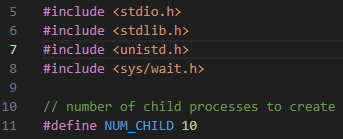
Alex Ryse

CS 470 Dr. Zhu

10/21/2025

Lab 2 Project Report

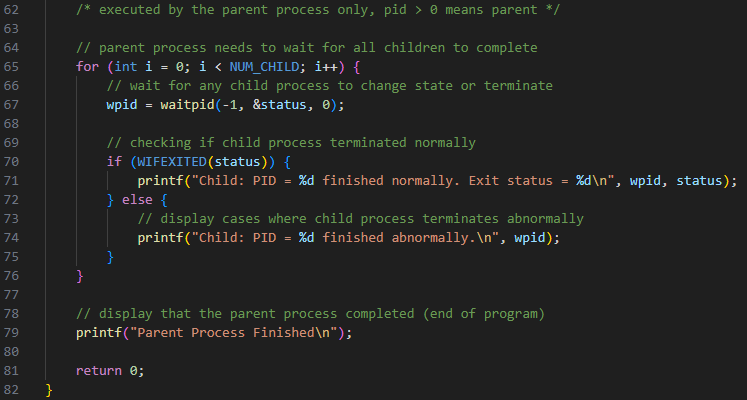
In the world of computer operating systems, a process is fundamental for program execution. Processes can be thought of as computer tasks that the operating system manages and controls. For example, opening a browser requires a process, most likley multiple processes, in order to fully render to the user’s screen. Additionally, a computer central processing unit (CPU) is the computer hardware that executes these instructions using its own processors. Depending on the type of CPU, single-core which has one processor or mult-core which has more than one processor, processes can be executed more efficiently. In this report, I will take you through my Process Management Simulator program, written in the C programming langauge for Linux, to demonstrate a high level overview of how operating systems manage their processes effectively.

The main goal of my Process Management Simulator is to demonstrate the use of the Linux process management commands like fork(), wait(), and execvp(), and how they are used by creating ten child processes that each peform a unique Linux command. Starting with Figure 1, I have included the neccessary C libraries: stdio.h for standard input/output, stdlib.h for standard library, unistd.h for process management, and sys/wait.h for waiting for process termination. Next I have defined NUM\_CHILD for the number of child processes to create. Next in Figure 2, we begin looking at the main function. The main function is essentially where the parent process begins its execution. Inside the main function, I have used the getpid() function to display the current parent processes unique id number. Following that I have also initialized a two dimensional A screen shot of a computer program

AI-generated content may be incorrect.array of chars for each of the execvp() function arguments that each child process will perform. Lastly, I also defined two int variables that will be utilized later by the parent process.

Figure

A computer screen shot of text

AI-generated content may be incorrect.Now we can move onto the child process creation and execution section of the program, located in Figure 3. First, I have a loop that iterates for however many times the NUM\_CHILD is specified. In each iteration of the loop, I create a new child process using the fork() function with the return value set to the typedef int pid. The next part is crucial for process execution and the goal of this lab. The pid variable has three categories determining the current process: pid < 0 for fork() function failure, pid == 0 for child process, and pid > 0 for parent process. We can first check if the fork() function failed which will result in an error message and an immediate exit of the program. If successful and the current process is a child process, we display the current pid and Linux command that will be used by the execvp() function. We also check if the execvp() function failed, resulting in another error message and immediate exit of the program. Since, the execvp() function terminates the child process for execution, the return statement should not be reached. The last section of my program, Figure 4, is the case where the pid > 0, which only is executed by the parent process. Since the parent process does not perform any of the operations in the previous loop, we must explicitley hold the parent process until all child process are complete. This can be achieved by using the waitpid() function, which will return the pid of the child that changed state. Additionally, I have used the WIFEXITED() function to check if the child process terminated normally using the status that was altered during the waitpid() function call. Finally, outside the loop I display a message to detail the end of the program, which means that all child processes have finished allowing for the parent process to finish too. The results of this program, depicted in Figure 5 and 6, details the creation A screenshot of a computer

AI-generated content may be incorrect.of all processes with their unique pid, the Linux command operated by the execvp() function, and the termination of all processes. Also, the program will detail wether the A screenshot of a computer program

AI-generated content may be incorrect.process termination was normal or abnormal. Lastly, we can see that the parent process finshed execution, meaning that the overall program has been completed.

Figure 3

Figure 2

Figure

Figure

Figure

Overall, this project report has detailed the theoretical and implementation features for my Process Management Simulator. We have defined what computer operating systems process are and how they work, along with an in-depth example of how to use processes in the C programming language for Linux. I hope that you have learned something valuable from my report and are able to utilize this knowledge to better understand processes in computer operating systems.