SYSC4001 - Assignment 1 (Part II): Interrupts Simulator
Ibrahim Komeha 101308485,
David Mea 101297581
Date: Oct 6, 2025

We have built a small simulator that turns a simple program trace (CPU bursts and I/O events) into a detailed timeline of what actually happens during interrupts. Each interrupt expands into six steps: switch to kernel mode, save context, find the vector, load the ISR address, run the ISR body, and return. We have verified the format and the timing math against the assignments example, then we ran 20 tests by changing context save time, ISR chunk size, and vector entry size.

Inputs:

- Trace.txt - a sequence like 'CPU, 50', 'SYSCALL, 14', 'END_IO, 14'
- Vector_table.txt - ISR address per device (device 14 -> 0X0165)
- Device_table.txt - total ISR time per device in ms (device 14 -> 456)

How time was modeled:
Kept a single clock 't'. For CPU, D, log one line "CPU Burst" and move time by D. For any interrupt (SYSCALL or END_IO) it is expanded into:
1- switch to kernel mode (1ms)
2- context saved (10ms baseline, later 20 and 30)
3- find vector for the device at ADDR_BASE + dev x VECTOR_SIZE (1ms)
4- load address <vectors[dev]> into the PC (1ms)
5- ISR body that sums exactly to delays[dev]
6- IRET (1ms)

Check for errors gets the remainder:
The assignments gives us only a total ISR time per device, not per sub-step. So we split the SYSCALL body as 40ms "run driver", 40ms "transfer" and remainder = check errors. For END_IO' it's 40ms " run driver ", remainder "check status". This guarantees the parts add up to the device's total delay.

Test Cases:
We have run over 20 test cases to ensure structural integrity that can be found within the github repository.

Repository:
https://github.com/itsdavidmea/Assignment-1-4001
Change the value of the save/restore context time from 10, to 20, to 30ms. What do you observe?
for 10 ms

```
int time_of_event = 0;
int context_save_time = 10;
int previous_CPU_time;
int isr_activity_time = 40;
```

```
0, 51, CPU burst
51, 1, switch to kernel mode
52, 10, context saved
62, 1, find vector 14 in memory position 0x001C
63, 1, load address 0X0165 into the PC
64, 40, SYSCALL: run the ISR (device driver)
104, 40, transfer data from device to memory
144, 1, IRET
```

After the CPU burst, the kernel switch happens at t=51, context save takes 10ms (52 -> 62), then we do a vector lookup, load the ISR address, run the ISR (40ms), transfer (40ms) and IREST at t=144. Which shows that a 10ms context time gives the shortest interrupt path.

for 20 ms

```
int time_of_event = 0;
int context_save_time = 20;
int previous_CPU_time;
int isr_activity_time = 40;
```

```
0, 51, CPU burst
51, 1, switch to kernel mode
52, 20, context saved
72, 1, find vector 14 in memory position 0x001C
73, 1, load address 0X0165 into the PC
74, 40, SYSCALL: run the ISR (device driver)
114, 40, transfer data from device to memory
154, 1, IRET
```

Same steps but the context save is now 20ms (52->72). Every downstream timestamp shifts but +10ms and IRET moves to t=154.

for 30 ms

```
int time_of_event = 0;
int context_save_time = 30;
int previous_CPU_time;
int isr_activity_time = 40;
```

```
0, 51, CPU burst
51, 1, switch to kernel mode
52, 30, context saved
82, 1, find vector 14 in memory position 0x001C
83, 1, load address 0X0165 into the PC
84, 40, SYSCALL: run the ISR (device driver)
124, 40, transfer data from device to memory
164, 1, IRET
```

context save is 30ms (52->82), vector lookup / load happen later and IREST slides to t=164. Again linear growth, the interrupt finishes about 20ms later than the 10ms case. Longer context handling directly stretches the response time and the completion time.

Vary the ISR activity time from between 40 and 200, what happens when the ISR execution takes too long?

for 200 ms

```
22667, 48, CPU burst
22715, 1, switch to kernel mode
22716, 30, context saved
22746, 1, find vector 4 in memory position 0x0008
22747, 1, load address 0X0292 into the PC
22748, 200, ENDIO: run the ISR (device driver)
22948, 1, IRET
22949, 63, CPU burst
```

Once in the ISR, the kernel stays busy for 200ms straight, IRET at t=22948. A long ISR utilises the CPU. Everything else waits, in a real system this may risk missed deadlines.

for 100ms

```
13767, 48, CPU burst
13815, 1, switch to kernel mode
13816, 30, context saved
13846, 1, find vector 4 in memory position 0x0008
13847, 1, load address 0X0292 into the PC
13848, 100, ENDIO: run the ISR (device driver)
13948, 1, IRET
13949, 63, CPU burst
```

ISR body is 100ms, IRET at t=13948. Halving ISR time roughly halves the in kernel hold time, the CPU returns to user work sooner.

for 70ms

```
11097, 48, CPU burst
11145, 1, switch to kernel mode
11146, 30, context saved
11176, 1, find vector 4 in memory position 0x0008
11177, 1, load address 0X0292 into the PC
11178, 70, ENDIO: run the ISR (device driver)
11248, 1, IRET
11249, 63, CPU burst
```

ISR body 70ms, IRET at t=11248. Shorter ISR leads to shorter total interrupt latency which leads the system to be more responsive.

for 40ms

```
8427, 48, CPU burst
8475, 1, switch to kernel mode
8476, 30, context saved
8506, 1, find vector 4 in memory position 0x0008
8507, 1, load address 0X0292 into the PC
8508, 40, ENDIO: run the ISR (device driver)
8548, 1, IRET
8549, 63, CPU burst
```

ISR body 40ms, IRET at t=8548. This is the best of the cases, minimal time spent in kernel, fastest return to user code.

what happens when the ISR execution takes too long?

When ISR time grows large, the CPU is in kernel mode longer, that pushes back the work (CPU bursts and other interrupts), increases I/O completion time and cause pressure in systems that receive frequent interrupts.

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

Raising context save by +10ms adds +10ms per interrupt, raising ISR body by +30ms adds +30ms per interrupt. Over many interrupts, these delays add up quickly and significantly increase total run time.

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

If we model the fetch time proportional to address size, loading a 4-byte ISR address would take about two times the load address step. In the traces that step is 1ms so it would be roughly 2ms, which is tiny per interrupt but will be noticeable over many interrupts.

What if we have a faster CPU

If CPU gets faster but the interrupt steps stay the same, more time is wasted on interrupts compared to the actual work, the system would spend more time handling interrupts instead of running the main program.