CS470 - Lab 2 Report

Cole E. Ralph

October 2025

1 Introduction

This lab demonstrates the use of Linux system calls `fork()`, `execvp()`, and `wait()` to manage processes in C. The goal is to create multiple child processes from a single parent process, execute various Linux commands in each child, and synchronize their completion through the parent process. This lab helps illustrate how the operating system handles multitasking, process scheduling, and process communication.

2 Implementation

The program begins by defining an array of command–argument pairs representing ten Linux commands to be executed by child processes. The parent process first prints its own process ID using `getpid()`. It then enters a loop that calls `fork()` ten times to create ten child processes.

Each child prints its process ID and the command it will execute, then replaces its code with the specified Linux command using `execvp()`. If `execvp()` fails, the child prints an error message and exits safely. The parent process remains active throughout this process, waiting for all child processes to finish using `wait()`. For each child that terminates, the parent reports its process ID and exit status.

3 Results

After compiling with `make`, the program successfully produced ten concurrent child processes. Each executed a command such as `echo`, `ls`, `pwd`, `date`, and `whoami`. The output order varied due to concurrent scheduling, which is expected behavior in multitasking environments.

Figure 1 shows the terminal output demonstrating correct execution, successful child process termination, and the parent waiting for all children to complete.

```
root@bce2b5d282f6:/work/cs470.lab2# make -f MakeFile
gcc -Wall -Wextra -O2 -o lab2 main.c
root@bce2b5d282f6:/work/cs470.lab2# ./lab2
Parent PID: 1614
Child 1 PID: 1615 executing echo
Child 2 PID: 1616 executing ls
Child 3 PID: 1617 executing pwd
Child 4 PID: 1618 executing date
Child 5 PID: 1619 executing whoami
Child 6 PID: 1620 executing uname
Child 7 PID: 1621 executing echo
Child 8 PID: 1622 executing echo
Child 9 PID: 1623 executing echo
Child 10 PID: 1624 executing echo
Hello Cole!
Learning fork() and exec()
Process 7 running
/work/cs470.lab2
Parent: Child 1615 exited normally, status=0
Parent: Child 1622 exited normally, status=0
Parent: Child 1621 exited normally, status=0
pr 21 17:08:54 UTC 2025 x86_64 x86_64 x86_64 GNU/Linux
Parent: Child 1624 exited normally, status=0
Parent: Child 1618 exited normally, status=0
Parent: Child 1620 exited normally, status=0
root
MakeFile  lab2  main.c
Parent: Child 1619 exited normally, status=0
Parent: Child 1616 exited normally, status=0
Parent process (1614) finished waiting for all children.
```

Figure 1: Execution results showing child creation, command output, and parent synchronization.

4 Conclusion

This lab effectively demonstrates process creation and control in a Linux environment. Using `fork()`, each child inherits the parent's context, while `execvp()` replaces that context with a new program. The parent's use of `wait()` ensures synchronization and prevents zombie processes. Overall, the program confirms a clear understanding of concurrent process management and interprocess co-

ordination in Unix-like systems.