARIA
221070036

OUTPUT:

```
PS D:\OS_lab> & C:/Users/Divyanshu/AppData/Local/Microsoft/Windo
item 0 is being produced
[1, 0, 0, 0, 0, 0, 0, 0]

item 0 is being consumed
[0, 0, 0, 0, 0, 0, 0, 0]

Access is blocked: Buffer is empty
item 1 is being produced
[0, 1, 0, 0, 0, 0, 0, 0]

item 1 is being consumed
[0, 0, 0, 0, 0, 0, 0, 0]

Access is blocked: Buffer is empty
item 2 is being produced
[0, 0, 1, 0, 0, 0, 0, 0]

item 3 is being produced
[0, 0, 1, 1, 0, 0, 0, 0]

item 2 is being consumed
[0, 0, 0, 1, 0, 0, 0, 0]

Access is blocked: Buffer is empty
item 4 is being produced
[0, 0, 0, 1, 1, 0, 0, 0]

item 5 is being produced
```

```
item 5 is being produced
[0, 0, 0, 1, 1, 1, 0, 0]

item 3 is being consumed
[0, 0, 0, 0, 1, 1, 0, 0]

Access is blocked: Buffer is empty
item 6 is being produced
[0, 0, 0, 0, 1, 1, 1, 0]

item 7 is being produced
[0, 0, 0, 0, 1, 1, 1, 1]

item 4 is being consumed
Access is blocked: Buffer is empty
[0, 0, 0, 0, 0, 1, 1, 1]

item 0 is being produced
[1, 0, 0, 0, 0, 1, 1, 1]

item 1 is being produced
[1, 1, 0, 0, 0, 1, 1, 1]

Access is blocked: Buffer is empty
item 5 is being consumed
[1, 1, 0, 0, 0, 0, 1, 1]
```

CONCLUSION:

Implemented a python program to simulate producer-consumer problem using semaphores.