SYSC 4001 A1: Part Two Report

Repository Link: https://github.com/Raghav3141/SYSC4001_A1.git

Student 1: Raghav Ramaswamy, 101310114
Student 2: Hariharan Thennarasu, 101304027

Purpose)

In part two of assignment one, teams were required to build a small interrupt system simulator to visualize different parts of the interrupt process. In this report, we will demonstrate our implementation and analyze the execution with respect to the given test cases and our own. By changing various variables in the program, the impact of those variables on overall runtime through CPU bursts, SYSCALLs, and END_IOs can be determined.

Implementation)

To implement the simulation code, the code base was forked off the given GitHub repository and the interrupts.cpp file was modified so that it was able to read specific instructions (said instructions being CPU, SYSCALL, and END_IO). Then depending on the instruction, specific steps were taken.

For the CPU instruction the given duration of the CPU burst was added to the overall timer of the simulation.

For SYSCALL and END_IO the specific delay time for the respective I/O device was obtained using the device_table.txt file. Then, the system switched to kernel mode, the context was saved, the memory of the ISR was obtained, the ISRs were processed, and IRET was executed. The ISRs are programmed so that whatever value is assigned to the ISR delay gets subtracted each time an ISR is executed in the SYSCALL or END_IO program and then whatever is left over is placed as the delay for the final instruction.

Different Starting Time)

The starting time was initially set to 0 and incremented during each activity. Upon experimentation, it was observed that the starting time had minimal influence, it only affects the time the program takes to start and has no influence on the program afterward. Additionally, positive and negative values can be used, meaning no matter how fast or slow the program begins execution it will function the same.

Context Switch Time)

Initially, the context time was set to 10 ms as instructed by the manual. When this value was increased, the CPU spent more time switching states. This is because the time change affects the program every time the context is saved. For example, if two I/O devices were used and the context time was changed from 10 to 30 ms. The program runs for an extra 40 ms. Additionally, this change is felt greater when more devices are used, and vice versa. Furthermore, increased context switch time means more overhead for the CPU.

Varying ISR Activity Time)

By default, the ISR delay is set to 40 ms, but when it is increased to 200 ms it drastically affects the runtime of the simulation. Especially in cases where the trace file contains many system and I/O calls. This is first because CPU bursts don't use interrupts so changing the ISR delay doesn't affect the CPU burst runtime. However, when running the system calls (which used 2 ISRs) and the I/O operations (which used 3 ISRs), the change in ISR delay value causes considerable changes to their overall delay.

Selectively Choosing I/O Devices)

Since each I/O device has its own assigned overall delay time when it runs, by having a majority of fast system and I/O calls, the overall speed of the system decreases quite a bit. The opposite happens when there is a large majority of slow-running I/O and system calls. The runtime increases quite a bit since each system call and END_IO operation takes so long.

Conclusion)

The difference in speed when it comes to changing the context and changing the ISR is quite different. While changing the amount of time it takes for the CPU to switch contexts does result in different runtimes, changing the ISR's activity time affects the system far more. This is because the rate at which the ISR changes is much more significant (going from 40 ms to 200 ms is a difference of 160 ms while for the context it's only switching from 10 ms to 30 ms which is a difference of 20 ms). Furthermore, there is only one context switch per system call/END_IO whereas there are multiple ISRs for both operations.