PUNJAB ENGINEERING COLLEGE (DEEMED TO BE UNIVERSITY) CHANDIGARH

Assignment 6

Submitted By :

Sugam Arora
SID : 21105021
Branch : ECE

Date : 30th March, 2025

1. Make a queue of user-defined length. Generate two threads, one would generate a random number and enqueue it. The second thread would help in dequeuing the same and printing those values.

2. In computing, the producer-consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, which share a common, fixed-size buffer used as a queue.

   - The producer's job is to generate data, put it into the buffer, and start again.
   - At the same time, the consumer is consuming the data (i.e., removing it from the buffer), one piece at a time.

Bash Script

Q1. Producer Consumer

```
#!/bin/bash

# ================================
# Q1: Producer-Consumer Simulation
# Author: Sugam Arora
# Description: Simulates a queue with two threads using Bash.
# ================================

QUEUE_FILE="shared_queue.txt"
> "$QUEUE_FILE"  # Clear previous data

# Ask user for queue size
read -p "Enter the maximum queue size: " QUEUE_SIZE

# Colors for output
GREEN="\033[0;32m"
RED="\033[0;31m"
NC="\033[0m"  # No color

# Producer Function
producer() {
    while true; do
        local size
```

```
        size=$(wc -l < "$QUEUE_FILE")
        if [ "$size" -lt "$QUEUE_SIZE" ]; then
            local value=$((RANDOM % 100))
            echo "$value" >> "$QUEUE_FILE"
            echo -e "${GREEN}[Producer] $(date +"%T") - Enqueued: $value${NC}"
        fi
        sleep 1
    done
}

# Consumer Function
consumer() {
    while true; do
        if [ -s "$QUEUE_FILE" ]; then
            local value
            value=$(head -n 1 "$QUEUE_FILE")
            sed -i '1d' "$QUEUE_FILE"
            echo -e "${RED}[Consumer] $(date +"%T") - Dequeued: $value${NC}"
        fi
        sleep 2
    done
}

# Start processes
producer &
consumer &

# Keep script running
wait
```

Q2. Buffer Problem

```
#!/bin/bash

# ==================================
# Q2: Bounded Buffer Problem (Bash Sim)
# Author: Sugam Arora
# Description: Simulates fixed-size buffer for producer-consumer using arrays.
# ==================================

BUFFER_SIZE=5
BUFFER=()
```

```
# Colors
YELLOW="\033[1;33m"
BLUE="\033[1;34m"
NC="\033[0m"

produce() {
    while true; do
        if [ "${#BUFFER[@]}" -lt "$BUFFER_SIZE" ]; then
            item=$((RANDOM % 1000))
            BUFFER+=("$item")
            echo -e "${YELLOW}[Producer] $(date +"%T") - Produced: $item | Buffer Size:
${#BUFFER[@]}${NC}"
        fi
        sleep 1
    done
}

consume() {
    while true; do
        if [ "${#BUFFER[@]}" -gt 0 ]; then
            item="${BUFFER[0]}"
            BUFFER=("${BUFFER[@]:1}")
            echo -e "${BLUE}[Consumer] $(date +"%T") - Consumed: $item | Buffer Size:
${#BUFFER[@]}${NC}"
        fi
        sleep 2
    done
}

# Run in background
produce &
consume &

# Wait forever
wait
```

Python Script

Q1. Producer Consumer

```
# ================================
# Q1: Producer-Consumer Queue Simulation
# Author: Sugam Arora
# Description: Creates a queue with two threads — one to enqueue, one to dequeue.
# ================================

import threading
import queue
import random
import time
from datetime import datetime

# Colored terminal output
def log(msg, color="default"):
    colors = {
        "green": "\033[92m",
        "red": "\033[91m",
        "blue": "\033[94m",
        "default": "\033[0m"
    }
    print(f"{colors.get(color, colors['default'])}[{datetime.now().strftime('%H:%M:%S')}]
{msg}{colors['default']}")

# Input queue size
queue_size = int(input("Enter the maximum queue size: "))
q = queue.Queue(maxsize=queue_size)

# Producer thread function
def producer():
    while True:
        if not q.full():
            item = random.randint(1, 100)
            q.put(item)
            log(f"Produced: {item}", "green")
        time.sleep(1)

# Consumer thread function
def consumer():
    while True:
        if not q.empty():
```

```
            item = q.get()
            log(f"Consumed: {item}", "red")
        time.sleep(2)

# Launch threads
producer_thread = threading.Thread(target=producer)
consumer_thread = threading.Thread(target=consumer)

producer_thread.start()
consumer_thread.start()

producer_thread.join()
consumer_thread.join()
```

Q2. Buffer Problem

```
# ================================
# Q2: Bounded Buffer Simulation
# Author: Sugam Arora
# Description: Classic producer-consumer using a fixed-size buffer.
# ================================

import threading
import time
import random
from datetime import datetime

BUFFER_SIZE = 5
buffer = []
lock = threading.Lock()

# Terminal logging with color
def log(msg, color="default"):
    colors = {
        "yellow": "\033[93m",
        "blue": "\033[94m",
        "default": "\033[0m"
    }
    print(f"{colors.get(color, colors['default'])}[{datetime.now().strftime('%H:%M:%S')}]
{msg}{colors['default']}")

def producer():
```

```
    while True:
        with lock:
            if len(buffer) < BUFFER_SIZE:
                item = random.randint(100, 999)
                buffer.append(item)
                log(f"Producer produced: {item} | Buffer: {buffer}", "yellow")
        time.sleep(1)

def consumer():
    while True:
        with lock:
            if buffer:
                item = buffer.pop(0)
                log(f"Consumer consumed: {item} | Buffer: {buffer}", "blue")
        time.sleep(2)

# Start both threads
threading.Thread(target=producer).start()
threading.Thread(target=consumer).start()
```