1. Project Purpose:
   1. This project was created to simulate a computer system, consisting of the CPU and Memory, where the CPU (aka the brain of the computer), and Memory (supplemental space for data that will be read, processed, and used by the CPU). This was used to take the knowledge about the computer system we learned in lecture and implement them in code as an inter-process communication project, in which the CPU and Memory run as two separate processes at the same time. Overall, this was a great application of knowledge about computer systems and inter-process communication programming skills, as we implemented the computer system’s components and functions in code, while using inter-process communication logic in Java or C/C++, also developing our inter-process communication programming skills (or complex programming project skills) in general.
2. How my Project was Implemented:
   1. My project was implemented in Java using inter-process communication from the build in Runtime class; and using rt.exec() on the runtime class object running a child process (which was all saved as an object of the built in Process class. The CPU class was the parent process for this project because this is the brain of a computer, or computer systems. The Memory class was the child process because all this did was listen to a command from CPU, read data and write it back to CPU, or just write data into an address from the CPU, given that CPU’s command specified an index and a value to insert in that index. I first started to program the CPU class, by calling an object of itself in its main, then running it. Before that I made sure that it took in 2 arguments from UNIX command, 1 for the filename and 1 for the timer, which the filename I added to the Memory class UNIX line for execution (“java Memory “ + filename was put into rt.exec()). I’ve then used the “Hello Greg”, where I needed a PrintWriter class to write commands to Memory, and a Scanner class to read things written back from Memory. I’ve realized I needed to make the processes cooperate, but writing back the file contents, instead of just “Hello Greg”, so then I started creating the Memory class. In the Memory class, I realized that it needed a Scanner to read commands from parent CPU, and a memory array of size 2000 for the project. Then I also called an object of itself in main, and set it to call a file input function, which printed (wrote back to parent CPU) each number eligible to be put into memory, just to test process cooperation and file reading. I went back to the CPU and read in lines written back from child Memory, which read all numbers put in memory, which meant that file is read and processes cooperated just fine. Then I realized that I needed functions in CPU to either ask to read value at an index or write a value to an index, asked by writing instruction with PrintWriter, in order to receive data and store/edit data, while Memory kept running as long as there were commands from CPU. From there, I’ve made the main loop of Memory read a command from parent CPU, then made methods to write back data at an index, or store data at an index, which would be called upon command. Also, I let the loop of Memory terminate upon command to end, assuming it only reads instructions and runs only for that purpose, and this concluded the Memory class’s main loop and its functions. Afterwards, I’ve created a function to fetch instructions and perform operatons based on instruction. I also had to implement methods to fetch instruction, and push and pop from stack (which implemented functions to read and/or write). Finally, I programmed the main loop of CPU, which would run endlessly, starting after the instruction is fetched, in which it would check if that instruction were 50 (the instruction to end), in which main loop ends. Then, it would check if time ran out, in which it would interrupt if it already did not and would reset timer counter and continue to next iteration; and then it would execute given instruction and fetch new instruction. I realized I had to create functions to go to system call (Kernel mode), whether it be interrupt or an instruction, and to return from system call (back to User mode), knowing that interrupt and some of instructions asked for such functions. After the loop, I made it write a command to Memory to end the process, in which Memory’s main loop would terminate, after which we wait for child process to finish, then printing exit code and ending parent process also.
3. My Personal Experience Doing this Project
   1. This project was hard for me at first, since I did not have a strong knowledge of inter-process communication, which should have been learned in CS 3377 (a prerequisite for this course), along with not being as attentive in lecture. However, after asking questions, looking at the example code, and knowing more about computer systems at the surface level, I was able to get started. The rt.exec() using java for inter-process communication was not hard at the surface level thanks to the example code but making file contents in memory and commands to cooperate via child and parent, as opposed to just a name and a greeting, was sure harder to implement due to more complex functions finding and creating data to pass back and forth. Reading the file, programming Memory class functions and main loop, and CPU class functions to read and write data were not as bad since they reflect theory and concepts in previous classes (e.g., CS 2 and Data Structures). Then, programming the fetch functions to execute instructions was various in difficulty, as some functions were operations that took a few lines at most, while others required more complex logic, and had to call helper functions (e.g., performing system call and returning from system call). Programming the main loop and making a system call method (for interrupt and instruction) were more challenging, since they had come edge cases I had to assume, and overall took more time to think about and debug. The hardest part of the main loop was that I had to fetch an instruction before the loop, or else the loop would be stuck infinitely. The system call was also difficult because I had to assume a case for timer interrupt and fetched instruction. For the interrupt, I realized that I had to decrement the PC because the interrupt is checked prior to instruction, in which another instruction is fetched after system call, therefore skipping an instruction fetched at a PC. For example, if instruction at PC = 5 fetched, and PC and incremented to 6 -> PC 6 had interrupt, fetch would get instruction at PC = 6, and PC would increment to 7 -> Then continue to next iteration, in which PC = 6 would execute. The problem here was that instruction at PC = 5 was never executed, but when I decremented PC, then the scenario decremented PC at PC = 6, changed to PC = 5, which let instruction at PC = 5 to execute. Also, I had many syntax errors, the most crucial ones having been using PC where I should have used AC, and not adding new line character at the end of my end execution, which had an infinite loop, despite my test case(s) having executed proper output. Overall, this project was hard to understand at first, but reviewing key concepts, breaking the project down into smaller problems facilitated my experience. However, this project was not easy, as it pushed me out of my comfort zone to understand and implement inter-process communication and operating systems concepts.