Telmen Enkhtuvshin

CS 470 Operating Systems

Dr. Shangyue Zhu

February 10th, 2025

Lab 2 – Process Management Simulator

Introduction

This lab simulates the parent and child processes in a Linux-based Ubuntu Operating System with a C program. As the simulation runs, multiple child processes stem out from the parent process and execute their assigned terminal commands accessed from a simple data structure. The resulting program is compiled with the MakeFile library dependency to easily control the compilation and the build process of the C program source file. When the resulting executable program file is executed, the assigned commands in the program get executed with the direction of the command line accessing execvp() function. Then, with multiple Command Line designated commands being executed, the results are immediately displayed in the following lines of the currently using terminal. After execution, the program exits with success.

Implementation Summary

The C program works as a simple short program with a few additional libraries like <sys/wait.h> get pulled in to use special functions to simulate parent and child processes of an Operating System. The program starts by declaring variables like *pid*, *status*, and a *numberofChildren* to use in the main part of the program. Then, a list of terminal-specific commands is stored in a 2D array using a pointer. After that, in a for loop, the child processes are created one after the other. A special *fork()* function is used to get the *pid* number of a process.

Then, if the number is negative, it indicates a failure, and the error is caught to print out a relevant message to the user. If the *pid* is zero, it means the fork process was successful, and the child process can be continued. After that, the corresponding command from the array is accessed to execute it, and a special function of *getpid()* is also used to access the child process *pid* and display it to the user. The command is executed by a special function called *execvp(),* and

if it fails, the following lines of code are ready to catch the execution error and display the error message to the program user. Moving on, when the for loop is fully executed, the special *wait()* function is used inside a while loop to wait for the child process to finish, and a finish confirmation message is printed for every child process in the console depending on if the *wait()* function returns zero or not. Finally, the program returns a value corresponding with success, ending the program.

Results and Observations

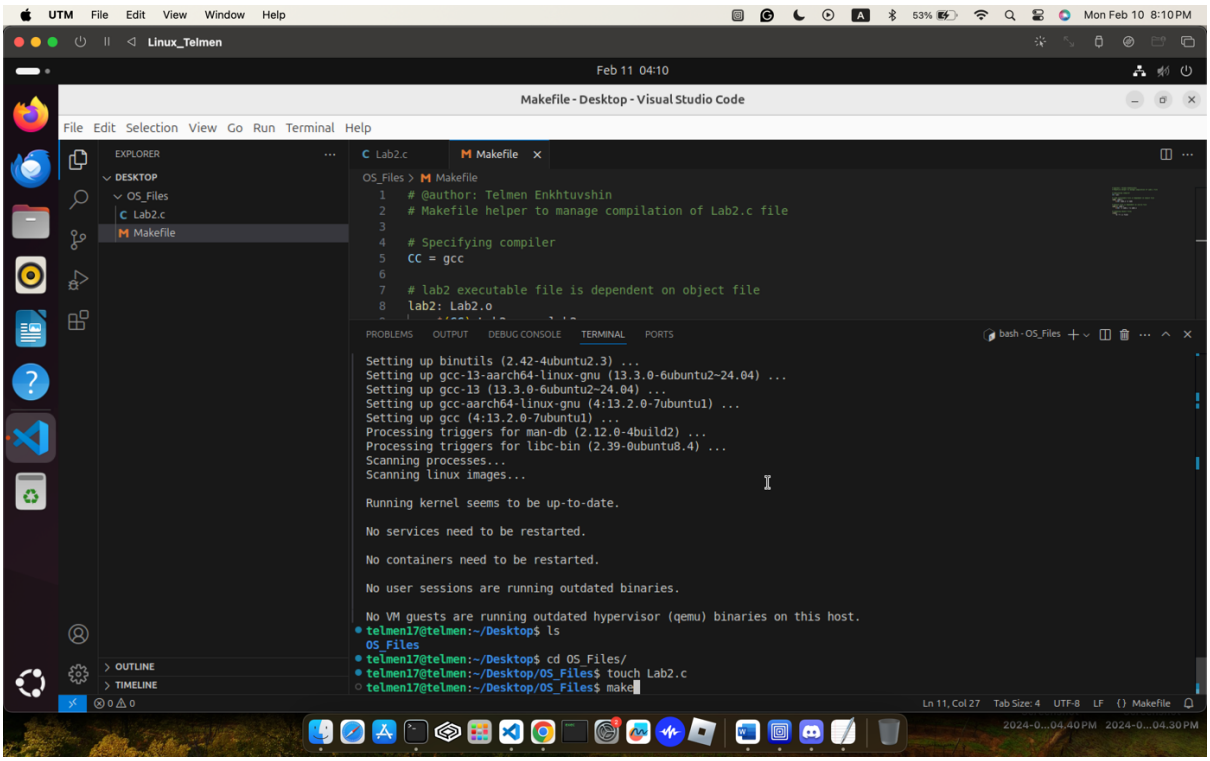
As explained in the implementation part of the C program, processes were created as a child using the special function *fork()* to get the designated *pid* for that specific process. Once that is successful, the process can execute its main purpose to run terminal commands with the information pulled from the 2D array.

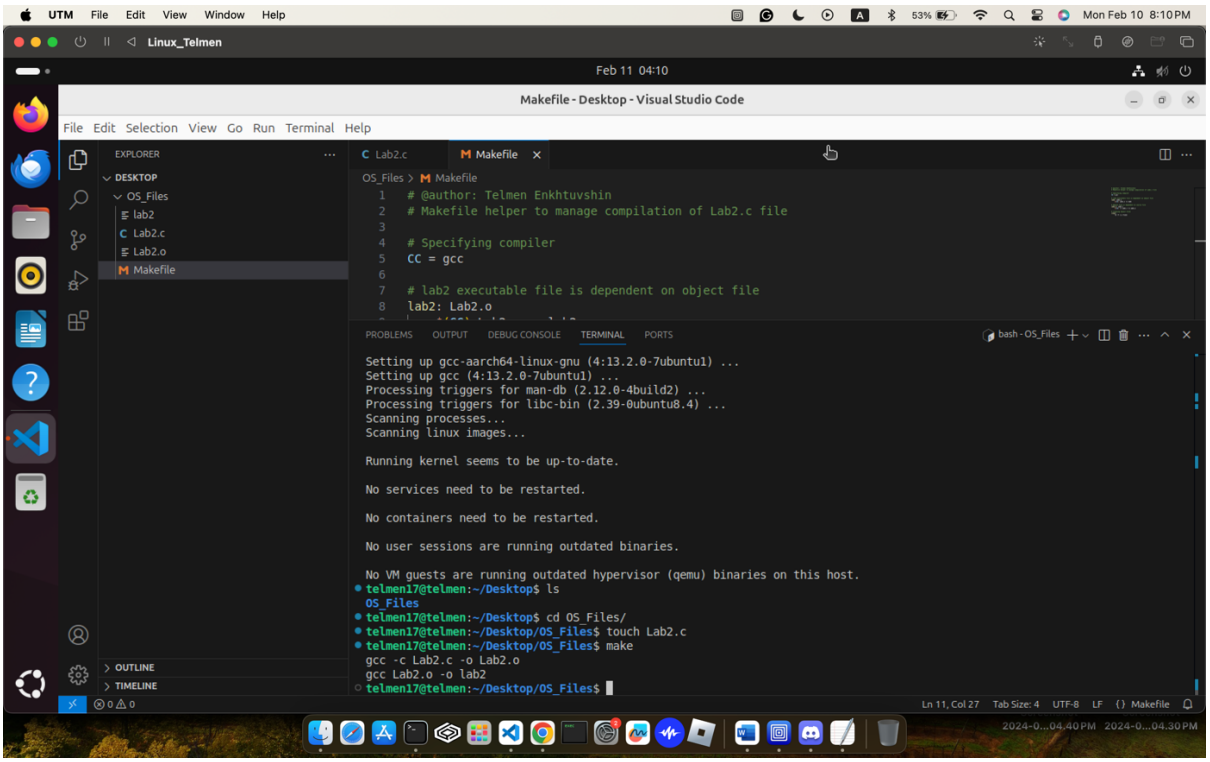
Relatively, the error cases are caught with the respective actions depending on the value of the *pid* variable for the current process. After that, the parent process, which is the main program in the activity, waits for the child processes to finish and prints out the confirmation messages. Finally, when all the child processes are complete, the parent process can finish its own execution and end the overall program with success.

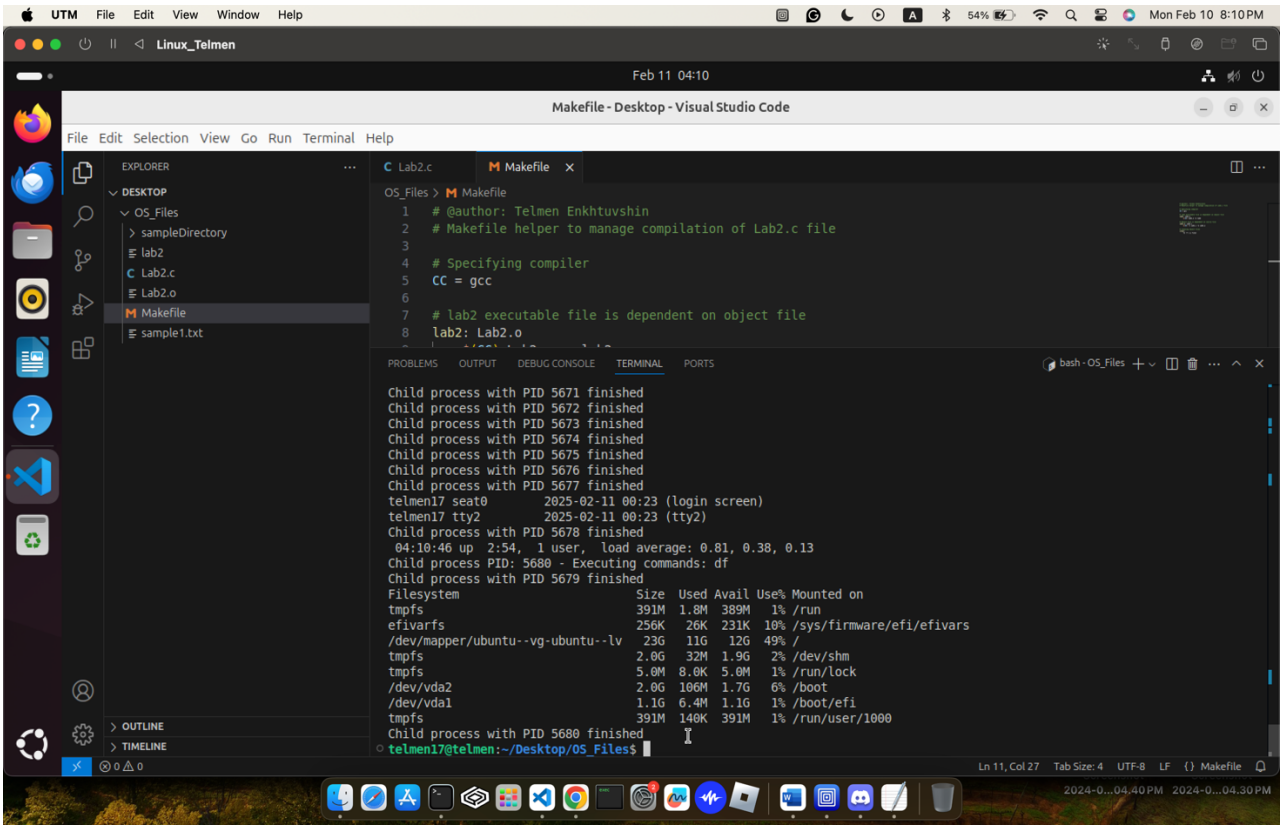
Conclusion

In conclusion, this lab was very helpful in aiding to understand the exact processes and phenomenon that happen inside an Operating System to manage tasks and processes. Within a main process, there can be many child processes that stem out from the parent like a branch coming out of the main tree body. The parent waits for all its child processes to finish and continues from there to complete the main task. I find this very fascinating and hope to learn more in the next Lab assignments.

Screenshots:


UTM File Edit View Window Help
Mon Feb 10 8:10 PM
Linux_Telmen
Feb 11 04:10
Makefile - Desktop - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
DESKTOP
OS_Files
Lab2.c
Makefile
Lab2.c
Makefile
OS_Files > Makefile
1 # @author: Telmen Enkhtuvshin
2 # Makefile helper to manage compilation of Lab2.c file
3
4 # Specifying compiler
5 CC = gcc
6
7 # lab2 executable file is dependent on object file
8 lab2: Lab2.o
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash - OS_Files
Setting up binutils (2.42-4ubuntu2.3) ...
Setting up gcc-13-aarch64-linux-gnu (13.3.0-6ubuntu2~24.04) ...
Setting up gcc-13 (13.3.0-6ubuntu2~24.04) ...
Setting up gcc-aarch64-linux-gnu (4:13.2.0-7ubuntu1) ...
Setting up gcc (4:13.2.0-7ubuntu1) ...
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for libc-bin (2.39-0ubuntu8.4) ...
Scanning processes...
Scanning linux images...
Running kernel seems to be up-to-date.
No services need to be restarted.
No containers need to be restarted.
No user sessions are running outdated binaries.
No VM guests are running outdated hypervisor (qemu) binaries on this host.
telmen17@telmen:~/Desktop$ ls
OS_Files
telmen17@telmen:~/Desktop$ cd OS_Files/
telmen17@telmen:~/Desktop/OS_Files$ touch Lab2.c
telmen17@telmen:~/Desktop/OS_Files$ make
OUTLINE
TIMELINE
Ln 11, Col 27 Tab Size: 4 UTF-8 LF {} Makefile


UTM File Edit View Window Help
Mon Feb 10 8:10 PM
Linux_Telmen
Feb 11 04:10
Makefile - Desktop - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
DESKTOP
OS_Files
lab2
Lab2.c
Lab2.o
Makefile
Lab2.c
Makefile
OS_Files > Makefile
1 # @author: Telmen Enkhtuvshin
2 # Makefile helper to manage compilation of Lab2.c file
3
4 # Specifying compiler
5 CC = gcc
6
7 # lab2 executable file is dependent on object file
8 lab2: Lab2.o
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash - OS_Files
Setting up gcc-aarch64-linux-gnu (4:13.2.0-7ubuntu1) ...
Setting up gcc (4:13.2.0-7ubuntu1) ...
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for libc-bin (2.39-0ubuntu8.4) ...
Scanning processes...
Scanning linux images...
Running kernel seems to be up-to-date.
No services need to be restarted.
No containers need to be restarted.
No user sessions are running outdated binaries.
No VM guests are running outdated hypervisor (qemu) binaries on this host.
telmen17@telmen:~/Desktop$ ls
OS_Files
telmen17@telmen:~/Desktop$ cd OS_Files/
telmen17@telmen:~/Desktop/OS_Files$ touch Lab2.c
telmen17@telmen:~/Desktop/OS_Files$ make
gcc -c Lab2.c -o Lab2.o
gcc Lab2.o -o lab2
telmen17@telmen:~/Desktop/OS_Files$
OUTLINE
TIMELINE
Ln 11, Col 27 Tab Size: 4 UTF-8 LF {} Makefile


UTM File Edit View Window Help
Mon Feb 10 8:10 PM
Linux_Telmen
Feb 11 04:10
Makefile - Desktop - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
DESKTOP
OS_Files
sampleDirectory
lab2
Lab2.c
Lab2.o
Makefile
sample1.txt
OUTLINE
TIMELINE
Lab2.c
Makefile
OS_Files > Makefile
1 # @author: Telmen Enkhtuvshin
2 # Makefile helper to manage compilation of Lab2.c file
3
4 # Specifying compiler
5 CC = gcc
6
7 # lab2 executable file is dependent on object file
8 lab2: Lab2.o
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash - OS_Files
Child process with PID 5671 finished
Child process with PID 5672 finished
Child process with PID 5673 finished
Child process with PID 5674 finished
Child process with PID 5675 finished
Child process with PID 5676 finished
Child process with PID 5677 finished
telmen17 seat0 2025-02-11 00:23 (login screen)
telmen17 tty2 2025-02-11 00:23 (tty2)
Child process with PID 5678 finished
04:10:46 up 2:54, 1 user, load average: 0.81, 0.38, 0.13
Child process PID: 5680 - Executing commands: df
Child process with PID 5679 finished
Filesystem Size Used Avail Use% Mounted on
tmpfs 391M 1.8M 389M 1% /run
efivarfs 256K 26K 231K 10% /sys/firmware/efi/efivars
/dev/mapper/ubuntu--vg-ubuntu--lv 23G 11G 12G 49% /
tmpfs 2.0G 32M 1.9G 2% /dev/shm
tmpfs 5.0M 8.0K 5.0M 1% /run/lock
/dev/vda2 2.0G 106M 1.7G 6% /boot
/dev/vda1 1.1G 6.4M 1.1G 1% /boot/efi
tmpfs 391M 140K 391M 1% /run/user/1000
Child process with PID 5680 finished
telmen17@telmen:~/Desktop/OS_Files$
Ln 11, Col 27 Tab Size: 4 UTF-8 LF Makefile
2024-0...04.40 PM 2024-0...04.30 PM