SYSC 4001 Assignment 1 Part II Report

Nate Babyak – 101310590
Ozan Kaya – 101322055

Overview

This report summarizes the results of numerous simulation tests conducted to understand the influence of the different steps of the interrupt process.

Methodology

We created 20 input trace files to thoroughly test the interrupt simulator under various conditions. Each trace file contained a mix of CPU bursts, SYSCALL operations, and END_IO events to simulate realistic program execution patterns. Our baseline trace file contained 40 lines (20 interrupt pairs) with varying CPU burst durations between 20-100ms and device numbers ranging from 0-19.

For each test category, we systematically varied a single parameter while keeping others constant, allowing us to isolate the effect of each component on system performance. All simulations used the provided device_table.txt with device delays ranging from 68ms to 1000ms.

The baseline configuration uses:
- Context save/restore time: 10ms
- ISR activity time: 40ms per activity
- Vector size: 2 bytes
- Standard CPU speed

Each test category isolates a single variable to understand its specific contribution to system performance.

Findings

Influence of changing the value of the save/restore context time

Using the first trace file as input, we ran simulations while changing the save/restore context time to 10, 20, and 30 ms, obtaining total program durations of 3822, 3922, and 4122 ms, respectively. These results show that increasing the save/restore context time increases total execution time linearly. This effect is directly related to the number of interrupts that occur, as each interrupt requires a context save operation.

Increasing the save and restore context time from 10 ms to 20 or 30 ms causes each interrupt to take longer, which slightly increases both the cycle time and total program execution time. Longer context switching also keeps the CPU busy for a greater portion of the interrupt period, thereby reducing the opportunity for other tasks or low-power modes. Overall, higher context times reduce system efficiency, particularly when interrupts occur frequently.

Influence of varying the ISR activity time

Using the second trace file as an input we varied the ISR activity time from 40ms to 200ms.

The ISR activity time has a multiplicative effect on total execution time because different interrupt types perform different numbers of activities:

- SYSCALL: Performs 2-3 activities (run ISR, transfer data, check errors)
- END_IO: Performs 1-2 activities (run ISR, check device status)

The total time contribution from ISR activities is:
Total ISR overhead = (number of SYSCALLs * activities per SYSCALL + number of END_IOs * activities per END_IO) * ISR activity time

If ISR execution time becomes too long, they create several problems:

- Interrupt latency increases, other devices must wait longer for service.
- CPU remains in kernel mode, user programs cannot execute.
- The system becomes interrupt-bound and spends more time handling interrupts than doing useful work in a meaningful amount of time.

When ISR time exceeds approximately 100-150ms per activity, the system transitions from compute-bound to interrupt-bound, meaning the majority of execution time is spent in interrupt handling rather than actual computation.

Key Insight: ISR execution time is the dominant factor in interrupt-driven system performance. Keeping ISRs lean (< 50ms) is critical for maintaining system responsiveness and efficiency.

What happens if we have addresses of 4 bytes instead of 2?

In our simulation model, changing the address size will have no measurable impact on execution time because:

- Memory access time is constant: We model “load address from memory” as taking exactly 1ms regardless of address size.
- Vector calculation takes negligible time whether the vector size is 2 bytes or 4 bytes long.

The only tangible difference is memory overhead:

- 2-byte vectors: Vector table size = N × 2 bytes
- 4-byte vectors: Vector table size = N × 4 bytes

Despite the memory overhead, modern systems use 4-byte (32-bit) or 8-byte (64-bit) addresses because of:

- Address Space: 2 bytes can only address 64KB of memory (2^16), while 4 bytes can address 4GB (2^32).
- System Architecture: 64-bit systems require 8-byte pointers for memory addressing.

Some real-world considerations (not in our simulation) in actual hardware, larger addresses might affect:

- Cache Performance: Larger vector tables may span more cache lines.
- Memory Bandwidth: Loading 8 bytes instead of 2 bytes consumes more bus bandwidth.

Key Insight: Address size represents a trade-off between memory overhead and addressable space. In terms of execution time, the impact is minimal because memory access speeds are fast relative to other interrupt processing steps. The choice of address size is driven by addressing requirements, not performance.

What if we have a faster CPU?

If we have a faster CPU, the total execution time will decrease. The amount of reduction depends on the length of the input trace files and the extent to which the processor speed is increased.

This could be tested by creating two, or more, identical trace files but changing the duration of the CPU burst execution times to simulate a faster CPU. Performing these tests yield the previously predicted results, with reduced CPU burst times leading to shorter overall execution times. However, the speedup achieved is not proportional to the CPU speed increase.

For example, if we simulate a 2× faster CPU by halving all CPU burst durations in the trace file, the total execution time does not decrease by 50%. This is because a significant portion of execution time is consumed by interrupt handling overhead (context switching, ISR execution, and device delays), which remains constant regardless of CPU speed.

Key Insight: In I/O-intensive programs with frequent interrupts, the interrupt overhead dominates total execution time. This demonstrates that in interrupt-driven systems, simply upgrading to a faster processor provides diminishing results, and optimization efforts should focus on reducing interrupt frequency or interrupt handling efficiency rather than solely relying on increased processor performance.

GitHub

https://github.com/natebabyak/SYSC4001_A1