Amogh Yatnatti

Professor Ozbirn

CS 4348.001

13 March 2021

**Project 1 Summary**

The purpose of the project was to learn and mimic how a CPU processes instructions with the incorporation of two processes: one responsible for memory and another responsible for the CPU. While we are not necessarily creating a CPU or memory, we are mimicking the functions of them through a high-level language. In our mock CPU, we have a PC, SP, IR, AC, X, and Y register. These registers still serve the same purpose as they would in an actual CPU. The PC and SP registers are used to access indexes in memory, the IR holds instruction values, and the AC, X, and Y registers are used to store data. With our mock memory, we are making array with 2000 elements with the first half being dedicated for the user program and the second half being dedicated for the system code. The CPU makes sure we are not accessing system code when we should be accessing the user program. We also have a user stack that is at the end of the user program memory and a system stack that is at the end of the system memory. We also are including mock interrupts as well. There are two types: timer and a system call. The timer interrupts execute at 1000 and the system calls execute at 1500. The interrupts first save the user program’s SP and PC before proceeding. These behaviors that we are tasked with implementing are similar to the behaviors of a real CPU and memory. This project is essentially an application of the low-level concepts we learned.

I implemented this project in C++ so I used the fork method to create a child process that would be used for the memory and a parent process that would be used for the CPU. The memory process first retrieves all the instructions/values/addresses from the text file and stores it into the memory at the desired indexes. Once this is done, it activates a loop that waits to read instructions from the CPU. Once it receives an instruction from the CPU, it stores the first character and stores the rest into as a memory address (index). If the first character is an ‘r’, then the process will read the value at that memory address and send it to the CPU. If the first character is an ‘w’, then the process will read the value from the CPU and store it at the given memory address. If the first character is a ‘q’, then the memory process will exit. The memory and CPU process can communicate through pipes. One pipe would be used for the CPU to communicate with the memory and another pipe would be used for the memory to communicate with the CPU. Since the CPU will mainly be using the pipes to read or write to/from memory, I made a read and write method for it. The read method pipes the index to the memory process and receives the value at that index from the memory process. The write method pipes the index to the memory address and then pipes the value that needs to be written at that address. Both methods make sure that the user program is not able to access memory outside of its bounds. Both the timer and system call interrupts store the user’s current SP and PC onto the stack and begin execution at the corresponding index. The IR is evaluated for all possible instructions via a switch statement, and we print out an error message if an invalid instruction is recorded.

My experience with the project was greatly impactful. While I had worked with forks and pipes in my UNIX class, I had not worked on a project that was as big as this one. The most difficult part was debugging errors in the child process. When I first started debugging, I noticed that only the parent process was being tracked in the debugging process and that the child process was ignored. It was not until much later that I found out the debugger I used in Visual Studio Code (GDB) had a command (“set follow-fork-mode child”) that would allow me to evaluate to debug the child process and its variables. This project also helped me develop my understanding of interactions with the CPU and memory. Before, I understood the general concepts of how the CPU, memory, registers, stack, and interrupts functioned. However, when I started the implementation of these concepts in my program, I grew a deeper understanding of how they exactly work and interact with one another. Ultimately, this project allowed me to learn familiar with these concepts as well as the fork and pipe methods in C++.