Practical No 5

Aim: Process Synchronization and Bounded Buffer Problem.

5.1: Stimulate producer-consumer bounded buffer using mutex and semaphores.
5.2: Buffer control with synchronized access.

```
import threading
import time
import random

BUFFER_SIZE = 5
buffer = []

mutex = threading.Lock()
empty = threading.Semaphore(BUFFER_SIZE)  # Starts full (5 slots empty)
full = threading.Semaphore(0)             # Starts empty

running = True  # Stop flag

def producer():
    global running
    while running:
        item = random.randint(1, 100)

        if not empty.acquire(timeout=1):  # Avoid deadlock on exit
            continue

        with mutex:
            buffer.append(item)
            print(f"Produced: {item}, Buffer: {buffer}")

        full.release()
        time.sleep(random.uniform(0.1, 0.5))  # Simulate variable production time

def consumer():
    global running
    while running:
        if not full.acquire(timeout=1):  # Avoid deadlock on exit
            continue

        with mutex:
            if buffer:
                item = buffer.pop(0)
                print(f"Consumed: {item}, Buffer: {buffer}")

        empty.release()
        time.sleep(random.uniform(0.1, 0.5))  # Simulate variable consumption time

if __name__ == "__main__":
    producerThread = threading.Thread(target=producer)
    consumerThread = threading.Thread(target=consumer)

    producerThread.start()
    consumerThread.start()
```

```
  time.sleep(10)  # Let the simulation run for 10 seconds
  running = False

  # Unblock threads if they're waiting
  empty.release()
  full.release()

  producerThread.join()
  consumerThread.join()

  print("Simulation Finished")
```

```
Produced: 13, Buffer: [13]
Consumed: 13, Buffer: []
Produced: 7, Buffer: [7]
Consumed: 7, Buffer: []
Produced: 35, Buffer: [35]
Consumed: 35, Buffer: []
Produced: 66, Buffer: [66]
Consumed: 66, Buffer: []
Produced: 8, Buffer: [8]
Consumed: 8, Buffer: []
Produced: 37, Buffer: [37]
Consumed: 37, Buffer: []
Produced: 58, Buffer: [58]
Consumed: 58, Buffer: []
Produced: 64, Buffer: [64]
Produced: 70, Buffer: [64, 70]
Consumed: 64, Buffer: [70]
Produced: 78, Buffer: [70, 78]
Consumed: 70, Buffer: [78]
Consumed: 78, Buffer: []
Produced: 26, Buffer: [26]
Consumed: 26, Buffer: []
Produced: 96, Buffer: [96]
Consumed: 96, Buffer: []
Produced: 75, Buffer: [75]
Consumed: 75, Buffer: []
Produced: 59, Buffer: [59]
Consumed: 59, Buffer: []
Produced: 74, Buffer: [74]
Consumed: 74, Buffer: []
Produced: 9, Buffer: [9]
Consumed: 9, Buffer: []
Produced: 16, Buffer: [16]
Consumed: 12, Buffer: [30, 44]
Produced: 86, Buffer: [30, 44, 86]
Consumed: 30, Buffer: [44, 86]
Produced: 10, Buffer: [44, 86, 10]
Produced: 99, Buffer: [44, 86, 10, 99]
Consumed: 44, Buffer: [86, 10, 99]
Consumed: 86, Buffer: [10, 99]
Produced: 13, Buffer: [10, 99, 13]
Consumed: 10, Buffer: [99, 13]
Consumed: 99, Buffer: [13]
Produced: 75, Buffer: [13, 75]
Produced: 74, Buffer: [13, 75, 74]
Consumed: 13, Buffer: [75, 74]
Simulation Finished
```

5.3: Circular queue technique(bounded buffer)

```
import threading
import queue
import time
import random

BUFFER_SIZE = 5
buffer = queue.Queue(BUFFER_SIZE)
running = True
SENTINEL = None

def producer():
    global running
    while running:
        item = random.randint(1, 100)
        buffer.put(item)
        print(f"Produced: {item}")
        time.sleep(random.uniform(0.1, 0.5))  # Simulate some delay

    # Send sentinel value to stop the consumer
    buffer.put(SENTINEL)

def consumer():
    while True:
        item = buffer.get()
        if item is SENTINEL:
            buffer.task_done()
            break
        print(f"Consumed: {item}")
        buffer.task_done()
        time.sleep(random.uniform(0.1, 0.5))  # Simulate processing delay

if __name__ == "__main__":
    producerThread = threading.Thread(target=producer)
    consumerThread = threading.Thread(target=consumer)

    producerThread.start()
    consumerThread.start()

    # Let it run for a while
    time.sleep(10)
    running = False  # Signal the producer to stop

    producerThread.join()
    consumerThread.join()

    print("Simulation Finished")
```

```
Produced: 28
Consumed: 28
Produced: 59
Consumed: 59
Produced: 51
Consumed: 51
Produced: 50
Produced: 93
Consumed: 50
Consumed: 93
Produced: 11
Consumed: 11
Produced: 92
Consumed: 92
Produced: 25
Consumed: 25
Produced: 9
Consumed: 9
Produced: 25
Consumed: 25
Produced: 58
Consumed: 58
Produced: 88
Consumed: 88
Produced: 25
Consumed: 25
Produced: 83
Consumed: 83
Produced: 33
Consumed: 33
Produced: 44
Consumed: 44
Produced: 74
Consumed: 74
Produced: 97
Consumed: 97
Produced: 11
Consumed: 11
Produced: 80
Consumed: 80
Produced: 29
Consumed: 29
Produced: 43
Consumed: 43
Produced: 78
Consumed: 78
Produced: 98
Consumed: 98
Produced: 46
Consumed: 46
Produced: 96
Consumed: 96
Produced: 83
Consumed: 83
Produced: 26
Consumed: 26
Simulation Finished
```