National University of Computer and Emerging Sciences

Operating Systems Lab

Lab Manual 10

Muhammad Hassan Raza

Fall 2024

Department of Software Engineering
FAST-NU, Lahore, Pakistan

Q1: Write C++/C code to implement a system with three producer threads and two consumer threads. Each producer generates random integers and writes them into a shared bounded buffer that can hold up to 10 integers. The consumers read from this buffer and print each integer, ensuring the buffer does not exceed capacity and that items are consumed in the order they were produced.

Each producer should stop after collectively generating 50 integers. Once all items are consumed, the program should free any resources and exit.

Constraints:

- Use semaphores exclusively for synchronization.
- Ensure no thread reads or writes outside buffer bounds.

Sample Workflow:

1. **Producer Threads**:
    - **Producer 1** generates an integer (e.g., 15) and writes it into the buffer.
    - **Producer 2** generates another integer (e.g., 42) and adds it next.
    - **Producer 3** generates an integer (e.g., 9) and adds it next.
2. **Consumer Threads**:
    - **Consumer 1** retrieves and prints the first integer (15), removing it from the buffer.
    - **Consumer 2** retrieves and prints the next integer (42).
3. **Process Continues**:
    - Producers and consumers operate concurrently, each waiting as necessary to avoid buffer overflows or reading from an empty buffer.
    - Once 50 integers are generated and consumed, the program exits.

Q2: Imagine a logistics company where every shipment must be verified before invoicing. Write a C++/C program that models this with two synchronized threads using semaphores.

1. **Verifier Thread:** Continuously generates a unique shipment ID (simulating verification) and stores it in a shared memory array representing verified shipments ready for invoicing.
2. **Invoice Processor Thread:** Retrieves each shipment ID from the shared array, removes it, calculates a simulated "invoice" (e.g., multiplying the ID by a fixed rate), and prints both the ID and calculated invoice amount.

The program should stop after processing 10 shipments, after which all resources are freed, and the program exits.

Constraints:

- Use only semaphores for synchronization.
- Ensure each ID is processed once, and both threads access the array safely.

Sample Workflow:

1. **Verifier Thread**:
    - Verifies and generates a shipment ID, say 1001, and adds it to the array.
    - Generates the next shipment ID, 1002, and stores it once the processor removes the previous ID.
2. **Invoice Processor Thread**:
    - Reads 1001 from the array, removes it, calculates the invoice (e.g., 1001 * rate = 15015), and prints:
      Shipment ID: 1001, Invoice: $15015
    - Proceeds to the next available ID, 1002, and repeats the process.
3. **Completion**:
    - After 10 shipments are processed, both threads stop, and the program exits.

This simulates a real-life shipment processing pipeline where synchronization ensures that no shipments are missed or double-processed between verification and invoicing.