Naman Gangwani
nkg160030@utdallas.edu

CS 4348.002
February 24, 2018

Project 1 Summary

The purpose of this project is to emulate the communication between a CPU and memory system with high-level languages. Of course, CPUs and memory systems are not actually controlled by high level languages, but for the learning process to understand the pipe communications between them, working in a high-level language can be quite convenient for this. Ultimately, this project is about the understanding of the communication between multiple processes and the low-level concepts that CPUs and memory systems are required to encounter in order to have a functioning operating system design that is capable of executing programs while also handling interrupts such as I/O. Some other concept that this project requires one to understand are the ways in which the processor has to interact with memory (such as reading from memory and writing to it with requests through pipes), the important jobs of registers with their specified roles, how to process certain instructions (based on project specifications), how to process the stack, protecting memory from being directly accessed by the CPU, saving and retrieving from different stacks (user and system), handling interrupts and I/O, calling procedures and the system, and working in different modes (user and kernel).

This project was implemented with the Java programming language through the concepts of object orientation run time execution methods for processes and stream communication. When a command initiates the project, the main method from Project1 validates whether the command line input was valid and creates & initiates the CPU and memory processes. The CPU and memory processes are handled through their own separate objects/classes. The CPU class is private to ensure that the memory cannot access it, however, the memory class is public, because it must be runnable as a process. When the main method initializes the memory process, it is given the file to read from, and it reads the file to store the program instructions into memory. It also contains a read and write function as a supplement to CPU requests, and it handles the CPU requests by infinitely scanning its input stream until it is told to end as a process. A singular CPU object has a plethora of functions that assist it with handling the low-level concepts required by this project, such as continually reading from memory to execute program instructions (fetch-execution cycle), handling timer and instruction interrupts, pushing and popping into stacks, writing to and reading from memory through pipes, and properly entering and exiting kernel mode. With these functions, it may communicate back and forth with the memory process to allow for the user program to function properly in a simulated OS-type environment.

When working on this project, I initially decided that I wanted to do it in the C programming language. However, I did not find the way C forks processes and pipe communications to be intuitive. To me, an object-oriented design for multiple processes made more sense, therefore, I scrapped my work in C and restarted the project in Java. Having a separate file/class for different processes allowed me to be more organized with my project and helped me understand processes in a better way. Writing to an output stream to speak to another process and reading from an input stream to read from another process seemed to be a much more efficient method of process communication to me. When testing my project with the sample test cases, I was able to quickly debug my problems through simple print statements that outputted the values of certain registers such as the IR to determine the order the program is

being executed. With that output, I was able to trace my program whenever an error occurred or some unexpected output was printed. Some of the bad code that I found with my program was improper reading of the input program, wrong order of saving to the stack when entering kernel mode, and incorrect implementation of some instruction cases. Throughout my experience of developing this project in Java and debugging, I feel like I was able to create the most efficient method of simulating CPU and memory processes that I possibly could.