Alex Boyce

Lab 2 Report

**Introduction**

The purpose of this lab was to learn process creation and management in C using system calls from the operating system. The goal was to create a parent process that spawns 10 child processes, each executing a unique Linux command. To accomplish this, the requirements of the lab meant that my solution had to use fork(), execvp(), and wait() to control and synchronize processes.

**Implementation**

My C program begins by defining an array of command argument lists, each representing a distinct Linux command.

A screenshot of a computer program

AI-generated content may be incorrect.

The parent process iterates through this array and creates a new child process for each command using fork(). In the child process, the command is executed using execvp(). If execvp() fails, an error is reported, and the child exits. A computer screen shot of white text

AI-generated content may be incorrect.

The parent process uses wait() in a loop to collect and report the status of each terminated child process.

A computer screen shot of white text

AI-generated content may be incorrect.

**Results and Observations**

A. Explain how processes were created and managed.

Processes were created using fork(). Each child process ran independently and executed a unique command via execvp(). The parent process maintains control by looping through wait() calls, ensuring its aware of when and how each child process exited by exit code.

B. Describe how the parent and child processes interacted.

The only direct communication between the parent and child processes was parent’s tracking of each child process.

**Conclusion**

This lab introduced me to process management in C. By implementing multiple child processes each running a distinct command, the program was able to control concurrent execution in the Linux operating system. The use of fork(), execvp(), and wait() in this lab grants some foundational knowledge and an example for use cases for OS-level programming.