Vicente Garcia

CS 470

Professor Zhu

February 10, 2025

Introduction

The goal of this lab was to simulate process management in an operating system. A process is program execution that must execute in a sequential function, and these processes can be scheduled to execute through a queue. There are also parent processes that can have child processes using the fork method, which can let the children execute a function program of their own. This child process is also a duplicate of the parent process and can have both the children and the parent execute together, or the parent waits, which is being done for this lab. The programming language C was used with many of its libraries that deal with process execution to simulate the execution of processes.

Implementation

In In this lab, the C libraries of unistd.h, wait/syst.h, stdlib.h, and sys/signal.h functions are used to be able to create this process management simulation. First, the pid_t pid variable to store process state information and the commands to execute were put into a 2D array using null as the way to terminate these commands. There was also a constant variable used to only create 10 child processes using the fork function and a for loop that terminated when it reached 10 child processes. When you create a child process, the number that the fork method returns tells you what state the child process is in. If it returns a negative, it failed; if it returns 0, it means it is in the child process, which then calls the function to execute a command using execvp, and if it is a positive, it is in the parent process. When the program reaches the parent process state, the wait and status methods are used to tell the parent to wait for the child process and tell if the child process is done executing. After the while loop is done executing, that means the parent process is done waiting for all the child processes to finish executing, and so the parent is also done executing.

Results and Observations

A. Explain how processes were created and managed?

The parent process is already the process that is running the code execution of the program. However, the way that the children processes are created for this program is using the fork method, which would return a certain value to let us know if it is in the child or parent process. The parent process would use the wait method to manage the child process and allow them to execute before taking over control again.

B. Describe how the parent process and children interacted?

With this program, it seems that the way the children process, and the parent interacts is by letting the child process the terminal operations like echo and ls to finish executing first, then file and directory manipulation commands, and the last to finish executing were commands that dealt with during calculations like the date and time commands. Therefore, it seems that the less the CPU must work, the faster the children will be done and give control back to the parent process.

Conclusion

This lab showed me the way that it seems in process management, at least for Linux: the easier the process is to do, the faster it will be done executing. Also, the parent process in a program is the program that you are on, and the child processes are created through the fork function in the program. When it comes to commands, when children processes are given, they do not execute in a sequential fashion, and instead, it is decided by the system scheduler. A place that I could improve would be running commands in a better order because some commands finished executing, resulting in some errors. Also, an area I would like to experiment further with is running the same commands but with more options to see how the process scheduler deals with the execution of those commands.