Summary of Exploring processes and IPC

Inter-process communication (IPC) is a mechanism that allows the exchange of data between processes. The IPC can be implemented as pipe, where data flow is unidirectional. A pipe is generally created by invoking the pipe system call, one of the main features of pipes is that the data flowing through a pipe is transient, which means data can be read from the read descriptor only once. If the data is written into the write descriptor, the data can be read only in the order in which the data was written.

In this project, I implemented a simple computer system consisting of CPU and memory, and the CPU and memory are implemented as two processes that can communicate with each other. I learn a lot of low-level concepts from this project, such as system calls, interrupt handling, memory protection, stack processing, how processor interact with main memory, I/O, role of registers, memory protection etc.

I use java to implement this project in linux environment. There are two file, one is CPU.java which implement class of CPU, the other file is Memory.java which implement class of memory. The memory process is created using Runtime.exec method by cpu process. The memory process can read input file by using FileReader class, store instruction and data to the memory array. There are two types of input, one is number (instruction and data) and other is dot with number (address). Pipe is created by BufferedReader to communicate with cpu. After input file is read by memory process, the instruction and data is fetched line by line to execute. 0-999 index is for user program and 1000-1999 index is for system program. BufferedReader and PrintWriter in cpu process are used to communicate with memory process. There are 31 instruction implemented in CPU class using switch-case structure. The cpu fetch instruction using PC and switch to appropriate instruction execution. A timer is implemented in cpu class, it will interrupt processor to resemble I/O interrupt and cause execution at address 1000. Call instruction pushes address to stack and jump to execute other function. Ret instruction pop address from stack and jump to execute specific function. Int instruction will cause execution at address 1500, and it will be disabled to avoid nested execution.

In the project, I learn how to design inter-process communication and I have better understanding of computer architecture basic knowledge such as interrupt, different cpu registers, system call, instruction fetch and execution flow etc. I also have more experience using Java to solve problem with OS knowledge.