

Introduction

This program creates 10 unique child processes in which the user is notified when finished. When a child process is successfully started, the ID will be printed and commands will be executed. If there are errors, they will be caught accordingly. Because of the header files, this program is OS specific and made for Linux.

Implementation

This program consists of two files; one is a C file that implements bash functionality and another is a basic Makefile.

The C program creates a 2D char array of 10 child processes, each a unique bash command. The ID of the parent process is printed and then we move into a for loop that goes through each child process. The ID is tied to the fork() function in C which will basically relay the status of the process. If the number is below 0, the user is notified of the failure and the program exits. If the ID is 0, the ID of the child process will be printed and the command will be executed. Within this if statement, is handling for an error if the execution is unsuccessful. Outside of the loop, once processes are finished, the user will be notified per child process.

The Makefile is a simple file that executes the gcc command renaming the file to a processes.exe file, which can be executed.

Results and Observations

The file executed quite easily. The only thing is that the commands were executed in the same terminal so the user will get a bit of clutter, especially in my example where the ls -l command was taking up a lot of space between the other echoes:

```
Parent process PID: 4276
Print out history -----
Child process PID: 4277 - Executing commands: ls
Child process PID: 4278 - Executing commands: echo
Child process PID: 4279 - Executing commands: ps
total 116
drwxrwxr-x 12 ubuntu ubuntu 4096 Jan 27 03:22 2025-01-27-03-22-26
-rwxrwxr-x 1 ubuntu ubuntu 17592 Jan 29 03:21 copy
-rw-rw-r-- 1 ubuntu ubuntu 416 Jan 28 23:53 copy.c
-rw-rw-r-- 1 ubuntu ubuntu 488 Jan 22 03:50 copyfile.sh
-rw-rw-r-- 1 ubuntu ubuntu 901 Jan 27 03:22 create_files_with_subdirs.sh
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 22 03:50 des470
drwxrwxr-x 3 ubuntu ubuntu 4096 Jan 29 03:23 des_dir
drwxr-xr-x 2 ubuntu ubuntu 4096 Jan 21 05:53 Desktop
drwxr-xr-x 2 ubuntu ubuntu 4096 Jan 21 05:53 Documents
drwxr-xr-x 2 ubuntu ubuntu 4096 Jan 21 23:04 Downloads
-rw-rw-r-- 1 ubuntu ubuntu 177 Feb 10 05:31 Makefile
drwxr-xr-x 2 ubuntu ubuntu 4096 Jan 21 05:53 Music
drwxr-xr-x 3 ubuntu ubuntu 4096 Jan 29 03:34 Pictures
-rwxrwxr-x 1 ubuntu ubuntu 18272 Feb 10 17:51 processes
-rw-rw-r-- 1 ubuntu ubuntu 1322 Feb 10 17:51 processes.c
drwxr-xr-x 2 ubuntu ubuntu 4096 Jan 21 05:53 Public
drwx----- 5 ubuntu ubuntu 4096 Jan 27 03:00 snap
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 22 03:50 source470
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 29 03:23 source_d
```

Here is the first picture of the execution where it's cut off since the ls command is so large.

Conclusion

From this lab, I have a much better understanding of process creation and how parent and child processes interact with each other.