Assignment One Report

Course: SYSC4001A – Operating Systems
Lab Section: L3
Group: 8

Group Members: Noor Karabala, Marwan Alakhras

Instructor: Gabriel Wainer

Submission Date: October 5, 2025

Introduction

In this assignment, we made a simulator to mimic how a computer handles interrupts. The program takes an input trace of CPU activity, system call, and device requests. It prints out a timeline that indicates every phase of the interrupt process. This includes normal CPU bursts, and also the specific actions taken in the event of a system call or completion of an I/O operation. By running the simulator on different input files and parameters, the effects of context saving and ISR run times varying on the overall performance of the system can be measured.

Methodology

The simulator was implemented in C++ based on the template provided within the assignment. The simulator accepts three input files: a trace file, a vector table, and a device table.

We created 4 trace files to execute different scenarios. io_focus.txt focusses on I/O activity, balanced.txt balances CPU processing and I/O events, multi_device.txt contains interrupts from a range of devices, and long_loop.txt simulates a longer alternating program loop. The vector table provides ISR addresses, and the device table initiates service times for every device.

During simulation, a burst of computation was added to each CPU entry in the trace file. SYSCALL and END_IO entries included switching to kernel mode, saving context, locating the ISR in the vector table, loading the address into the program counter, executing the ISR for the time specified in the device table, and finally returning with an IRET instruction.

The ISR is divided into smaller steps to show what happens during an interrupt. For a SYSCALL, the steps are: running the ISR (device driver), transferring data from the device to memory, and checking for errors. For an END_IO, the steps are: running the ISR (device driver) and checking the device status.

Experiments

We ran 20 test cases on four different trace files. Our experiments were divided into two groups. In the first group, we modified the context save time (10, 20, and 30 ms) by changing its parameter in the code while keeping ISR delays from the given device table. In the second group, we fixed the context save time to 10 ms and changed the total ISR delay to 40, 100, and 200 ms by editing the device tables.

To test different ISR durations, we adjusted the code so that the total time of the ISR steps (three steps for SYSCALL and two steps for END_IO) added up to the selected delay value. This was done using separate device tables: device_table_40.txt (testing 40 ms), device_table_100.txt (testing 100 ms), and device_table_200.txt (testing 200 ms). These changes allowed us to compare how different total ISR execution times affected the overall performance.

Table 1: Effect of Context Save Time (10 cases)

Trace file	save/restore context (ms)	ISR delay (ms)	Total time (ms)	Execution file
io_focus.txt	10	given device table	679	execution(test_1).txt
io_focus.txt	20	given device table	719	execution(test_2).txt
io_focus.txt	30	given device table	759	execution(test_3).txt
balanced.txt	10	given device table	700	execution(test_4).txt
balanced.txt	20	given device table	740	execution(test_5).txt
balanced.txt	30	given device table	780	execution(test_6).txt
multi_device.txt	10	given device table	1753	execution(test_7).txt
multi_device.txt	20	given device table	1833	execution(test_8).txt
long_loop.txt	10	given device table	750	execution(test_9).txt
long_loop.txt	30	given device table	830	execution(test_10).txt

Table 2: Effect of ISR Delay (10 cases)

Trace file	save/restore context (ms)	Total ISR delay (ms)	Total time (ms)	Execution file
io_focus.txt	10	40	245	execution(test_11).txt
io_focus.txt	10	200	885	execution(test_12).txt
balanced.txt	10	40	266	execution(test_13).txt
balanced.txt	10	100	506	execution(test_14).txt
balanced.txt	10	200	906	execution(test_15).txt
multi_device.txt	10	40	457	execution(test_16).txt
multi_device.txt	10	200	1737	execution(test_17).txt
long_loop.txt	10	40	316	execution(test_18).txt
long_loop.txt	10	100	556	execution(test_19).txt
long_loop.txt	10	200	956	execution(test_20).txt

Results

Based on the first set of tests, we noticed that adjusting the context save time from 10 ms to 20 ms and to 30 ms always contributed to increased total execution time. The increment was roughly linear for all traces. The effect was small for CPU-balanced traces, but much larger in traces with more interrupts, like multi_device.txt and long_loop.txt. As every interrupt takes CPU context saving, traces with heavy I/O interrupts demonstrate the largest variation.

For the second experiment, in which we changed the ISR delay and maintained the context save time as a constant, the overall time difference was much larger than the first set of tests. Raising the ISR delay to 200 ms from 40 ms doubled the total execution time. In io_focus.txt and multi_device.txt, where interrupts dominate, longer ISR times were most impactful. This means that when interrupts are of high frequency, lengthening the ISR run time imposes great overhead and reduces CPU time available for regular execution.

Additional Observations

Using 4-byte addresses instead of 2 would slightly increase the vector lookup step but have little effect overall. A faster CPU would finish bursts quicker, making interrupt overhead more noticeable compared to useful work.

GitHub repository link: https://github.com/NoorTamim/SYSC4001_A1.git