National University of Computer and Emerging Sciences

Laboratory Manual # 10
Operating Systems

Course Instructor	Mubashar Hussain
Lab Instructor	Muhammad Hashir Mohsineen
Section	BCS-4E
Date	22-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:

- Process Synchronization
- Shared memory / memory mapping

Learning Material:

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

---

1. Exercise: [00]

(solved…. Read and practice from learning material document)

Synchronize two processes so they alternately print "ping" and "pong" using shared memory or named semaphores.

- Use shm_open and mmap to share synchronization state or
- Create two named semaphores: sem_ping and sem_pong.
- Create two child processes using fork():
  - One prints "ping" when sem_ping is available.
  - The other prints "pong" when sem_pong is available.
- Alternate printing between them exactly 20 times each, with correct order.
- Ensure correct synchronization without busy-waiting. Cleanup shared memory and semaphores properly.

2. Exercise: [10]

Implement a shared counter that is safely incremented by multiple processes using semaphores.

- Use shm_open, and mmap to create and access a shared memory region.
- Use a named semaphore (sem_open) to protect access to the shared counter.
- Spawn 5 child processes using fork(). Each child will increment the counter 100,000 times.
- The parent process should wait for all child processes and print the final value of the counter.
- The final counter value should be 500,000.

- Ensure that race conditions are avoided via semaphore synchronization.

3. Exercise: [10]

Simulate a resource allocation table shared among processes and synchronized using semaphores.

- Create a shared resource table in memory with 5 slots (e.g., representing 5 printers).
- Multiple processes (e.g., 3 children) request and release resources.
- Each process randomly tries to allocate and release a resource.
- Use one binary semaphore (mutex) to control access to the table.
- Simulate at least 10 resource requests per process.
- No two processes should hold the same resource simultaneously.
- The shared table should consistently reflect resource allocation.

4. Exercise: [10]

Implement safe concurrent access to a file mapped to memory, where multiple processes write to it without corrupting data.

- Use mmap to map a file (e.g., output.txt) into shared memory.
- Create a named semaphore to control access to the file.
- Use fork() to spawn 3 processes.
- Each process writes a specific string (e.g., "Process 1\n", "Process 2\n") 50 times to the file using mapped memory.
- Synchronize writes using a semaphore to avoid overlap.
- The output file should have interleaved (but not overlapping) clean lines from each process.
- No data corruption, mixed lines, or missing content.