## Introduction

Process management is a very important concept in operating systems, but also in general. In this project, the C program creates multiple child processes, assigns them specific tasks, and manages their execution and termination.

## Implementation Summary

The program creates 10 child processes using the “fork()” system call. Each child process is assigned a unique task based on its order in the sequence of a for loop.
The first child process executes a specific task: running the “echo” command to print "Hello + <Your Name>". The rest of the processes execute a generic task: some processes include “uptime”, “ps”, and “hostname”. The parent process waits for all child processes to complete using the “wait()” call, and then prints the exit status or termination signal of each child process.
New code: I will not be going over commands previously used in lab 1 (like echo).
This is also in a different language than lab 1, so I will not be going over things like what a switch case is.
- fork(): Creates a new child process by duplicating the parent process. The child process starts execution from the point where fork() was called.
- execvp(): Replaces the current process with a new one by specifying a command (like “echo” or “ls”).
- wait(): Makes the parent process wait for the child process to terminate.
- PID: Process identifiers are assigned to each process, and can be used to manage or monitor the process.

- WIFEXITED: Checks if the process ended normally.
- WIFSIGNALED: Like WIFEXITED, but not normal, since the process was told to exit by a signal.
- WTERMSIG: Applies to WIFSIGNALED: gives the process exit signal.
- WEXITSTATUS: Applies to WIFEXITED, and returns exit code of the child process.

## Results and Observations

### A. Process Creation and Management

The “fork()” system call creates the child processes, and each one inherits the parent's address space and resources. There are 10 of these processes in a loop, with their PIDs being stored. The first process is assigned to execute the “echo” command to print a message; the remaining processes execute 9 unique commands.The default case is an “echo” statement, which should not be reached. “wait()” is used to allow each child process to terminate. Each exit status is checked by using “WIFEXITED” and “WIFSIGNALED” to determine if the process exited normally or by a signal.

### B. Parent and Child Process Interaction

The parent process creates child processes and waits for their completion: PID and exit status is printed after termination. Child processes execute their specific task by using “execvp()”. If “execvp()” fails, the child process terminates with an error message. These child processes run simultaneously. The parent process ensures that all child processes complete before it terminates, using the “wait()” call.

## Conclusion

This project demonstrates process management using C with calls like “fork()”, “execvp()”, and “wait()”. The “fork()” call can be used to create multiple processes, and the “execvp()“ call allows processes t be replaced dynamically. While this program can be seen as trivial, these ideas exist because of how complicated managing processes can be.