Operating Systems – Lab 2 Report

This lab explored process creation and management using the **fork()** system call in C. Three programs (**prog1**, **prog2**, and **my3proc**) were compiled and executed. The experiments demonstrated how parent and child processes interact, how their outputs interleave, and how the operating system scheduler determines execution order.

Lab Questions & Answers

- **What does fork() return?**
  In the child process: 0. In the parent process: the child's PID (>0). On error: -1.
- **How do you tell parent vs child?**
  By checking the return value of fork(). 0 → child, >0 → parent.
- **Why does the output order change each run?**
  Because parent and child execute concurrently. The OS scheduler interleaves them nondeterministically.
- **Why do variables look identical right after fork()?**
  The child receives a copy of the parent's address space (copy-on-write). They start identical but become independent after.
- **Why use write() in prog1 instead of printf()?**
  printf() is buffered, so outputs from different processes can clump or reorder. write() is unbuffered, making the interleaving visible.
- **What does wait()/waitpid() do?**
  It blocks the parent until a child process exits and provides its exit status (use WEXITSTATUS(status) to decode).

In conclusion, this lab demonstrated how **fork()** creates new processes, how their execution is interleaved by the operating system, and how **wait()** ensures the parent can synchronize with child termination. The hands-on experiments with prog1, prog2, and my3proc reinforced key concepts of process management in Unix-like systems.