CS 4348 Project 1

Daniel Crawford

9/23/2018

1. Project Purpose

This project will simulate the behaviors of a CPU and RAM, and then their relationships. The CPU is going to determine all the instructions to run, and the RAM will hold these instructions. After RAM loads a program and data, the CPU will run those instructions. However, the CPU will also handle more than that. It will also take in a timer argument to decide when it should issue an interrupt. These interrupts are handled based on what was loaded into RAM. The input files themselves will contain programs that will be interpreted and run by the CPU. Now, the process simulating CPU will execute these instructions and end when it reads instruction 50(End). The process simulating RAM will end around this time as well.

1. How the Project was Implemented

This project was implemented using C++ and modularizing each section as a class. To be specific, the CPU and RAM were their own class, and there was another class as well for reading an input file line by line. This way, it was easy to fix bugs because they were concentrated in one area. To start on the specifics, the RAM and the CPU communicated with each other using two pipes, cpupipe and rampipe. Cpupipe writes from cpu to ram, and rampipe vice versa. The RAM was a bit tricky to implement at first, but I ended up following this design: read address from CPU, write value at address, read type of instruction, then write, end or reiterate. Typically, the address would be the PC but sometimes it will also be from an instruction wanting to request data a certain point in memory. In the CPU, the registers are implemented as members of the function. These are all initialized at the constructor, with all being set to 0 except for the IR and the SP, where the stack pointer is set to the start of the user stack(999). The stack would simply increment and decrement the SP, and write values and read from them. Values are not deleted or set to 0 after they are popped, the stack within the CPU simply just stops acknowledging that memory. The CPU will overwrite this memory if it encounters that address in the stack again. Interrupts in the CPU are implemented in the runInterrupt() function. This function works like a switch, it is called once, it switches to kernel mode. If it is called yet again, it will switch back to user mode and restore the PC and SP. The RAM will end after that CPU encounters address 50, in which case the CPU will send type exit(represented as 2) to the RAM and it will also close.

1. Personal Experience

My personal experience with this project started with planning the implementation. Keeping a modular design to this project was a necessity to me because it would help focus on getting the code working one step at a time. However, I needed to understand how exactly pipes worked beforehand so I decided to jump into the implementation earlier than usual. Learning pipes was quick thanks to resources online and from the class examples. One of the problems I kept encountering on this program is that the processes deadlocked each other if not coded carefully. I realized later that the programs would reach the read() pipe functions and be waiting on each other to write something. This required me to plan out how the RAM handled these things, but I realized a bit later that my current process for handling RAM was done incorrectly. I had to re-implement the way RAM will read from the processor then write back to it. The complicated part was choosing how to decide will the RAM will do a write. Later on, I made a more robust implementation of RAM and this helped avoid all those deadlocks. At this point, after RAM was implemented correctly, all I had to do was implement the interrupt and system call functions. Once I had these implemented, I encountered problems with the program not ending when it was supposed to and I learned that some of the instructions I implemented were done wrong due to misunderstanding. After a series of debugging, I learned that my load Spx function needed to be incremented by 1, because my SP represented the current empty space. There were other bugs as well, and I fixed these by logging the information about registers at different points. Finally, once the main code was implemented, I made the sample file. I wanted to experiment with nested for loops, so I made a program that printed a pyramid shape using two loops and utilized the stack. By keeping an idea of what strategies I used to manage a for loop, I finished this sample quickly, and finished the project.