Andrew Paettie
4348.002

Project Description and Purpose

The project simulates a basic computer system consisting of a CPU and Memory. It simulates the general fetch → decode → execute life cycle of program execution. The CPU and Memory will each be contained in their own process and communicate with each other over pipes.

The purpose of the project is to examine and understand interprocess communication, as well as the low-level concepts important to an operating system. The CPU will implement a simple given ISA, and the system will be designed with stack processing, procedure calls, system calls, interrupt handling, memory protection(kernel mode vs user mode), as well as I/O.

Implementation Details

In order to organize the system, there are separate objects which represent the Memory and CPU. The CPU contains several registers. Since instructions are represented in memory as integers, these registers are also integers. Also in the CPU object is methods for handling instructions, as well as communicating over the pipe. The Memory object holds an array of integers which represent the instructions (and their corresponding operands). The operands for instructions are stored 1 address past the address of its corresponding instruction. Also in the memory object is the logic to load the instructions.

The system is run from a main file, cpuMemSim.cpp. Here, the arguments are validated, objects are initialized, and then the program forks. The child process interacts with the memory object, while the parent process interacts with the CPU object. The 2 process communicate over 2 separate pipes, 1 for communication from the CPU to the Memory (cpu2Mem), and 1 for communication from the Memory to the CPU (mem2CPU).

Communication over the pipe was a point which required a little bit of thought on my part. Eventually, I settled on designing a protocol to facilitate this communication. Enumerations are defined in Protocol.h which represent the commands the CPU and memory can send over the pipe. The memory sits in a loop in which it waits for a command to be read from the CPU over the pipe. When it reads a command from the CPU (over the cpu2Mem pipe), it sends a confirmation to the CPU (over the mem2CPU pipe). After hearing this confirmation, the CPU sends an address to be read or written. If the command is for reading from the memory, the Memory will respond back with the value which resides at the address specified by the CPU. If the command is for writing to memory, the Memory responds with a confirmation, and then listens on the pipe for the CPU to write the data. I chose to have the memory respond with confirmations to ensure that the CPU and Memory communication do not get out of sync and end up reading commands out of order or anything crazy like that. That is probably unnecessary, but I like to err on the safe side when dealing with asynchronous execution.

Personal Experience

Writing this program was a pretty fun experience for me, especially when I realized that there was a need to design some kind of protocol to facilitate the interprocess communication.

My experiences in C++ in Unix and Computer Architecture from last semester helped me not only to break the system down into manageable components, but also in writing the 4th sample program. All in all, I enjoyed writing this, however given that this was assigned almost a month ago, I feel like it should have been a little bit more complex, possibly requiring caching of some kind or enabling some kind of system calls to get input from the keyboard. Also, the lack of comparator operations in the ISA made making th 4th sample program kind of a pain, which is why I choose to stick to a simpler printing program with subroutines rather than my initial intension of finding the fibonacci number of a random integer between 1 and 100. Maybe in the future adding an option to slightly expand the ISA would make for some more interesting sample programs.