**Operating Systems Project**

**Application of a Linux-based system-level multicore programming (Threading & Process Management)**

**Instructor: Muhammad Monis**

**Section: BCS-5B**

**Group Members:**

**Syed Ali Ahmed 22K-4681**

**Osaid Ur Rehman 22K-4763**

**Zeerak Shahzad 22K-4692**

**1. Introduction:**

Multithreading and inter-process communication are essential concepts in computer programming, especially in systems programming. This project aims to demonstrate the practical implementation of multithreading and processes in a C program. By utilizing semaphores, mutexes, and sockets, the project showcases how to coordinate and synchronize multiple threads and processes effectively.

**2. Objectives:**

- Implement a C program that involves multiple processes and threads.

- Demonstrate the use of semaphores and mutexes for synchronization.

- Showcase inter-process communication using sockets.

- Ensure proper coordination between threads and processes to achieve desired execution order and resource management.

**3. Project Details:**

The project comprises three main files: `a2.c`, `a2\_helper.c`, and `a2\_helper.h`.

**- a2.c:** This file contains the main program logic. It orchestrates the creation of multiple processes and threads, each with specific tasks and synchronization requirements. Process creation, thread creation, synchronization, and resource management are handled within this file.

**- a2\_helper.c:** This file implements helper functions for inter-process communication and logging. It utilizes sockets for communication between processes and a semaphore for mutual exclusion when accessing shared resources. Additionally, it provides functions to handle forking in multi-threaded applications safely.

**- a2\_helper.h:** This header file declares the functions implemented in `a2\_helper.c`, providing an interface for other source files to utilize.

**4. Results:**

The project successfully demonstrates the coordination and synchronization of multiple processes and threads. Through detailed logging, it provides insights into the execution flow, showing the proper order of actions taken by each process and thread. The use of semaphores and mutexes ensures that critical sections are properly protected, preventing race conditions and ensuring data integrity.

**5. Conclusion:**

In conclusion, the project achieves its objectives of showcasing multithreading and inter-process communication in C. By leveraging semaphores, mutexes, and sockets, it demonstrates effective synchronization and coordination between multiple threads and processes. The detailed logging provides clarity on the execution sequence and highlights the importance of proper synchronization in concurrent programming. Overall, the project serves as a practical example of implementing multithreading and processes in C for systems programming applications.