PROCESS MANAGEMENT

Magnus Hayden

CWU Operating Systems

3 February 2026

Table of Contents

Introduction

Process management includes creation/termination, scheduling, resource management, state management and more. It is the steps to set up and run processes either one at a time or simultaneously. It also involves the steps taken while the process is actively running. It is important in operating systems because you need to be able to run multiple processes at once and without process management then said processes would collide with one another and break the system. Or you would only be able to run one process at a time but then the system would be too slow to be of any practical use. Process management is what allows the system to function as well as we have it today.

Implementation Summary

My program starts out by declaring many local variables that are used throughout the rest of the program. It then creates a child process with fork() and runs the first command in that child by calling execvp() on a separate shell script. The PID of this child gets printed out to the console. The parent waits for the child to return with waitpid() before it moves on to print out the details of the return. This process is repeated fifteen times with a separate command for each iteration. In these commands, two of them intentionally abort returning a signal rather than an exit code. Another two intentionally call an invalid statement which then returns a non-zero exit code of 127. After all the commands are run the program prints out a summary including the number of zero exits, non-zero exits, and signal terminations.

fork() was used to create fifteen different children processes.

execvp() was used to run the fifteen separate Linux commands from their respective shell scripts.

waitpid() was used in the parent process to wait for all operations inside the child process to finish before moving on.

Results and Observations

Child processes were created using the fork command. The PIDs were managed in the parent process by keeping track of them in an array. This array was populated in order of creation.

Process creation is the order in which the processes have been initialized inside the program. This differs from process termination which is the order in which the process has

finished all its instructions and has exited either smoothly or abruptly. These orders can vary wildly depending on the instructions within each process and whether or not they are running simultaneously or are waiting for previous processes to exit.

The parent is able to determine whether a child has existed normally or with a signal by using macros such as WIFEXITED(status) and WIFSIGNALED(status). These tell you whether a process returned normally with exit or aborted sooner with a signal. You can use other macros such as WEXITSTATUS(status) and WTERMSIG(status) to get the values of the exit and signal codes.

Conclusion

In this lab I learned how to create and run multiple different processes at once and manage them to be in a clean unified environment. I learned how to switch between the parent process and the child process and how to how to run commands through execvp(). I learned more about clean and abrupt exits and the differences between what they return and what those codes mean. Overall, I learned a ton about how processes are created, managed, and terminated.