# Practical No 1

## Aim: Process Communication using Shared Memory.

**1.1: Understand shared memory concepts in inter-process communication**
**1.2:Implement producer-consumer synchronization using shared memory and semaphores**
**1.3: Explore issues of race conditions and how to avoid them**

```
import threading
import time
import random

# Shared Memory Variables
capacity = 5
logBuffer = ["-EMPTY-" for _ in range(capacity)]
inIndex = 0
outIndex = 0

# Declaring Semaphores
mutex = threading.Semaphore(1)        # Ensure mutual exclusion
empty = threading.Semaphore(capacity)  # Count of empty slots
full = threading.Semaphore(0)         # Count of filled slots

# Sample log messages for simulation
sampleLogs = [
    "User login detected",
    "File uploaded successfully",
    "System warning: Low memory",
    "Connection timeout",
    "New user registration",
    "Payment transaction complete",
    "Background job started",
    "Error: Invalid input received",
    "Database backup completed",
    "Security alert: Multiple failed logins"
]

# Producer Thread Class - Simulates log creation
class LogProducer(threading.Thread):
    def run(self):
        global capacity, logBuffer, inIndex
        global mutex, empty, full
        logsProduced = 0

        while logsProduced < 20:
            empty.acquire()        # Wait for an empty slot
            mutex.acquire()        # Lock the buffer

            logMessage = random.choice(sampleLogs)
            logBuffer[inIndex] = logMessage
            inIndex = (inIndex + 1) % capacity
            print(f"[Producer] Generated Log: {logMessage}")

            mutex.release()        # Unlock the buffer
```

```
            full.release()          # Signal that buffer has a new item
            time.sleep(random.uniform(0.5, 1.5))  # Simulate variable log generation time
            logsProduced += 1

# Consumer Thread Class - Simulates log processing
class LogConsumer(threading.Thread):
    def run(self):
        global capacity, logBuffer, outIndex
        global mutex, empty, full
        logsConsumed = 0

        while logsConsumed < 20:
            full.acquire()          # Wait for a full slot
            mutex.acquire()         # Lock the buffer

            logMessage = logBuffer[outIndex]
            logBuffer[outIndex] = "-EMPTY-"
            outIndex = (outIndex + 1) % capacity
            print(f"[Consumer] Processed Log: {logMessage}")

            mutex.release()         # Unlock the buffer
            empty.release()         # Signal that a slot is now empty
            time.sleep(random.uniform(1.0, 2.0))  # Simulate variable processing time
            logsConsumed += 1

# Create Threads
producerThread = LogProducer()
consumerThread = LogConsumer()

# Start Threads
consumerThread.start()
producerThread.start()
ra
# Wait for Threads to Finish
producerThread.join()
consumerThread.join()
```

```
[Producer] Generated Log: Background job started
[Consumer] Processed Log: Background job started
[Producer] Generated Log: Payment transaction complete
[Consumer] Processed Log: Payment transaction complete
[Producer] Generated Log: Payment transaction complete
[Producer] Generated Log: Security alert: Multiple failed logins
[Consumer] Processed Log: Payment transaction complete
[Producer] Generated Log: Database backup completed
[Consumer] Processed Log: Security alert: Multiple failed logins
[Producer] Generated Log: Background job started
[Consumer] Processed Log: Database backup completed
[Producer] Generated Log: Security alert: Multiple failed logins
[Producer] Generated Log: Error: Invalid input received
[Consumer] Processed Log: Background job started
[Producer] Generated Log: User login detected
[Consumer] Processed Log: Security alert: Multiple failed logins
[Producer] Generated Log: File uploaded successfully
[Consumer] Processed Log: Error: Invalid input received
[Consumer] Processed Log: User login detected
[Consumer] Processed Log: File uploaded successfully
```