The purpose of this project is to show how an operating system executes multiple processes at the same time through multiprocessing. Instead of directly interacting with a system CPU, this program simulates the core functions of CPU interaction to an OS with its related registries (PC, IR, AC, etc..).

Program executions are simulated by taking user and system instructions to and from the stack on a simulated main memory. This simulated transfer of data also includes interrupts, proper buffering, and mode switching depending on if a user and system instruction is needed.

I programmed this project in C++. Some of the methods used are older C based functions, however I tried to as congruently as possible use C++ methods. I do somewhat regret not possibly writing the project in Java however. I chose C/C++ because that is the languages I used to do multithreading/multiprocessing in my Unix course. Having spent many hours trying and failing to implement the program in a ‘object-oriented’ manor I ran into too many errors and opted to have it is a giant nested main method.

All the nested core functions are rooted back to a fork between the simulated processor and memory. This splits off into three while loops divided between CPU doing reads and writes until completion, the main memory, and nested registry sub routines. My goal from this part of the code originally was to have the splitting be a class itself. I also wanted to have the repeated parts of the registry subroutines to be their own classes too. I learned the hard way splitting these into separate header files while using ongoing reads/writes/pipes is more difficult than I expected. For this reason I decided to just nest everything under its corresponding ‘parent’ function. This level of layering probably has unforeseen bugs, but that is what I ultimately landed on due to time constraints.

A lot of the code is basic combing through the given sample set with inputs and outputs. Most code related to this is largely is duplicated throughout the whole program. Hence why I wanted to turn those routines into separate classes. This being the first time I’ve programmed something with so many split actions happening at once, it has made me really respect the power of this ‘type’ of processing. It definitely hurts my brain to try to visualize the program executing in this manor versus most of my programs in the past executing in a straight line and sometimes jumping over to other classes.