Interrupt Simulation Trace Analysis Report

Team members: Manoj Kuppuswamy Thyagarajan (101166589)
Shabesa Kohilavani Arunprakash (101258619)

Introduction

This report presents a performance analysis of an interrupt-driven system simulator. The simulator models CPU execution, system calls (SYSCALL), and interrupt handling (END_IO) to evaluate the overhead introduced by interrupt mechanisms in operating systems.

How it works

All traces in this analysis follow a pattern similar to an interrupt-driven one. Each trace logs show these events:

- CPU execution intervals: Periods when the processor executes normal user-mode application code
- Interrupt triggers: SYSCALL (initiated by software) or END_IO (device I/O completion) events
- Mode transitions: Switches between user mode and kernel mode
- Context management: Saving and restoring CPU state to preserve program integrity
- Vector table operations: Memory lookups to locate interrupt service routine addresses
- ISR execution: Device-specific interrupt handling code execution
- Return mechanisms: IRET instructions and mode switches returning to normal execution

Common Pattern Across All Traces

Typical Interrupt Handling Sequence

Every interrupt in all traces follows this identical 8-step sequence:

1. CPU execution - Normal user-mode processing (duration varies: 10-100 time units)
2. Switch to kernel mode - Privilege escalation (1 time unit)
3. Context saved - Store CPU registers and state (10 time units)
4. Find vector - Lookup interrupt number in vector table at memory address (1 time unit)
5. Load ISR address - Retrieve interrupt handler address into program counter (1 time unit)
6. Execute ISR - Run device interrupt handler (110 time units for device 0)
7. IRET - Execute interrupt return instruction (1 time unit)
8. Context restored - Reload saved CPU state (10 time units)
9. Switch to user mode - Return to normal privilege level (1 time unit)

General Trace Characteristics

Workloads

- CPU bursts range from 10 time units to as long as 100 time units
- Some traces have longer initial CPU execution periods (75-95 units)
- Others start with shorter bursts (49-73 units), which show different application behaviors
- It reflects real situations where applications have different computational demands between I/O operations

Consistent Interrupt Handling

Despite workload differences, all traces are maintained:

- Identical 110-unit ISR execution times for device 0 operations
- Fixed 26 units of overhead per interrupt (context switches, mode transitions, vector lookups)
- Predictable interrupt handling latency regardless of CPU burst patterns
- Equal treatment of both SYSCALL and END_IO interrupts for the same device

Interrupt Frequency Patterns

Different traces show interrupt frequencies:

- Traces with shorter CPU bursts between interrupts show more interrupt density
- Longer CPU execution periods result in fewer interrupts per unit time
- The nature of operations (SYSCALL followed by END_IO) appears consistent
- Equal numbers of SYSCALL and END_IO events across all traces

Important Observations Traces

Consistent Interrupt Overhead

Every single interrupt incurs exactly 26 time units of overhead:

- Context save: 10 units (38% of overhead)
- Context restore: 10 units (38% of overhead)
- Mode switches: 2 units (8% of overhead)
- Vector lookup and PC load: 2 units (8% of overhead)
- IRET: 1 unit (4% of overhead)
- ISR execution: 110 units (additional device-specific work)

This shows that 19% of each interrupt's total time (26 out of 136 units) is overhead, providing no functional value.

Impact Caused by the Interrupt

Traces with shorter CPU bursts have more frequent interrupts, resulting in:

- Higher cumulative overhead as a percentage of total execution time
- More context switches use CPU cycles
- High memory traffic from state load and stores

- Lower system throughput despite the same amount of actual work

Device I/O Pattern

All traces show the paired SYSCALL/END_IO pattern:

- SYSCALL initiates an I/O operation (e.g., read from a device)
- CPU continues executing during I/O operation (simulated as immediate)
- END_IO signals I/O completion, triggering another interrupt
- Both interrupts for the same device use identical ISR code (110 units)

This pattern is similar to that of a real operating system, where I/O operations are asynchronous, allowing the CPU to remain active while peripherals operate independently.

System Responsiveness

The traces also show the interrupt handling latency components:

- Immediate response: 1 unit to switch to kernel mode
- State preservation: 10 units to save context
- Handler dispatch: 2 units to locate and invoke ISR
- Total latency to ISR start: 13 time units from the interrupt signal

This 13-unit latency represents the minimum time that is needed for device-specific code to begin executing, a very important detail for real-time system responsiveness.

What Can be Learnt from the Simulation

1. The simulation illustrates many crucial operating system concepts:
2. Privilege separation: Clearly distinguishing between kernel and user modes
3. Preservation of the state: Program accuracy is ensured by context save/restore.
4. Vectoring interruptions: Flexible handler assignment is made possible by a memory-based dispatch table.
5. Asynchronous I/O: Using an interrupt-driven design to overlap processing and I/O
6. Costs of overhead: Impact of interrupt infrastructure on performance that can be measured

Conclusion

The crucial balance between kernel-mode interrupt handling and user-mode processing is evident in all simulation traces. The steady timing patterns throughout traces confirm the accuracy of the simulator and offer a fundamental comprehension of interrupt overhead expenses. The goal is the same for all traces, regardless of how long or short the CPU bursts are: interrupt-driven systems give up raw computing efficiency in exchange for responsiveness and device control.

Github Link: https://github.com/shabesa/Assignment-1-L1-5