**Project1**

**1.Project Purpose:** The project aimed to enhance my comprehension of various course topics, including cycle instruction, memory, CPU, fork, pipes, timer, and disruption. By delving into concepts like forking processes and communication via pipes, the project played a crucial role in solidifying the somewhat abstract notion of the process. Despite the simplicity of the project, which fortunately lacked complexity, it effectively covered fundamental systems within a processor. The primary objective was to explore how an operating system functions, emphasizing the communication and collaboration of multiple processes through the CPU and Memory. The CPU, with its registers and instructional capabilities, interacts with the memory for reading and writing. The simulation involves separate processes for the CPU and memory, with the memory containing a program executed by the CPU before the simulations conclude, offering a practical and hands-on approach to understanding operating systems.

**2. How the project was implemented:** The project was implemented as a multi-process simulation of a simple operating system, emphasizing process communication and cooperation through CPU and memory interaction. The program reads an input file containing instructions and data, initializing a memory array accordingly. The child process, representing the memory and CPU, continuously awaits instructions from the parent process. The parent process, acting as the central processing unit, executes a set of commands based on the instructions fetched from the memory array. The project incorporates features such as forking processes, communication via pipes, and timer interrupts. Notable components include the implementation of system calls, stack manipulation, and interrupt handling. The system demonstrates user and kernel modes, enforcing memory protection by restricting user access to system addresses. The timer interrupt mechanism facilitates periodic interruptions, illustrating the project's versatility and providing a hands-on understanding of fundamental operating system concepts. Interrupt handling was enhanced through a three-signal system is implemented to distinguish between system calls is signal 1, timer interrupts signal 2, and a non-interrupt state is signal 0. Rather than tying the timer to the actual PC value, it was implemented as a local variable, incrementing each time the CPU command switch statement was called. The stack operation follows a decrement-then-write approach, optimizing the stack manipulation process. This design choice ensures efficient management of the stack within the overall simulation of an operating system.

**3. Personal Experience:** Working on the sample file input was by far the trickiest part of the project. Understanding how to interpret the dot, especially when it indicates the position in the array to store the next line's value, added a significant layer of complexity. Then, building a simulated operating system in C++ gave me a real hands-on experience with the complex concepts we covered in the course. Deciding to make the parent process the CPU and the child process the memory brought in an extra layer of complexity, making me ponder deeply about how these processes communicate. As I delved into implementing interrupts, especially dealing with system calls and timers, I got a solid understanding of the inner workings of operating systems in a very tangible way. Handling flags to control interrupts and coming up with a flexible timer system sharpened my troubleshooting skills. Working with stack operations, especially using the decrement-then-write method, provided a deeper insight into memory management. Though debugging and refining the code tested my patience, successfully wrapping up the project not only strengthened my grasp of OS basics but also left me feeling accomplished and geared up for more coding challenges.

**My sample file program:** In crafting the sample file for the project, I employed ASCII encoding to generate instructions that would result in printing my name. This choice allowed for a straightforward representation of characters, contributing to the clarity and useful of the instructions within the simulated operating system.