Hieu Luong

CS 470

Feb 9, 2025

Project management is a fundamental concept in operating systems that allows multitasking by building and controlling many processes. In this lab project, we are required to build a C program that shows the process creation and management using system calls such as fork(), execvp(), and wait(). In this program, it created multiple child processes. In this project, we need to make at least 10 child processes. More than that, this project delivers insight into how a parent process spawns and manages child processes in a Unix-based environment.

Now, let us summarize how we are implementing this lab. The program shows an array containing ten different shell commands; we can do more than that, but at least 10 are required. It helps to execute the separated child process. Following the code, the parent process creates these child processes using the fork() system call. Also, it helps to replace the child's execution context with the respective command using the execvp(). The parent process, after that, uses the waits for all the child processes to be completed using the wait() function to ensure the lab meets the proper requirements.

There are four key components to this program. The **process Creation** used the function fork() to create the child processes. Next, **command Execution,** which has the function of the execvp(), which replaces the child process imaged with the specified command. The third one will be the **Synchronization process, which has** the function of waiting for all the child processes to complete using the wait. Last but not least, the **error handling** which we make the program to handle the potential error in those two functions, fork() and execvp(), to ensure robustness.

Next, there are two results that I can see for the project: **process creation and management** and **parent and child process interaction,** which I can see in this lab. First of all, the program successfully creates a total of ten child processes using fork(), more than that, each executing a predefined command, for example: “date”, “us”, “echo”,... Every child for that prints its PID before executing the command. The function we are required to use is the execvp(), which helps to replace the child process image with the specified command, allowing it to execute within the child process. More than that, if the execvp() fails, an error message will be printed, and the child process exits with the failure status for that.

Furthermore, the next part is about the interaction between the child and the parent. We can have in the project the parent process prints its PID before creating the child processes, which for the output we can see at first. After spawning all the child processes, the parent process enters a loop where they wait for each child to finish using the wait() function. Lastly, the parent process collects and prints the exit status of each child of the process, if they are terminated normally or due to a signal. I have three outputs for that so that if there are problems, it will show the developer.

In conclusion, this project shows the fundamental process management. By creating and executing multiple processes, we can explore inter-process communication and synchronization using the system call of the function. The implementation provided valuable insights into how operating systems handle all the process execution, and we can do it the understanding process control in C