CS 470 Lab 2 Project Report

Introduction

The purpose of this project is to demonstrate the use of the system calls `fork()`, `wait()`, and `execvp()` in a C program for creating and managing child processes. It aims to showcase both our existing knowledge and any new insights gained from recent lectures and discussions throughout this quarter. The objective is to simulate a scenario where multiple child processes execute different tasks while the parent process waits for their completion. Additionally, this project will demonstrate our ability to create a `Makefile`.

Implementation Summary: Summary of the Structure and Logic of the C Program:

For this project, the goal of the C program is to create 10 child processes, each performing a unique task. One of the child processes will print a greeting message that includes the user's name. Another child process will execute the `` `ls` `` command, while the remaining processes will run various `` `gcc` `` commands, including `` `pwd` `` and `` `date` ``. The parent process will wait for each child process to terminate and will report the completion status of each one.

Results and Observations:

A. An Explanation of How the Processes are Created and Managed.

The C program successfully creates 10 child processes, and each executes a different command. One of those commands is `fork()`, which is used to create child processes, and another command is `execvp()`, which is used to replace the child processes with the specified commands.

B. A Description of How the Parent and Child Processes Interact.

The parent process uses the command `wait()`, which makes sure that it does not exit before all child processes have finished. The parent process is supposed to accurately report the exit status of each child, whether they terminated normally or due to a signal.

Conclusion

CS 470 Lab 2 Project Report

This project helped to solidify the understanding of process creation, organization, and inter-process communication in Unix-like systems. The relationship between parent and child processes was proficiently handled using `fork()`, `execvp()`, and `wait()`, demonstrating how a parent process can create a numerous amount of child processes and coordinate their termination.