The purpose of this project was to understand how multiple processes can communicate and cooperate and to understand low-level concepts important to an operating system. Project One accomplished both goals. Pipes and forks were needed to create processes, and we had to use everything ranging from processor interaction with main memory to virtualization/emulation.

The way in which I implemented the project may or may not be different from how other people would have done it. I envisioned CPU and Memory as two classes or objects in my head with a main function as the driver of everything. So, I did just that. I translated all the problem details for the CPU and Memory into classes for each of them. The Memory class housed the memory array and the Read and Write functions. It also parsed each line of the user’s file and stored the instructions in the memory array. The CPU class housed all the registers and contained all the supported functions listed in the Project word file. Furthermore, I also gave it an additional function for running the user’s instruction file. That function would essentially iterate through the memory array and carry out whatever instruction was stored at that memory address. If time reached 0, or if an interrupt system call occurred, the CPU would go into Kernel Mode, where it would store the SP and PC registers in addresses 1000 and 1500 respectively, change the SP register to 1999 and then continue iterating through the memory array. For the main function, since I was required to use pipes and forks, I used them here. I called the pipe and fork functions to create a child and parent process. The parent process would ultimately not be used, but I used the child process to check for invalid command line arguments and hand a string vector of each line in the user’s file to the memory. In this same child process, I used pipes and forks to call another parent and child process. This parent process would also go unused; however, the new child process would take the newly formed memory object and hand it to the CPU object and run it. When finished running, the entire program would end.

I can say with complete confidence that I have been given only two programming assignments where I was legitimately unsure if I would be able to finish it in time. One was a programming assignment involving Bison and YACC. The other I can now say is this assignment. But not quite for the same reason as the Bison and YACC assignment. Where for the Bison and YACC assignment I had absolutely no clue at all how to program it due to a complete lack of tutorials and other material helpful to me, for Project One I simply underestimated how much time the program would take me to make. Creating each function was a simple task in and of itself. The problems cropped up with the pipes and forks, and when I began feeding it the test instructions. Debugging was nightmarish for me, especially for sample 2. I was forced to output every variable relevant to a called function to figure out why the smiley face was not displaying properly. For the forks and pipes, the trouble came from getting them to do what I wanted them to do. Originally, I planned to have the first child process send the memory object back to the first parent process and then have the parent process give the memory object to the CPU object and run the instruction file. After hours of trying, I determined it was not possible to send a string vector through the write function. That is why I have a child process that contains another child process. Overall, this was a good experience for me, albeit tough when you have to juggle homework from three different classes in addition to this project.