Introduction

This lab helped me understand how to create and manage processes in a Linux system using C. The goal was to simulate how a parent process spawns multiple children that each run their own task. I used three main system calls to do this: `fork()` to create new processes, `execvp()` to run commands inside those processes, and `wait()` to make sure the parent process stays in control and waits for the children to finish. This gave me practical experience with process handling and system-level programming.

Implementation Summary

The program begins by storing 10 Linux commands in a 2D array. These include `ls`, `whoami`, `date`, and one that prints "Hello Mariatu" using the `echo` command. A `for` loop is used to create 10 child processes using `fork()`.

Each child prints its PID and the command it's about to run, then calls `execvp()` to execute the command. If the command fails, an error is printed using `perror()`. Meanwhile, the parent uses `wait()` to wait for each child and prints whether it exited normally using macros like `WIFEXITED()` and `WEXITSTATUS()`.

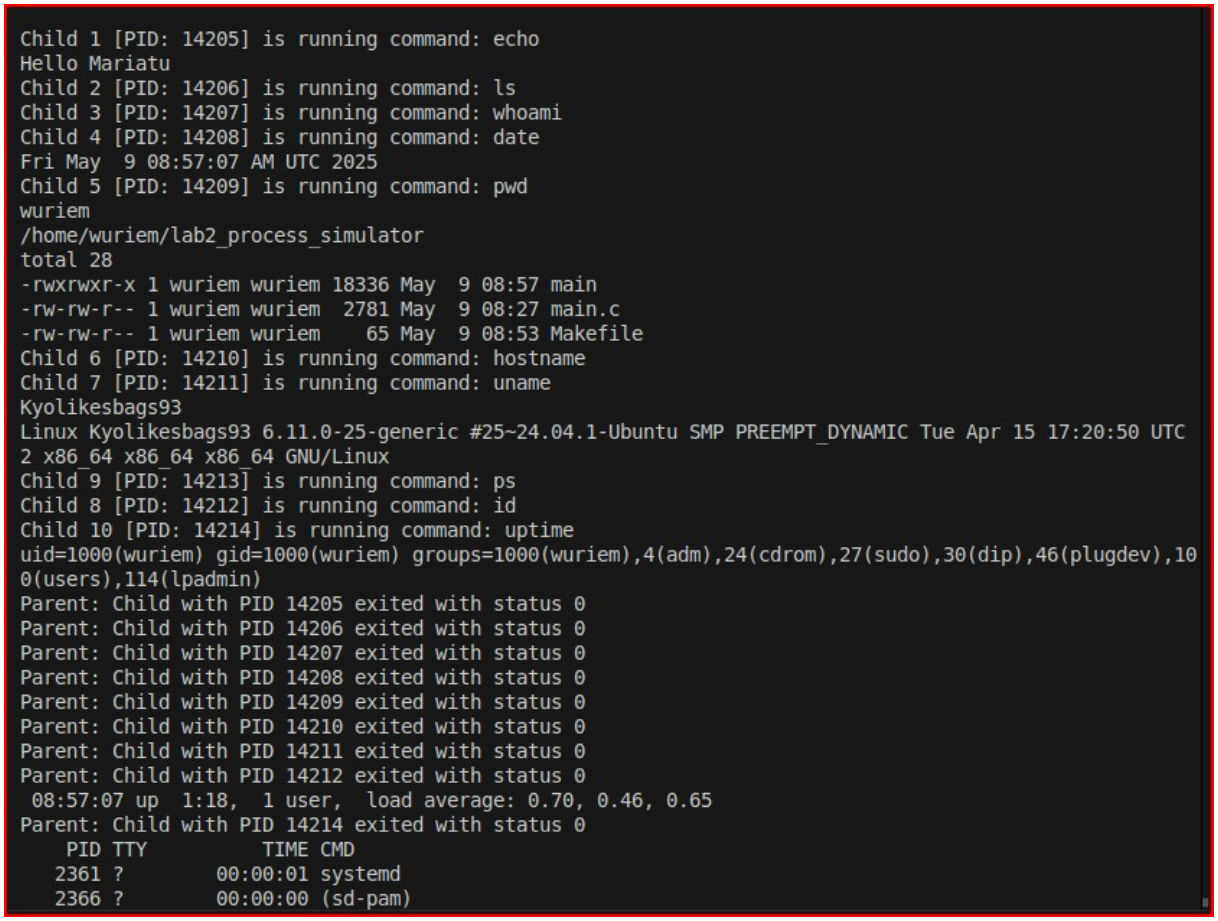
Results and Observations

A. Process Creation and Management

Processes were created using `fork()` inside a loop. Each successful child process replaced its own memory using `execvp()` to run the assigned command. The parent kept track of child termination using `wait()` and interpreted the exit codes using predefined macros.

B. Parent and Child Interaction

The parent controlled the overall flow by initiating the children and waiting for them to complete. Each child process operated independently but reported back indirectly by exiting. The parent used this to log process completion and confirm that each command ran as expected.



As seen in the output above, each child process printed its own message, and the parent reported status messages as each child exited. The `echo` command correctly printed "Hello Mariatu," and all commands completed with exit status 0.

Conclusion

This lab helped me connect theory with actual system programming. I now understand how operating systems manage multiple processes and how `fork()`, `execvp()`, and `wait()` work together to make it possible. Writing this program made me more confident in using system calls in C and showed me how real programs can manage multiple tasks efficiently. This will be useful in future classes or jobs where multitasking and process control are required.