Introduction

In this lab the objectives were made clear, learn to understand the creation of processes using the fork() method, executing them with exec functions like what I used execvp, and using the wait() function to manage synchronization and the termination of processes. I found that processes didn't run in order, and it seems some terminal commands execute quicker than others, the amount of time is maybe even smaller than a millisecond.

Implementation

My implementation followed the in-class example closely. I create the necessary variables, one to hold the child processes, one to hold the status code, and another to represent the amount of child processes there are. Each array position has a unique command with a max of 3 indexes per row which all end with NULL to represent the end of an array, reminds me of SML which I'm working on in another class.

Below the first block is a simple print statement to print the Parent Process ID using the getpid() function.

The first loop is a for-loop and goes from 0 to 1 less than the numberOfChildren which is initialized at the beginning of the main function. Iterating by 1 each loop I assigned the fork() value to the "pid" variable which will now hold the Child Process ID of the current process in the loop. If an error happens, then the perror will send a message that the "Exec failed" and the exit() function below will be triggered, or if the "pid" is less than 0 this means that the fork failed, because no process is a negative integer value.

The final loop is one I took inspiration (wrote along in class) from the in-class code, and I broke it down like the following: If the wait(&status) system call is above 0 then that means it successfully returned a Child Process ID that has been terminated. Once all 11 of the processes are finished then the wait(&status) will return a -1 value and store it into “pid”, this results in breaking the while loop and returning 0 to end the program.

Results and Observations

The processes, starting with the parent, is executed as the program is executed and printed out in line 38. Each iteration of the first for-loop pointed each child process, fork() in c syntax, to the “pid” variable and prints out the currently executing child process. Each process had their own wait(&status) to block the parent process until no more child processes were available to input sending a -1 wait(&status) system call.

The child and parent processes interacted through the wait(&status) system call to pause the parent process to complete all child processes. I found that Echo got done quicker than other child processes to give results in the terminal. “man” seems to override priority and you have to press ‘q’ to exit it at the beginning of execution. Chmod seems to be a slow command compared to echo in terms of processing time according to my terminal results.