Assignment 1: Interrupt Handling Analysis Report

Esli Emmanuel Konate 101322259 and Nitish Grover 101324174

Carleton University

SYSC 4001 : Operating Systems

Introduction

Today’s operating systems make use of interrupt mechanisms to handle the asynchronous events from the hardware devices in an efficient manner. It is crucial to understand the way performance works in interrupt handling and this especially for the design of systems and their optimization. This report analyzes all the characteristics linked to the performance of a simulator system driven by interrupts. We made use of various parameters affecting the way the overhead of interrupt handling.

The main objective of this analysis is to help measure how different parameters, especially in the context switch time, the ISR execution time, and the device I/O delays can impact the overall system execution time. By directing these experiments with multiple tests, it provides knowledge into the exchanges/differences between the efficiency of interrupt handling and the performance of the system.

HOW

The simulator resembles a simplified version of an operating system with components synch as:

- Mode switching: It can switch in between user and kernel modes (1ms)
- Context saving and restoring : It can save CPU state during the interrupts (10ms, 20ms, 30ms)
- Lookup of the vector table: It can find ISR addresses (1ms)
- PC Loading : It can Load the ISR address into the PC (1ms)
- Execution of ISR: It can run interrupts which are made by devices (40ms, 60ms)
- It has device operations: Like I/O data transfer and also error checking with various delays in the delay table.

We created 4 distinct test files (trace files) to represent and test different ways the CPU might be used:

- There is trace.txt, which is the normal workload with various CPU and I/O activities
- There is trace_test2.txt: Which represents more of an intensive workload by I/O with multiple interrupts.
- There is also trace_test3.txt: Which is more intensive on the CPU and it has more and longer bursts in computation.
- Finally, there is trace_test4.txt, which has an average balanced workload with a medium amount of interrupts.

In addition to all of this, we switched very precise parameters in order to conduct our experiments.

- First, we changed the switch time: So we tested with 10ms first, then we moved on with 20ms and finally we stopped at 30ms context save/restore times.
- Second, we changed the ISR execution times and compared 40ms which was our normal baseline with 60ms ISR execution.
- Finally, in third, we changed the device delays by doubling the delays seen in the original table.

Every single category of tests were run on all 4 of our previously mentioned test files and it resulted in an amount of 20 simulation runs.

Results

Context Switch Time Impacts

Test File	10ms Context (ms)	20ms Context (ms)	30ms Context (ms)	Overhead Increase (%)
Trace.txt	~ 1554	~ 1614	~ 1674	7.72
Trace_test2.txt	~ 1313	~ 1413	~ 1513	15.23
Trace_test3.txt	~ 2290	~ 2350	~ 2410	5.24
Trace_text4.txt	~ 1995	~ 2095	~ 2195	10.03

In this table it is possible to see that tripling the context switch time increases the total execution time from 5 to 15% In addition, the impact will greatly vary based on the workloads. To add, trace_test2.txt has the highest increased percentage (15.23%). This means that it had the most recurring interrupts. Finally, trace_test3.txt had the smalles increased percentage (5.24%). This means that it had the least amount of context switches.

ISR Execution Time Impacts

Test File	ISR 40ms (ms) **Baseline**	ISR 60ms (ms)	Time Increase (%)
Trace.txt	~ 1554	~ 1674	7.72
Trace_test2.txt	~ 1313	~ 1513	15.23
Trace_test3.txt	~ 2290	~ 2410	5.24
Trace_text4.txt	~ 1995	~ 2195	10.03

Here in the table, it is possible to see that increasing the ISR time by 50% (from 40ms to 60ms) produces a total time percentage increase from 5 to 15%. In addition, the results in this table resemble greatly that of the context switch table results, this in turn, confirms that our results are realistic. Finally, each of the parameters adds around the same amount of overhead per interrupt which is 20ms. Which confirms the 50% increase in ISR time.

Device Delay Impacts

Test File	Normal Delays (ms)	Doubled Delays (ms)	Time Increase (%)
Trace.txt	~ 1554	~ 2748	76.83
Trace_test2.txt	~ 1313	~ 2533	92.92
Trace_test3.txt	~ 2290	~ 3010	31.44
Trace_text4.txt	~ 1995	~ 3135	57.14

In this table, the device delays have a large impact on the context and ISR overhead. The time increases from a large range of 31% to 93%, and the average increased time percentage is 65%. In addition, the trace_test2.text file had the highest impact with 92.92%. This confirms the fact that the I/O is really intensive. Finally, even the CPU trace_test3.txt fill still has an increase of 31% which shows the importance of the I/O.

Analysis

After conducting those tests, we have found three key points about the speed of interrupt handling and how it affects the performance of a system:

First, the context switch time and ISR execution time both increase in a linear way with the total execution time. Therefore, adding 20ms per interrupt ends up making the same percentage increase across the different workloads. This confirms the fact that our simulator is correct and also showcases that the overhead of the interrupt is somewhat proportional to the interrupt frequency.

Second, different workloads can showcase different ways they will react to interrupt parameters. Workloads which are I/O intensive like trace_test2.txt have more context switch and ISR delays because they have to process more interrupts per time. In addition, workloads which are intensive for the CPU like trace_test3.txt have less of an impact, because they have less interrupts.

Finally, with the device delay experiment, we found out that I/O operations essentially control the performance of a system way more than that of an interrupt handling overhead. When we tripled context switch time, it only added from 5 to 15%, but in contrast, doubling the I/O delays added from 31 to 93%. This shows that in real systems, it is important to improve the speed of an I/O to have the best performance possible and it is way more efficient than just optimizing the inner mechanisms of interrupt handling.

How does the difference in speed of these steps affect overall execution time?

The speed of the steps in interrupt handling affects the execution time in a proportional manner to the frequency of interrupts. Each time a millisecond is added, the context switching or ISR execution multiplies through all the interrupts in the workload. But, it is not as effective as the speed of the I/O device. In addition, for the workloads with 6 interrupts per 100ms of time of CPU as seen in trace_test2.txt, the increase of 20ms adds an extra 120ms of overhead per 100ms of CPU execution. This also is connected to the increase of 15% . While at the same conditions, the I/O delays can do twice or even triple more of that total execution time. This explains why they are the primary concern in terms of performance. So in other words, a faster interrupt handling can improve the response time and also reduce overhead, but a faster I/O device can improve the throughput in a more significant manner. Therefore, a good system will need both optimization, but it has to first prioritize a good/fast I/O performance if it wants better/faster outputs.

Conclusion

In this analysis, we measured the performance impact of parameters in interrupt handling in a simulation of an operating system. We found out that the context switch overhead increases in a linear way with the frequency of interrupts and could add from 5 to 15% execution time when it is tripled. In addition, ISR execution time has a similar impact to context switching which confirms the validity of our simulation. In third, the device I/O delays have a significant control over performance, it contributes way more on overhead than the mechanisms of interrupt handling. Also, I/O workloads are highly dependent on parameters linked to interrupts in terms of time percentage increase. Our final results showcase that even though interrupt handling is important to reduce CPU waste in a system, the most significant attribute of performance is the speed of the device itself. So, in order to be as efficient as possible, today’s operating systems have to optimize both the interrupt handling parts, but also the I/O itself to reach a maximal throughput.

KEY to the repository : https://github.com/J3Sli3/SYSC4001/tree/main/A01