**Project 1**

For project one in Operating Systems class, we got to go deep into the process of how the processor and memory interact with each other. We did this by writing a program that emulates the behavior of a processor accessing instructions from a block of memory. In our program we had to really understand the roles that the CPU and the Memory play. For the CPU, we had to understand how the registers PC, AC, IR, X and Y interacted with each other. We had to understand what operations the Main memory performed which are read and write to its data. In addition to this we had to deploy the concept of a Stack into our program to help store our PC when an interrupt occurs or when the need to store data into the user stack, when in user mode, or the system stack , when in kernel mode. We also had to make our program handle interrupts and switch between kernel mode and user mode. When an interrupt occurs we save the PC and SP into the system stack and once that interrupt has been completed we proceed to retrieve the PC and SP from the system stack and continue to execute the instructions in the user section of the main memory.

To technically implement the project, I chose to use a blend between C and C++ I thought that these languages would be a good fit for this project since C is as low level as it gets when in comparison to assembly and C++ is just superset that allows for Object object oriented concepts to be applied. First in the program I read in two arguments from the user filename and Interrupt timer value. I then proceed to open the file and check if it is in the same directory as the program if so I begin to parse it and feed each line of the file into the main memory. Before I get to parsing the file I call the fork function to generate a child process and within the child process I parse the file into main memory.The CPU is running in the parent process and the Main memory is running the child process. The CPU and memory processes communicate via 2 way pipe communication using pipes. For the two pipes I block either the write or the read section but not both. To keep both processes alive I insert them in a while loop.

Main memory is an array of 2000 elements. Main memory is divided into two sections from elements 0-999 the user instructions are stored and the user stack resides here. The user instructions are stored in ascending order starting at element 0. The user stack is stored at 999 and grows towards zero. For the system side of main memory, the Int system instructions reside in the elements 0-500 and the interrupt instructions are in the addresses 500 - 1000. The system stack pointer points at address 1999 and grows towards 500. I use a flag variable, Kernal\_mode, if it is set to 1 we are in system mode else we are in user mode. The flag keeps interrupts from occurring when I am already executing code in the system mode.

To implement the CPU, I have a massive switch statement. Each case in the switch statement checks the value of the IR register. The cases represent the instruction that the cpu will process. For example if IR =30 we execute the 30th instruction. The registers PC, AC, X, Y and IR are represented by a variable of an object called processor\_obj and SP is an array of length two. The first element of the SP array holds the address in memory that the SP points at and the second element holds the value that will be written into that address.