4348 Project 1 – Operating System Simulation

Summary

# Project Purpose

The purpose of the project is to understand how multiple processes can communicate and cooperate. For example, it is to understand how fork() can generate a child process with a process ID of 0, which means we can use control structures like if statements to differentiate between the parent process and child process to delegate them different tasks. Furthermore, the project is also meant to help understand how to use read() and write() to communicate between processes, parent and child.

Another project purpose is to understand the low-level concepts important to an operating system. For example, the use of registers to hold instructions and values, the interaction between CPU and memory, how stacks are used to hold memory addresses so execution can be returned to where it was left off before a procedure call and how they can hold arguments to those procedures, how system calls are made isolated from user control using interrupt handling, how a timer interrupt is handled, how system memory is protected from user access, how I/O operations are performed, and the concepts of virtualization and emulation.

# Project Implementation

1. Memory.cpp
   1. Main objectives of Memory.cpp
      1. Handle reading text files
      2. Parsing text files into instructions
      3. Storing instructions into Memory array
   2. Memory::Memory(**string** user\_prog, **int** addr) – constructor
      1. Takes user input from terminal to find and open text file
      2. Parses each line of text file correctly and stores into memory array
   3. **int** Memory::read(**int** addr)
      1. Checks for memory violation aka user access to system memory
      2. Returns element at addr in the memory array
   4. **void** Memory::write(**int** addr, **int** val)
      1. writes val to the memory array at index addr
   5. **void** Memory::set\_usr\_mode(**bool** mode)
      1. Sets the user mode
   6. **int** Memory::get\_instr(**string** str)
      1. parses lines in text files for instruction or value
      2. checks if line is empty
      3. checks if line is a new load address (e.g. .1000)
      4. returns parsed instruction or value integer
2. main.cpp
   1. **int** main(**int** argc, **const** **char** \* argv[])
      1. declare file descriptor array for pipe
      2. declare stack array
      3. declare registers
      4. declare pid
      5. declare boolean checks for user mode, system calls, timer interruption and loop control
      6. declare variables for program string input, timer input, and timer counter
      7. check for correct user input
      8. declare Memory object, read input file and create memory array
      9. pipe and error check
      10. fork process and error check
      11. while loop to ensure communication between parent and child process until instruction 50 (end) is reached
          1. cpu\_is\_on remains true until instruction == 50, so loop will continue until cpu\_is\_on is set to false under if(IR == 50)
          2. child reads from memory array and store instruction into IR register
          3. child writes IR to the parent
          4. parent reads it from child, checks if there is a timer interruption, and executes the instruction accordingly via calling cpu\_exec function
          5. timer interruption occurs when tmr\_ctr == timer value, when it happens reset tmr\_ctr to 0, check for nested interruption, set SP register to 1999, save original SP and PC values, and PC to resume execution at 1000.
          6. parent updates PC register and writes it back to child
          7. child reads the updated PC and reads from the memory array at updated index
          8. increment tmr\_ctr at every loop
   2. **static** **void** cpu\_exec(**int** &AC, **int** IR, Memory &M, **int** &PC, **int** &SP, **int** &X, **int** &Y, **bool** &cpu\_is\_on, **int** \*stack, **bool** &usr\_mode, **bool** &sys\_call, **bool** &timer\_int)
      1. Takes all the registers, interruption and system call checks, and stack as arguments
      2. Goes through if statements to find matching IR value
      3. Execute if statement body at matching IR
      4. Return to control to while loop in main
      5. When IR == 50, program ends

# Personal Experience

In the beginning of the project, I didn’t know what the registers meant so I had to look them up and learn what their functionalities were. I had a lot of trouble with the stack, because I didn’t know what the stack was supposed to do. Even after I had everything figured out, I had a lot of problems with the instructions that required jumping and going to a different memory index, because it would never return back to the correct PC value to resume execution. To solve the problem, I printed out the register values at checkpoints to debug and to edit the code in the if statements. Overall, I had a really memorable time doing this project, and I did feel like I learned a lot. There were moments where I struggled a lot, but the process gave me a sense of achievement.