|  |
| --- |
| CWU |
| Process Program Documentation |
|  |

|  |
| --- |
| Caden-Gamer14  2-10-2025  CS470  Dr. Shangyue Zhu |

Contents

[Introduction 2](#_Toc190094260)

[Implementation 2](#_Toc190094261)

[Results and Observations 3](#_Toc190094262)

[Process Creation 4](#_Toc190094263)

[Process Interaction 4](#_Toc190094264)

[Conclusion 5](#_Toc190094265)

## Introduction

This document encloses information based on the process\_program.c program written in the C programming language. The knowledge contained within this document includes implementation, results and observations, various processes, and how each child interacted with one another. This C program contains different types of child and parental processes which execute various commands and functions.

### Implementation

The implementation of this C program begins with creating a taskCommands function for the purpose of executing all the child processes. This method takes in an int value, which will be used later for the ten child processes. Afterwards, the primary focus of the taskCommands method is the switch statement, which contains information relating to each child process within this program. I have not used switch statements very much in the C programming language, and seeing as a switch statement makes the most sense for having a wide variety of various processes to execute, I thought it made the most sense to store each process in the ten cases of the switch statement. The first case begins with an echo stating my name, followed by a ls command, date command, pwd command, another echo command, whoami command, uptime command, a third echo command, a uname command, a final echo command, and a default case for outputting errors. Although it is stated to have a unique command for each child process, I was unable to think of ten distinctly unique commands to use. There might be some other commands to add instead, however for simplicity I mostly used echoes and other commands I am more familiar with. Anyways, once the switch statement comes to an end, an execvp method is used to execute the commands, alongside a perror message if anything goes wrong. Afterwards, this is the end of the taskCommands method, leading to the main method. The beginning of the main method includes a pid for the ids of the children. Furthermore, a for loop is initiated which begins the primary sequence. Starting off the for loop, the pid in an array within the loop is assigned to the fork() method to create the child processes. Once this is completed, an if else statement is executed to determine if the program successfully runs or not. This ends the first for loop, however there is also a second for loop to add onto this one. The second for loop begins by using the wait() method to allow the parent process to finish completing the task assigned. Additionally, a series of if, else if, and else statements are used to see if the processes are successfully executed, or if they fail and terminate. Once this is completed, the for loop is finished, and underneath the for loop a simple printf statement is declared to mark the end of the program.

### Results and Observations

After carefully experimenting and tinkering with this program, a few results have been concluded based on the findings. Every process within the program successfully runs as intended, using the switch statement accordingly. However, after discreetly looking through the output message within the terminal, it seems one aspect occurred which was unintended but is a minor issue. The minor issue is the first set of child processes executed first, and afterwards the print statement saying the process is completed appears. The intent of the print statement is to allow for the message to be output directly after each child process. This seems to happen for the remaining child processes however, something which I am unable to discover the cause of. In any case, the program is fully functional, and other observations include how each process is executed within the terminal. Additionally, each process shows a unique output message in the terminal, which is entirely intentional given each process is meant to be unique in nature.

### Process Creation

The primary method each child process is created is using the switch statement within the taskCommands method. The switch statement allows for each process to be declared with specifications, meaning that each process is created beforehand. Furthermore, this continues to elaborate on the idea of each process being unique, and clearly displays which command is being used for each process. After each process is declared within the switch statement, the main method uses two for loops to process and execute the various processes. The first for loop goes through each process and executes it by using the fork() method, instantiating each process to be used within the program. Afterwards once the first for loop processes everything, the second for loop ensures that each parent process waits for each process to complete by using the wait() method. Once this is completed, each process finishes executing, and a print statement is displayed which means the program has ended.

### Process Interaction

The main interactions between parent and child processes are rather quick, as the primary method is with the wait() method. The wait() method allows each process to wait for each process to finish executing, allowing the next process to execute without overlapping any previous processes. Although this is the primary method of interaction, the wait() method is indispensable as it provides a way of organization between each process. Furthermore, each process waits its turn and does not execute until the previous process is completed. This is an example of saving memory, as using multiple processes at once would cause more memory to be used. Considering this is the case, the use of the wait() method is used for processes to interact with one another and wait for previous processes to complete before executing.

## Conclusion

In conclusion, the process\_program.c program allows for users to execute various processes using different functions. The use of switch statements, for loops, execution commands, and useful methods such as fork() and wait() are needed for proper execution and are necessary for completing the program efficiently and without an overuse of memory. Despite how this program is compiled, various other methods can be implemented to enhance the performance and further the functionality even more.