Patrick Dayton

CS 5348-Prof. Ozbirn

Due 24 February 2018

# Project 1 Summary: Forks & Pipes

# Project Purpose

The project allowed me to gain a better understanding of many of the topics we’ve covered so far in the course. Creating processes established helped significantly in solidifying the somewhat abstract idea of the process.

Though very light and featureless (lucky for us), this project covered a great deal of the basic systems at play within a processor. And we could actually build cool stuff using it!

# Implementation

The project was implemented in C++. I decided to make the parent process my CPU and the child process my memory. The read/write commands in memory are housed in an infinite while loop. When the parent/CPU encounters an exit command, the parent exits which terminates the child process in turn.

To signal the memory code to whether it should read or write, I implemented a write flag. When the PC is set to -1, memory knows to accept the next two piped parameters as the value to store and the address for where to store the value.

I utilized a 3-possibility flag for interrupts to differentiate between system calls (labeled 1) and timer-interrupts (labeled 2). Label 0 is used to note that we are not in an interrupt state. An interrupt can only be called if the flag is set to 0.

Rather than try to base the timer on the actual PC value, I implemented it as a local variable that incremented each time the CPU command switch statement was called.

I utilized the decrement then write approach for the stack.

# Personal Experience

I initially attempted to write it in C, a language in which I’ve never worked, but the lack of features in that language was frustrating to the point that I made the switch to C++. At first this project was a bit intimidating, as forks and pipes are a relatively abstract process when you are used to programming without parallelization.

Once I was able to implement the piping from one process to the other, it was simply a matter of hooking up all the pieces and implementing the flags correctly. It was a bit finicky but was an enjoyable project. Seeing the smiley face and other programs show up on the screen was highly satisfying, and then actually writing a program to be compiled by your own code was really great.

# Example Program Description – DNA Pattern Recognition

For my sample program, I wanted to use a non-trivial number of the commands. I decided to create a simplified version of a DNA sequence pattern recognizing program. Using the Random number generator command, the program continuously prints either C, G, T, or A.

Using the jump comparisons, we keep track of how many Ts we see in a row. When the program finds 5 Ts in a row, it prints ‘MUTATION!’ Use of the stack was necessary to store an extra variable during the comparison process.