Summary

These efforts to design and implement a simulated CPU and memory system were made in an attempt to break down the complexity of the relationship between the memory and processor to an atomic level where it can be better understood. By taking these systems apart and attempting to rebuild them, we are able to truly grasp the intricacy and detail that gives life to these systems. In the same stride, this project introduces the idea of multiple processing and interprocess communications. By simulating these components across separate processes, we begin to explore the realm of interprocess communication and establish these channels so that these processes can feed valuable information to each other. Together, these facets allow us to learn about how operating systems work in a way that truly internalizes them. It's hard to forget that a system call saves the context before switching to kernel mode, for example, when you've spent hours trying to ensure that it worked properly.

Designing a system that was both modular and simple proved to be a challenge. I found that this project was best approached from the vantagepoint of components. For this reason, I started by creating a Memory class and CPU class. The Memory class would be responsible for two main functions: initializing the memory via an input file, and listening and responding to cpu instructions. The first part was simple: take a file and read each line into a 2000 cell memory array (handling . jumps where necessary). The second part, while seeming quite difficult, ended up being nothing more than a continuous while loop and scanner read functions. Each instruction from the cpu would be processed and subcategorized as a read or write instruction and handled accordingly. Next up was the CPU class. This was relatively more complex, needing to be broken down into the following functions: initializing interprocess communication channel, memory permission authentication, sending read to memory, sending write to memory, instruction process, and lastly, interrupt-handling. The communication initialization would begin by running the Memory process and feeding its output into a scanner, and tying its input to a printwriter. This establishes a pipe between the two processes that allows them to talk to each other. Memory permission authentication was nothing more than a simple check on the mode when attempting to access kernel memory. Reading and writing to memory would be done by sending either a 0 for read, or a 1 for write with their following parameters to memory. The instruction processing portion was implemented via switch case that would match the instruction received from the memory to its implementation on the cpu end. Lastly, the interrupt handling portion would save context and switch the cpu from user mode to kernel mode.

Throughout this project, I faced a plethora of different emotions: excitement, frustration, relief, joy, and more. It was an absolute roller coaster. I found that the biggest struggle was trying to figure out one of the many reasons why the communication pipe was faltering. There was no easy way to figure out exactly what was going on with it and so coming across a "no line found error" generally put me in a bad mood. Otherwise, it was very exciting to watch something I

have always found too confusing to properly understand being built from the bottom up. It truly gave me an in-depth understanding of how these components work together and it unveiled a lot of the complexity behind computers. I feel as though there were curtains that were suddenly opened and I got to finally see what makes a computer work. Overall, I am proud of myself for really sticking through and overcoming the challenges. It was really exciting to see what my own hands had built after the long journey that came before it.