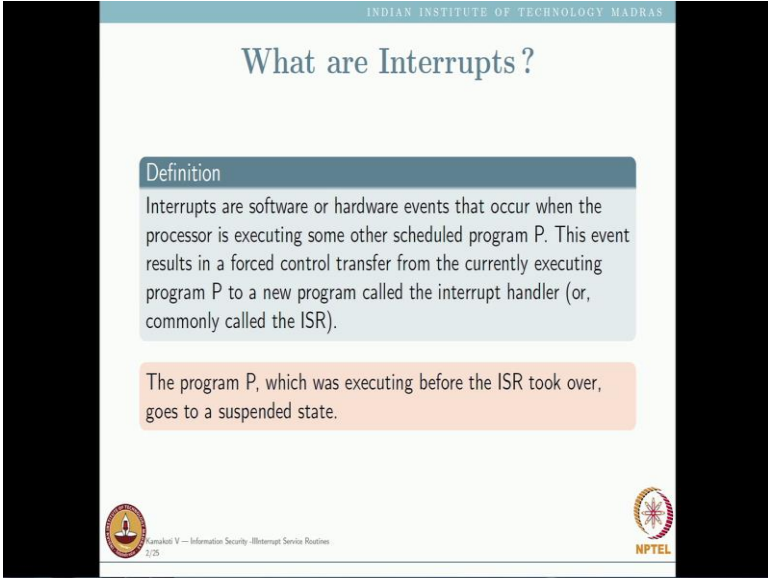


**Information Security - II**  
**Prof. V. Kamakoti**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture - 24**  
**Interrupt Service Routines**

Good afternoon all of you, and welcome to this session. We now proceed to what we term as Interrupt Service Routines. And, so, in this session, we will cover interrupt service routines, a little more deep-dive than what I gave, talked in the previous sessions. And then, we will also give you some examples; we will run the code and show you, how interrupt service routine works. Please be ready with your laptops, so that, as and when I give certain instructions, you can basically follow, and see the execution for yourself. Now, very quickly, what are interrupts?

(Refer Slide Time: 01:08)



INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## What are Interrupts ?

**Definition**

Interrupts are software or hardware events that occur when the processor is executing some other scheduled program P. This event results in a forced control transfer from the currently executing program P to a new program called the interrupt handler (or, commonly called the ISR).

The program P, which was executing before the ISR took over, goes to a suspended state.

Kamakoti V — Information Security — Interrupt Service Routines  
4/25

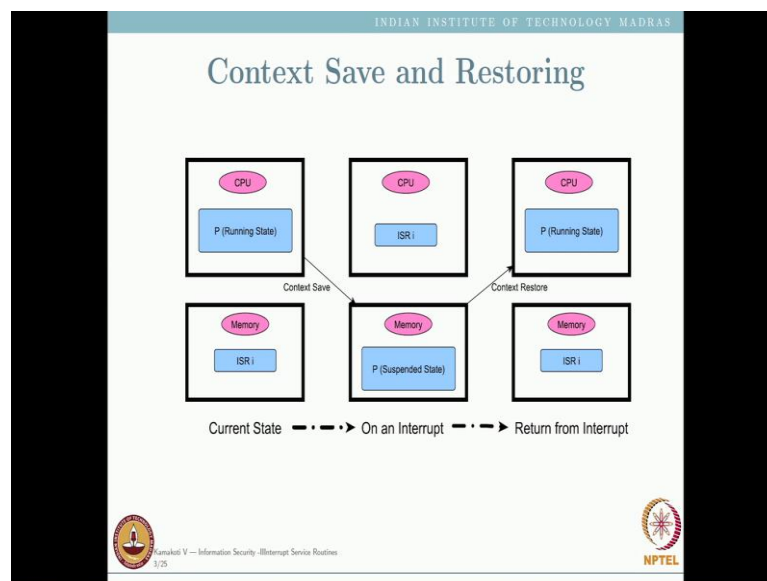
NPTEL

Actually, interrupts are driven either by software or they are driven by hardware. So, the hardware interrupts are those that come from the external world; for example, a keyboard interrupt is a hardware interrupt; a device interrupt which comes through the external interrupt pins to the processor are hardware interrupts. There is also software interrupts, which are basically, these interrupts come into place, because of certain problems with your software. So, like, divide by zero; this basically creates an interrupt. So, there is a

terminology that is followed, between what is the difference between an interrupt, a trap, exception. But, as I mentioned yesterday, for this course purpose, we will have one exception, as they, as the word for exceptional behavior, and if that is caused by the program, then it is a trap; when it is caused external to the program, it is an interrupt.

So, what happens when an interrupt comes up? The current program that is executing gets suspended, and the interrupt service routine takes over; it executes, then, it shall see whether it can return back the control to the program. For example, if you have a divide by zero, it will never return it back to you. If you are having a segment overflow, it gives you a seg fault and never returns back to you; but, if there are some other small, small things, for which it can return, back to you. And so, the interrupt service routine does the service, and it decides, whether it can return back to you or not and based on that, it will return, or it will terminate your program.

(Refer Slide Time: 02:57)



So, this is how the whole thing works. So, currently, there is a process running; that is what you see on the left hand side and there is an interrupt. If there is an interrupt then, the interrupt routine service goes to the CPU, while the program goes to a suspended state. Then, after the interrupt service routine finishes, the program again comes back to the running state, if the need be, the interrupt service routine again goes back to the

memory. So, this is how the interaction between a program and the associated interrupt service routine, which needs to be executed, when an interrupt is caused, when the program is executing.

(Refer Slide Time: 03:42)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## Importance of Interrupts

If Interrupts weren't there..

- External hardware devices cannot request/respond to the CPU asynchronously.
- Scheduler can not perform preemptive scheduling.
- Software might not be able to perform privileged operations.
- Software might not be able to recover from run-time errors.
- Operating system might not be able to perform demand paging.

4/25

NPTEL

So, what are the importances of interrupts? Without interrupt, nothing can happen. So, you cannot, for example, no external hardware device can communicate; that is a most important thing and the external devices communicate in an asynchronous fashion, so that is also very, very important. Asynchronous in the sense that, the CPU has a clock; it is not necessary that, my interrupt has to come exactly at the clock, raising edge of the clock. It can be any time, any point of time.

For example, the scheduler cannot perform preemptive scheduling. Yesterday, we were talking about round-robin scheduling. So, when a process is executing and there is only one CPU, the processor actually executes on that CPU; when the processor actually executes on that CPU, process executes on that CPU, how will it be pulled out after a particular time quantum? The operating system cannot pull it out; why, because the operating system is also software and it needs to execute to pull it out. The operating system cannot pull it out, because it is not executing. So, who will pull it out? There will be timer that is set by the operating system before giving control to the process and that

timer will give an interrupt and pull this process out. So, this is very, very important, that also is caused because of an interrupt.

Software may not be able to perform any privileged operations because if I am a user level privilege level 3 operation, and I want to do something, for example, I want to print on the screen, or I want to go access the disk, all those things have happened, happens through a much more privileged code; for example, a device driver, which is much more, which has higher privilege than the normal application code. And, how will I switch on to a device driver? It is happening because of an interrupt. If I do not have the interrupt, I cannot basically go and switch to a higher privileged code. And, more importantly, software may not be able to recover from run time errors.

When there is a run time error, today, interrupt is the one which will service, and see that, if that run time error can be salvaged, salvageable, then it will salvage it, and then, give back control to the program. And, the operating system may not be able to perform demand paging. Today, when we use virtual memory, when a page, when I go and try to access address, the page corresponding to that address is not there, then, in the memory, then immediately, an interrupt is raised. Similarly, all your privilege checking, from a security point of view, everything depends upon the interrupt. If you do not have these operations, if you do not have interrupts, many of these cannot be achieved. Actually, a system cannot be built without interrupts. So, interrupt is a very, very important thing that we need to keep in mind.

(Refer Slide Time: 06:50)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## Interrupts Vs Exceptions

**Interrupts**  
Interrupts are asynchronous events intended to shift the focus of the CPU towards a different process. It can occur before the start, during the execution or after the execution of an instruction.

**Example**  
Example : Timer interrupts, Request to service h/w device.

**Exceptions**  
Exceptions occur when the processor detects an error when executing a particular instruction.

**Example**  
Example : Page faults, Protection violations.

© 2014 IIT Madras — Information Security — Interrupt Service Routines  
1/25

NPTEL

So, some of the manuals actually has, does not have a notion of a trap, but they have interrupt process exceptions. And, why I am basically covering these things are, that, when you, when you actually start reading multiple books, then, you will have some varying terminology; first and foremost, you should understand that, that variance exists and this is one variation. So, this is from the Intel manual.

Interrupts are asynchronous events, intended to shift the focus of the CPU towards a different process. It can occur before the start, during the execution, or after the execution of an instruction. So, I as a processor, I am executing an instruction; an interrupt can happen before I start executing it, while I am executing it, and after I execute it, right. And then, what happens, there is a shift of focus from the current process, and it will go through, go to the interrupt service routine. For example, timer interrupts, and request to service hardware device. So, these are all some of the asynchronous, or interrupts that we call. These are all some of the events that we call as interrupts.

On the other hand, there is something called exceptions. Exceptions occur, when the processor detects an error, when executing a particular instruction, right. So, whatever is happening inside the processor, there was some error that happens while executing the

instruction and then an exception is caused. So, anything that come external to the processor, we call it as interrupt; anything that happens internal to the processor, we call it as an exception. So, this is another way of looking at interrupt and exception. This is another way by which books define interrupt and exception.

So, there are different types of interrupts. Some interrupts can be hardware generated; some interrupts can be software generated, right. So, interrupts are not because of some instruction executing wrongly. If an instruction is executing wrongly, then it is called an exception. Now, what is hardware generated interrupt? Interrupt from printer, keyboard, mouse, etcetera. These all come through in the Intel architecture as something called a LAPIC, which is actually a programmable interrupt controller. And, this programmable interrupt controller is, will be responsible for bringing the interrupt from the device on to the processor. But sometimes, you also generate interrupts by software.

If you had carefully looked at the code that we covered in the last session, one of the lines in that code has INT 3. So, it is actually a software generated interrupts; and that is not because of an error in the program that the program voluntarily executes INT 3, to get into the interrupt service routine given by 3. So, for example, INT 64, which can invoke a custom ISR, an interrupt service routine written by the user. So, I can have, as I told you, we could have our own customized interrupt service routine for each of this interrupts, and we can handle it in the way we want it to be done.

(Refer Slide Time: 10:18)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## Types of Exceptions

Fault	Trap	Abort
A <b>recoverable</b> exception whose ISR takes corrective action. The instruction that caused the fault is re-executed.	A <b>recoverable</b> exception whose ISR takes corrective action. The instruction that caused the trap is ignored, and the program execution continues from the next instruction.	An <b>irrecoverable</b> exception used to report severe errors that cannot be corrected. Program execution cannot continue after this.
<b>Example</b> Page Fault	<b>Example</b> System calls	<b>Example</b> Hardware errors

© 2014 IIT Madras. All rights reserved. NPTEL

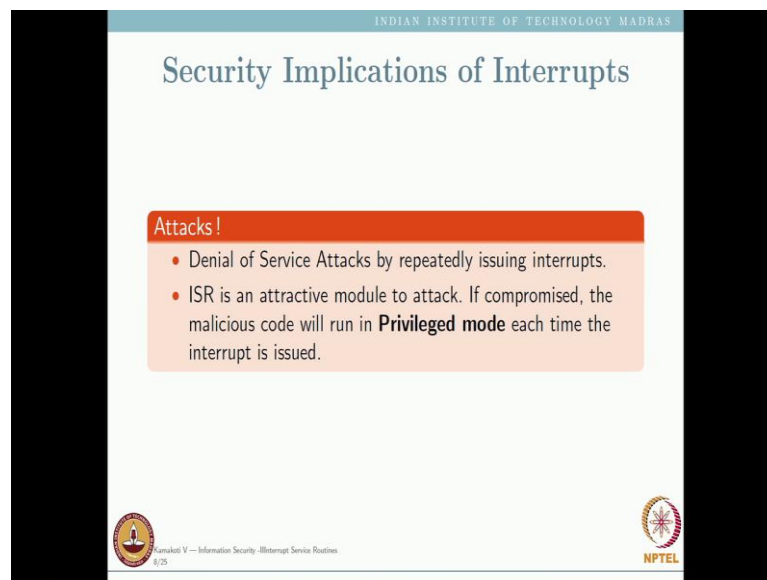
Now, there are 3 types of, you know, events that could happen, and that, specifically exceptions; that is, when I am executing an instruction, there are three ways by which an error could happen. One is a recoverable error, recoverable event, that is, I do something wrong, but I can recover; and, the interrupt service routine will help you recover that, and this is actually called a fault. For example, page fault. I try to access a page; it is not actually available; then, what do I do? I do an, I go and get that page into the memory, and then, execute the same instruction again. So, page fault is something that is recoverable.

A trap is actually a recoverable exception, which ISR takes corrective action. The instruction that caused the trap is ignored, and the program execution continues from the next instruction. In a page fault, I am trying to access some data; it is not available. So, the page is loaded, and again, I restart the same instruction. In a trap, I restart from the next instruction; I ignore this instruction that caused a trap. Then, abort is something that happens when there is an irrecoverable error. For example, there could be a hardware error; there could be a divide by zero; we cannot recover from these things. So, that is actually called an irrecoverable exception.

But anyway, who will abort? The interrupt service routine will abort; who will correct?

The interrupt service routine will correct. So, the decision of whether to abort a program, or keep the program, and when I keep the program, should I execute that instruction which caused, caused me the exception, or the instruction following it; can I ignore the instruction; all these decisions are made by the interrupt service routine. So, today, we get divide by zero error, and your program aborts, that divide by zero is actually, is actually printed by an interrupt service routine. And, the interrupt service routine decides that, it needs to come, kill your program, because, there is a divide by zero. So, these are all some of the very different ways by which, you know, events of the nature of interrupt and exceptions happen in a system.

(Refer Slide Time: 12:44)



INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## Security Implications of Interrupts

**Attacks!**

- Denial of Service Attacks by repeatedly issuing interrupts.
- ISR is an attractive module to attack. If compromised, the malicious code will run in **Privileged mode** each time the interrupt is issued.

Information Security - Interrupt Service Routines  
4/25

NPTEL

So, the security implications of interrupts are very, very important. So, whenever you talk about secure operating system, or whatever, how does the operating system handle interrupt? So, this is some question that we need to answer very emphatically.

For example, I could get a denial of service attacks by repeatedly issuing interrupts; because, interrupts actually switches. So, you are working; it is an asynchronous event; I could give an interrupt; again it can, it can go. So, the processors, CPU will be made to keep serving interrupts, rather than serving the normal CPU processes. So, this is one way by which, if I have a control and I start invoking the interrupt service routine quite



frequently, then the processor cannot do the normal functionality but it can do something else; and it will keep on, you know, servicing the interrupts, and that is one example of a denial of service attack.

Similarly, the ISR is an attractive module to attack because, once I am sitting inside an ISR, I normally run with very high privilege, right. So, by attacking the ISR, and getting, penetrating into it, I get the privilege level of the ISR, which essentially becomes extremely important for me to compromise execution, to get into the system, in a highly privileged mode because most of the ISR works at privilege level zero. So, if I get into the ISR, nothing like that, I could create much more havoc, than other entry points.

(Refer Slide Time: 14:30)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## How do we disable Interrupts ?

- Non Maskable Interrupts**  
These interrupts cannot be disabled by the user
- Maskable Interrupts**  
Clearing the IF flag of the EFLAGS processor register disables Maskable interrupts. We can use CLI instruction to clear, and STI instruction to set the flag.
- Special Cases**  
PUSHF, POPF, Task Switching & IRET affect EFLAGS register implicitly.

9/25

NPTEL

So, there are non maskable interrupts, which cannot be disabled but there are maskable interrupts which can be disabled; yesterday we talked about that. So, if I clear the IF flag, in the EFLAGS processor register, it disables all maskable interrupts. And so, we can use that IF flag, the interrupt flag; I can use CLI to clear that interrupt flag; I can use STI to set back that interrupt flag. And, there are some special cases like, PUSHF, POPF, task switching and IRET, which will affect EFLAGS register implicitly, which because, it will, PUSHF will actually push the value of the flag registers into the stack but it cannot touch some values; while POPF will, write from the stack on to the EFLAGS register,

and some fields may or may not get updated. So, similarly, IRET, task switching and IRET can affect EFLAG register implicitly. Now, what happens is, all the interrupts today are all called vectored interrupts. So, whether it is software generated, hardware generated, exceptions, for every interrupt, there is a basic number. So, there is an interrupt descriptor table.

The interrupt descriptor table can have up to 256 entries; 0 to 255 and each interrupt in the system, as I mentioned yesterday, is mapped onto one such number. So, in the range 0 to 255, 0 to 31 is hardware defined; 32 to 255 is user defined, though not very strictly, but this is how they follow the standard.

(Refer Slide Time: 16:21)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## Interrupt Descriptor Table

**Need For IDT**  
To support relocatable ISR's and to configure permissions.

**Structure of IDT**  
Table of 256 entries, one corresponding to each interrupt. Each entry holds the descriptor that points to the ISR. This table is programmable.

**How do i find it?**  
Processor register **IDTR** holds its address

Praveen V — Information Security — Interrupt Service Routines

And, when I generate an interrupt 0, or interrupt 1, or interrupt 2, please note, on your right hand side you have the figure; you go to that corresponding gate for interrupt, and that gate will give you the base address of the code of the interrupt service routine and so, you can, and it also gives you the offset into that code. So, you can go there, and start executing the interrupt service routine. So, what will the interrupt gate have? We are going to see it now. It will have a pointer to a code segment descriptor; it will have a pointer to a code segment descriptor because that is the code, that is the descriptor which points to that segment of the memory, which has the interrupt service routine, and it can

also have offset into that. So, I can go and start executing.

(Refer Slide Time: 17:12)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## IDT Entries a.k.a Descriptors

**Task Gate Descriptor**

Used when the ISR is coded as a separate task.

**Interrupt & Trap Gate Descriptor**

Used when you want to transfer control to ISR by using a far jump.

The processor uses the IF Flag bit of EFLAGS register differently.

**Task Gate**

	P	D	L	0	0	1	0	1		4
										0

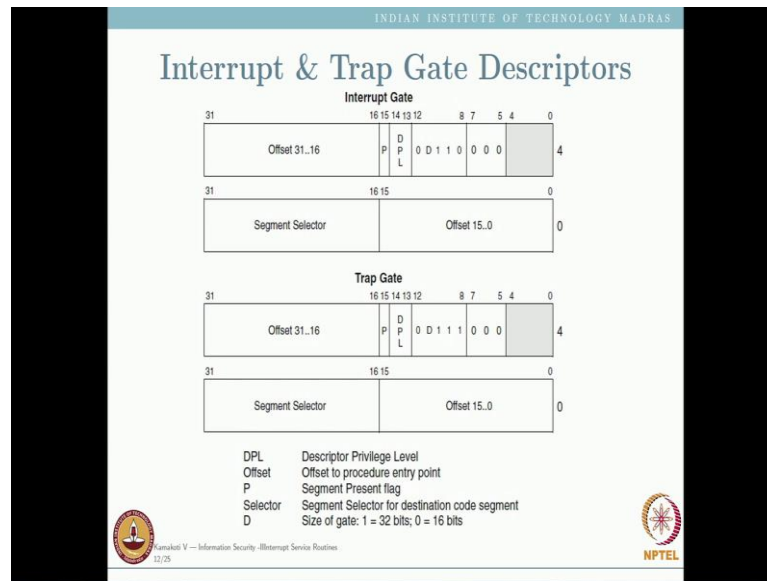
  

TSS Segment Selector		0
		0

Samadhi V — Information Security — Interrupt Service Routines  
11/25

So, there are three types of; so, in a GDT, we saw the GDT this morning. There were different types of descriptors there. In the IDT, there can be 3 types of descriptors. One is the task gate descriptor, which I, which is defined like this; which we saw yesterday, right. What was the task gate descriptor trying to do? It does a task switching completely, right. And, we also told, some classes of interrupts which will require task switching, right. The other two types of descriptor are the interrupt and trap gate descriptor.

(Refer Slide Time: 17:47)



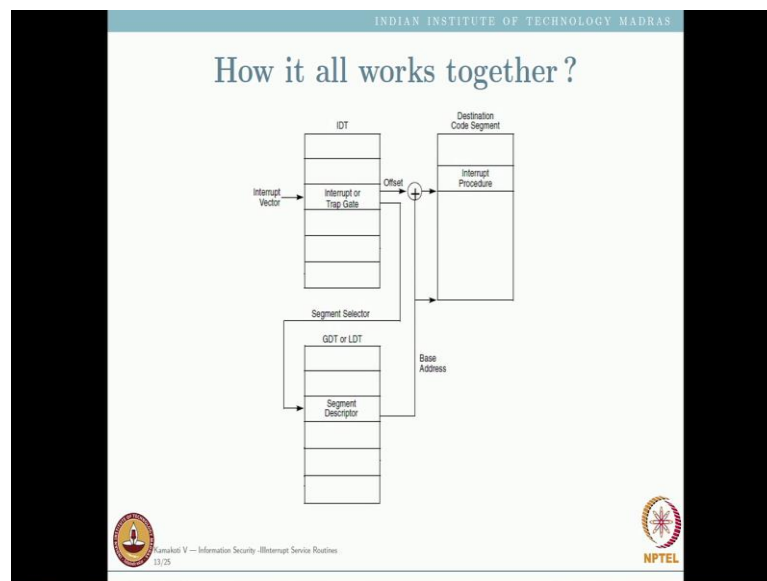
The interrupt and trap gate descriptors will allow that interrupt service routine to work like a function call. Yesterday, I told you, an interrupt service routine can be, in the previous session it was mentioned that, the interrupt service routine can be like a function call. It can also be a complete task switch. If I use a task gate, this also I had mentioned in the previous session, if I use this as a task gate, then, you will do a complete process switch; if you use an interrupt or a trap gate, then, you use, then, you treat it like a function call. So, the interrupt and trap gate looks like this.

Interrupt gate, again, there is a segment selector, which points to which code segment is going to execute me; and then, there is an offset, which basically says, from in that code segment, from which offset should I, from the base of that code segment, which offset should I start executing; similarly for the trap gate. So, the difference between interrupt gate and trap gate is that, in the case of an interrupt, what happens, in the case of an interrupt, the INT, the instruction at which the interrupt happens can be restarted, right. So, when I see an interrupt gate, what is going into the stack, that is where I should return, will be the current instruction pointer.

When I use a trap gate, what goes into this, will be the next instruction pointer; because, that instruction, I am going to ignore; you differ. So, that is the difference between an

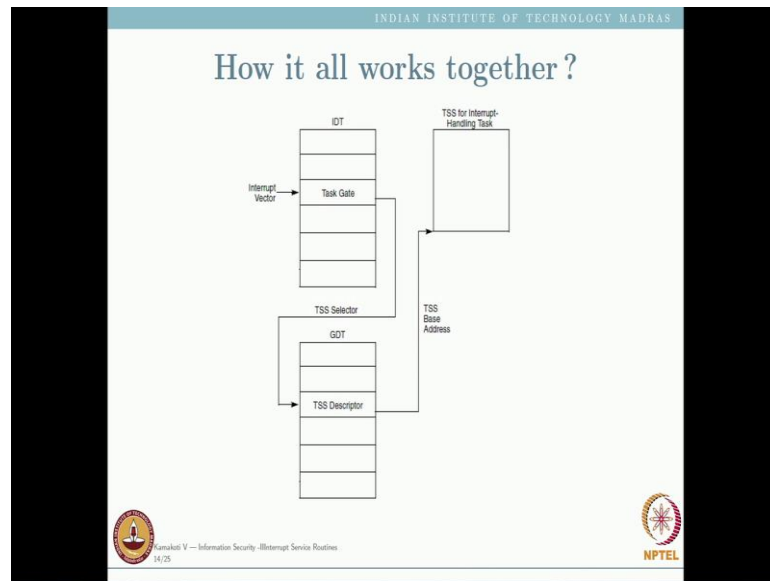
interrupt gate and a trap gate. The interrupt gate, when it returns back, it will store the return address of the interrupting instruction. A trap gate will store the instruction of the next possible instruction. So, that is difference between interrupt and trap gate. In terms of organization, please note that the eighth bit is 0; now the eighth bit is 1; that is the only difference between interrupt gate and trap gate.

(Refer Slide Time: 19:57)



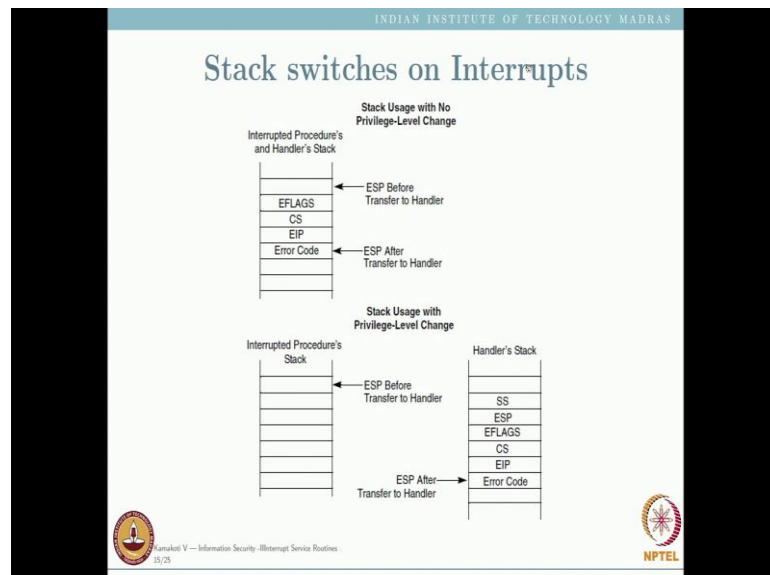
So, what happens when an interrupt comes? There is an interrupt vector; this goes to the IDT. There is an interrupt or trap gate there. If it is an interrupt gate, that interrupt gate has what? It has a segment selector. So, that segment selector actually points to a segment descriptor; that segment descriptor gives a base address to which I will have the offset; and, that is where the interrupt procedure will be stored, and you start executing on it. This is how normal interrupt works. But, in case I need to do a complete task switch for double fault etcetera, that I explained yesterday, what will happen? The, the same thing; so, there is a interrupt vector; it sees a task gate.

(Refer Slide Time: 20:49)



That task gate will point to a TSS descriptor; the TSS descriptor will point to a TSS, and you start executing. So, this is a complete task switch that is happening here.

(Refer Slide Time: 21:11)



And, when an interrupt service routine comes, if the interrupt service routine and the, if the interrupt service routine and the program that is interrupted are there in the same

privilege level, there is no stack switch that will happen. But, if the interrupt service routine and the process that created the interrupt are in two different privilege levels, then, there will be a stack switch. The interrupt service routine will use the stack corresponding to its privilege level. So, this diagram basically explains you.

If I do not have a stack switch, the stack pointer before transfer to handler will be like that, in the top, and the stack pointer after transfer to handler. So, what will happen is, what will be stored there is EFLAGS; your entire flag will be stored. Then, your current code segment selector, the EIP, the current code segment selector; then, the EIP is what, the instruction pointer, and the error code, all these thing will be stored in your stack. If the interrupted, interrupt service routine and the task which got interrupted, if both are at the same privilege level, then the stack switching will not happen; and, this is what will be stored in the interrupted procedures stack.

If in case both are different, then what will happen? In case both are different, then, that is, the interrupt service routine and the process which is interrupted, are in, are in two different privilege levels, what will happen; there will be a stack switch. And, in the interrupt service routine stack, right, in the handlers stack, all these error code, EIP, CS, everything will be stored. In addition, your SS also will be stored, because, I want to restore back my original stack, right. So, so, this is how stack switches happen on the interrupts, and this is extremely important from a security point of view. We did elaborately, yesterday, in the last session, on this.

(Refer Slide Time: 23:30)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## Protection Mechanisms in Intel x86

**CPL**  
Current Privilege Level : Privilege level of the descriptor of the currently executing task.

**DPL**  
Descriptor Privilege Level : Privilege level of the destination segment, stored as a part of its descriptor.

**RPL**  
Requested Privilege Level : Override privilege level assigned to the destination segment selectors.

**Example**  
if the DPL of a data segment is 1, only programs running at a CPL of 0 or 1 can access the segment!

**Protection Rings**

The diagram shows four concentric circles representing protection rings. Level 0 is the innermost ring, labeled 'Operating System Kernel'. Level 1 is the second ring, labeled 'Operating System Services'. Level 2 is the third ring, labeled 'Applications'. Level 3 is the outermost ring, also labeled 'Applications'. Arrows point from the text labels to their respective rings.

© 2004 IIT Madras — Information Security — All Rights Reserved  
16/25

NPTEL

Now, there are many, there are many protection mechanisms in the Intel X86 architecture. There is a current privilege level; there is a descriptor privilege level, and there is a requested privilege level. We have seen that in great detail, right and this basically, why we are talking about here is that, your level 0 could be operating system kernel, level 1 and level 2 can be operating system services, level 3 can be applications. So, this is how the stack is organized, and the, this 4 levels, the operating system corresponding bits of the operating system are put in each level, and there is enough amount of protection across these privilege levels.



(Refer Slide Time: 24:23)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## Privilege Checks on Control Transfer

Types of Control Transfer  
JMP, CALL, RET, SYSENTER, SYSEXIT, INT i, IRET, Interrupts and Exceptions.

Source Segment Descriptor

CPL

>= of

DPL

>= of

RPL

Destination Segment Descriptor

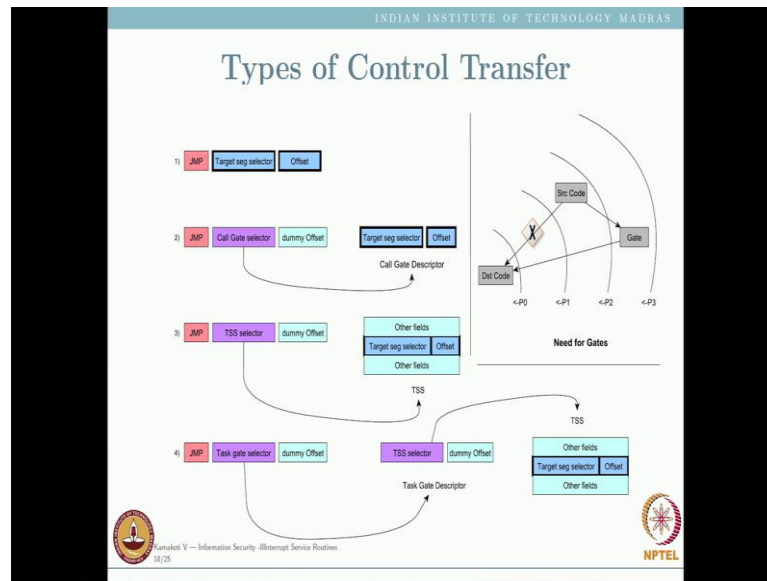
Destination Segment Selector

NPTEL

Copyright © 2008 IIT Madras. All rights reserved. Information Security - Interrupt Service Routines 17/25

So, what happens is that, when I do a control transfer, Jump CALL, Return SYSENTER, SYSEXIT, INT i, IRET, interrupts and exceptions, all these things are control transfer from one program to another program. There we go and see, if the CPL is less than or equal to DPL, or RCPL is, or your RPL is greater than or equal to that of DPL. So, all these checks will be done; we have seen very, very extensively on this check; and these checks need to be done for privilege, while I am doing a control transfer.

(Refer Slide Time: 25:09)



So, what are the types of control transfer? We just saw Jump 0 x 8 colon something; 58 or something. There, I just directly give the segment, code segment. So, I give the target segment selector, and I jump; that is the first number, time of control transfer. The second thing is Jump, give a call gate selector, and dummy offer, and from the, from the call gate selector which is a target segment selector, that call gate selector will give me which is the target segment collector, and then, the offset is also given in the call gate. So, I will start executing from that. I could also say Jump TSS selector and dummy offset. Why dummy offset, because, the task stage segment to which I am jumping will have which instruction that needs to be executed, and so, you start executing from that.

Similarly, I could have jump task gate collector dummy offset. So, I go through a task gate, to a, to a task gate segment selector, to a task gate segment and start executing. So, these are 4 types of control transfer, wherein I, we can go from one code to another. The first type of control transfer can happen between one flow and another flow; any, any flow and any other flow. While the second type can happen if there exists a call gate sector that supports this, the third sort of jump is that, directly we use the TSS selector and go there; and the fourth jump is, we use the task gate selector and do this act. So, these are all 4 types of control transfer under this secure privileged environment.

(Refer Slide Time: 26:57)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## Interrupt Control Transfer

Interrupts do not have RPL. RPL checks are ignored!

CPL checks are ignored **only** for hardware generated interrupts!

ISR is always placed in P0. Think why?

a) Successful Access from Source Program      b) Successful Access from Source Program

Checks for Software Generated Interrupts

© 2014 NPTEL. All rights reserved. NPTEL V — Information Security — Interrupt Service Routines 19/25

Now, note that, we also introduced a notion of RPL. Please note that, interrupts do not have a RPL. So, RPLs are removed while the CPL checks that we had, are ignored only for hardware generated interrupts. So, for all the software interrupts, we still do a current privilege level check because if I do not do a current privilege level check, this has a very big security implication; any Tom, Dick and Harry can create lot of interrupts, and thereby give you denial of service. So, in your definition of security, the confidentiality, integrity and availability, the availability will take a big task. If you do not go and you now, do a check on who can execute an interrupt service routine.

So, beyond that 31, or 32, there is a check that, you can execute the interrupt service routine only, if your CPL is less than your descriptor privilege level. So, the interrupt service routine will have a corresponding descriptor, that will have a privilege level; the fellow who is calling that, should have at least that privilege level, if not numerically low and that is the reason, why this, this thing has become extremely complex.

Now, let us see here, when I do a software generated interrupt, right, always if there is a source code, if the interrupt gate, if the interrupt descriptor in the IDT has a privilege, is a descriptor privilege level much lesser than mine, then, there is no way by which the interrupt could be served. So, I need to have lesser; lesser means, numerically higher

privilege. For example, if I am a, if I am a privilege level 3 code, but I am generating a software generated interrupt, say INT 50 and if the DPL of that code is only 2, for example, then as a 3 level privileged fellow, I cannot go and execute that ISR; then, this is true for all software generated interrupts.

(Refer Slide Time: 29:27)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## Error Codes for Exception handlers

**Who generates it?**  
Processor pushes the error code onto the ISR's stack.

**EXT**

Set on External hardware interrupt

**IDT**

1 - index into IDT to find ISR.

**TI**

Valid when IDT bit is 0  
1 -index into LDT.  
0 indicates GDT index.

31	3 2 1 0						
Reserved	Segment Selector Index						
	<table style="border-collapse: collapse;"> <tr> <td style="border: 1px solid gray; padding: 2px;">T</td> <td style="border: 1px solid gray; padding: 2px;">I</td> <td style="border: 1px solid gray; padding: 2px;">D</td> <td style="border: 1px solid gray; padding: 2px;">E</td> <td style="border: 1px solid gray; padding: 2px;">X</td> <td style="border: 1px solid gray; padding: 2px;">T</td> </tr> </table>	T	I	D	E	X	T
T	I	D	E	X	T		

Anantakshi V — Information Security, Interrupt Service Routines  
28/25

So, then, for every exception, there is error code that is generated. The error code normally is 32 bits, in which the 16 to 31 are reserved. The remaining, the first to, first 16 bits, there will be 13 bits which will give you a selector because the selector, last 3 bits are any way 0. So, it will, it will give us a selector index; this corresponds to which code segment actually went and created this error and that code segment can be in IDT.

It can be, first and foremost, that can be a external hardware interrupt, and that is given by that EXT pin, EXT bit; if it is 1, then, it is an external hardware; if it is 0, then, it is a, it is not an external hardware that created an interrupt. The next thing is IDT; if it is 1, then, go and look for this particular selector in the IDT, because I could create an exception while executing an interrupt service routine also. Any instruction inside an interrupt service routine can also cause an exception. So, in that case, I have to go and find out, if it is there in the IDT and if your TI bit is zero, right, sorry, your IDT bit is zero, then, then only your TI bit becomes valid. If your IDT bit is 1, you know that, it is

caused because of an interrupt. So, there is nothing more I need. But, if your IDT bit is zero, then, if your TI bit is 1, then, look for that code, which caused interrupt in the LDT, or look, if it is zero, look for the code that causes this interrupt in the GDT. So, your exception handler, the error code of the exception handler, gives very good, you know, infrastructure for doing, for understanding the error, and going and locating which particular code actually caused the error.