

Introduction

This lab helped me understand how to create and manage processes in a Linux system using C. The goal was to simulate how a parent process spawns multiple children that each run their own task. I used three main system calls to do this: `fork()` to create new processes, `execvp()` to run commands inside those processes, and `wait()` to make sure the parent process stays in control and waits for the children to finish. This gave me practical experience with process handling and system-level programming.

Implementation Summary

The program begins by storing 10 Linux commands in a 2D array. These include `ls`, `whoami`, `date`, and one that prints “Hello Mariatu” using the `echo` command. A `for` loop is used to create 10 child processes using `fork()`.

Each child prints its PID and the command it’s about to run, then calls `execvp()` to execute the command. If the command fails, an error is printed using `perror()`. Meanwhile, the parent uses `wait()` to wait for each child and prints whether it exited normally using macros like `WIFEXITED()` and `WEXITSTATUS()`.

Results and Observations

A. Process Creation and Management

Processes were created using `fork()` inside a loop. Each successful child process replaced its own memory using `execvp()` to run the assigned command. The parent kept track of child termination using `wait()` and interpreted the exit codes using predefined macros.

B. Parent and Child Interaction

The parent controlled the overall flow by initiating the children and waiting for them to complete. Each child process operated independently but reported back indirectly by exiting. The parent used this to log process completion and confirm that each command ran as expected.

```
Child 1 [PID: 14205] is running command: echo
Hello Mariatu
Child 2 [PID: 14206] is running command: ls
Child 3 [PID: 14207] is running command: whoami
Child 4 [PID: 14208] is running command: date
Fri May  9 08:57:07 AM UTC 2025
Child 5 [PID: 14209] is running command: pwd
wuriem
/home/wuriem/lab2_process_simulator
total 28
-rwxrwxr-x 1 wuriem wuriem 18336 May  9 08:57 main
-rw-rw-r-- 1 wuriem wuriem 2781 May  9 08:27 main.c
-rw-rw-r-- 1 wuriem wuriem  65 May  9 08:53 Makefile
Child 6 [PID: 14210] is running command: hostname
Child 7 [PID: 14211] is running command: uname
Kyolikesbags93
Linux Kyolikesbags93 6.11.0-25-generic #25~24.04.1-Ubuntu SMP PREEMPT_DYNAMIC Tue Apr 15 17:20:50 UTC
2 x86_64 x86_64 x86_64 GNU/Linux
Child 9 [PID: 14213] is running command: ps
Child 8 [PID: 14212] is running command: id
Child 10 [PID: 14214] is running command: uptime
uid=1000(wuriem) gid=1000(wuriem) groups=1000(wuriem),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),100(users),114(lpadmin)
Parent: Child with PID 14205 exited with status 0
Parent: Child with PID 14206 exited with status 0
Parent: Child with PID 14207 exited with status 0
Parent: Child with PID 14208 exited with status 0
Parent: Child with PID 14209 exited with status 0
Parent: Child with PID 14210 exited with status 0
Parent: Child with PID 14211 exited with status 0
Parent: Child with PID 14212 exited with status 0
08:57:07 up 1:18, 1 user, load average: 0.70, 0.46, 0.65
Parent: Child with PID 14214 exited with status 0
  PID TTY          TIME CMD
 2361 ?            00:00:01 systemd
 2366 ?            00:00:00 (sd-pam)
```

As seen in the output above, each child process printed its own message, and the parent reported status messages as each child exited. The `echo` command correctly printed “Hello Mariatu,” and all commands completed with exit status 0.

Conclusion

This lab helped me connect theory with actual system programming. I now understand how operating systems manage multiple processes and how `fork()`, `execvp()`, and `wait()` work together to make it possible. Writing this program made me more confident in using system calls in C and showed me how real programs can manage multiple tasks efficiently. This will be useful in future classes or jobs where multitasking and process control are required.