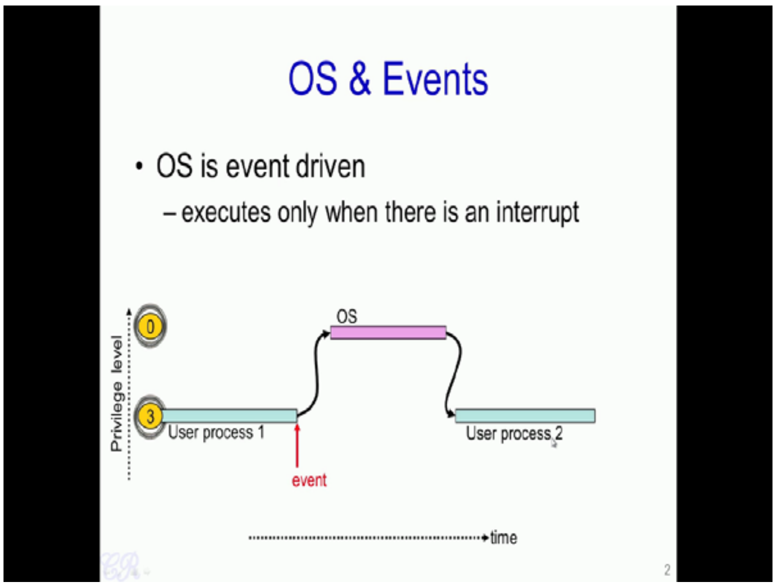
Introduction to Operating Systems

Prof. Chester Rebeiro
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Week – 04
Lecture – 14
Interrupts

Hello and welcome to this video. In today's video we will look at interrupts which forms a crucial part in all modern day operating systems. So, unlike normal software or normal programs that we write, operating systems are event based that is whenever an event occurs only then the operating system executes. To see this, what this means, let us look at slides.

(Refer Slide Time: 00:43)



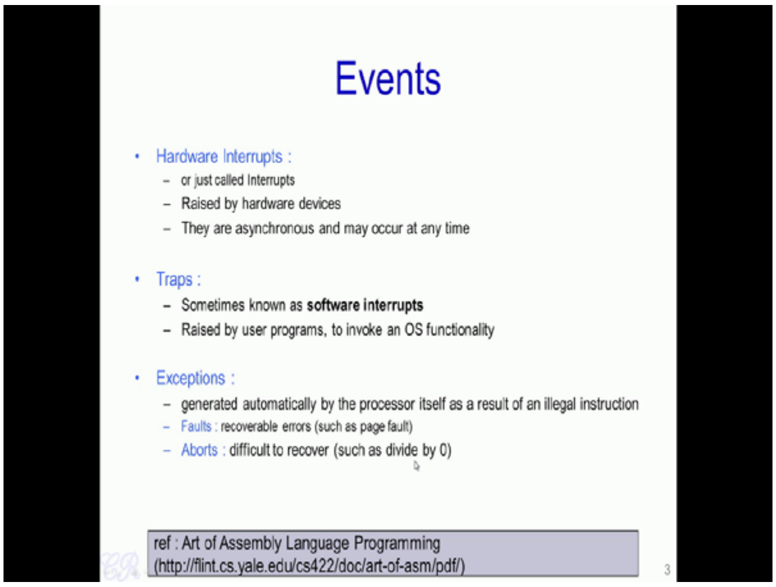
OS & Events

- OS is event driven
  - executes only when there is an interrupt

Suppose you have a user process that is running, so as we have seen before user process runs in user space and in the Intel nomenclature this is in ring 3. Now this user process continues to execute on the processor until an event occurs. So, when this event occurs it would trigger the operating system to execute. So, this triggering of the operating system will also result in a change in the privilege level. The system would no longer be executing in the user space, but rather it will be in privilege level 0 or that is executing in the Kernel space.

The operating system would essentially execute and service this particular event and at the end of that execution, the control is fed back to user space and the process will continue to execute. The User space process which would execute after the OS completes could be the same process that is User Process 1 or some other User Process, in this example it is User Process 2.

(Refer Slide Time: 02:01)



Events

- Hardware Interrupts :
  - or just called Interrupts
  - Raised by hardware devices
  - They are asynchronous and may occur at any time
- Traps :
  - Sometimes known as **software interrupts**
  - Raised by user programs, to invoke an OS functionality
- Exceptions :
  - generated automatically by the processor itself as a result of an illegal instruction
  - Faults : recoverable errors (such as page fault)
  - Aborts : difficult to recover (such as divide by 0)

ref : Art of Assembly Language Programming
(http://flint.cs.yale.edu/cs422/doc/art-of-asm/pdf/)

3

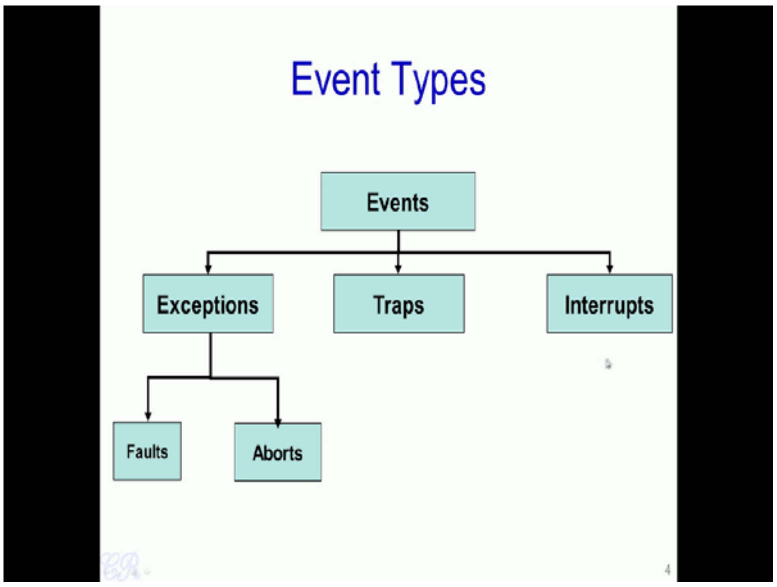
Let us look at how events are classified. So, various literature categorizes events in different ways, but we will do is we will follow the categorization of events based on this particular book called the Art of Assembly Level Programming which can be downloaded from this website. So, in this book events are classified into 3 different types these are Hardware Interrupts, Tarps and Exceptions.

Hardware Interrupts or sometimes just called as Interrupts are raised by external hardware devices. For example, the network card when it receives a packet could possibly raise an interrupt or other device such as the keyboard mouse or a USB device when plugged in could rise in hardware interrupt. So, these hardware interrupts are asynchronous and can occur at any time.

Besides hardware interrupts there are Traps and Exceptions. Traps are sometimes known as software interrupts, they are raised by user programs in order to invoke some operating system functionality. For instance, if a user program wants to print something on the monitor it would invoke a trap which essentially would be a system call to the operating system and then OS will then take care of writing the particular text or writing the particular data on to the screen.

The third type of event is known as Exceptions, these events are generated automatically by the processor itself as a result of an illegal instruction. So, there are 2 types of exceptions these are Faults and Aborts. Very common example of a fault is a page fault. So, faults are essentially exceptions from which the processor could recover. For instance, when there is a page fault that occurs when a process is executing it would result in the OS or in the operating system executing and loading the required page from the swaps space into the RAM. On the other hand, an exception which is of the form abort would be very difficult to recover, such as a divide by 0 exceptions.

(Refer Slide Time: 04:38)

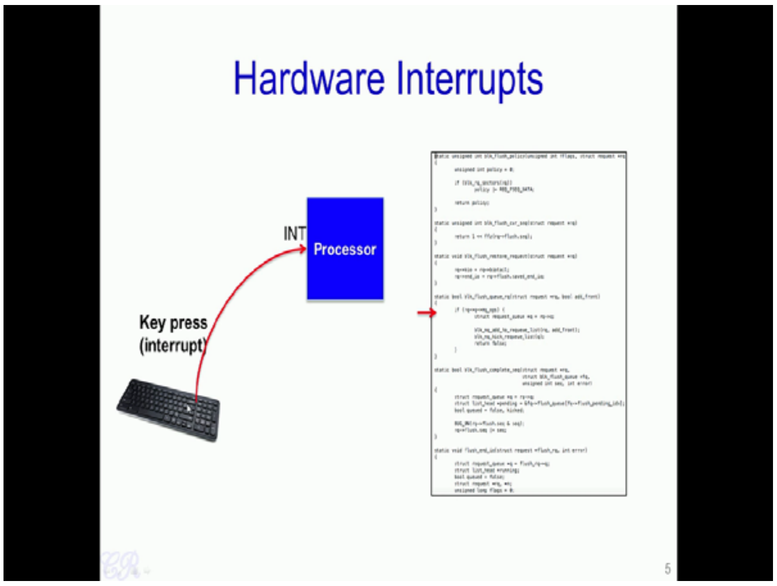


Event Types

$$\begin{tikzcd}
 & \text{Events} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{Exceptions} \arrow[d] \arrow[dr] & \text{Traps} & \text{Interrupts} \\
\text{Faults} & \text{Aborts} &
\end{tikzcd}$$

So, when a divide by 0 exception occurs in a program typically the program would be terminated, essentially the operating system has no way to recover from such a divide by 0 exceptions. This particular slide shows the various classifications of events into exceptions, traps and interrupts. Exceptions are further classified into Faults and Aborts. So, we will now take a specific case of interrupts that is of Hardware Interrupts.

(Refer Slide Time: 04:58)



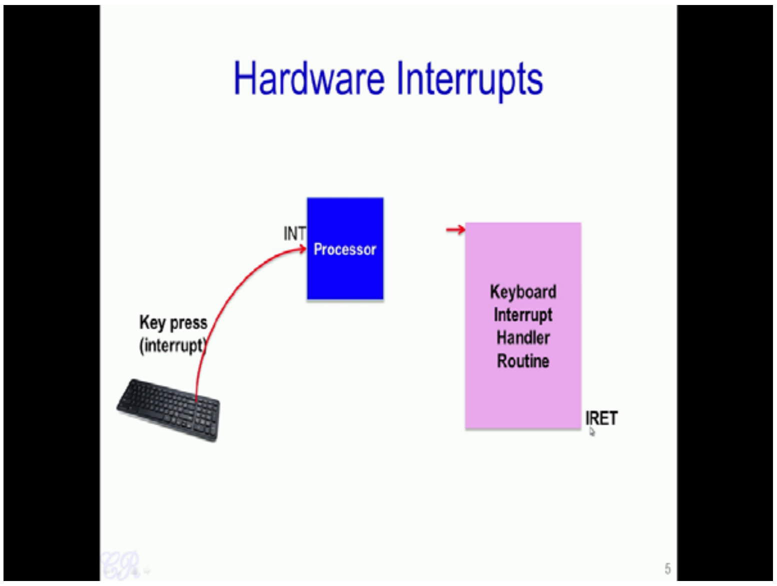
Hardware Interrupts

```
static unsigned int blk_flush_policy(unsigned int fflags, struct request *rq)
{
        unsigned int policy = 0;

        if (blk_rq_sectors(rq))
                policy |= REQ_FSEQ_DATA;

        return policy;
}

static unsigned int blk_flush_cur_seq(struct request *rq)
{
        return 1 << ffz(rq->flush.seq);
}

static void blk_flush_restore_request(struct request *rq)
{
        rq->bio = rq->biotail;
        rq->end_io = rq->flush.saved_end_io;
}

static bool blk_flush_queue_rq(struct request *rq, bool add_front)
{
        if (rq->q->mq_ops) {
                struct request_queue *q = rq->q;

                blk_mq_add_to_requeue_list(rq, add_front);
                blk_mq_kick_requeue_list(q);
                return false;
        }
}

static bool blk_flush_complete_seq(struct request *rq,
                                   struct blk_flush_queue *fq,
                                   unsigned int seq, int error)
{
        struct request_queue *q = rq->q;
        struct list_head *pending = &fq->flush_queue[fq->flush_pending_idx];
        bool queued = false, kicked;

        BUG_ON(rq->flush.seq & seq);
        rq->flush.seq |= seq;
}

static void flush_end_io(struct request *flush_rq, int error)
{
        struct request_queue *q = flush_rq->q;
        struct list_head *running;
        bool queued = false;
        struct request *rq, *n;
        unsigned long flags = 0;
```

So, let us look at hardware interrupts. In general, processors today have a dedicated pin on the IC known as the interrupt pin. So, this pin is often shortened or just called by the INT pin or in some processors as the INTR pin. So, devices such as the keyboard will be connected to the processor through the INT pin. When a particular key is pressed on the keyboard it would result in an interrupt being generated to the processor.

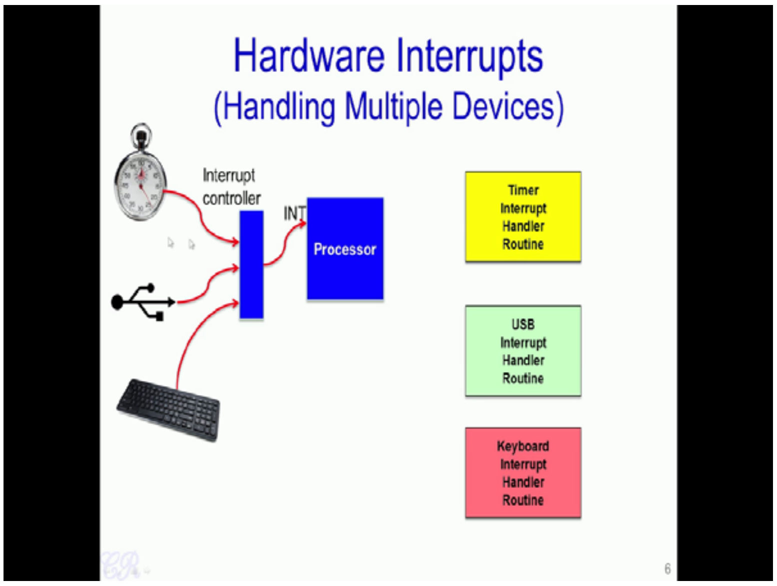
Now, let us see how this particular interrupt takes place and what happens in the processor. So, the processor typically would be executing a program and would be executing some instructions. Now, when a key is pressed and interrupt is generated to the processor and that would result in a switch in the processor to what is known as the interrupt handler routine.

(Refer Slide Time: 05:58)


Hardware Interrupts

So, in this particular case since it is the keyboard which has resulted in the interrupt, the keyboard interrupt handler routine would be invoked. The processor would then begin to execute this keyboard handler routine until an instruction such as the IRET is obtained. When the IRET instruction gets executed, the context is switched back to the program which was originally being run. So, in this way we see that interrupts could occur in time during the programs execution, it would result in a new context being executed and at the end of that execution the processor goes back to the original context.

(Refer Slide Time: 06:41)

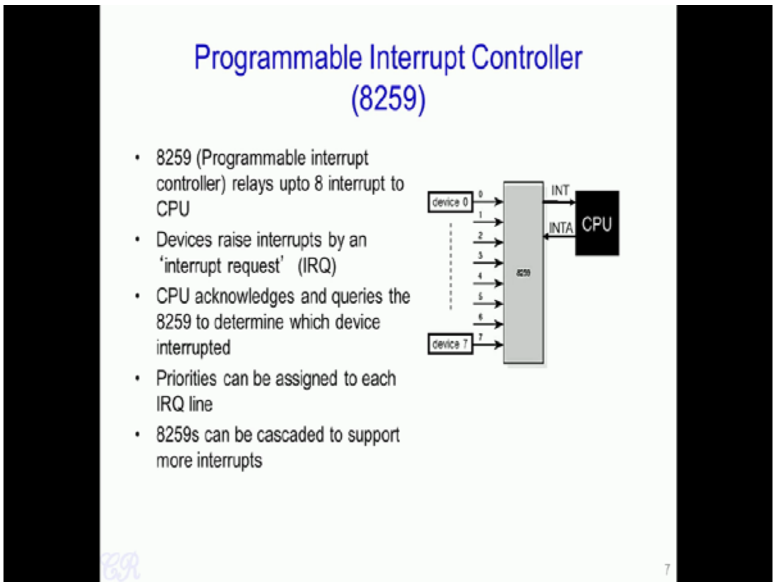

Hardware Interrupts
(Handling Multiple Devices)

Interrupt controller
INT
Processor
Timer Interrupt Handler Routine
USB Interrupt Handler Routine
Keyboard Interrupt Handler Routine
6

Typically, systems do not have just one device connected to the processor that could be several devices. For instance systems could have timers, USB drives, keyboard, mouse, network cards and so on. However, as we have seen previously the processor just has a single interrupt pin, so how is it possible then that several devices share the single interrupt pin? In order to achieve this, a special hardware is used in systems. So, this is known as the Interrupt Controller. The job of the Interrupt Controller is to ensure that the single pin of the interrupt is shared between multiple devices. So, the interrupt controller would receive interrupts from each of these devices and then channelize that interrupts to the INT pin of the processor.

The processor would then communicate with the interrupt controller to determine which of these devices had actually generated the interrupt. As a result, the processor would then execute the corresponding interrupt handler routine. For instance, if the timer had resulted in the interrupt then the timer interrupt handler routine would be invoked. On the other hand, if a USB device had resulted in the interrupt the USB interrupt handler routine would be invoked and so on. Thus, the interrupt handler routine invoke is going to be very specific to the device that cause the interrupt to occur.

(Refer Slide Time: 08:15)



Programmable Interrupt Controller (8259)

- 8259 (Programmable interrupt controller) relays upto 8 interrupt to CPU
- Devices raise interrupts by an 'interrupt request' (IRQ)
- CPU acknowledges and queries the 8259 to determine which device interrupted
- Priorities can be assigned to each IRQ line
- 8259s can be cascaded to support more interrupts

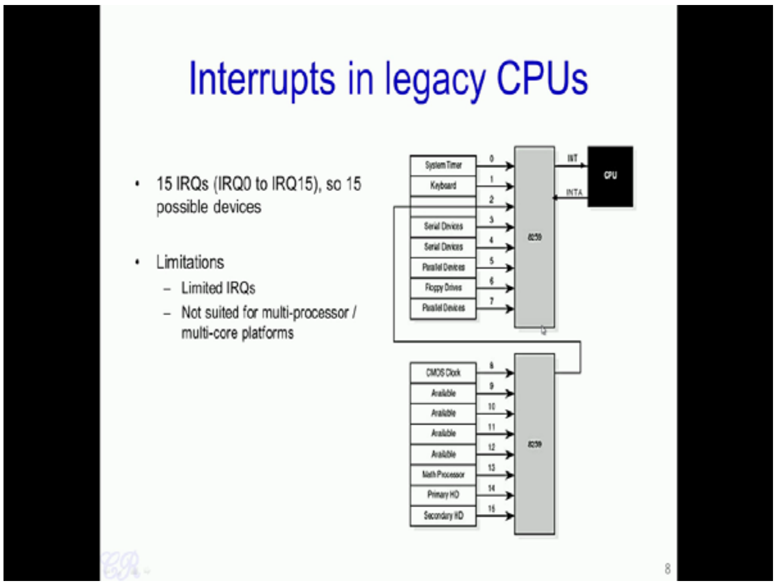
device 0
0
1
2
3
4
5
6
7
device 7
8259
INT
INTA
CPU
7

One commonly used interrupt controller is known as a Programmable Interrupt Controller. So, it is numbered as 8259 and pictorially this is how it gets connected. So, the 8259 has two sides this is the input side and the output side. The output is connected to the INT pin of the CPU, there is also an INTA pin which is an interrupt acknowledge pin.

On the other side we have 8 IRQ lines. IRQ stands here for Interrupt Requests. So, on the input side there are up to 8 devices that can be connected to the 8259. These devices are labeled device 0 to device 7, all these devices could independently request an interrupt from the CPU. The 8259 would then channelize that interrupt through the INT pin of the CPU. The CPU would acknowledge the interrupt through the INTA pin and also determine which of these 8 devices had requested the interrupt.

Now, what would happen if two devices request the interrupt at exactly the same time? In such a case, 8259 would use some priority encoding algorithm to determine which of these devices should be given the privilege to request for the interrupt first. Another feature of the 8259 is that it can be cascaded to support more than 8 devices, thus more than 8 devices could cause interrupts to the CPU.

(Refer Slide Time: 09:52)



Interrupts in legacy CPUs

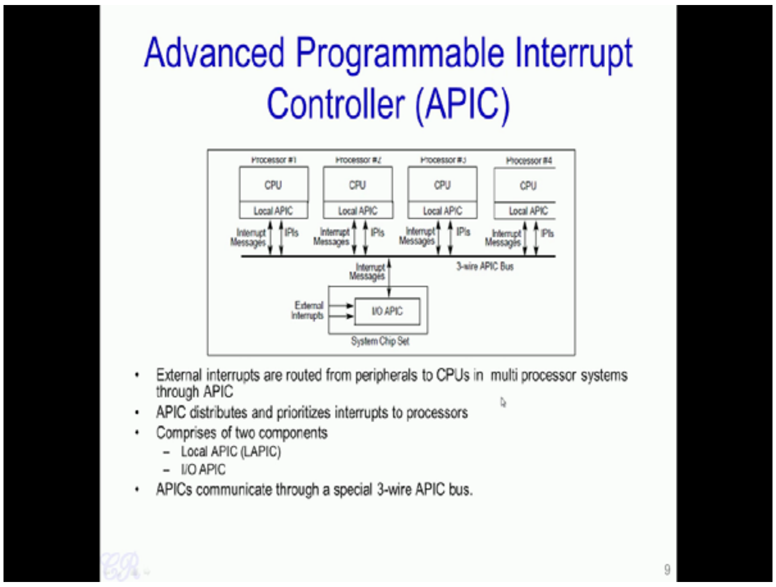
- 15 IRQs (IRQ0 to IRQ15), so 15 possible devices
- Limitations
  - Limited IRQs
  - Not suited for multi-processor / multi-core platforms

System Timer
Keyboard
Serial Devices
Serial Devices
Parallel Devices
Floppy Drives
Parallel Devices
CMOS Clock
Available
Available
Available
Available
Math Processor
Primary HD
Secondary HD
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
8259
8259
INT
INTA
CPU
8

In Legacy computers typically there are two 8259 controllers present - one is configured as the master, while the other is configured as the slave. The slave 8259 controller is connected to one of the input channels of the master 8259. So, if any of these devices connected to the slave 8259, request an interrupt, this interrupt is channelized to the master 8259 and the master 8259 would then channelize this interrupt to the CPU.

So, one limitation of this legacy configuration of the 8259s is the limited IRQs. So, each device could as we sink and support only 8 devices and therefore, if you have large number of devices in your system then you would require several 8259 controllers to be present. Another major limitation of this configuration is that, the support from multi-processor and multi-core platforms is difficult. Essentially as we seen over here, there is only one CPU that is connected to the master 8259 programmable interrupt controller. This particular INT pin cannot be sent to other CPU, which may result in some problems.

(Refer Slide Time: 11:14)



Advanced Programmable Interrupt Controller (APIC)

Processor #1
Processor #2
Processor #3
Processor #4
CPU
Local APIC
Interrupt Messages
IPIs
3-wire APIC Bus
Interrupt Messages
External Interrupts
I/O APIC
System Chip Set

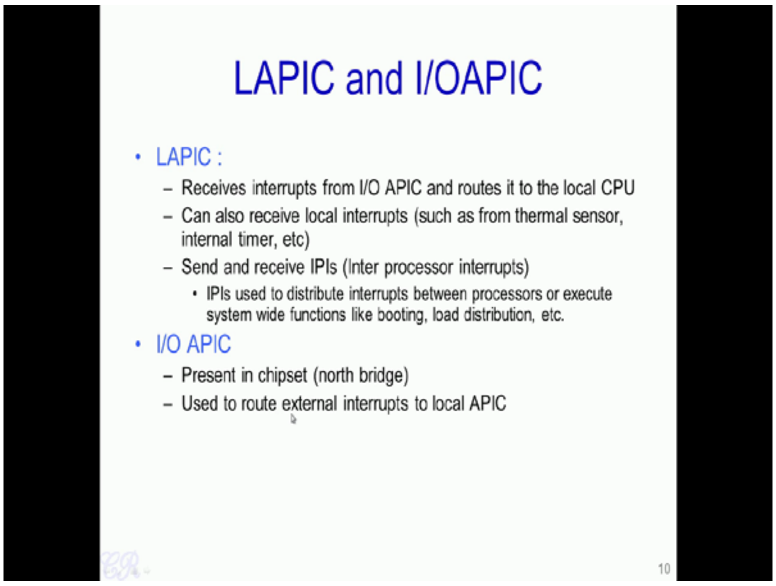
- External interrupts are routed from peripherals to CPUs in multi processor systems through APIC
- APIC distributes and prioritizes interrupts to processors
- Comprises of two components
  - Local APIC (LAPIC)
  - I/O APIC
- APICs communicate through a special 3-wire APIC bus.

9

In current systems, the 8259 programmable interrupt controller is replaced by something known as APIC or Advanced Programmable Interrupt Controller. The configurations for the APICs in modern systems are shown over here. So, each CPU in a multi-core or multiprocessors system would have a local APIC as shown here. So, Processor 1 has a local APIC, Processor 2 has its own APIC, Processor 3 has its own local APIC and so on.

In addition to this, there is something known as an IO APIC which is present in the System Chip Set. Now, external devices such as keyboards, mouse, network cards, and so on, would request interrupts through the IO APIC which is then channelized to one of the local APICs in each CPU. Thus interrupts can be distributed and prioritized between CPUs, also the local APICs as well as the IO APICs communicate through interrupt messages and IPIs that is Inter Processor Interrupts.

(Refer Slide Time: 12:24)



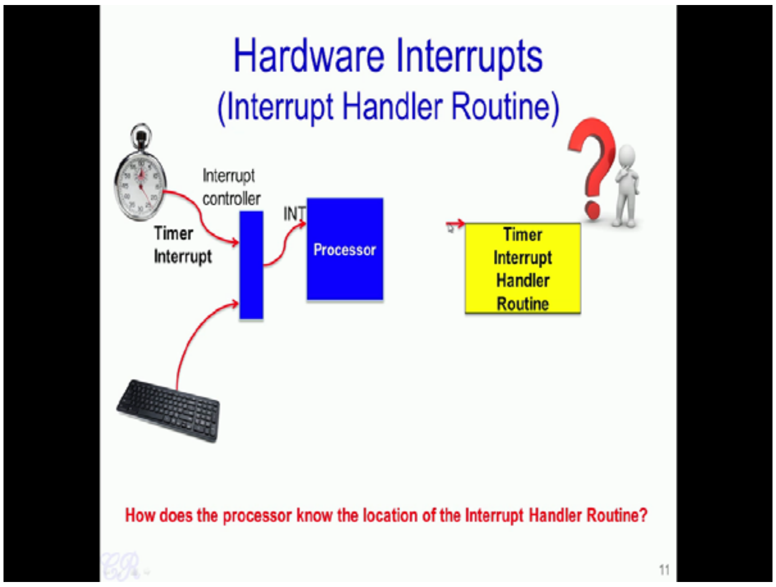
LAPIC and I/OAPIC

- LAPIC :
  - Receives interrupts from I/O APIC and routes it to the local CPU
  - Can also receive local interrupts (such as from thermal sensor, internal timer, etc)
  - Send and receive IPIs (Inter processor interrupts)
    - IPIs used to distribute interrupts between processors or execute system wide functions like booting, load distribution, etc.
- I/O APIC
  - Present in chipset (north bridge)
  - Used to route external interrupts to local APIC

10

The Local APIC receives interrupts from the IO APIC and routes it to the corresponding local CPU. The local APIC can also receive some local interrupts such as interrupt from the thermal sensor which is present on the CPU, an internal timer and so on. The Local APIC could also send and receive IPIs which is Inter Processor Interrupts, which allows interrupts between processors that it allows processors to do system wide functions like booting, load distribution, and so on. The IO APIC is present in the System Chip Set which sometimes known as north bridge. This particular APIC is used to route external interrupts into the Local APIC.

(Refer Slide Time: 13:12)



Hardware Interrupts (Interrupt Handler Routine)

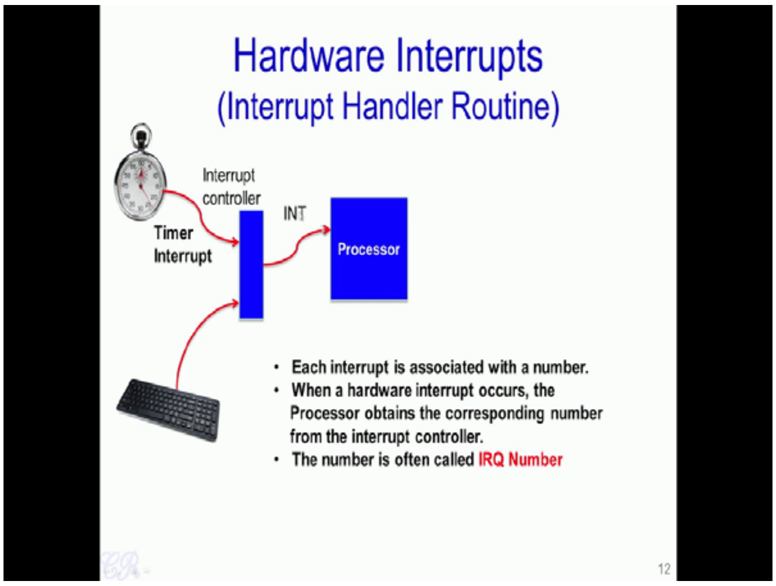
How does the processor know the location of the Interrupt Handler Routine?

11

So, we have seen so far that we have multiple devices which are connected to an interrupt controller which could be either an APIC or in legacy systems based on the 8259 controller. Even any of these devices request an interrupt, the interrupt is channelized to the processor and it would result in the corresponding interrupt handler routine to be invoked.

How does the processor know the location of the interrupt handler routine? So, the interrupt handler routine, just like all other software codes or all other codes, so the important question that one needs to ask here is how does the processor know the location of the interrupt handler routine. So, the interrupt handler routine is just like any other software code is also executed from the RAM. So, the question that we are posing here is how does the processor know the starting location or the address of the interrupt handler routine.

(Refer Slide Time: 14:13)



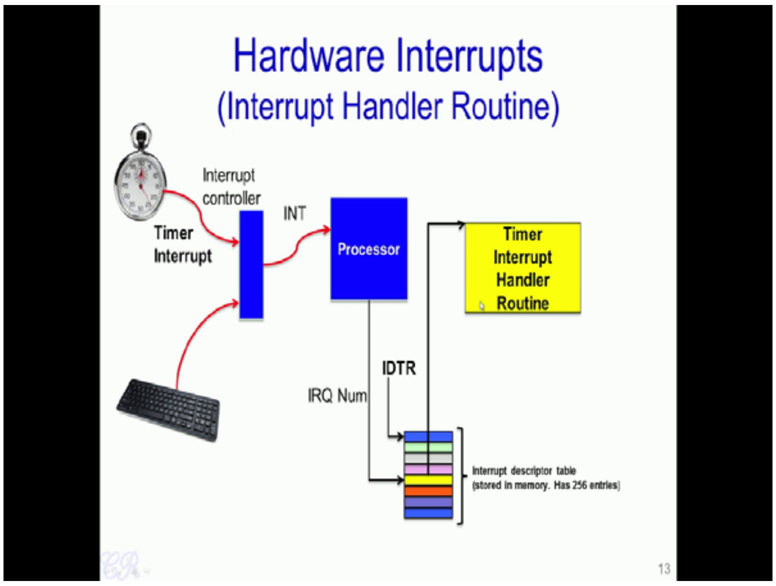
Hardware Interrupts
(Interrupt Handler Routine)

- Each interrupt is associated with a number.
- When a hardware interrupt occurs, the Processor obtains the corresponding number from the interrupt controller.
- The number is often called IRQ Number

12

In order to achieve this, what happens is the following. Each of the devices that are connected to the interrupt controller also has a dedicated number known as the IRQ number. So, when a device request an interrupt and the INT pin in the processor gets asserted the processor would then obtain the IRQ number from the interrupt controller. In this way the processor now determines which of these devices has requested for the interrupt.

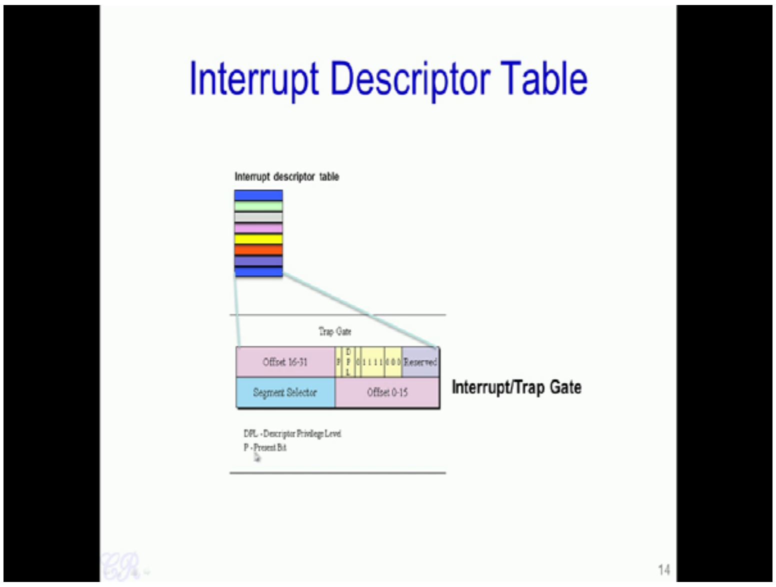
(Refer Slide Time: 14:43)


Hardware Interrupts
(Interrupt Handler Routine)

Interrupt controller
Timer Interrupt
INT
Processor
Timer Interrupt Handler Routine
IDTR
IRQ Num
Interrupt descriptor table (stored in memory. Has 256 entries)
13

We will now see how the processor uses the IRQ number. In the memory of the system, a table known as the interrupt descriptor table or IDT is stored. This IDT table is pointed to by a register stored in the processor known as the IDTR. This is interrupt descriptor table register. So, each entry in this descriptor table contains information of where in memory the corresponding interrupt handler routine is present. So, the processor would use the IRQ number to look into the interrupt descriptor table, from this descriptor the corresponding location of the interrupt handler routine is obtained and there after the processor can then execute instructions from this handler routine.

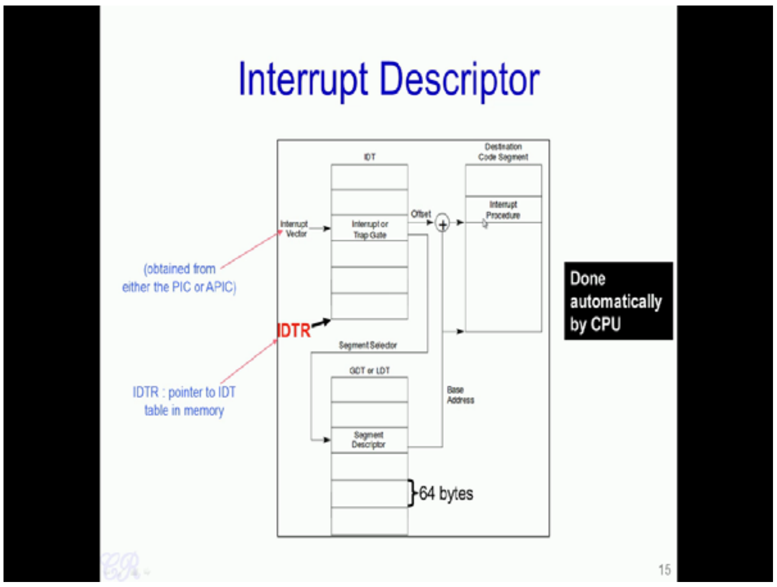
(Refer Slide Time: 15:36)


Interrupt Descriptor Table

So, what is the content of the interrupt descriptor table? In the x86 systems, since it also has segmentation. Therefore, the each interrupt descriptor would contain the segment selector as well as the offset. The segment selector as we have seen is of 16 bits and while the offset is of 32 bits. So, bits 0 to 15 are here and the bits 16 to 31 are present over here.

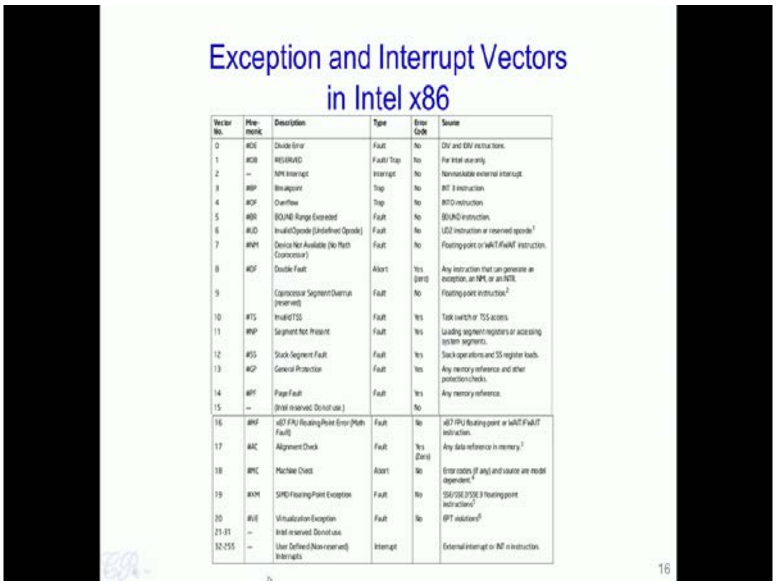
There are other aspects in the interrupt descriptor such as the Present bit and the Descriptor Privilege Level.

(Refer Slide Time: 16:11)



Let us see how this Interrupts Descriptors are used in reality. So, the processor would obtain the IRQ number or the interrupt vector from the programmable interrupt controller or the APIC, and as we have seen this interrupt vector is used to index into the IDT table, and the corresponding interrupt or trap gate contains the segment selector as well as the offset. The segment selector is then used to index into the GD or the LDT table from which, the base address of the code segment is obtained. The offset part is then taken and added, to the base address to obtain the final address where the interrupt procedure is present.

(Refer Slide Time: 17:01)



Exception and Interrupt Vectors in Intel x86

Vector No.	Mne-monic	Description	Type	Error Code	Source
0	#DE	Divide Error	Fault	No	DIV and IDIV instructions.
1	#DB	RESERVED	Fault/ Trap	No	For Intel use only.
2	—	NMI Interrupt	Interrupt	No	Nonmaskable external interrupt.
3	#BP	Breakpoint	Trap	No	INT 3 instruction.
4	#OF	Overflow	Trap	No	INTO instruction.
5	#BR	BOUND Range Exceeded	Fault	No	BOUND instruction.
6	#UD	Invalid Opcode (Undefined Opcode)	Fault	No	UD2 instruction or reserved opcode.[1]
7	#NM	Device Not Available (No Math Coprocessor)	Fault	No	Floating-point or WAIT/FWAIT instruction.
8	#DF	Double Fault	Abort	Yes (zero)	Any instruction that can generate an exception, an NMI, or an INTR.
9		Coprocessor Segment Overrun (reserved)	Fault	No	Floating-point instruction.[2]
10	#TS	Invalid TSS	Fault	Yes	Task switch or TSS access.
11	#NP	Segment Not Present	Fault	Yes	Loading segment registers or accessing system segments.
12	#SS	Stack-Segment Fault	Fault	Yes	Stack operations and SS register loads.
13	#GP	General Protection	Fault	Yes	Any memory reference and other protection checks.
14	#PF	Page Fault	Fault	Yes	Any memory reference.
15	—	(Intel reserved. Do not use.)		No	
16	#MF	x87 FPU Floating-Point Error (Math Fault)	Fault	No	x87 FPU floating-point or WAIT/FWAIT instruction.
17	#AC	Alignment Check	Fault	Yes (Zero)	Any data reference in memory.[3]
18	#MC	Machine Check	Abort	No	Error codes (if any) and source are model dependent.[4]
19	#XM	SIMD Floating-Point Exception	Fault	No	SSE/SSE2/SSE3 floating-point instructions[5]
20	#VE	Virtualization Exception	Fault	No	EPT violations[6]
21-31	—	Intel reserved. Do not use.			
32-255	—	User Defined (Non-reserved) Interrupts	Interrupt		External interrupt or INT n instruction.

16

The Intel x86 systems supports 56 different types of events. Some of these events are used internally for faults and aborts, while others can be configured for hardware interrupts as well as for software interrupts.

So, this particular table gives some of these interrupts and exceptions that are supported by Intel x86 systems. Especially we would like to see that interrupt with vector number 0 to 31 are internal to the processor, of these the vector number 0 to 20 are used for various faults or aborts. The interrupt vector numbers from 32 to 255 can be user defined. So, some of these user defined interrupts are configured as hardware interrupts, while others are used as software interrupts.

In the next video, we will look at how interrupts are handled in the CPU as well as in the operating system.

Thank you.