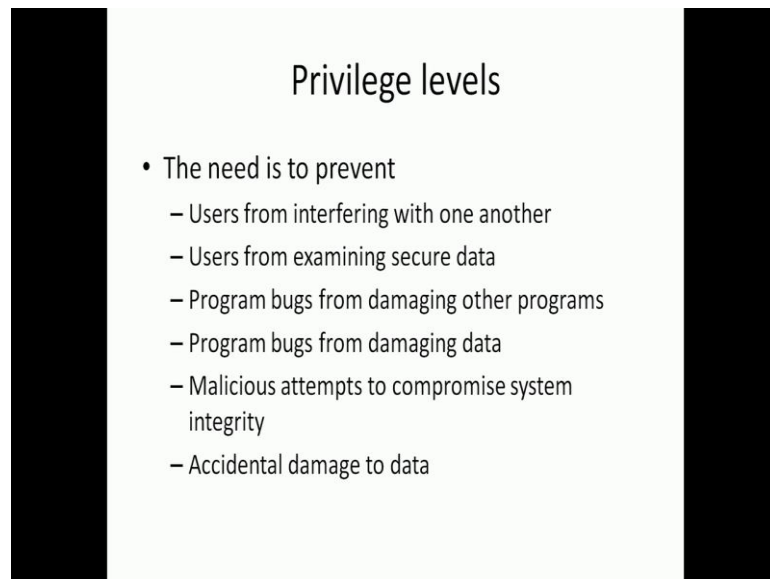


Information Security - II
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 18
Architectural Aid to Secure Systems Engineering
Session - 17: Memory Segmentation Deep Dive - 3 (Privilege Checking)

So, welcome back. We are going to the session-17, which is memory segmentation, specifically from the privilege checking point of view. Now, let us again try to summarize, what is the need to have this privilege level checking. We will just list many of this, because at some point of time, there could be a justification for doing certain things.

(Refer Slide Time: 00:45)



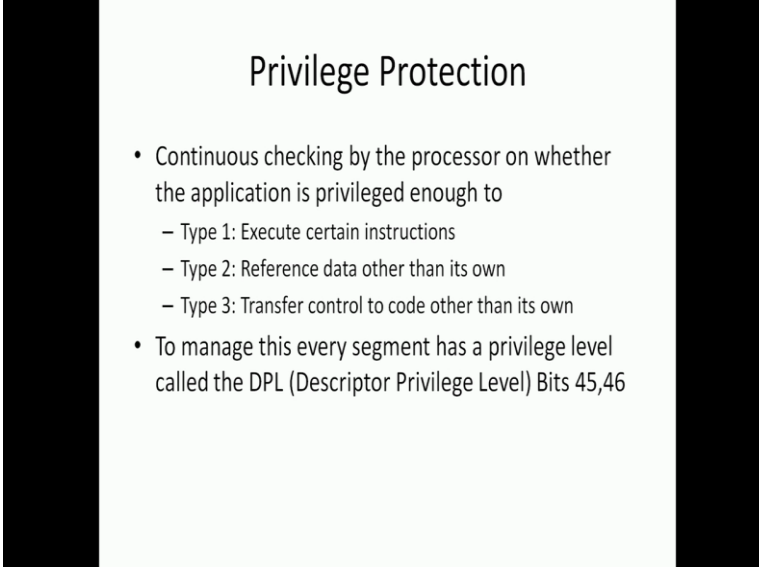
One of the things is, the users it is to prevent several things. Number one, users from interfering with one another or meaning, the process is from interfering with one another; process is from examining some secure data.

More importantly, the program bugs from damaging other program; if one program fails, especially in a multi user operating system, reliability is very, very important. So, how do

we define reliability? If one program fails, the other program should not be affected. It should not know that something has happened like that. What is the relation between this reliability and security? When you look at security, there is confidentiality, integrity and availability; these 3 define security. Now, if my system is not reliable, that is by crashing one program, I can bring the system down, then, it has some say on the availability.

So, in some sense, the isolation should not just be that, I try to go and poke my nose into somebody else, but, by a process killing itself, or process trying to do something; it should not kill the affect the entire system, by bringing down the system. So, there is a relationship between reliability and security from availability perspective. And, the program bugs should not damage the data. So, this bug, how do I define between a bug and interference? Bug is unintentional; even without any intention, without any malicious intent, intention, there should not be error that could happen and the malicious attempts to compromise system integrity, again that is very, very important at this stage. And then, accidental damage to data; so, all these things give us one shot of, one slide of why I want to do this privilege level checking.

(Refer Slide Time: 02:57).



Privilege Protection

- Continuous checking by the processor on whether the application is privileged enough to
 - Type 1: Execute certain instructions
 - Type 2: Reference data other than its own
 - Type 3: Transfer control to code other than its own
- To manage this every segment has a privilege level called the DPL (Descriptor Privilege Level) Bits 45,46

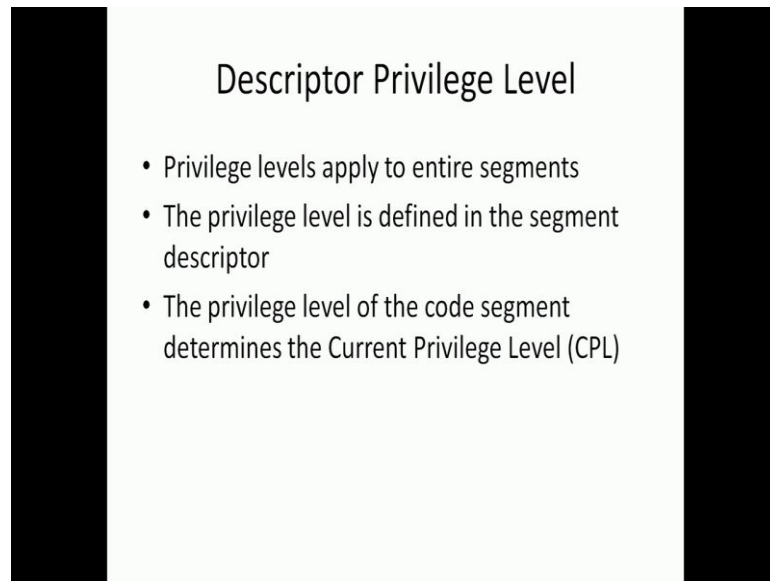
So, one of the most important architectural aid to information security is that, the architecture performs instruction by instruction or micro instruction by micro instruction.

A continuous checking, by the processor on whether the application, any application that is running is privileged enough to one, execute certain instruction; two, reference data other than it is own; three, transfer control to code other than its own. And, it is continuously monitoring; it is not that, there is an occasional monitoring. And I think, this is what we have seen, by all that we covered in these sessions so far, I think we should be in some way you know, satisfy these 3 aspects are continuously being monitored by the architecture. If at all there is going to be vulnerability then, it means that, we have not understood the architecture properly or we are not utilizing some of the features that it is providing to us; and that would be the case in many operating systems.

So, when we actually go through the practical examples, we will, practical, you know, training from, starting tomorrow morning, we will talk about this in much more detail. So, to manage this entire stuff, every descriptor has a privilege level; that is, bits 45 and 46 of your descriptor table. If you go back to the diagram that I have been showing on how a descriptor looks like, the 45 and 46 bits are the descriptor privilege levels. So, I could have 4 privilege level 0, 1, 2 and 3. Now, this means, to just repeat what I told some several sessions before, every memory location will have a privileged level attached to it. So, by this setup, that what we have discussed so far I can go and mark every memory location saying this is the privilege for you. And so the code, which has that privilege, can go and touch it. The code which does not have the privilege cannot go and touch it. And then, this privilege level has another type of variations that we see.

See, here is that, even though I have a segment which is of privilege level 3, since it is not listed in my LDT, or in the GDT I will not be in a position to access that. It is not that, if a segment is at privilege level 3, I can go and access it, if I am a privilege level three code. I may not be in a position to access it, if that segment does not figure in my GDT and LDT. So, this two level of protection is basically given for us, to go and perform this isolation; and that is something that we need to understand and keep in mind.

(Refer Slide Time: 05:55)

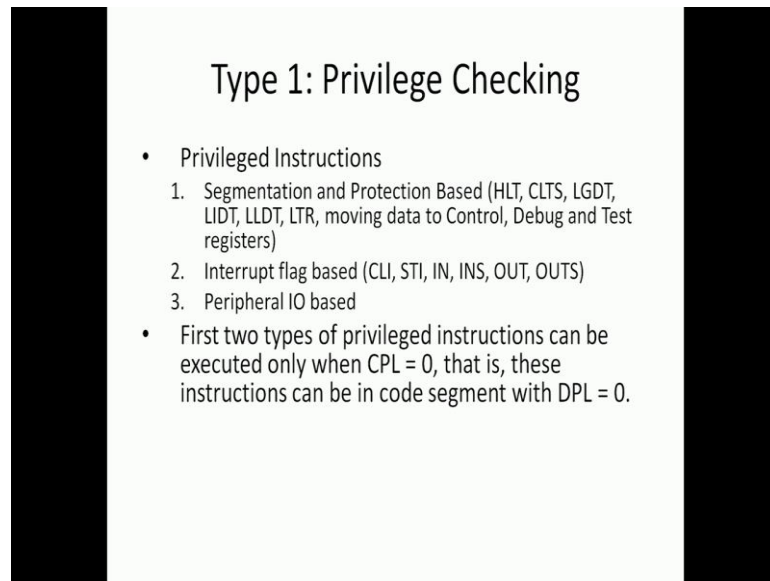


Descriptor Privilege Level

- Privilege levels apply to entire segments
- The privilege level is defined in the segment descriptor
- The privilege level of the code segment determines the Current Privilege Level (CPL)

So, the privilege levels actually apply to entire segments. So, the privilege level is defined in the segment descriptor itself as we see and the privilege level of the code segment determines the current privilege level. This is what we have been explaining so far; I am just recapping it, and I will recap it more number of times, at least 7 times, so that, by the end of the course, all of you know it by default. In the midnight, when you are fast asleep, if I wake you up and say, what is CPL; you will be able to tell. So, the, if I achieved that, that is my goal in this course.

(Refer Slide Time: 06:38)



Type 1: Privilege Checking

- Privileged Instructions
 1. Segmentation and Protection Based (HLT, CLTS, LGDT, LIDT, LLDT, LTR, moving data to Control, Debug and Test registers)
 2. Interrupt flag based (CLI, STI, IN, INS, OUT, OUTS)
 3. Peripheral IO based
- First two types of privileged instructions can be executed only when CPL = 0, that is, these instructions can be in code segment with DPL = 0.

So, now, privileged instructions; I am giving you a summary of these privilege instructions. We have been seeing LLDT, LGDT, LIDT, they are all privileged instructions. But then, there are more privileged instructions; for example, HLT, halt, LTR, load the task register. So, all these and moving data to control debug and test registers; these are all segmentation and protection based instructions, which are privileged. So, only OS, privilege level zero code can do this.

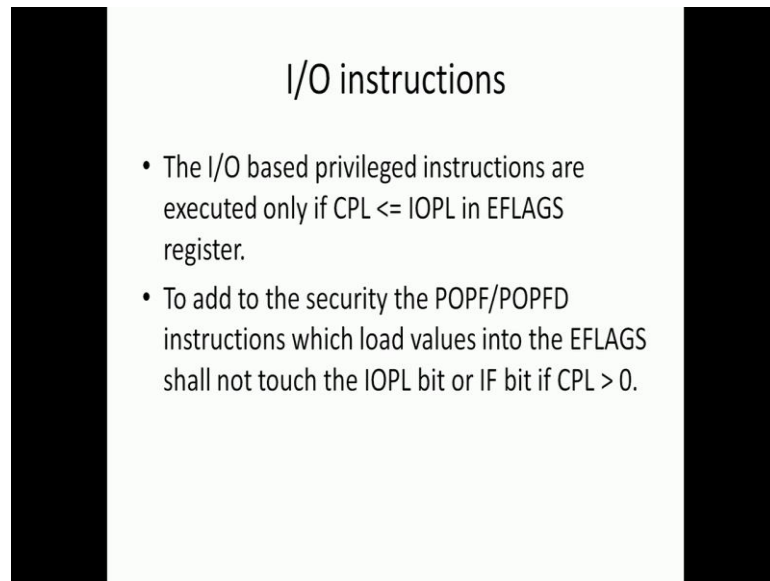
Similarly, setting and clearing an interrupt flag; in in as, out out as, these are all for the interrupt flags. These are all privileged instructions. Peripheral IO based instructions are all privileged. Please note that, lot of things by which some external fellow can enter into your system, all those are privileged. For example, I want do stealth. I have entered the system; if I want to complete an action, at that point of time, if an interrupt comes, I will be in problem, I cannot ensure stealth. So, what I do normally is, I will disable interrupt and then, enable after I finish that stealth. But, I can be; how will I disable interrupt by set i or CLI, right? And, enable it back by doing STI. But that, I can do only if I am at privilege level 0. Please understand this. So, and if I could not make to PL 0, then I get a reasonable amount of protection from lot of external poking, as you see. So, that is why I put this slide; basically, it will tell you these are all the privilege instructions, right. The first two types of privilege instructions that we see here, namely the segmentation and

protection based and interrupt flag, can be executed only when CPL equal to 0; that is, these instructions can be in code segment with DPL 0. What do you mean by it is a current privilege level 0? There should be a code segment with DPL 0, which is that code segment selector, is loaded in to my c s, and I am executing it.

So, this can be executed by privilege level zero code, right? So, if I am not allowing, so please understand, how we can approve security, right? If I am not going to allow creation of descriptors with p l zero, other than the fellows who I have certified, right, then, I can avoid lot of things. You are getting my point? The implication of the last statement is, if I am not going to permit creation of descriptors with p l 0, other than those which I have certified, then all these things are under control; you are following? So, how are these under control, because, they cannot be executed; they can be executed only by my code, and the all those fellows with p l 0, I go and hash it, I sign it, so that nobody can go and change that values. Then, I get very good amount of security. I get very very good amount of security. See, see, these are some things that you know, that we need to keep in mind, and we need to think about it.

Beyond some point, security is art, right? It is no more engineering; so, if you know how to play a violin, you will do much good security then you know, getting a good grade in this course. So, it does not matter. So, it is art; it is more of imagination but, please note that, this is a point here that we need to note. So, creation of descriptors with privilege level 0 is something which is very, very important. And, all the privilege level 0 descriptor are signed; I know what they will do, and they are secure; then many, many issues we can solve.

(Refer Slide Time: 11:04).

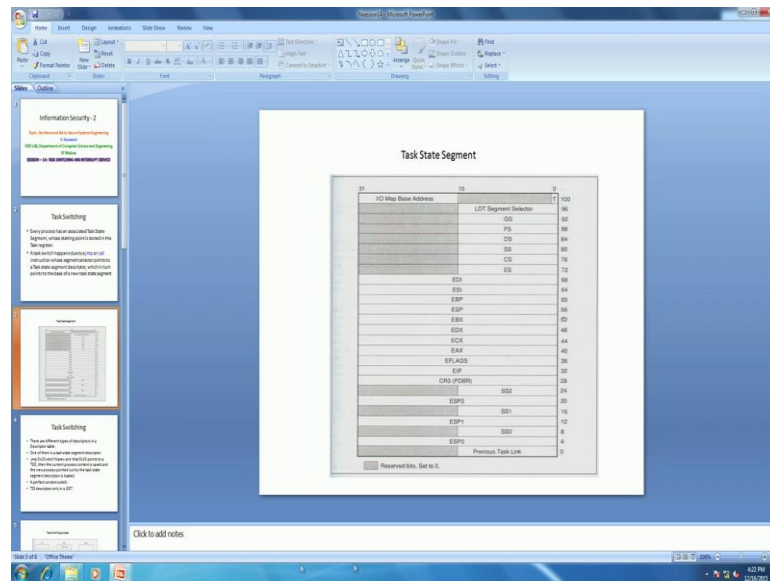


I/O instructions

- The I/O based privileged instructions are executed only if $CPL \leq IOPL$ in EFLAGS register.
- To add to the security the POPF/POPFD instructions which load values into the EFLAGS shall not touch the IOPL bit or IF bit if $CPL > 0$.

Similarly, IO instructions that is the last of those instructions; Please note that, the IO based privileged instructions are executed only if the current privilege level is less than or equal to the IOPL, IO privileged level flags in the EFLAG registers, right? So, the EFLAG actually becomes very, very important. To add to the security, the POPF and POPFD instructions; What is POPF? Popping of the flags that is, for the normal 16 bit flags in the 8086, POPFD is 32 bit extended flags. So, what you can store the flag and then you can pop, we can retrieve the flags. Where do I store the flags and where do I retrieve the flag? When I am doing task switching, right? There is a in the task stage segment, there is one thing for the flags register; do you understand? Do you remember? Did you notice, right? So, let us go back to that, one minute yes.

(Refer Slide Time: 12:15)



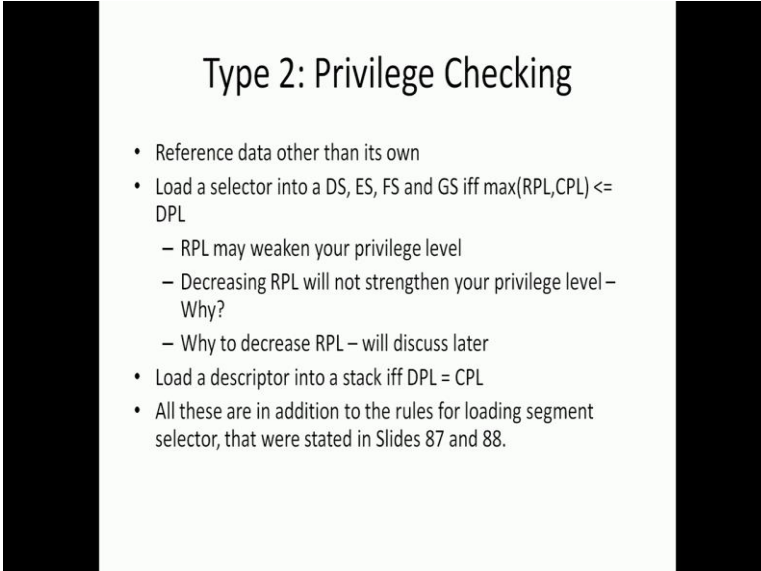
So, now you see, there is an EFLAGS here. So, what I can do is, the e EFLAGS come here. Now, it is in my task stage segment. If somebody is going to go and change that value there and you again go and reload it there, then you can start doing IO p operations. If I make it IOPL is 3, then any code can do that IO operations. So, this POPF and POPFD type of instruction, which is going to write and read from the EFLAGS, which load values into the EFLAGS, shall not touch the IOPL flag bit or the IF bit, the interrupt flag bit, if your CPL is greater than 0. So, unless I am a privileged code, even if I try to write into the EFLAGS register, that IOPL I cannot change; IOPL and the interrupt flag can be changed only if IMPL equal to 0.

So, please note that, PL equal to 0 is very, very, very powerful. And, if you are very careful about entry into PL equal to 0, the X86 architecture can provide you very good amount of (Refer Time: 13:42). Rather than talking about lot of other things, like you know, I will add hyper (Refer Time:13:49), that thing, this, that, that, and you know, like you know, the same (Refer Time: 13:53), as I told you yesterday, you sell the operating system, along with this you sell security patches for it.

Rather than doing that, if we are going to have a very good way of developing this system, where, you know, PL 0 is treated with lot more respect, than with less entry, I

think we will get much, much more from the entire system. Please understand, and all these things have come up in this architecture. Why I am very fascinated personally by this is that, all these things have come up in this architecture much, much before people realized about information security. These are existing, these are something which were available in Turley's book, some 20 years before, 25 years before, at least to my knowledge and he has written that book much earlier. So, this is very, very important. And, these have come up with lot of thought process; but unfortunately, the software era has neglected these things and that is why we are here.

(Refer Slide Time: 12:55)



Type 2: Privilege Checking

- Reference data other than its own
- Load a selector into a DS, ES, FS and GS iff $\max(\text{RPL}, \text{CPL}) \leq \text{DPL}$
 - RPL may weaken your privilege level
 - Decreasing RPL will not strengthen your privilege level – Why?
 - Why to decrease RPL – will discuss later
- Load a descriptor into a stack iff $\text{DPL} = \text{CPL}$
- All these are in addition to the rules for loading segment selector, that were stated in Slides 87 and 88.

Type 2 privilege checking is about referencing data, other than its own. So, please note that, load a selector into a DS, ES, FS and GS if and only if the maximum of your RPL comma CPL is less than or equal to DPL. Now, what is RPL? Request Privilege Level. Where it will be? In the selector, when I am loading the selector move DS comma something, I am loading the selector. So, when I am loading the selector, the last 3 bits, one of the bit, the third bit will be whether GDT or LDT, the remaining two will have a privilege level, and that privileged level will be what, privilege level will be the RPL. So, what is that RPL? That is where you are using it here.

Now, please note this very interesting thing. I will load a selector into a DS, ES, FS and GS, if and only if, the maximum of the RPL comma your current privilege level, should be less than or equal to DPL. Currently, what we have seen so far in the previous things, if your CPL is less than or equal to your DPL, I can load. So, if my current privilege level is 2, I can load data 3, but by introduction of the RPL what will happen? It will weaken my privilege level. I am privilege level 2; there is a data, which is also a data segment, which is also privilege level 2. I can go and access it by whatever we have discussed so far but, if I make a request RPL with that RPL as 3, then what will happen? Max of RPL comma CPL will become 3; your DPL is 2; I will be stopped from accessing it.

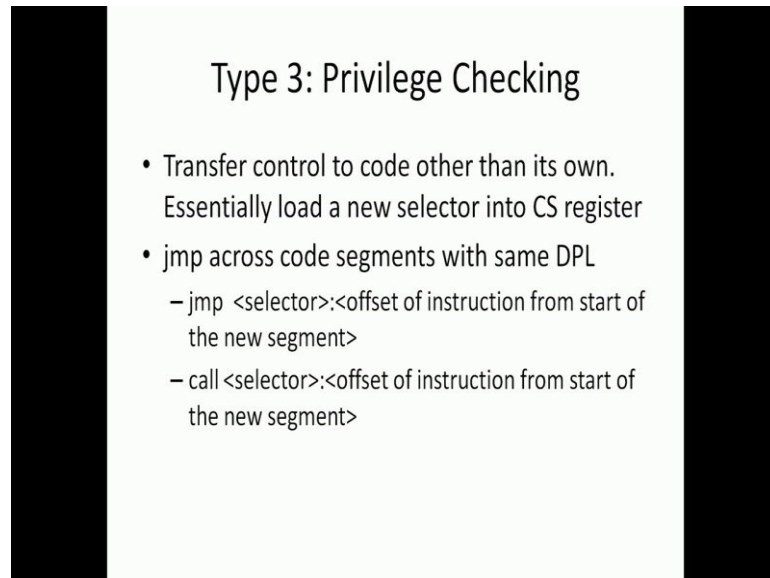
So, RPL can actually weaken your privilege level; but if I decrease my RPL, it will not strengthen my privilege level, because it is max. So, I am privilege level 2, I want to go and look at privilege level zero code, I cannot put my RPL zero, and say try and go; because it will be max of zero comma 2 will be 2; so, then, why this confusion? Why should I decrease this RPL, right? I leave it for today's night dream. So, dream about it, but tomorrow, I can tell you some good answers for this. We will discuss about that later, but please note that, this is one check that comes here. And for stack, it is always, I will load as descriptor into a stack, only if my DPL is equal to CPL; it is not DPL to be greater than or equal to CPL.

If I am, current privilege level is 2; I will load a stack segment which is exactly privilege level 2. I could have a data segment, which is privilege level 3, but stack, I will use it exactly with privilege level 2, because this stack is going to be used by many; I could have interrupts; I could have different things; the stack is vulnerable to be shared. And, if some other privilege level code gets access to my stack in some form or other, then, I will be in a bigger soup.

So, with respect to stack, which is the way by which many many information leaks from one process to another, through interrupt service routine or you know, not erasing the stack etcetera, we now clearly see that, that I will load a stack, if and only if, the DPL should be equal to CPL; it is not that greater also. So, I have a very much, I have a bigger privilege level checking for this stack. All these are, in addition to the rules for loading

segment selectors that were stated, I am sorry, I do not have the slide number; these were stated in the previous session. We stored lot of Type 1, Type 2, Type 3 checking, right? These are all some additional checks that is happening in addition to what we have done in the past, OK.

(Refer Slide Time: 19:26)



The slide is titled "Type 3: Privilege Checking" and is presented on a light green background with black vertical bars on the left and right sides. It contains the following text:

Type 3: Privilege Checking

- Transfer control to code other than its own. Essentially load a new selector into CS register
- jmp across code segments with same DPL
 - jmp <selector>:<offset of instruction from start of the new segment>
 - call <selector>:<offset of instruction from start of the new segment>

Now, the next thing is, this is all about data; now, the next thing is about privilege checking. Now, I can transfer control to code other than it is own; say I jump to my something within my code segment that is called a short jump. Short jump means, it is not that you know, the way we, is not that I just jump within some reasonable offset, that is one notion that exists. Short jump means next instruction; next to next instruction, it is not like that. Short jump is within my own code segment I jump, it is called a short jump. My code segment can be even 4 GB in size; but it is a short jump.

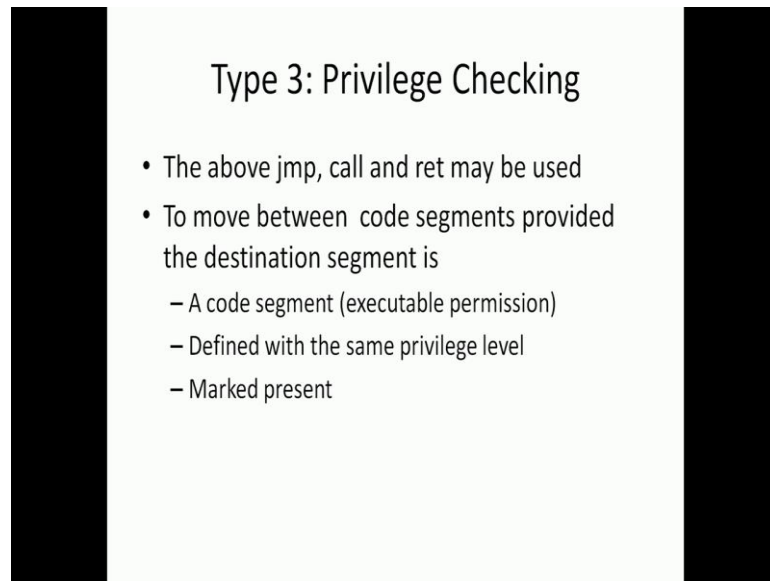
What is a long jump? I go and do a jump into somebody else's segment that is not my own, right. I go and jump into some other, somebody else's, jump; I go and call some routine in somebody else's segment, or some other segment; it need not be somebody else's, but it can be some other segment, then it becomes a long call or a long jump. That is how I am defining it, because it is an utter confusion when you go through many of these books. So, I will define like this; if you all follow it for at least this course. I hope

you are not going to write an examination based on this course, other than what I am going to set. So, I will stick to this. So, now, when I jump across code segments with same DPL, I am DPL 3, you are also DPL 3, I jump from myself to my code segment to your code segment then it is very easy.

So, I put jump, the selector of that code segment; you have a code segment, you will have a selector that code segment, selector, colon, whatever offset of the instruction from the start; I may not always execute from the top; I will go and start from the middle of your segment. So, I can do that. Similarly, because your code segment will be a collection of library functions; one such example you will see is in system calls. When you, when you look at the, how the Linux is organized. So, you may have several functions put into one code segment, right. So, depending on which function I want to execute, I will use the same code segment, but I will go into different offsets there. Do you understand this?

So, similarly, call; I call into a particular code segment, and within, I have an offset of instruction from start of that new segment. So, when I am going jump between code segments of the same privilege levels, I just do this; there is nothing no [FL] in this. I can do this; but note that, that code segment that whatever selector that code segment should be accessible to me, it should be either in the LDT or in the GDT. So, the operating system can say that, whatever I can access, those are the things they will put in the LDT and make it available in the GDT; some things which I do not access, they will not put it in the GDT or they will put in GDT with other privilege levels and the vice versa here. So, that way this becomes a; I can have a good check on where I can go.

(Refer Slide Time: 26:01)



Type 3: Privilege Checking

- The above jmp, call and ret may be used
- To move between code segments provided the destination segment is
 - A code segment (executable permission)
 - Defined with the same privilege level
 - Marked present

The above jump, call and return may be used, to move between code segments, provided, the destination segment is a code segment; and it is defined with the same privilege level, and it is marked present. So, I can use jump; I can use call; I can use return back, after call. All these things I can do, if I am, when I am executing, my privilege level is equal to the target jumping privilege level, and that is also a code segment with executable privilege permission; in the type, you know, you saw, right? Execute read only, execute etcetera, and it should have the same privilege level, and it should be marked present.

So, why I am making all this statement is, tomorrow when you will be asked to write a segment descriptor; then only you will understand the complications there and you will also understand many intricate details about this. And, these are all very, very important there.