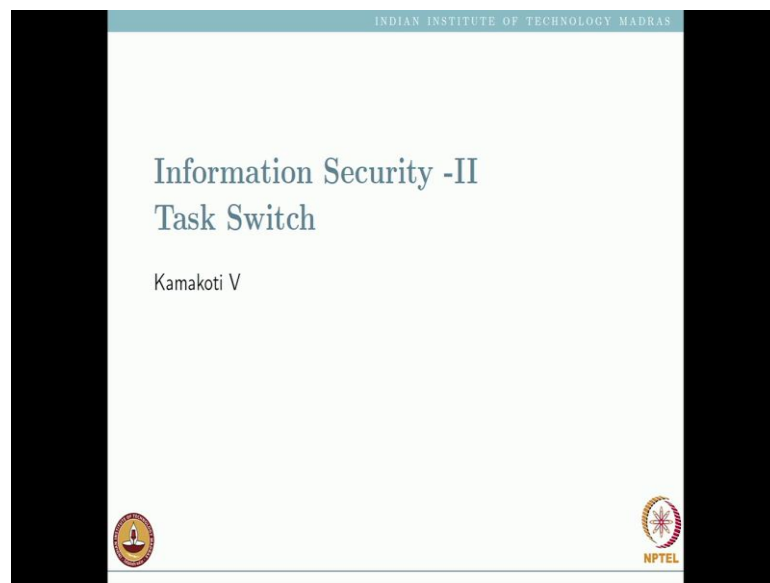


**Information Security – II**  
**Prof. Kamakoti**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture – 30**  
**Task Switch recap - Week 6**

(Refer Slide Time: 00:09)



So welcome back and we go onto the next demo session which will be task switching. We discussed task switching in a very broad way in a previous session. Now we will have much more better deep understanding of task switching and then followed by one code which will be demonstrate this task switching in a full perspectives.

(Refer Slide Time: 00:40)



INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## What is a task ?

**Definition**

Task is a unit of work performed by processor to :

- a) execute process
- b) serve operating system utility
- c) handle interrupt or exception

 Prasad V - Information Security - IIT Madras 

Now, let us see what is the task, task is nothing but a unit of work performed by the processor which is to primarily to execute a process or serve some operating system utility or handling interrupt or exceptions. So, task can be of three forms; one it can be application program executing or it can be a device driver executing or some operating system utility that is executing, a scheduler executing etcetera or it can also be in a interrupt service routine that is executing. So, there are three different type of task by this type of a classifications. So, everything that is executing in the processor is essentially a process and that is called a task. So, everything that is executable on the processor essentially is a task.

(Refer Slide Time: 01:39)



INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## What is Task Management and why it's important for information security?

**Task management**  
How hardware handles multiple tasks and protection among them

- a) Task execution
- b) Task switch(Context switch)

**If task management is not made protected**  
access or corruption of secured data(belong to higher privilege user task) by lower privilege user task—**threat to information security**

 Module V — Information Security: IT Task Switch 

Now, what is task management, and why it is very much important for the information security. Now what do we understand the term task management, how hardware handles multiple tasks and protections among them, this is the most importance thing that we need to look at. So, we have to look at from two perspectives, namely the task execution perspective and the task switch perspective. When I am executing in a task, I have certain privileges, I do not have certain privileges. When I move from one task to another task, I can make the move; I cannot make the move. I can make move from my task to some task a, but I may not allowed to move from my task to another task b. So, when we talk of task management the main issue is executing task is the programmers problem, but when I execute the task what are the privileges that I have what are privileges I do not have essentially is the part of task management. And when I complete the task and move out the during task I switch to some other task which are the task and supposed to switch and which I cannot switch this is also part of the task management. And these two conditions basically governed by security.

Why should I not go to some other task for security reasons, why should I not when I am executing, why should not you know access certain routines or access certain objects that is again due to privilege reasons or security reasons. So, security basically dictates task management from these two perspectives. Again repeating that when I am executing the

task what is it I am allow to access and, but I am not allow to access. And from a task, which are the other tasks I could switch on and which I cannot switch on. So, if task management is not made protected then anybody can access anything else and essentially there is nothing call secure data at all and so that is the basic threat to the first and probably the largest threat to information security. So, this task management forms a hard cracks for information security. Now what will do is how x 8 6 architecture handles these both task is execution and task switch is what we will see as a part of this session.

(Refer Slide Time: 04:39)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## What information a task should have ?

**Task Structure**

- 1 task execution space : code , stack, one or more data segments used by the task
- 2 task -state segment(TSS) :
  - 1 metadata of task execution space (to identify the memory location of each segment)
  - 2 task state information/context of a task (highly important for task switch)

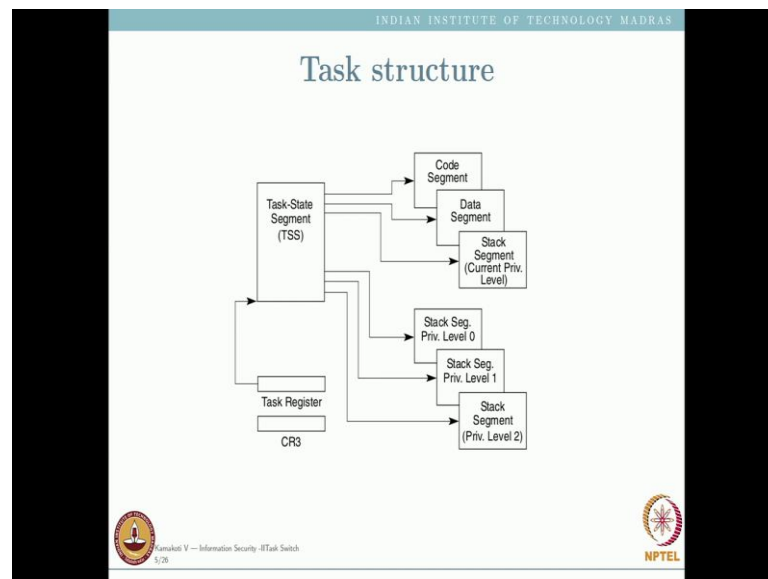
© 2014 IIT Madras — Information Security — ITTask Switch 1/26

NPTEL

Now, what is inside a task, a task has in an execution space. What you mean by an execution space, space means some amount of memory in which I carry out my execution using which I carry out my executions. So, the memory basically has code it has stack and it has one or more data segments that is used by tasks. So, where are these information captured like as a task what is the data segment I can use what are the you know stack segment I can use etcetera, these are all captured what we call as the task state segment. In a task state segment, essentially, I could say that these is a metadata of task execution space essentially used to identify the memory location of each segment right. So, we have already saw the structure of a task state segment it as hundred and four bytes, but I am going to go much deeper into that in this session. So, what will the task state segment hold, it hold all details about the tasks and what are those details, one of

the most important thing is that it holds the context of the task. What is the context of a task, I explain that in the earlier sessions, but I repeat here, it is the basic minimum information require for me to restart a tasks after once it gets suspended and this information is basically stored in task state segment.

(Refer Slide Time: 06:16)



Now, this is the basic task structure. There is a task state segment. And in the task state segment, you store all the select task to the code segment, to the data segment, to the stack segment, to the stack segment of the current privilege level and then you have pointers to stack segment of privilege level of 0, 1 and 2, and then you can store your LDT - local descriptor table then also you can store what you call as CR 3 which is the start space table. As you go into the version of Unix, you will understand that a space table every task and have its own space table also, so that is also captured in the task state segment. In addition, where will the task state segment start that location is basically pointed to directly or indirectly by a task register. So, this is how a task state segment looks like, and the task state segment essentially dictates the structure of the task.

(Refer Slide Time: 07:37)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## What are the task state information ?

Task state information : define the state of currently executing task

- 1 task's current execution space-segment selectors(CS,DS,SS, ES, FS, and GS)
- 2 general purpose registers(EAX,EBX,ECX,EDX)
- 3 EFLAGS, flag register information
- 4 EIP – instruction pointer
- 5 CR3 –physical address of the page directory used by the task
- 6 Task register(TR)
- 7 LDTR
- 8 ESP of PL0, PL1,PL2 stacks
- 9 Link to previous executed task ( for task switch purpose)

NPTEL

So, the information essentially has in addition to the segment selectors the current segment selector CS, DS, SS, ES, FS and GS, please note that in point number seven in your slide you have the LDTR, it essentially has pointer to its current LDT. So, all the segment associated with the stack is basically captured through that LDT, it also stores at the time of suspension, whatever was where the current values of current selectors that is stored in CS, DS, SS, ES, FS and GS, that also it captures exclusively. In addition, it store a value of the all the general purpose registers EAX, EBX, ECX, EDI, ESI, ESP all the eight general purpose registers sorry, there are only four, totally eight, eight of them these are also stored at the time of suspension that values stored. At the time of suspension, whatever is the value of e flags that is also stored. At the time of suspension, whatever the value of EIP - extended instruction pointer that is also stored.

Then the physical address of the page directory used by the task because each can have its own page directory as I mentioned here that is also stored there. And the task register value that is load for the task register values, you can also store that and then the stack pointers of PL 0, PL 1, PL 2 stacks and the actually the selectors of SS 0, SS 1, SS 2, SS 3, all these things are SS 2, all selectors for these different stacks are also stored. And then there is also link to previous executer task which you can use for task switch purposes.

(Refer Slide Time: 09:39)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## Task switch management

### Task switch management data structures

- Method 1
  - 1 Task register- direct pointer to TSS descriptor
  - 2 TSS descriptor- descriptor for TSS(like any other segment)
  - 3 Task-state segment (TSS)-already discussed
- Method2
  - 1 Task-gate descriptor - indirect/protected pointer to TSS descriptor
  - 2 TSS descriptor- descriptor for TSS
  - 3 Task-state segment (TSS)
- NT flag in the EFLAGS register- Controls the chaining of interrupted and called tasks

7/28

NPTEL

So, to sum up the data structures that need for a task switching, there two ways I jump or call directly to a TSS descriptor. And immediately there will be a task switch because the TSS descriptor will pointed task state segment; and in the task state segment, there will be a EIP. So, whichever task state segment that is TSS descriptor points to I jump to the that task. So, that is method one. Another way is, I go through a task gate and from there task gate I go to the TSS descriptor and then I go to the task state segment and I get the new address and start executing. So, there are two methods by which I can do task switching. So, the method one, involves a task register that directly points to the TSS descriptor and the TSS descriptor is a descriptor for the TSS and that will point to the TSS and I get a EIP which is a next instructions to be executed. In the method two, I go to the task gate descriptor which is indirect task protect pointer to the TSS descriptor and from there I go to the TSS descriptor and from there I go to the task state segment. So, these two methods do exists; well, the second method is introduced to get more security for than the first method.

And there is one thing called nested task flag - NT flag in the extend flags register which actually controls the chaining of interrupted and call tasks. So, one task call another task automatically that NT flag becomes one. When one task essentially gets an interrupt and by that it goes to the interrupted task, then NT flag is basically set to one. Why is NT flag

set to one, if I do a jump which is not set a one, because there is no way by which going to return back; but if I do a call, then I will return back. So, this is why the NT flag basically tells is there a task switch that will gives out a possibility of going back to the original task at all; if that possibility is exists then your NT flag is set to one otherwise NT flag is set to zero. So, this is something which has being you know followed by the x 86 architecture.

(Refer Slide Time: 12:13)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## Method 1 : Task register

The diagram shows a Task Register with two parts: a 16-bit Selector (Visible Part) and a 32-bit Base Address and 16-bit Segment Limit (Invisible Part). The Selector points to a TSS Descriptor in the GDT. The Base Address and Segment Limit are used to access the TSS descriptor's visible and invisible parts respectively.

- Two components :
  - 1 **Visible** /software controlled part - 16-bit segment selector
  - 2 **Invisible** /processor controlled part- 32 bit base,16 bit limit and descriptor attributes for TSS descriptor(caching purpose)
- Instructions :
  - LTR (Load task register) : Visible and invisible portion from TSS
  - STR (store task register) : Visible portion into GPR or memory

Semester V — Information Security — ITTask Switch
NPTEL

So, this is method one for task switching. So, there is a task register which will which stores a selector that selector points to a TSS descriptor inside the GDT. Please note that TSS descriptors can be stored only in the GDT. Now this TSS descriptor has a base address and the segment limit. So, the moment I stored, when I load LTR with that selector value the base address and the limit goes into the invisible part, why I do that for performance reason. Yesterday, we also saw in the segmentation that there is always a hidden part, and the data goes stays in the hidden part, the limit base and the privilege level goes and stays there. So, here the base address and limit goes to an invisible part and based on that now this how a TSS is basically defined and stored. So, now, the TSS has two components as I see, the visible is the software controlled part which 16 bit segment selector, and what does segment will select, it will selected TSS descriptor. And there is a invisible part which is controlled part which has 32 bit base, 16 bit limit and



descriptor attributes for TSS descriptor right. So, there are some caching purpose etcetera. So, all this things are invisible part. Why I need that invisible part as programmer I do not really care about that invisible part, but why I need that because for performance reason.

So, if I go and change something in that TSS, I need to go and do again in LTR - load the task register to see that those thing becomes effective, you got this. If I go and change something in the TSS descriptor have to do in LTR again, so that the hidden parts get updated. So, this is very similar to what we are seen in yesterday's lecture right. If I go and change a segment descriptor base address, unless I load the selector again into the corresponding segment register that change will not be effected, because the base limit etcetera are stored in the hidden part and there is no way by which I could link this hidden part to the actual memory locations. So, when I go and change the memory location, it is the duty of the compiler to generate a code which will again move, once I change that it will again come and move the corresponding selector inside, so that we start execute it. So, this is very, very important.

Now there are instructions like LTR and STR which load task register will load the visible part that is given by the programmer and the invisible portion is given by the TSS descriptor. Similarly, STR will store the visible portion into the some GPR some general purpose memory or general-purpose register or memory. So, both of this are possible here.

(Refer Slide Time: 15:40)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## TSS descriptor



- segment descriptor for TSS
- each TSS should have only one TSS descriptor( to ensure one busy flag per task)
- task can be dispatched only if TSS descriptor is **accessible**  
i.e CPL  $\leq$  DPL of TSS
- Busy flag (B) : whether the task is busy (task is currently running or is suspended)

**TSS Descriptor**

31																24 23 22 21 20 19																16 15 14 13 12 11																8 7				0			
Base 31:24																G		V		A		Limit		D		P		Type		Base 23:16																									
																1		0		1		15:16		1		0		1																											
31																16 15																8																0							
Base Address 15:00																Segment Limit 15:00																																							

AVL Available for use by system software  
B Busy flag  
BASE Segment Base Address  
DPL Descriptor Privilege Level  
G Granularity  
LIMIT Segment Limit  
P Segment Present  
TYPE Segment Type

© 2008 V — Information Security — IIT Madras  
9/28



So, we actually saw about the TSS descriptor. Now let us now look at this TSS descriptor in great detail. So, TSS descriptor is pretty simple it has base address of 32 bits as you see in the lower part of the slide; it is between the 0 to 32. And then there is segment limit which is again 20 bits, it is like any other descriptor. And the type is 1 0 B 1 that B is available bit and basically do not care and then P the present bit and DPL is there then there is available there is a G bit. So, B bit is basically the busy flag weather the task is busy or task is currently running or is suspended. So, this is basically as far as the user is concerned, it is a do not care you can fill it whatever you want, the system will be update this B bit accordingly. Now what we need to get here is in the TSS descriptor there is already DPL. And very slowly we are trying to build up a security; we have to extremely carefully understanding this security because this is the crucial thing about you know transitions, because all vulnerable thief at all that could come these this should come in these type transitions.

Now there is a DPL here in this TSS descriptor. So, if some other code wants to use this descriptor to do a task switch that CPL should be less than or equal to this DPL. So, there is one first level check that is made. So, as an operating system, I can put the TSS descriptor only in the GDT, I cannot in the LDT. And once if I put in the GDT, there also as an operating system can set a privilege level so that only some privilege code can

access I could have a control that and that is very important for us to realize.

(Refer Slide Time: 18:08)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## Task-state segment (TSS)

31	15	0
32-bit Base Address	LDT Segment Selector	160
	ES	168
	FS	172
	GS	176
	SS	180
	EIP	184
	CR3 (PDBR)	188
	ES0	192
	ES1	196
	ES2	200
	ESP0	204
	ESP1	208
	ESP2	212
	Previous Task Link	216

- LDT segment selector : segment selector for the task LDT
- Segment selectors : ES, CS, SS, DS, FS, and GS registers prior to the task switch.
- EIP : instruction pointer prior to task switch
- CR3 : base physical address of the page directory to be used by the task

Sankar V — Information Security: Task Switch 10/26

Now, this is the actual task state segment that is at 104 bytes that we seen yesterday, and all that we described here is still here in this in this diagram on your left side. And you have place holders for all the six segment registers, you have LDT segment selector that is for every process can have its own LDT you have a place holder for all the eight general purpose registers EAX to EDI. Then you have E flags which is also the flag at the time of suspension what would be the suspension of the task what would be the value of the flags. You have EIP which is the instruction pointer. Then we have CR3 which is the page directory ah base register. So, you can have your own page for every task. And then you have SS2, SS1, SS0 which are the segment selector for this stacks then ESP2, ESP1, ESP 0 which are the you know stack pointers. And then you have also a link for previous task.

If a task A calls task B then the task A thing is link to task B. So, there will be a link from task B to task A, because after you finish executing task B and you do return then it has to go back to task A. So, this in operating system terminology, this can be viewed as a process control block or a context of the process. And this is essentially necessary for me to restart the process after once it is suspended I have to get back. Where it will restart,

please note that 32 byte here is EIP, and it will restart at that instruction pointer the address stored at EIP. So, this how the task state segment is conceived off and you see that there are lot of black or gray areas and those things are undefined and they are they are reserved. So, you can put any do not care value when you are building this task state segment.

Now let us take a very quick view here that when the operating system wants to execute a task you always seen in the text books that you create a task. What it means to create a task, there to create a task which you going to do as a part of assignment. To create a task what we do, we create the data segment, the stack segment, code segment, their LDT, their page table if necessary then we initialize all the registers, then we initialize C flags, we load the programs say, now we loaded in the previous session we loaded the program in a thousand. So, we also load the program somewhere and that starting point you store in that EIP, then create stacks for the different privilege levels put everything and now go and task switch into this TSS. This is what we mean by operating system creating a process. So, the operating system creates the TSS fills it with the correct entries and then it transfers control to the task state segment; it jumps into the task state segment or it calls the task state segment, so that it start executing and when the task returns, it will come back to the operating system. You got this, so this is how, this is what we mean by creating a task from an operating system prospective.

And what things can go wrong here, for example, if your CS-code selector has privilege two, your SS should also have privilege two. So, things like that can go on. For example, if your CS has privilege one and but your SS has privilege zero or your DS has privilege zero then it will not allowed to hold. So, when the task switch is happening, the architecture checks all these things before it allows the task switch right. For every segment selector it will go and check if the first it will load the code segment right, so then now the current privilege level is set, based on that it will check for all the other segments whether certain segment can go or not. So, there is lot of architectural level checks that happens before you do a task switching. So, even if the operating system makes a mistake, if you fix the code segment correctly, all the other things will essentially be privileges at least at the privilege level all those checks will be done by the architecture. We are going to describe those steps in a very quick way as you progress in

this lecture, but please note that this check essentially adds lot more confidence to task switching. And if we call that this is one of the most important architectural aid for security. So, the TSS descriptor is a segment...

(Refer Slide Time: 23:44)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## Method1 : Task-state segment(TSS)

32	16	8	0
TSS Base Address			
TSS Segment Selector			
SS0	SS1	SS2	
ESP0	ESP1	ESP2	
EBX	EBP	ESI	EDI
EAX	EIP	EIP	
IOPL			
Previous Task Link			

- PL 0,1,2 Stack :
  - logical address formed by stack segment(SS0,SS1,SS2) and offset(ESP0,ESP1,ESP2).
  - static fields to the task
  - Though SS and EIP change for task switch
- T : raising debug exception when task switch occurs from the task
- **Previous task link** : segment selector for the TSS of the previous task

© 2011 IIT Madras. All rights reserved.

So, there is also some small things like there is a T bit at the hundred byte; if this T bit is set then it will raise debug exception when the task switch occurs from the task right. So, when I move from this task to another task, and this T bit is set to one then it will raise the debug exception, debug exception is exception number two or exception number three one of those. So, it will automatically go to that service. So, whenever the task is switching, I know that it is switching right. So, this is another small feature that is available as a part of task state segment.

(Refer Slide Time: 24:35)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## Method 2 : Task gate descriptor

DPL Descriptor Privilege Level  
P Segment Present  
TYPE Segment Type  
Reserved

- indirect reference to a task
- can be stored in GDT, LDT or IDT
- (TSS Segment Selector) → TSS descriptor
- TSS descriptor access rule :  
CPL/RPL calling Task gate selector  $\leq$  DPL of Task gate

© 2006 NPTEL

Now, the other way of accessing or going to another task is through a task gate to a TSS descriptor to a TSS. The first method was directly to the TSS descriptor to the task state segment. The second method is I go through a task gate to a TSS descriptor to the task segment. So, why do you need this, can anybody tell me why do you need this?

Student: ((Refer Time: 25:22))

Student: ((Refer Time: 25:26))

So, I can do it at the TSS descriptor level. If you say, I can make it more secure then I can make it happen in the TSS descriptor stage itself.

Student: But in this segment always ((Refer Time: 25:38)) flag of which process is ((Refer Time: 25:42))

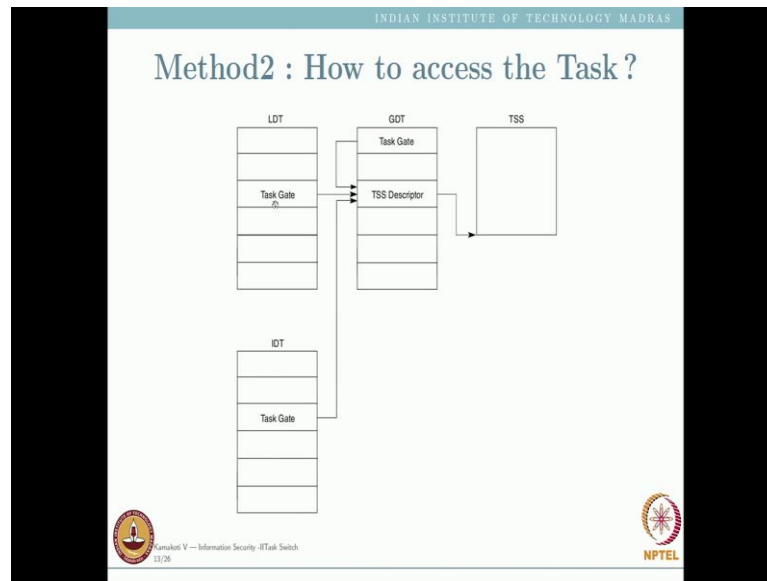
No, the reason is that task gate is like a call gate right. So, please note that if I, so let us take this scenario, I have three privilege level, three codes A, B and C. I want A alone to jump to some task, B and C should not jump to the task. If I want to permit A to jump to the task using only the TSS descriptor, what will happen, I have to put the TSS descriptor

where in the I can only put TSS descriptor in the GDT; and in the GDT, I have to create a privilege level three. Now if I create privilege level three there then all these fellows can access that, but I want only A to access not B and C. So, what I do, I make it privilege level zero, by put another gate there which point to that right and that gate I make it privilege level three. So, I can go through that gate through this task gate through the TSS descriptor and then basically go and do the task switch. And this gate, I can put it in the LDT.

So for process A alone I will put a task gate in its LDT which will point to task stage segment descriptor in the GDT and it can go there. And for the but the task stage descriptor will have privilege zero, so I will go only through the task gate. While for the other two, I will not put the task gate there in their LDTs. So, they cannot access, but I can still access right. So, there is something like many point of time, you will have certain objects which you want only certain people of a privilege level to use, I do not want everybody in a particular privilege level to use me. I want to classify that subset of people who need to come and use me, correct. I do not want every privilege level three to come and use me, but I want one fellow to use me correct. And to achieve this, I use the notion of this task gate right. So, I make myself PL 0. So, no PL 3 fellow can jump to me, but for one fellow who wants to access me for that person alone I will give a task gate in its LDT, so that he can use that to access me, but other fellows cannot access. So, this is very, very important.

So, the structure of the task gate is there is a TSS segment selector here, there is a present bit DPL 0 0 1 0 1, this is five, this is a type and this is a system descriptor. So, it is zero present DPL and I just have TSS segment selector. So, the task gate descriptor is very, very simple to see. So, the thing is my CPL or RPL of the calling task gate selector should be less than or equal to the DPL of the task gate. So, when I am calling the task gate, I say I jump to the task gate my CPL should be less than or equal to DPL of task gate; if that is done that is enough then there is no more check right. So, this is how this works.

(Refer Slide Time: 30:04)



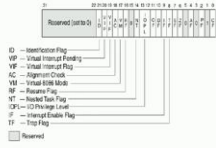
So, this is the LDT, IDT, so the task gates can be in LDT and IDTs, for example IDT also yesterday I told there are instances where I cannot use a interrupt gate or a trap gate, I need to use a task gate actually I need to do a context switch for solving certain interrupt related routines. Why should I do that we took the case of the double fault. A privilege level zero code creating a stack fault having a stack fault, and it tries to push more things in to the stack and that creates a double fault. In one of the sessions, I also clearly explained a difference between double fault and a fault got when executing interrupt service routine, so that is very these are something that we need to very carefully understand. So, the IDT will also have a task gate, your LDT will also have a task gate, and this task gate will go to the TSS descriptor, your GDT can also have a task gate and then it can go to a TSS descriptor, and this is a TSS.



(Refer Slide Time: 31:12)



INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## NT flag in EFLAGS register



The diagram shows the 32-bit EFLAGS register with bits 31 down to 0. Bit 14 is highlighted as the NT flag. A legend on the left identifies the flags: ID (Identification Flag), VP (Virtual Interrupt Pending), VIF (Virtual Interrupt Flag), AC (Alignment Check), MM (MMX State), RF (RFX), NT (Nested Task Flag), OF (Overflow Flag), OFE (Overflow Enable Flag), and TF (Trap Flag). Bit 14 is marked as 'Reserved'.

- Nested Task flag(14th bit)
- for the chaining of caller and called tasks
- **caller** task CALL instruction, interrupt or exception sets NT flag for **called** task
- IRET modifies this flag, on return from the called task

 © 2008 IIT Madras. All rights reserved. Information Security - IIT Madras 

And note that there is a nested flag NT flag in the EFLAG register, this is the fourteenth flag of EFLAG register. So, what will happen is there is if I am going to do a task switch for example, there is a caller task which is going to call another TSS descriptor through a call instruction or it can come through a interrupt or exception, the moment I call a task state segment selector or a task gate either through interrupt or through normal calls or any exceptions, immediately I go and set the NT flag. What do you mean by NT flag, NT flag for called task essentially; that means that, if my task finishes, there is somebody else that I need to transfer control it is not that it is completely over. Because I have been this task switch has happened because of a call. So, when I finish then I have to return back to the caller, I need to get that captured, so this NT flag is basically done. So, the IRET actually modifies this flag on return from the call task. So, caller task calls task and the call task finish finishes its work, and it is comes back to the caller task while it is coming back it will reset the NT flag because there is no more nesting. I called you, so there was a nesting; when I come back, I lose the nesting.

(Refer Slide Time: 32:56)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## Types of Task switching

caller task -> called task

- JMP/CALL instruction to TSS descriptor in GDT
- JMP/CALL instruction to task gate descriptor in GDT/LDT
- interrupt/exception vector refers to task gate descriptor in IDT

called task -> caller task

- called task executes IRET (when NT flag is set)

©

Information Security - RTT& Switch

15/26

NPTEL



So, what are the types of task switching. So, there is a caller task, and there is a called task. So, jump or call instruction to TSS descriptor in GDT is one way of task switching jump or call instruction to task gate descriptor in GDT or LDT this is another way of task switching. There can be a interrupt or exception vector in the IDT in your interrupt descriptor table, I have something called interrupt gate, I have something called the task gate. Now I could have I have a trap gate or a task gate. If it is going to be a task gate then there is a task switching. So, because of these three instances, I could have a task switch between a caller task and called task. From the called task to the caller task, you basically go back using an IRET instruction. So, IRET instruction will take you from the call task back to the caller task NT flag is set right. The nested caller task should be set because NT flag essentially says that somebody has called you, so we go back to this.

(Refer Slide Time: 34:14)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## How task switch happens ?

1. Obtain TSS segment selector
  - get the segment selector from CALL/JMP
  - example -JMP 0X79 :0  
TSS segment selector : 0x78
  - For IRET instruction : from the **previous task link** field of TSS

 Prashanth V — Information Security: RTask Switch 16/26 

So, what are the stages in this task switching. So, I am repeating its quiet I have been covering this again for this is third time I am covering this, but every time I am slowly going into depth, because first teach you the complete depth it actually drives you mad. So, let me slowly go into a little more depth into what happens when we want to do a task switch. First I say call some segment selector colon dummy, because that segment selector can be a task a pointer to a task gate or it can be a pointer to a task state segment descriptor. So, get the segment selector from calls slash jump for example, if I say jump 0x79:0, this is essentially means the TSS segment selector 79 sorry it is not 78, 79. So, for IRET instruction from the previous task link field of TCS you get this. So, if I do an IRET, it is also a return from the task it is also a task switch. So, what happens I returning from a particular task to another task. So, I have a link as I told you in the previous slides there is a previous task link you see on your left hand side. So, I use that previous task link to the go to the previous task state segment and there I reset the NT flag, and I actually get the field of TSS from the previous task link. So, this is how I do a IRET.

(Refer Slide Time: 36:06)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## How task switch happens ?

2. check privilege

- whether the current task is allowed to switch to new task
- Through TSS descriptor :
  - CPL of caller task and RPL of segment selector of called task  $\leq$  DPL of TSS descriptor
- Through Task gate :
  - CPL of caller task and RPL of segment selector of called task  $\leq$  DPL of Task gate being referred

**Note**  
Privilege level **does not inherit** from caller to called task. Each task's separate privilege level is maintained in hardware.

© 2014 IIT Madras — Information Security: ITTask Switch 17/26 NPTEL

So, then after doing this, I go and check the privilege; now I am going to go from something, now, let me say that I am going to go to 0x79. Now what is the 0x79, it can be a TSS descriptor or it can be a task gate then only the task switch going to happen. So, if it is a TSS descriptor, go and see if the current privilege level of the caller task that is the fellow who has executed jump 0x79 and RPL of segment selector. 0x79 will have an request privilege level, the max of that is it less than or equal to the DPL of the TSS descriptor. If it is less than or equal to the DPL of CSS descriptor ok now we can go ahead and do the change.

Now on the other hand, if we say, if you want to go through a task gate again you do the same thing you compare the CPL of the caller task and RPL of the segment selector of called task ah is less than or equal to DPL of task gate being referred. So, what you mean by RPL, I say no jump 0x79 that has an RPL; the 0x79 is a task state segment descriptor of or a task gate of the called task right. So, if that max of RPL comma CPL, it is less than or equal to DPL then I permit; so this is one privilege that I make here. And please note that privilege level does not inherit from caller or called task. Each task separate privilege level is maintained in hardware right I cannot say, so task of privilege level three calls another task, it is not that also will not become privilege three unless it is a confirming code segment right. So, every task has its own privilege level and we

maintain it.

(Refer Slide Time: 38:17)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## How task switch happens ?

3. check other information

- called TSS should be available(Present) in memory
- caller task's TSS, called task's TSS, and all segment descriptors used in the task switch are paged into system memory

Information Security - ITask Switch

NPTEL

Now, I go and check other information, for example, the TSS is it present in the memory, s, that is a present bit for the TSS descriptor, I go and set it is present. Then if it is present, then if the caller task TSS, called task TSS, and all segment descriptor used in the task switch are paged into system memory. Why, because my caller task TSS should be now gone into memory my call task TSS should be loaded from the memory into the registers and all segment descriptors used in the task switch or all paged into system memory.



(Refer Slide Time: 39:02)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## How task switch happens ?

### 4. Setting up flags for caller task

- Busy flag(B) in TSS descriptor
  - Initiated by JMP/IRET : B=0
  - Initiated by CALL : B=1
- NT(Nested task) in EFLAGS
  - Initiated by IRET : NT=0
  - Initiated by CALL/JMP : NT=1

 Prashant V — Information Security: RTTsk Switch 19/26 

After doing this what is happening here the set up flags we have to set up flags for caller task. So, the busy flag B in the DSS descriptor is set to one if it is initiated by call. And if it is initiated by jump or IRET it is set to 0. For example, I am a caller routine, I call somebody that means, still he finishes I cannot execute, so I am busy correct. But if I am just jumping to somebody I am no more busy, because I have gone there and he may come back or not, so I am not busy. But if I call somebody till he finishes I am still busy, so that that is the notion of. If I initiate by busy, we initiate it by jump or IRET then the busy bit 0, but if I initiating it by a call then the busy bit is 1. Similarly nested task if it is initiated by IRET it means a return that means, a nesting is over, NT is 0. If it is initiated by a call slash jump, your nested task can become 1.

(Refer Slide Time: 40:16)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## How task switch happens ?

5. Save the context of caller task

- Obtain base address of caller TSS descriptor from TR
- Copies the task state information into TSS

© 2006 IIT Madras. All rights reserved. NPTEL

Then what we do is we save the context of the caller task. So, we obtain the case address of caller TSS descriptor from TR and copy all the task state information into that TSS because TR will point to the selector with base limit etcetera where the TSS is stored. So, through task register I can find out where the TSS is stored. So, you go and load everything into the TSS descriptor. All caller task data is loaded into its TSS descriptor. Then for the called task, if it is initiated by a call or jump or IRET, I become B equal to 1, because now I start executing, I am the called task, I may be the interrupt service routine or I can be another switch task. But and nested task in EFLAGS, if it is initiated by IRET and jump NT is equal to NT flag value from called task TSS, if it is initiated by call then NT equal to one right.

So, if I initiated by jump then my what is the NT it is the NT flag of my fellow who is jumping into me. If I am initiated by call, my NT is 1. If I am initiated by jump, why should I be the NT flag of my NT flag should be the NT flag of who jumped me, because please note that there might be a call to a routine, and he may jump and that fellow may also jump and somewhere I will go and return my call, followed. So, there can be a jump inside a routine, but somewhere I will go and return it back. So, when a particular caller jump into a particular another caller which is the another called task, the NT of the called task should be equal to the NT of the caller task, because it can be part of some basic

call, but I am doing intermediate jumps, but there is still a call there. So, this is also very important.

(Refer Slide Time: 42:39)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## How task switch happens ?

7. Load the context of called task from TSS

- load (task register) TR
- load the task state information
- load and qualify all the segments

Information Security - IT Task Switch 23/26 NPTEL

Now, I will load the context of the called task from TSS. First I load that that TR to point to the new segment selector; the TR old segment is over, now new segment selector. I load the task and information from that new segment selector into the corresponding registers etcetera, and load and qualify all the segments. Importantly, I qualify all the segments right. Because if I am privilege level three code current privilege my code segment is privilege two and my data segment is privilege zero, then I do not allow loading it right, because if I am three, I can only access privilege three from PAPL to I can only access PL 2 and 3. So, if my code segment has a privilege in the task state segment if my code segment as privilege 1, it will not allow me to load PL 0 as a data segment.



(Refer Slide Time: 43:36)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## How task switch happens ?

Called Task resumes (If no error)  
Otherwise **error/exception**

© 2012 IIT Madras. All rights reserved. NPTEL

After task switch, the called task will start resuming if there is no error so far; otherwise, it will create an error or exception if a depending upon what issue is.

(Refer Slide Time: 43:49)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

## Task switch and task linking

Top Level Task (TSS, NT=0) with Previous Task Link  
Nested Task (TSS, NT=1) with Previous Task Link  
More Deeply Nested Task (TSS, NT=1) with Previous Task Link  
Currently Executing Task (EFLAGS, NT=1) with Task Register

© 2012 IIT Madras. All rights reserved. NPTEL

So, this is how, so there could be a top-level task which was calling another task, which called another task, which called another task. So, that current task register will actually

point to the previous task link, which are essentially points to a previous task link and like this. So, I will have when I do a nested of calls, I actually create several TSS in a list. So, in the operating system terminology, there is a process control block right, there is something called a process control block, and this process control blocks are all part of a list and they can be in ready list or they can be in the wait list or they can be in the suspended list right. So, how do you form such list, this is how the such type list can be formed using hardware. Many of the OS does not use it, but still you can use it.