Xavier Santiago

10/25/2025

CS470

# Lab 2 Report

This project explores core Unix process management concepts through a C-based simulator that demonstrates the use of fork (), execvp(), and waitpid(). These system calls are foundational for multitasking and process control in Unix-like operating systems. The goal was to create a parent process that spawns multiple child processes, each executing a distinct command, and to manage their lifecycle with proper synchronization and reporting.

## Implementation Summary

The program is structured around three main phases. First, the parent process uses a loop to spawn ten child processes via fork (). Each child is assigned a unique command from a predefined list, including standard Unix commands such as ls, date, whoami, and a personalized echo command that outputs “Hello Xavier.” Second, each child process replaces its image using execvp() to run its assigned command. This demonstrates how child processes can execute independent tasks by invoking external programs. Third, the parent process uses waitpid() to monitor each child’s termination. It reports whether the child exited normally or was terminated by a signal, along with the corresponding exit status or signal number. The program includes robust error handling for both fork() and execvp() failures, and each child prints its PID and command before execution to aid in traceability and debugging.

## Results and Observations

The parent successfully created ten child processes using fork(), each of which executed a distinct command. This demonstrated parallel task execution and effective use of process creation. The use of execvp() ensured that each child replaced its process image with the intended command, showcasing how Unix systems manage program execution. The parent tracked all child PIDs and waited for each to complete using waitpid(), allowing for precise synchronization. Status reporting was accurate and informative: normal exits were captured using WIFEXITED() and WEXITSTATUS(), while signal-based terminations were handled via WIFSIGNALED() and WTERMSIG(). Each child printed its PID and command before execution, which provided clear visibility into the process flow and helped verify correct behavior.

## Conclusion

This project successfully demonstrated key principles of Unix process management by simulating multitasking through fork(), executing distinct tasks with execvp(), and synchronizing child termination using waitpid(). The parent process effectively managed multiple child processes, each performing a unique command, including a personalized echo. Status reporting provided clear insights into process outcomes, distinguishing between normal exits and signal-based terminations. Overall, the simulator offers a reproducible and transparent model for understanding parent-child interactions in Unix environments, laying a strong foundation for more advanced topics such as inter-process communication and concurrent programming.