**SUMMARY**

**Project Purpose**

The purpose of this project was to simulate the way a processor fetches, decodes and executes instructions from memory. The main requirement for the implementation was to have the memory and the processor simulated by separate processes that communicate. This implementation was to help us to reinforce various concepts about an operating system such as the role of registers, the way I/O operations are handled, interrupt handling, system calls, stack processing, memory protection.

**Implementation**

This project is required to run on cslinux1.utdallas.edu and I opted to write it in java. There are two classes included: Memory.java and Processor.java.

The memory program starts by creating an integer array of 2000 entries which is initialized by reading a file entered as an argument when executing the program. It parses the file, only reading integer values. A dot signals a jump to the memory address indicated by the integer following it. After initializing the memory, the program then waits for input from the InputStream (System.in) which will later be connected to our processor program. The first character from the input is set as a flag to signal the type of request: 0 for read and 1 for write. Those requests are then handled from the private methods read() and write() to read from or write to memory.

The processor program starts by ensuring a file name is entered as an argument and that the file exists. It then executes Memory.java as a process passing it the file name and timer flag as arguments. The timer flag is to trigger a timer interrupt after each X instructions (with X being the argument entered). The program then enters a fetch-decode-execute phase where it reads each entry from memory starting at address 0 before decoding and executing the instruction using the pre-defined instruction table provided. I have chosen to use a switch statement to handle instruction decoding, separating those without operands from those with one. In this way, I was able to differentiate instructions from operands quite easily while keeping the program counter incrementing correctly. I created a variable to keep track of the number of instructions (numOfInstructions) processed in order to compare it with the timer flag mentioned earlier to trigger timer interrupts. Each time an interrupt is being processing an interrupt flag would be raised to avoid nested interrupts. This applied to both, system call interrupts from the instruction 29 and timer interrupts. Along with the methods to read from and write to memory, several other methods were implemented to handle aspects such as memory control, timer interrupt and some of the instructions.

**Personal Experience**

This project has taught me a lot, from process creation in java to interrupt handling. I found that implementing the features weren’t the hardest part of the project but rather debugging problems that arose. I chose to code it exclusively in vim rather than use an IDE to force myself to think through possible issues without the help of a debugger. At one point, one of my samples was not producing any output at all and it took me going to the professor to get a hint as to what the problem was. I liked the way he narrowed down the possible root cause based on the symptoms, the problem was actually with the memory class while I was focusing on the processor class. I later on used the same approach successfully when I encountered another problem. Overall, I learned a lot from this project, from how to create and establish communication between processes to the lower level inner working of an operating system interacting with memory.

**Sample Outputs**

