**Summary**

The project will simulate a simple computer system consisting of a CPU and Memory. The CPU and Memory will be simulated by separate processes that communicate. The main objective of this project is to learn how multiple processes can communicate and cooperate. And, to understand low-level concepts important to an operating system such a processor interaction with main memory, processor instruction behavior, role of registers, stack processing, procedure calls, system calls, interrupt handling, memory protection and I/O.

The CPU consists of 6 registers namely PC- program counter, IR-instruction register, AC- accumulator, registers X and Y and a stack pointer register SP. The user program starts at address 0 and the CPU supports several instructions. The instructions are fetched to IR, and each instruction is executed before the next instruction is fetched. The user stack resides at the end of user memory and the system stack resides at end of system memory. Both will grow towards 0. The user program cannot access system memory. The memory consists of 2000 integer entries, 0-999 for user program and 1000-1999 for system code. It supports two operations which are read- reads the data from the given address and write-write the given data to the given address. The memory will be initialized by reading an input program file. A timer will interrupt the processor for every X instructions. For the timer interrupt the system code starts at 1000. There is one more interrupt, system call, for which the code starts at 1500. In both the interrupts the processor enters kernel mode. Stack pointer is switched to system stack. SP and PC are saved to the system stack. When returned form the interrupt both are restored back to their previous values.

The project is written in C language. For creating processes fork call is used. It will create two processes. The one which creates the new process is called parent process and the newly created process is called the child process. The processes will have a process Ids. The child’s process Id will be zero and the parent’s process Id will be greater than 0. Pipes are used for communication between the two processes. In this project child process acts as a main memory and parent process acts as the CPU.

The inputs for the program are given from the command line arguments. They are input test filename and the timer value. If we don’t provide any of them, the program will throw an error. There are some functions used to provide modularity for the program. The functions are

1.memoryInitialize – This function will read the input file which is given through command line arguments and stores the instructions and data which are in the file to the memory.

2. readFromMemory- This function will take address as an argument and returns the data stored in the memory at that address. It also takes processor mode as an argument to avoid accessing system code in user mode.

3. writeToMemory- This function will take address and data to be written as arguments and returns the data stored at the address. It also takes processor mode as an argument to avoid accessing system code in user mode.

4. endFucntion- This is used to end the child function whenever the End instruction is executed

5.processInstruction-This will take all registers and some flags as the arguments and processes the instruction based on those values and returns to main.

In main program, the processes are created using fork call and pipes are used for the communication between the child and the parent. In this the child process, which acts as main memory, reads the input file and stores the instructions and data in the memory. The main function of the child process is to read the required input from the parent and write whatever the parent wants from the child. It supports both read and write operations.

The parent process acts as the CPU. It will fetch instructions from main memory to IR and executes it. After the instruction is executed the program counter is incremented. It also counts the number of instructions executed. The processor will check for timer interrupt for every X instructions, where X is a command line parameter. If there is a timer interrupt, it will execute the same. Both the programs will end when the End instruction is executed.

This project tests all the 4 samples that are given as per the project. It also tests the sample5.txt file which is written by me as a part of this project.

I learned how to create processes and how inter-process communication can be established. I understand low-level concepts which are - processor interaction with main memory, processor instruction behavior, role of registers, stack processing, procedure calls, system calls, interrupt handling, memory protection and I/O. I have faced some problems while doing the project and took this an opportunity to learn new things like how to read files, how to create processes and establishing their communication like reading from and writing into pipes, construction of stack and processing of procedure calls and interrupts.