National University of Computer and Emerging Sciences

Laboratory Manual

for

Operating Systems Lab

Department of Computer Science

FAST-NU, Lahore, Pakistan

Objectives:
In this lab, students will practice process synchronization using semaphores

Question 1.

Write C++/C code for a program that takes as command line argument a source file name. The program acts as a producer and reads each time 20 character from file and writes the characters to a shared memory buffer created by the producer. The shared memory buffer can store at most 20 characters. Now there is another program which acts as a consumer, attaches the shared memory to its address space, and reads those 20 characters from shared memory, prints the characters on the screen, and waits for the user to press enter key. The producer can only write the next 20 character of file to shared memory after the consumer has read the previous 20 characters from the shared memory. So you need to synchronize the two processes using semaphores. You are not allowed to synchronize the processes any other way except using semaphores. When the file has been complete written to the shared memory, the producer writes a $ to shared memory which indicates to the consumer that file has been finished. After that all resources must be freed. (Assume the file's size is always multiple of 20, such as 20, 40, 60, etc.)

For example: Suppose there is a file of 40 characters.

Producer writes 20 characters, and waits for the consumer to read these 20 characters.

Consumer reads 20 characters and prints on the screen.

Producer now writes the next 20 characters and waits for the consumer to read these 20 characters.

The consumer reads the 20 characters and prints the data on the screen

The end-of-file has been reached, so the producer writes $ to the shared memory. Then the producer detaches all shared memory portions from its address space and deletes the shared memory portions and exits.

The consumer reads $, which indicates that the file's data has completely been read. So the consumer detaches all shared memory from its address space and exits.

Question 2: You are tasked with implementing a resource management system for a print shop using semaphores. In this shop, there are multiple printing machines and a limited number of ink cartridges.

Requirements:

- There are 3 printing machines (Machine 1, Machine 2, Machine 3).
- There are 2 ink cartridges that can be shared among the machines.
- Use a binary semaphore to control access to the printing machines (ensuring only one machine can print at a time).
- Use a counting semaphore to manage the ink cartridges (allowing up to 2 machines to print concurrently, provided they have access to an ink cartridge).

Task:

1. Implement a PrintShop class that includes:
    - A binary semaphore for machine access.
    - A counting semaphore for ink cartridges.
    - A print method that simulates a machine trying to print. The method should:
        - Acquire the binary semaphore before printing.
          Check if an ink cartridge is available; if so, acquire it and simulate printing (e.g., using time.sleep).
        - Release the ink cartridge after printing.
        - Release the binary semaphore once printing is complete.
2. Simulate the operation of the print shop with multiple threads representing the printing machines trying to print at the same time.

Question 3:

You are tasked with solving the classic "Dining Philosophers" problem using semaphores. In this scenario, five philosophers are sitting around a table and need to eat spaghetti. Each philosopher requires two forks to eat, and there is one fork between each pair of philosophers.

Requirements:

- Implement a DiningPhilosophers class that manages the philosophers and their access to forks using semaphores.
- There are 5 philosophers and 5 forks. Each fork can be represented as a semaphore initialized to 1 (indicating it's available).
- Each philosopher will alternate between thinking and eating, and each eating session requires acquiring both forks.

Task:

1. Implement the DiningPhilosophers class with:
    - An array of semaphores for the forks.

    - A `wantToEat` method that allows a philosopher to pick up the forks and eat.
    - A `doneEating` method that allows the philosopher to put down the forks after eating.
2. Simulate the philosophers' actions using multiple threads. Each philosopher should:
    - Think for a random amount of time.
    - Try to pick up the forks and eat.
    - After eating, put down the forks.