CSSE332 Operating System Final Project Report

Weimu Song
Yuqi Zhou

Table of Contents

Overview

In this report, we will first cover any challenges that we have met during the developing, then introduce the extra features added to the system and how to use them. And finally, we will talk about what we have learned in this project.

Key Challenges (from m3)

a. m3:
    i. In terminal function of m3, we at first used “shell”, which is a string to start the shell function. However, it turned out that it would never restart shell again if a function calls the terminal function. The same problem happened in some other cases where printString function did not work because string is used as the parameter. This happened because the strings are stored in the sections of the files themselves, so when another function called the terminal function, for example, it could not find the string. Therefore, we changed the strings to char arrays and the problem was solved.
b. m4:
    i. The biggest challenge we had was testing. Because we were unable to test many of the functions right after we finished them, it took us some time to find the errors. For example, we need to use write sector function in all the functions we wrote later. If we had made it wrong, it would have taken us hours to figure the problem was from delete file function or write sector function. Fortunately, those functions (that cannot be tested themselves, like write sector and write file) did not go wrong, but it still took us more time to debug than we expected.
    ii. For create text file function, at first it did not work because we failed to deal with the change line input (when pressing enter button on the keyboard), so we did have some weird bugs. Besides, when we wrote write file function, the file names was not correctly passed from shell to kernel (we don’t remember exactly how since it has been a while). After we made sure the write file function was correct, then we added checking to the create text file function and it worked.
c. m5:
    i. When we added handleTimerInterrupt function and change the launchProgram function to initializeProgram function in the executeProgram function, the system called executeProgram on “shell” nine times before actually executing it. Also, when we were executing phello, it kept printing “H” instead of “Hello World”. At first we guessed the problem was that we might have not set the stack pointers correctly. But finally we figured out that in main function, we did not let the function wait in a while loop for the interruption. The system will instead execute the instruction after the last line in the main, which is after the shell interruption. It worked in the previous milestones because we used launchProgram function, which was the next line in the

memory. But after we changed it to initializeProgram, it will go back to main function after it makes the function ready to run. In this case, the system will run into error, and qume will keep running the last command before the error, which is printing “H” of phello and calling shell execution.

ii. Another problem we had in m5 was the segment pointer. In the instruction, it only let us set and restore the segment pointer in handleTimerInterrupt and terminal functions. However, it also needed those in kill and execforeground functions. We did not figure that out at first and it caused some trouble.

Special Features

- When using “dir” for show the files, we also showed the size of the files.

- In shell, type command “clear” clears all texts on current screen.

What We learned

- Weimu Song:
    - lessons:
        - Some of the work in this project (eg. m4, m5) can spend infinity time if we get into some unknown bugs, It is necessary that we start the project as soon as possible, so we can have enough time to deal with some hard situation.
        - The special bcc compiler that we used in this project taught me that compiler may not be 100% reliable and I need to know exactly how the code should be written to get the output I want. It is important to be aware of potential source of errors.
    - Techniques things learned
        - I am feeling much confident debugging on a situation where multiple process contexts are in presence. In addition, I am feeling more confident to inspect lower level data or programs. For example, hexedit and assembly.
        - Almost all the features were implemented through interrupts, so I am feeling more confident on how to use interrupt in an operating system.

- Yuqi Zhou:
    - lessons
        - Use hints cleverly: The hints in the instructions are very important and can be keys to the problems. Like it says “use char array instead of strings” and “set and restore segment pointers every time make access to the process table”
        - Test before going to the next step: When working on m4, I did not fully tested copy file function before going to the create text, so actually

something was wrong in the write file function, but did not show up in the tests I did for copy file. It took a long time to debug that.

- technique things learned
  - how things work in the system: Basically, each file has their own spaces in the memory and the system uses interrupts to get contents from them, make changes to them, or run the functions stored in them. Knowing the processes helps a lot. Since we cannot use what we had for debugging, the most convenient way was to print some messages and numbers in the process and find out what is going run. In most case, this was really helpful.