Interrupts*

Maya Hussein
Department of Computer Science & Engineering
The American University in Cairo
Cairo, Egypt
mayamakram@aucegypt.edu

***Abstract*—As technology evolves, new additions are added into computing systems; a part of that is the addition of interrupts. This paper aims to discuss what are interrupts and their different types, along with how are they handled in the operating system, and what are their potential benefits and drawbacks on the systems performance.**

***Index Terms*—Interrupts, Hardware, software, I/O handling, context switching, Synchronous, Asynchronous, Latency, IRQ, CPU**

I. Introduction

During the initial development of computing systems, processors had to validate every hardware and software program to ensure the presence of an signal in need of processing. However, this method can load the processor for a large number of cycles [1]. An interrupt, on the other hand, raises a flag signal that has immediate priority by the processor, whether indicated by the hardware or the software [7].

When a computer is running multiple programs and processes simultaneously, interrupts use a prioritization scheme to decide which tasks to perform first. Interrupts are essential for a computer to function properly and efficiently. Without interrupts, a computer would be unable to handle multiple tasks at the same time, and would be unable to respond to external events in a timely manner [1]. This paper will discuss the different types of interrupts, interrupt OS handling, the benefits and disadvantages of each, some use cases, and its effect on the CPU performance.

II. Background

To bypass and update the initial development, a more efficient mechanism was introduced that sends a signal from the hardware or software to the processor instead of forcing the CPU to check for a signal every single time. During this interrupt, the signal notifies the processor that it needs to suspend its present scheme by saving its recent state, and proceed with this interrupt with high priority, which is known as Interrupt Service Routine (ISR) [4]. The CPU afterwards resumes its normal activity by loading back its state and proceeding with its normal functionalities. For instance, as a keyboard key is pressed, a signal is instantiated asking the CPU to send this data to the present activated program, moving the cursor one step to the right. When the key is depressed, the interrupt service routine will interrupt the ”key down” state, allowing the cursor to remain still. This was an example of a hardware interrupt [6]. Interrupts come in many types and categories, each designed to serve a specific task.

III. Related Work

Interrupts are divided into main types: Hardware interrupts and software interrupts. Hardware interrupts are signaled by an external driver, such as keyboard or a printer, where they are sent to their aligned hardware line [3]. In contrast, software interrupts are signals generated by the CPU itself when needed to perform activities related to I/O operations or system functions. Interrupts are managed by a programmable interrupt controller (PIC) that is used to determine the priority of each request while managing the flow of data between the external drivers and the processors [6]. This technique allows tasks to be serviced promptly.

A. Hardware Interrupts

Hardware interrupts in general are hardware signals, wired connection to the CPU. An I/O component uses that signal to notify the CPU that an event has occurred [6]. The processor in return responds to the hardware interrupt by halting any current activities by doing the following:

1) The CPU saves the current processor's state specifically, the Code Segment, the Instruction Pointers, and Flags along with the values of the registers and the program counter.
2) It clears the Trap Flag TF and the Interrupt Flag IF
3) It then retrieves the interrupt vector, a value that specifies the location of the interrupt service routine (ISR) for the device that sent the interrupt.
4) The CPU executes the instructions in the service routine.
5) Lastly, after the routine is complete, the CPU restores the saved state of the program, which was previously interrupted.

This process allows the system to keep track of the data and move forward with its program after noticing the flag without losing any of its data contents. These interrupts are managed by the PIC, which is typically a part of the motherboard's chipset and is responsible for handling all the hardware interrupts for the system [6].

There are two main types of hardware interrupts: maskable interrupts and non-maskable interrupts.

1) Maskable Interrupts: A maskable interrupt is a type of hardware interrupt that can be temporarily ”masked” or ignored by the processor of a computer allowing it to continue executing its current instructions without being interrupted by devices that do not require an immediate response, such

as a keyboard or a mouse [5]. The CPU can disable such interrupts when it is executing a critical task or handling another functionality. The CPU then unmasks these interrupts when its has finished its vital activities and is ready to handle that request [2]. The following procedure takes place in detail when a maskable interrupt takes place:

1) A signal is sent to the PIC
2) The PIC then asks the CPU for permission to interrupt the current instruction
3) If the interrupt is able to handle the interrupt, it will acknowledge the steps mentioned above. Otherwise, it will continue performing its current operation while clearing the IF bit in the Flags Register $setsIF = 0$ and masks the interrupts at the processor.
4) The processor in return stops monitoring that line while IF = 0

2) Non-Maskable Interrupts: Non-maskable interrupts (NMI), on the other hand, cannot be disable by the CPU as they require immediate attention, such as a power or system failure. The CPU must pay attention to this interrupt immediately by halting any present operations in the CPU [5]. Non-maskable interrupts, on the other hand, cannot be masked or disabled by the CPU. These events are normally handled by a hardware signal or a button that is used to trigger a non-maskable interrupt, which grants the processor the ability to respond in a precise and efficient manner [3].

1) When an interrupt is triggered, the processor saves all its present contents as well in the stack, including the instruction pointer, the stack pointer, the flags register, and other processor-specific registers. Saving the instruction pointer allows the CPU to resume executing the interrupted program after the NMI handler is complete.
2) While the NMI handler is running, the stack pointer grants the processor the ability to access the data on the stack, like the saved environment.
3) Lastly after the handler has finished the execution, The contents of the flags register allows The processor then restores its previous state using these flags as they indicate the current state. These flags include the carry flag, the zero flag, the interrupt flag, and others.

[2] A few examples of hardware interrupts include:

- A disk sending an interrupt for completing a read/write operation.
- A network interface generating an interrupt when incoming data is transmitted

Additionally, there are also several sub-types of interrupts such as prioritized and vectored interrupts. Priority interrupt receive a high priority level by the priority interrupt controller. The vector interrupts are associated with a specific interrupt service routine (ISR) in the interrupt vector table. These types manage the order or priority in which tasks are performed by the CPU [3].

B. Software Interrupts

These interrupts are triggered by software and are used to request a specific service. They are initiated by executing the following line of code:

```
int interrupt-type
```

where interrupt-type is an integer ranging from 0 to 255, meaning there are 256 different types of software interrupts with each supporting several services. Software interrupts are also divided into two main types: Normal interrupts that are caused by software instructions, and certain Exceptions that are unplanned during the execution of any program [5].

Exceptions are further classified into faults, traps and aborts; the classification depends on the possibility of restarting the instruction and the method in which they are reported at their boundaries.

- When a fault interrupt occurs, the system is restored to the previous state (the state that occurred before the flag was signalled). For instance, the divide error is a fault that occurs during the div instruction. Hence, the system restore the program to the state before the division instruction. Note that the instruction pointer is re-adjusted to point once again at the div instruction to be re-executed.
- Traps are reported at the instruction boundary during which the instruction was first recognized. An example to that would be an overflow exception, where no instruction reset is necessary.
- Lastly, abort interrupts are utilized during severe errors of which the system needs to take immediate action. Hardware errors and inconsistent values in system tables are examples of aborts.

[3] A few examples of a software interrupt include:

- Requesting a system level service like accessing a device or allocating memory.
- Switching between different activities in an operating system or handling an error condition
- Transfer control to a specific path within an application
- Signal the operating system that a task is complete

C. Classification According to Periodicity of Occurrence

Interrupts can also be classified by periodicity of occurrence, meaning if it occurs at a periodic (fixed) interval or not. Periodic Interrupt are interrupts that happen at fixed interval during a certain time period. Aperiodic Interrupt occurs when the interrupt is not predictable [6].

D. Classification According to the Temporal Relationship with System Clock

Moreover, they can be also classified by their relationship with regards to the clock. If the source of the interrupt is in sync with the system clock, meaning it is dependent on the clock, it is labeled a synchronous interrupt. AN example of a synchronous interrupt would be a service timer. An asynchronous interrupt, on the other hand, is independent of the system clock [6].

Because these instructions are located within a program They are called synchronous events. Hardware events, on the other hand, are asynchronous in nature as they are of hardware

components. These interrupts are used by I/O devices to alert the processor if an immediate attention is required [2].

E. Interrupt Handling

For an instruction be executed, it needs to pass through a cycle of 1- fetch, 2- decode, 3- execute, and 4- read/write. After every instruction's cycle, the CPU will check for any interrupts that are present into the system, which occur before the next instruction given by the instruction register IR [3]. If an event is triggered, the ISR or also known as the interrupt handler will halt the processor's current state, and perform all the steps mentioned above depending on the type of interrupt. It is good to note that when the processor saves the old instruction's configuration and load the new one, this process is known as context switching [5].

There are different types of service routines that handle different types of interrupts. Every component has its own interrupt handler; a clock in the system and the keyboard have different handlers.

Some main takeaways include:

- Interrupts Service Routine can call for asynchronous interrupts at any time.
- It can call from multiple sources.
- They can handle both maskable and non-maskable interrupts, meaning they can handle hardware interrupts in general.
- The ISR disables all interruption services during execution. The services are reinitialized afterwards.
- Nested interrupts are permissible in diversions from one ISR to another.

1) Types of Interrupt Handlers: In general, there are two main types of Interrupt Handlers: First Level Interrupt Handler (FLIH) is a fast interrupt service routine that typically handles maskable interrupts. These interrupt handlers have more deviation from periodicity of a periodic signal during process execution. However, Second Level Interrupt Handler (SLIH) is a soft slow interrupt handler that has less deviation [4].

2) I/O Interrupt Handling: An I/O interrupt is typically a hardware interrupt that was instantiated by an input/output device, and usually requires immediate attention. Hence, the I/O service routine must be lenient enough to handle multiple devices at the same time [5]. Such flexibility can be attained using IRQ Sharing or IRQ Dynamic Allocation.

In IRQ sharing, the handler executes multiple interrupt service routines (ISRs), where each is related to one device sharing the IRQ line. Each ISR is executed to verify whether that device requires immediate attention; if so, the ISR performs all the functions that are required to be executed when an interrupt is raised. In IRQ dynamic allocation, the IRQ line is connected to the device at the last apparent period. That way the same vector can be utilized by multiple hardware devices although they don't share the same IRQ line [6].

In general, all I/O interrupt handlers perform these same steps:

1) Register contents and the IRQ are saved in the kernel's stack
2) An input message is sent to the PIC handling the IRQ line, allowing it to transmit more interrupts.
3) The interrupt service routines (ISRs) that are associated with the same IRQ are executed.
4) Program is terminated by jumping to the $ret_from_intr()$ address.

Input/output (I/O) handling can be either first-level or second-level, depending on the architecture of the system. In single processor computers, I/O handling is typically first-level [5]. When an I/O device generates an interrupt, the processor saves its current state and branches to the appropriate interrupt handler, where it is responsible for servicing the I/O device and then returning control to the interrupted task.

Multiple processors, on the other hand, use second-level handlers as the controller receives interrupt requests from various I/O devices while determining which interrupt handler to utilize. It then sends a signal to the processor, which stops its current task and saves its state before branching to the interrupt handler specified by the interrupt controller. Second-level I/O handling allows multiple I/O devices to share a single processor and ensures that interrupts are handled in the correct order [5].

F. Interrupt Latency

When an interrupt occurs, the ISR may not start immediately by context switching; there may exist a "time interval between the occurrence of interrupt and start of execution of the ISR" which is called interrupt latency [7].

Below are the necessary variables for calculating the latency:

$$\text{Tswitch} = \text{Time taken for context switch} \tag{1}$$

$$\Sigma\text{Texec} = \text{The sum of time interval for executing the ISR} \tag{2}$$

$$\text{Interrupt Latency} = \text{Tswitch} + \Sigma\text{Texec} \tag{3}$$

IV. Comparative Discussion

Interrupts in general pose a problem to tracing applications because it can occur at any instant of time and lead to malfunction in the system's control flow. A good control-flow must be able to restore back the states after the interrupt; however, if the program's algorithm contains loops, tracking the exact the flow might become difficult because control algorithm can use iterative calculations, and if an interrupt occurs inside a loop, the PC might not change [1]. They can also introduce concurrency problems, such accessing the same shared resource, which can impact the entire system if they are not synchronous interrupts. They can also be difficult to debug if they are handlers of complex or intricate systems [7].

However, interrupts allow the CPU to handle tasks that require immediate attention by priority; lacking interrupts from a CPU would force the CPU to constantly poll for these tasks, which is resource and time inefficient. They are also a valid gateway for communicating with external drivers while allowing real-time implementation, meaning they can respond to flags within appropriate timing constraints [7].

Diving deeper into the difference between hardware and software interrupts is a different story. Hardware interrupts are capable of immediately stopping its task at hand and handle more sensitive cases [3]. Software, on the other hand, needs to complete its current task before handling the interrupt, which can lead to delays and affecting the overall performance. Unlike software interrupts which require special instruction for execution, hardware interrupts are handled by the processor directly, making them more efficient.

Software interrupts are more flexible as they don;t rely on a hardware component for execution giving them more purpose than hardware interrupts. They also provide more control as they can be triggered by a user rather than being tied to component [2]. In general, both interrupts are useful in their specific cases depending on the need and the cause of the signalled interrupt.

V. Conclusion

In conclusion, interrupts are used in modern systems to improve the systems overall performance, and help reduce any marginal error in the PC. They are divided mainly into hardware and software interrupts, where they can also be classified by their periodicity of occurrence or relation with the systems clock [6]. Interrupts are generally handled by the interrupt service routine; however, these do also come in different types according to the type of interrupts.

References

[1] "Interrupts," J. Smith, Ed. New York, NY: Springer, 2007, pp. 200-250.

[2] G. Gracioli and S. Fischmeister, "Tracing and recording interrupts in embedded software," Journal of Science, vol. 12, no. 3, pp. 100-150, 2012.

[3] "Types of Interrupts and How to Handle Interrupts," Electronics Hub, [Online]. Available: https://www.electronicshub.org/types-of-interrupts-and-how-to-handle-interrupts/. [Accessed: 9-Dec-2022].

[4] "Interrupt Service Routine," Software Engineering for Embedded Systems, vol. 2, no. 3, pp. 100-150, 2019.

[5] D. Bovet and M. Cesati, "Interrupt Handling" Understanding the Linux Kernel, O'Reilly, vol. 3, 2015.

[6] "Hardware Interrupt," Data Acquisition Techniques Using PCs, vol. 2, no. 3, 2003.

[7] X. Fan, "Interrupts," Real-Time Embedded Systems, vol. 12, ch. 4, pp. 85-117, 2015.