# National University of Computer and Emerging Sciences


## Laboratory Manual # 08
## Operating Systems

| | |
|---|---|
| Course Instructor | Mubashar Hussain |
| Lab Instructor | Muhammad Hashir Mohsineen |
| Section | BCS-4E |
| Date | 08-April-2025 |
| Semester | Spring 25 |

## Instructions:

- Submit a world/LibreOffice file containing screenshots of terminal commands/ Output
- Submit your .c (Code files)
- In case of any explanation you can add a multiline comment.

## Objectives:

- Mutex
- Semaphores
- Thread synchronization using semaphores

---

**Reading material:**
https://docs.google.com/document/d/1OvZZ-MAkXwX8xKyqZh4ay4uPhf_Q2yReEAwARfOOgAA/edit?usp=sharing

# 1. Exercise: [10]

Implement a multi-threaded ticket booking system where multiple customers (threads) try to book tickets concurrently. The system should ensure that:

1. No two threads book the same ticket (avoid race conditions).
2. Only available tickets can be booked (prevent overbooking).
3. The booking process is thread-safe using mutex locks.

Details:

- Use an array to store available tickets.
- Each thread represents a customer trying to book a ticket.
- Use a mutex (pthread_mutex_t) to ensure mutual exclusion.
- If tickets are available, the customer books one ticket and updates the system.
- If no tickets are left, the customer is informed that booking is not possible.

Test your program with 15 customers (threads) and 10 tickets.
[read the example in reading material for better understanding]

# 2. Exercise: [10]

We have two kinds of threads, the writer thread and the reader thread. The writer thread asks the user for input, and writes the input to shared memory. The reader thread then reads the data from shared memory and prints the data on the screen. Write a program which has 1 writer thread and 10 reader threads. The program will do the following:

1. The writer takes data from the user and writes it to shared memory
2. 10 readers then read the data written to shared memory by the writer. However, all readers will not read in parallel, they will read the data from shared memory according to the parameter passed by the user (through command line). For example if user passed

1, then thread 1 will read the data first. While thread 1 is reading no other thread is allowed to read. After thread 1 has finished reading, thread 2 will read. After thread 2 has finished reading, thread 3 will read. The order in which threads read does not matter.

3. When all threads have finished reading, the writer will execute, take data from the user, and put it to shared memory. After that, step 2 will occur. (So, there is a loop in writer and reader thread, and both readers and writers will run infinitely).

[Hint: use counting semaphore]

## Practice task:

Solve the Dining Philosophers problem using mutexes

1. Write a program where five philosophers (threads) alternately think and eat.
2. Use mutexes to ensure that no two philosophers can eat simultaneously using the same fork.
3. Avoid deadlock and ensure fairness among philosophers.

Details:

- Define the number of philosophers (e.g., 5). Create an array of mutexes representing forks (one for each philosopher).
- Each philosopher should alternately think and eat.
- When a philosopher wants to eat, they must pick up the left fork first and then the right fork (or vice versa) using mutexes.
  – Avoid Deadlock:
- To avoid deadlock, ensure that the philosophers pick up both forks in a consistent order (e.g., always pick up the lower-numbered fork first).
- Alternatively, use a timeout mechanism to release the fork if the second fork cannot be acquired within a certain period.
  – Philosopher Threads:
- Create a thread for each philosopher.
- Implement the actions for each philosopher in a loop where they think, try to pick up forks, eat, and then put down the forks.
- Ensure that all philosophers get a chance to eat by alternating between thinking and eating, and by using appropriate synchronization mechanisms (e.g., a semaphore or mutex) to control access to the forks.

---