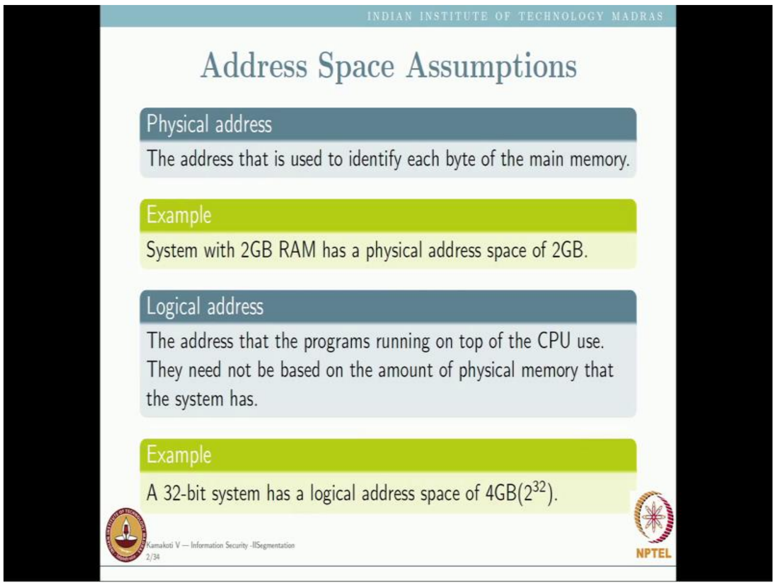
Information Security - II
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 20
Segmentation Recap - Week 4

(Refer Slide Time: 00:13)



INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Address Space Assumptions

Physical address
The address that is used to identify each byte of the main memory.

Example
System with 2GB RAM has a physical address space of 2GB.

Logical address
The address that the programs running on top of the CPU use. They need not be based on the amount of physical memory that the system has.

Example
A 32-bit system has a logical address space of 4GB(2^{32}).

Kamakoti V — Information Security -IISegmentation
2/34

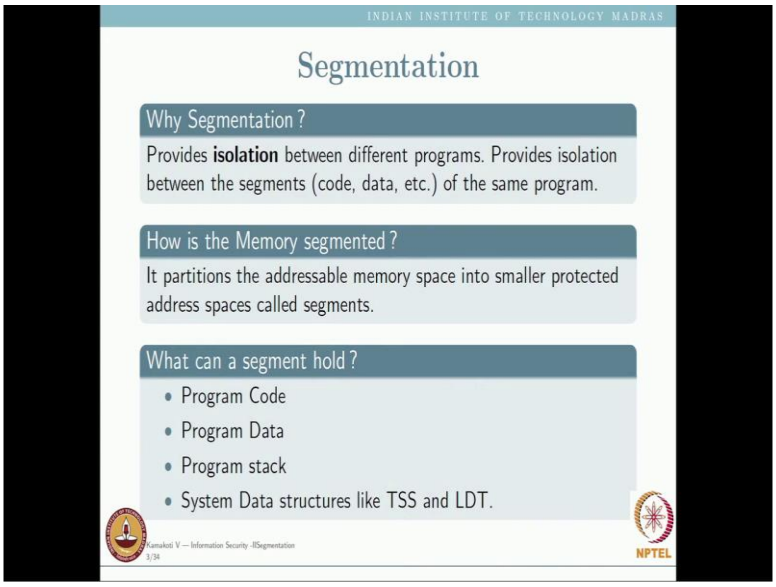
NPTEL

Let us very quickly, in next 10 to 15 minutes, basically get to know what were the things that happened yesterday? When we talk of segmentation, now there are two addresses that we are talking of; one address is the actual address in the ram, another address is what the compiler generates. The compiler, when it compiles the code assumes that your code starts at 0, your data starts at 0, your code segment starts at 0, your data segments starts at 0 and from that point onwards, it will assemble your data and the code. So, all the addresses generated by your compiler that is part of your assembly code which is compiled, will be with respect to a 0. So, that is what we saw in yesterday's class in the previous sessions.

But now what happens further is that every set logical address; that is a logical address because you assume that it starts from a logical 0 and everything with every offset that the compiler generates is with respect to that logical 0. Now that value gets added to a segment base and you get the actual physical RAM address. So, there are two addresses we have understood yesterday, one is the physical address, another is the logical address. So, the physical address is used to identify each byte of the main memory, the x86 architecture that we are talking of; is a byte addressable architecture and so every memory location, every byte can be addressed individually.

So, since we have 32 bit you can address 4 Giga bytes. For example, a system with 2 GB RAM will have a physical address of 2 GB. The logical address is the address that the programs running on top of the CPU use, they need not be based on the amount of physical memory that the system has. A 32 bit system has a logical address space of 2 power 32, so if we are not using virtual memory then what will happen, the logical addresses generated by your system plus the segment base should be within the limits of the physical address because there is no virtual memory given to you.

(Refer Slide Time: 02:52)



INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Segmentation

Why Segmentation ?

Provides **isolation** between different programs. Provides isolation between the segments (code, data, etc.) of the same program.

How is the Memory segmented ?

It partitions the addressable memory space into smaller protected address spaces called segments.

What can a segment hold ?

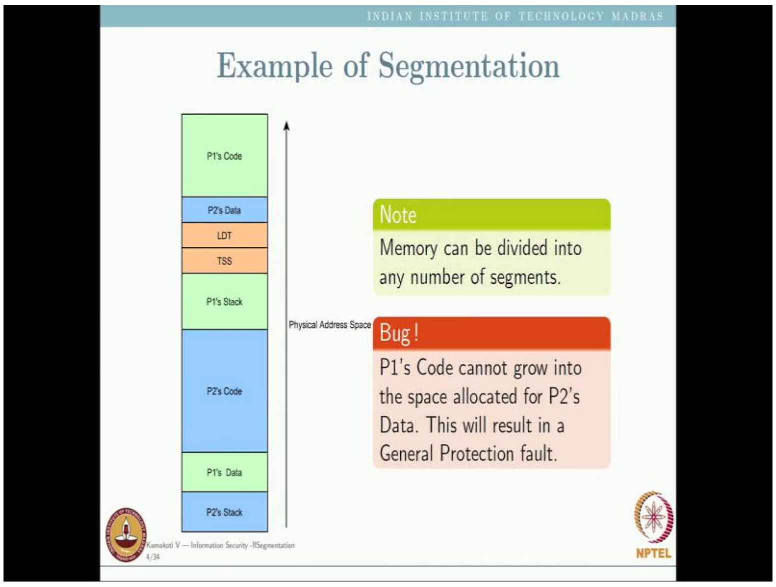
- Program Code
- Program Data
- Program stack
- System Data structures like TSS and LDT.

Kamakoti V — Information Security -IISegmentation
3/34

NPTEL

Now, we know that segmentation provides isolation between different programs. It also provides isolation between the segments, which talked about it in great detail yesterday. So, what is memory segmentation, it partitions the addressable memory space into smaller, protect that address spaces called segments. We have also introduced lot amount of protection across these spaces. So, a segment as we saw yesterday can hold code, it can have hold data, can hold stack and there are also some system data structures like TSS and LDT. The task state segment and local descriptor tables and your GDT, so there is a segment can hold all these tasks.

(Refer Slide Time: 03:46)



INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Example of Segmentation

Note

Memory can be divided into any number of segments.

Bug!

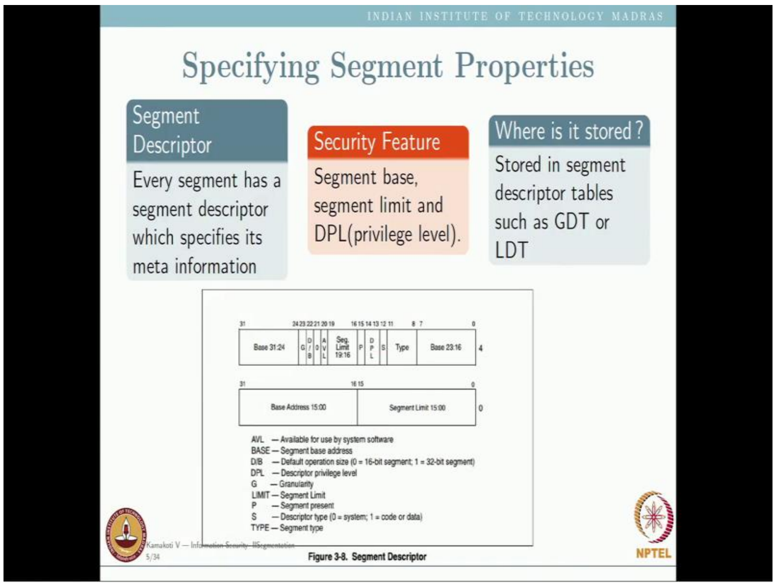
P1's Code cannot grow into the space allocated for P2's Data. This will result in a General Protection fault.

Kamakoti V — Information Security -ISegmentation
4/34

NPTEL

Now, this is how a typical memory we look when it is in run. So, there what you see here is there are two process is running, one is P 1 another is P 2. You see P 1 scored P 1's data, P 1 stack, P 2's data, P 2's stack and P 2's code and then you will have some LDT's and TSS. So, this is typically how a segment will look and very interestingly the segmentation at x86 does not allow anything, any code or data or stack to over grow its limit. So, whenever there is a over growing like for example, your P 1's code cannot grow into the space allocated for P 2's data. For example here, if that happens then automatically the architecture will catch you and it says you are trying to do something illegal. So, this is the architectural aid that we talked of yesterday and then coming back to this.

(Refer Slide Time: 04:48)



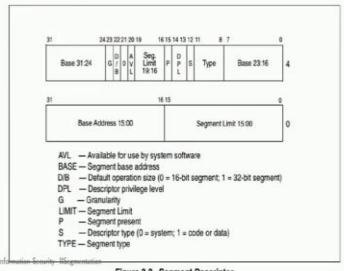
INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Specifying Segment Properties

Segment Descriptor
Every segment has a segment descriptor which specifies its meta information

Security Feature
Segment base, segment limit and DPL(privilege level).

Where is it stored?
Stored in segment descriptor tables such as GDT or LDT



AVL — Available for use by system software
BASE — Segment base address
D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
DPL — Descriptor privilege level
G — Granularity
LIMIT — Segment Limit
P — Segment present
S — Descriptor type (0 = system; 1 = code or data)
TYPE — Segment type

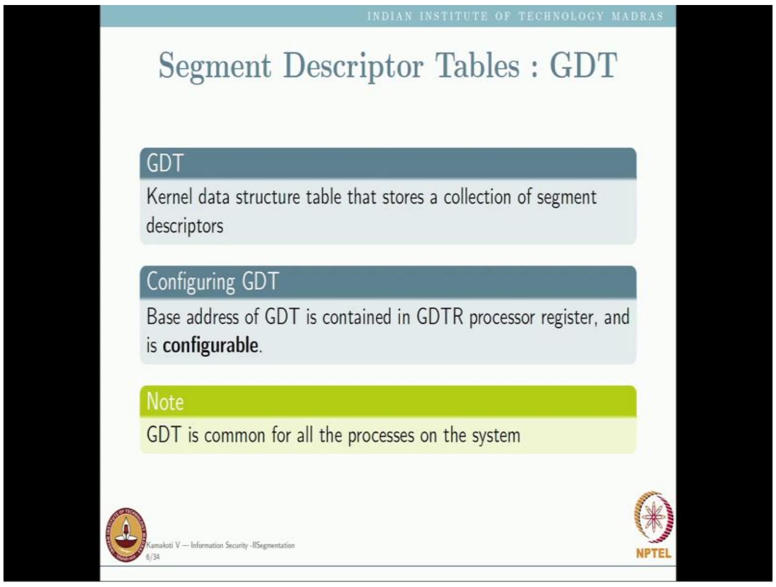
Figure 3-8. Segment Descriptor

Kamakoti V — Information Security -IISegmentation
5/34

NPTEL

We had a notion of a segment descriptor, we had a notion that this segment descriptor introduces some amount of you know privileges and then these descriptors are stored in the local descriptor and the global descriptor tables. This also we covered yesterday and this is whatever you see down here is actually the diagram of a segment descriptor, it describes the content of that segment descriptor so yesterday we also covered this.

(Refer Slide Time: 05:36)



INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Segment Descriptor Tables : GDT

GDT
Kernel data structure table that stores a collection of segment descriptors

Configuring GDT
Base address of GDT is contained in GDTR processor register, and is **configurable**.

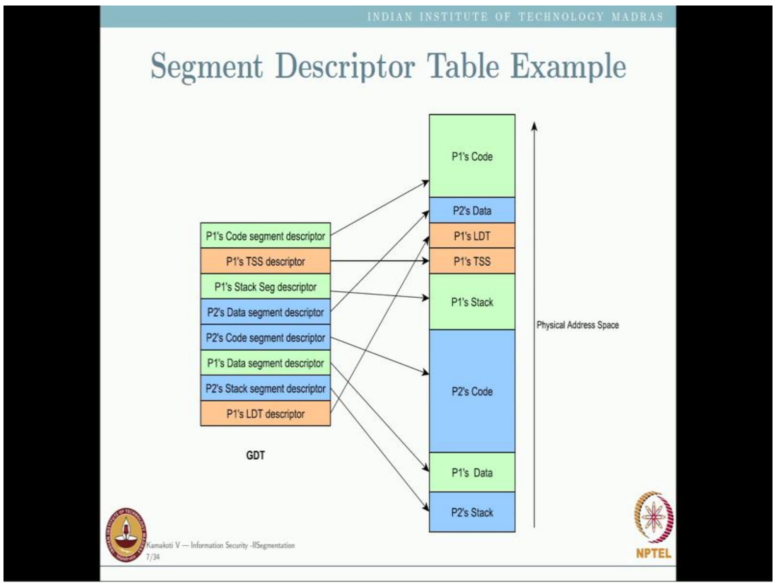
Note
GDT is common for all the processes on the system

Kamakoti V — Information Security -IISegmentation
6/34

NPTEL

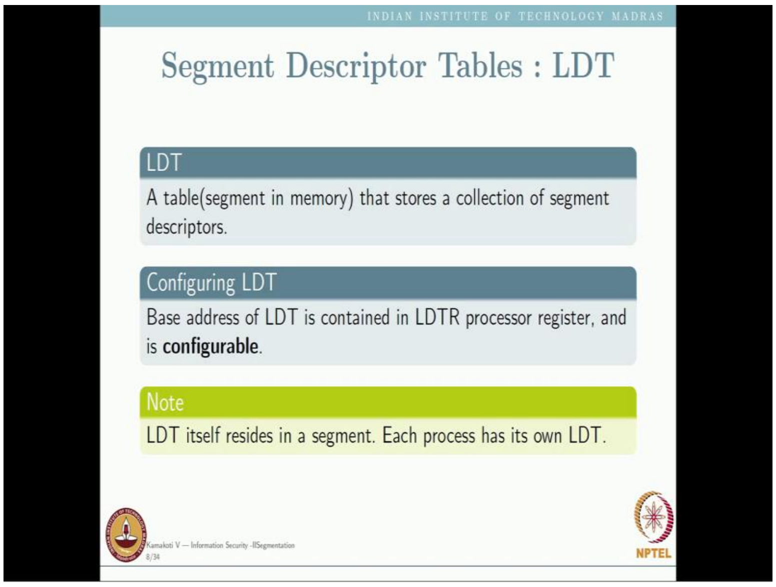
These descriptors are stored in descriptor tables, there are two segment descriptor tables; one is called the global descriptor table, another is called the local descriptor table. The global descriptor table typically is used to hold the kernel data structures and you can load the global descriptor table at any point by using the global descriptor table register. So, in that sense the global descriptor table is configurable, in the sense it can be loaded at any point in the memory. Now, the global descriptor table is called global because it is common to all the processes and we also cover this in great detail in last classes, last set of sessions.

(Refer Slide Time: 06:23)


INDIAN INSTITUTE OF TECHNOLOGY MADRAS
Segment Descriptor Table Example
P1's Code segment descriptor
P1's TSS descriptor
P1's Stack Seg descriptor
P2's Data segment descriptor
P2's Code segment descriptor
P1's Data segment descriptor
P2's Stack segment descriptor
P1's LDT descriptor
GDT
P1's Code
P2's Data
P1's LDT
P1's TSS
P1's Stack
P2's Code
P1's Data
P2's Stack
Physical Address Space
Kamakoti V — Information Security - ISegmentation
7/34
NPTEL

Now, this is typically how a global descriptor table looks. So, a global descriptor table or any descriptor table essentially has code segment descriptors, stack segment descriptors for different processes and it could have task state segment descriptor. We also understood what task state segment essentially means it can also have a descriptor to the LDT of a process. So, many things can be there inside a GDT but typically GDT will have the code stack, the data plus TSS plus LDT's, but that code stack and data would be for only the kernel processes and for all the local processes, all the stack data and code for the local processes will be stored in the corresponding LDT's.

(Refer Slide Time: 07:28)


INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Segment Descriptor Tables : LDT

LDT

A table(segment in memory) that stores a collection of segment descriptors.

Configuring LDT

Base address of LDT is contained in LDTR processor register, and is **configurable**.

Note

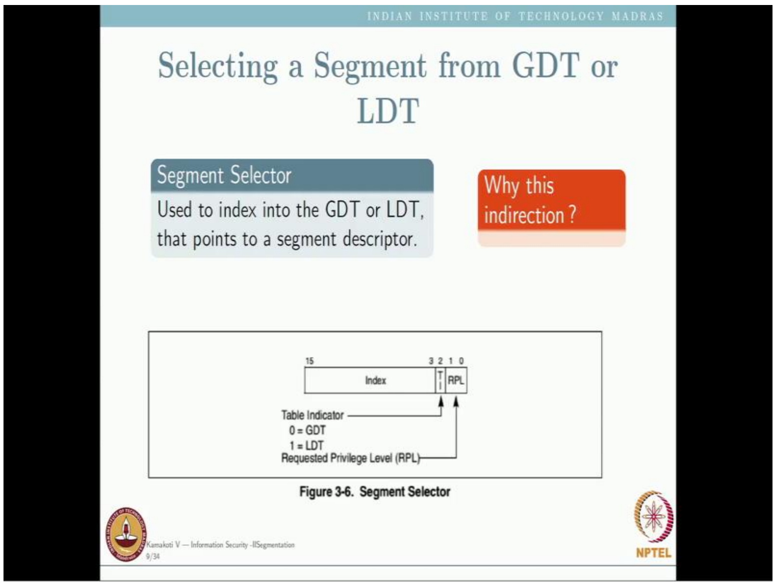
LDT itself resides in a segment. Each process has its own LDT.

Kamakoti V — Information Security -IISegmentation
8/34

NPTEL

Similarly, the LDT is a per process stuff and so it is given, the start of the LDT is given by LDTR and the LDT under GDT themselves will reside in segments and that will be inside the operating system memory, those segments would be described for in the inside the operating system memory.

(Refer Slide Time: 07:55)


INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Selecting a Segment from GDT or LDT

Segment Selector

Used to index into the GDT or LDT, that points to a segment descriptor.

Why this indirection ?

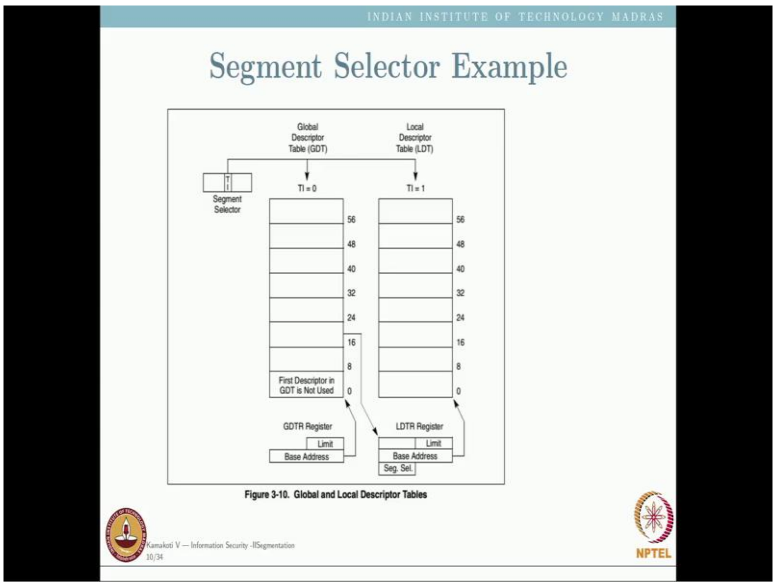
Figure 3-6. Segment Selector

Kamakoti V — Information Security -IISegmentation
9/34

NPTEL

Then we also yesterday discussed when a process is trying to access a segment, when it is trying to access segment it has to load the corresponding segment register. So, whether that offset in the segment register, is it pointing to the GDT or LDT and we did see that there is 1 bit, there are 3 bits that are wasted. Instead of putting 0's, we use for some constructive purpose and there is a bit, the second bit if it is 0, then it is pointing to a G D T and if it is 1, it is pointing to a LDT, so this also we saw yesterday.

(Refer Slide Time: 08:42)


INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Segment Selector Example

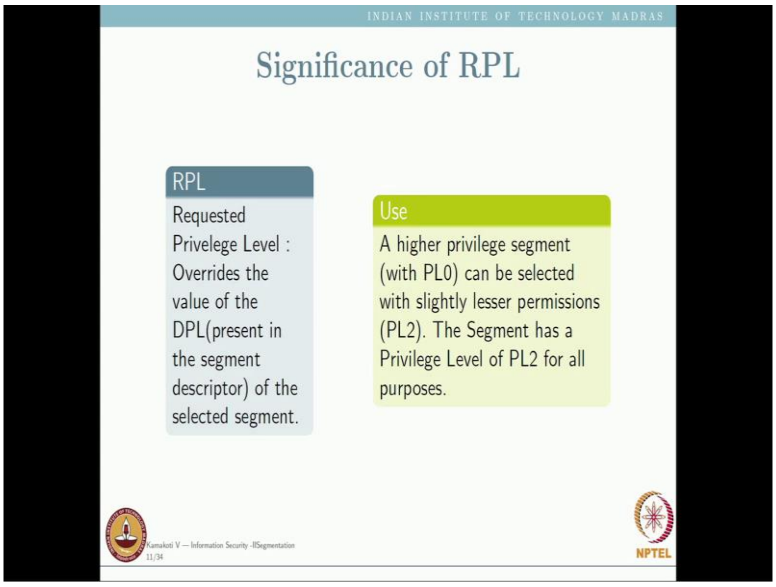
Figure 3-10. Global and Local Descriptor Tables

Kamakoti V — Information Security -IISegmentation
10/34

NPTEL

So, this is just a diagrammatic; you know illustration of what I talked in the previous slide. So, if the T 1 is 0, then the segment is selected from the global descriptor table, if that T 1 is 1, then it is selected from the local descriptor table.

(Refer Slide Time: 09:09)



INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Significance of RPL

RPL

Requested Privelege Level : Overrides the value of the DPL(present in the segment descriptor) of the selected segment.

Use

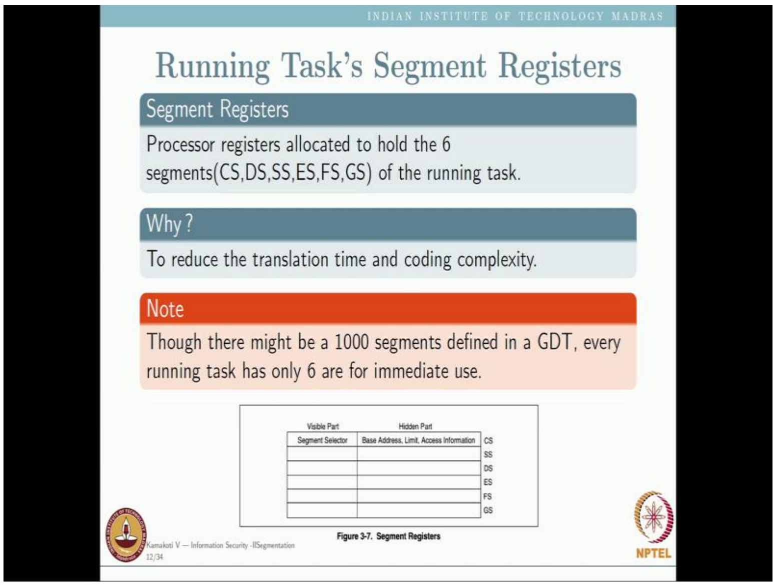
A higher privilege segment (with PL0) can be selected with slightly lesser permissions (PL2). The Segment has a Privilege Level of PL2 for all purposes.

Kamakoti V — Information Security -IISegmentation
11/34

NPTEL

Then we also saw that there were 3 bits in this global and local descriptor, when we try to address these tables because all the segment descriptors start at 8 byte boundary, so we had this 3 bits that are actually wasted and in that 3 bits, 1 bit we use to indicate whether it is GDT or LDT, the remaining 2 bits we used it for what we call as a requester privilege level. By using this requester privilege level, I can weaken my privilege so it actually over writes your DPL, you take the maximum of what it is given in the DPL and the RPL and this can be used to actually execute some higher privilege; that means numerically lower privilege code with a numerically higher privilege level. So, there is the RPL which is also path, which is used when the code is requesting to load the selector.

(Refer Slide Time: 10:25)



INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Running Task's Segment Registers

Segment Registers

Processor registers allocated to hold the 6 segments(CS,DS,SS,ES,FS,GS) of the running task.

Why?

To reduce the translation time and coding complexity.

Note

Though there might be a 1000 segments defined in a GDT, every running task has only 6 are for immediate use.

Visible Part	Hidden Part	
Segment Selector	Base Address, Limit, Access Information	CS
		SS
		DS
		ES
		FS
		GS

Figure 3-7. Segment Registers

Kamakoti V — Information Security -IISegmentation
12/34

NPTEL

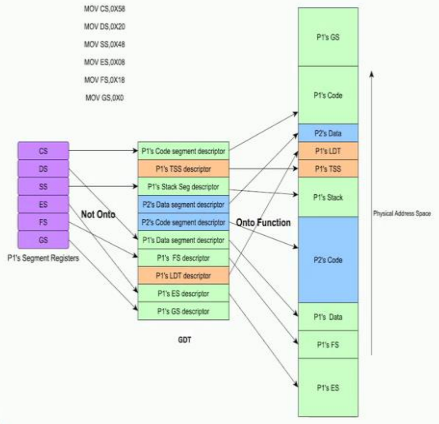
Then we also talked about the segment registers, the segment register actually points to an offset within the global descriptor table and the local descriptor table. Along with the segment register, there is also a hidden part which basically stores the base address limit and access information of the current segment that particular segment register is pointing to and this is actually important for performance reasons.

(Refer Slide Time: 10:55)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Segment Register Example

```
MOV CS,0X58
MOV DS,0X20
MOV SS,0X48
MOV ES,0X08
MOV FS,0X18
MOV GS,0X0
```



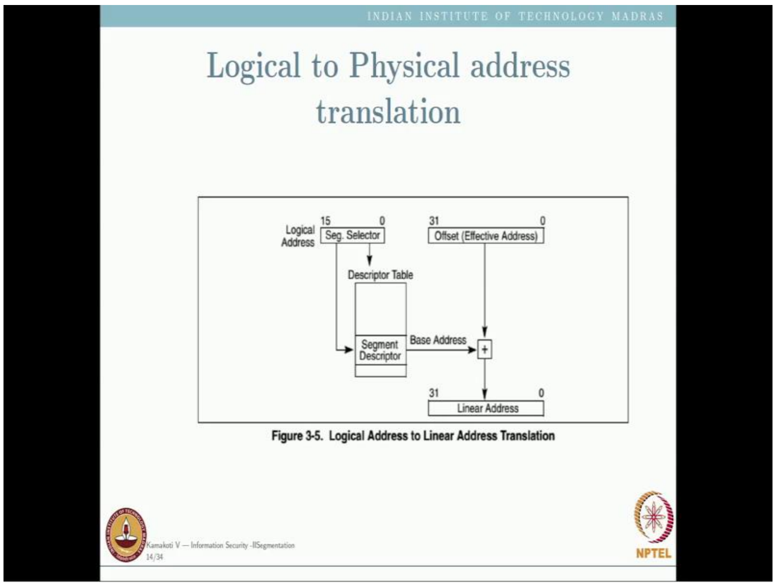
Kamakoti V — Information Security -IISegmentation
13/34

NPTEL

So, this is it. For example, let us say this is the GDT, so the GDT grows from a bottom. The bottom 0, actually 0 cannot store anything. So, please note the see the MOV's statements on the left hand side top, MOV GS comma 0. So, I want GS to store actually a null descriptor. Then MOV FS comma 0 x 18, so 0 x 18 is 16 plus 824, which is the third segment, 0th is P 1 GS, first is P 1 CS, second is P 1; LDT and third is P 1's FS. So, 0, 1, 2, 3 please note that FS, now actually takes that fourth descriptor form bottom.

Similarly MOV ES comma 0 x, 0 8; 0 x 0, 8 essentially stands for the second, that is the first segment that is 0 and the first. So, you now see that ES is taking the P 1's descriptor that is the first descriptor from the bottom. Similarly you can see that TSS is taking the 64 plus 8; 72, 72 is ninth descriptor, 0, 1, 2, 3, 4, 5, 6, 7, 9 right. So, SS is basically taking this 64 plus 8; 72, 72 is the ninth descriptor, so 0, 1, 2, 3, 4, 5, 6, 7. I think there is some small error in the slide, so SS is taking the seventh descriptor. So, what should be the seventh descriptor, 7 should be 7, 8; 56 right because each is 8 bytes so the seventh descriptor should be 56 and what is 56? 38, so it should be 38; 38 is 3 into 16 is 48 plus 856, so MOV SS comma 0 x 38. Similarly we go on and so CS should be 48, so this is how you load the segment register and depending on the value that you are loading on the segment register the corresponding a segment in GDT will be mapped on to.

(Refer Slide Time: 13:57)



INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Logical to Physical address translation

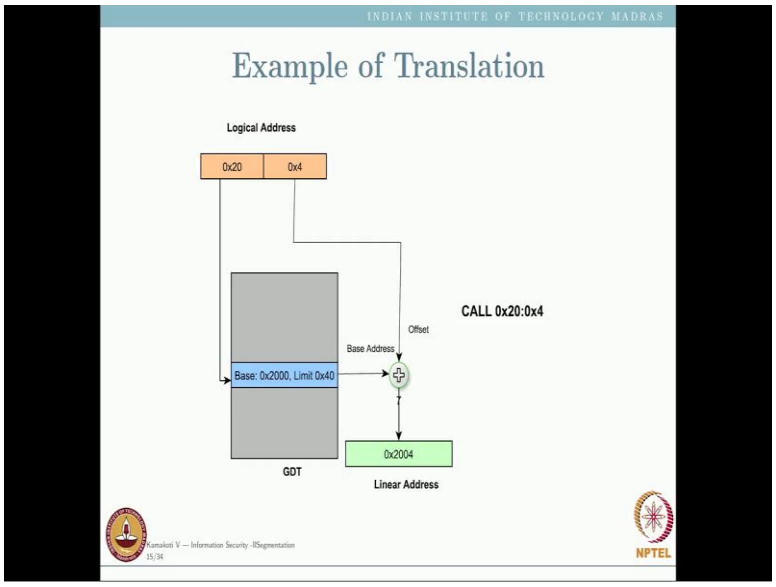
Figure 3-5. Logical Address to Linear Address Translation

Kamakoti V — Information Security -IISegmentation
14/34

NPTEL

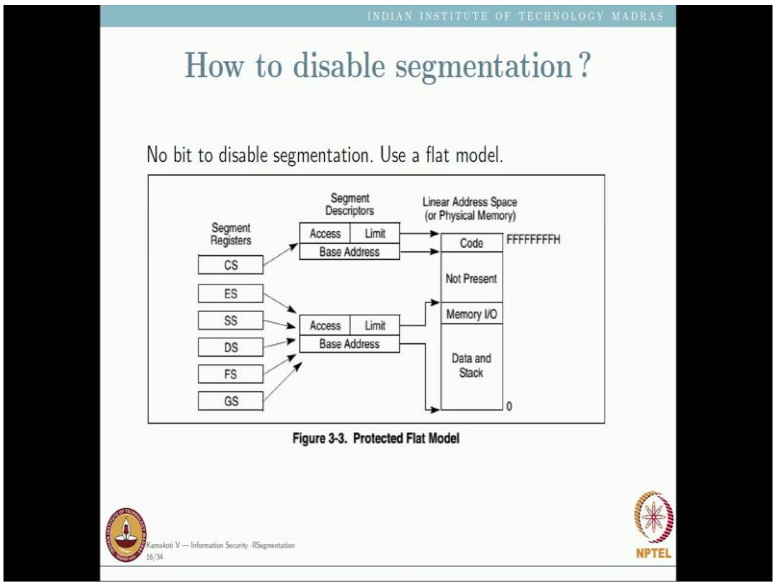
This is the logical to physical address translation in the absence of virtual memory. So, you have a segment selector that segment selector will point to a segment descriptor that will give you a base address to which add the effective address that is generated by the compiler to actually get the address in the RAM. So, this is how basically segmentation works.

(Refer Slide Time: 14:20)



If your compile program says DS colon 4 and DS is pointing to 0 x 20, so the 0 x 20 descriptor is nothing, but will have base 0 x 2000 and 0 x 4 is given here, so the linear address would be 0 x 2004. So, this is how you work.

(Refer Slide Time: 14:51)


INDIAN INSTITUTE OF TECHNOLOGY MADRAS

How to disable segmentation?

No bit to disable segmentation. Use a flat model.

Segment Registers
CS
ES
SS
DS
FS
GS
Segment Descriptors
Access
Limit
Base Address
Linear Address Space (or Physical Memory)
Code
FFFFFFFFH
Not Present
Memory I/O
Data and Stack
0

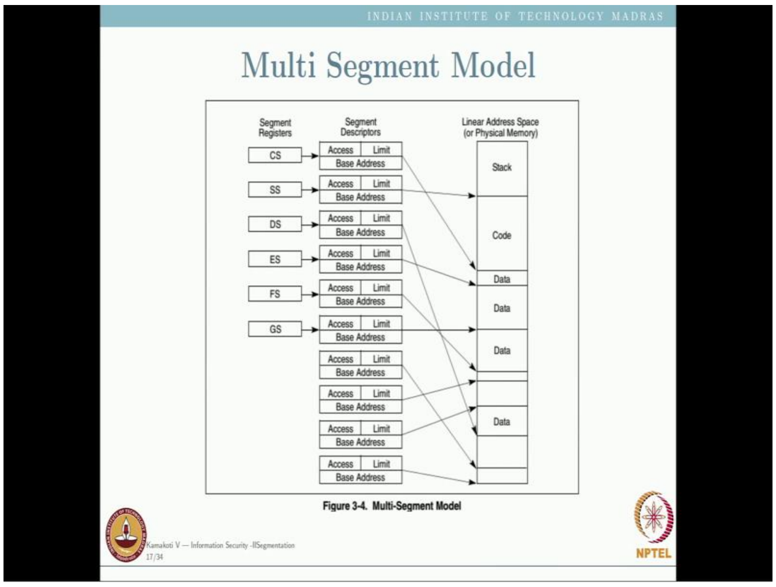
Figure 3-3. Protected Flat Model

Kamakoti V — Information Security - IISegmentation
16/34

NPTEL

We cannot disable segmentation in the protected mode, but what we can do like many operating system does, is that you just say that the entire 4 GB is one segment. You map the code, stack everything to one single segment so that there is no segmentation, so it is equivalent to saying there is no segmentation, but for protection and security purpose it is not advisable to do for all the things that we have seen so far, a best preference would be it have a multi segment model.

(Refer Slide Time: 15:14)


INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Multi Segment Model

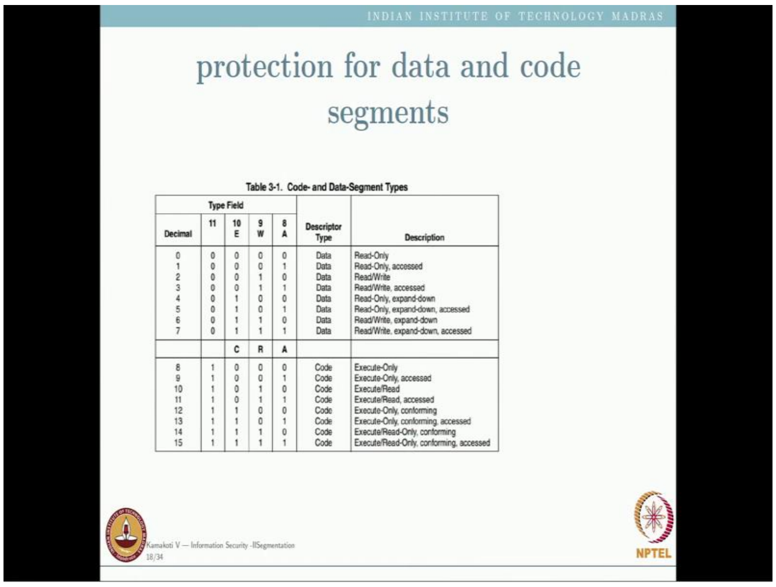
Figure 3-4. Multi-Segment Model

Kamakoti V — Information Security -IISegmentation
17/34

So, here you see here there are many segments that are loaded and some of the active segments are selected using the segment registers and they are currently active and when they become inactive somebody else becomes active, the corresponding segment register will be updated with that offset. So, currently if you see the first 1, 2, 3, 4, 5, 6 are active, so what you see on the left most side is the segment registers, what you see on the middle is the segment descriptor and what you see on the right hand is your physical memory.

Now all the segment registers are mapped on to one segment here segment descriptor here and so the first six segment descriptors are currently active. Suppose the third descriptor, which DS is pointing to becomes inactive and you want some other to be the DS, you just make that DS point to this right. So, I could have multiple amounts of segment descriptors in my local descriptor or the global descriptor table, but some of them will actually be mapped, some which are really active will be mapped on to the corresponding segment registers and these are the types of user level descriptors.

(Refer Slide Time: 16:44)



INDIAN INSTITUTE OF TECHNOLOGY MADRAS

protection for data and code segments

Table 3-1. Code- and Data-Segment Types

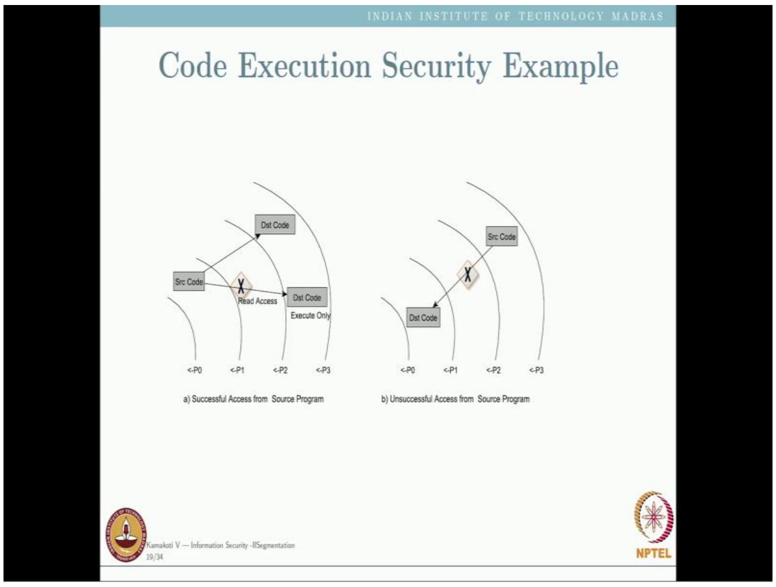
Type Field					Descriptor Type	Description
Decimal	11	10 E	9 W	8 A		
0	0	0	0	0	Data	Read-Only
1	0	0	0	1	Data	Read-Only, accessed
2	0	0	1	0	Data	Read/Write
3	0	0	1	1	Data	Read/Write, accessed
4	0	1	0	0	Data	Read-Only, expand-down
5	0	1	0	1	Data	Read-Only, expand-down, accessed
6	0	1	1	0	Data	Read/Write, expand-down
7	0	1	1	1	Data	Read/Write, expand-down, accessed
		C	R	A		
8	1	0	0	0	Code	Execute-Only
9	1	0	0	1	Code	Execute-Only, accessed
10	1	0	1	0	Code	Execute/Read
11	1	0	1	1	Code	Execute/Read, accessed
12	1	1	0	0	Code	Execute-Only, conforming
13	1	1	0	1	Code	Execute-Only, conforming, accessed
14	1	1	1	0	Code	Execute/Read-Only, conforming
15	1	1	1	1	Code	Execute/Read-Only, conforming, accessed

Kamakoti V — Information Security -IISegmentation
18/34

NPTEL

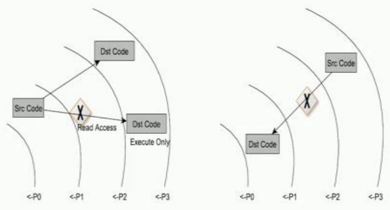
We said there are two types of descriptors, the system descriptor and the user descriptor. So, these are the 16 types of user descriptors that we have seen yesterday, the top 8 are basically data descriptors and the bottom 8 code descriptors.

(Refer Slide Time: 17:14)



INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Code Execution Security Example



a) Successful Access from Source Program

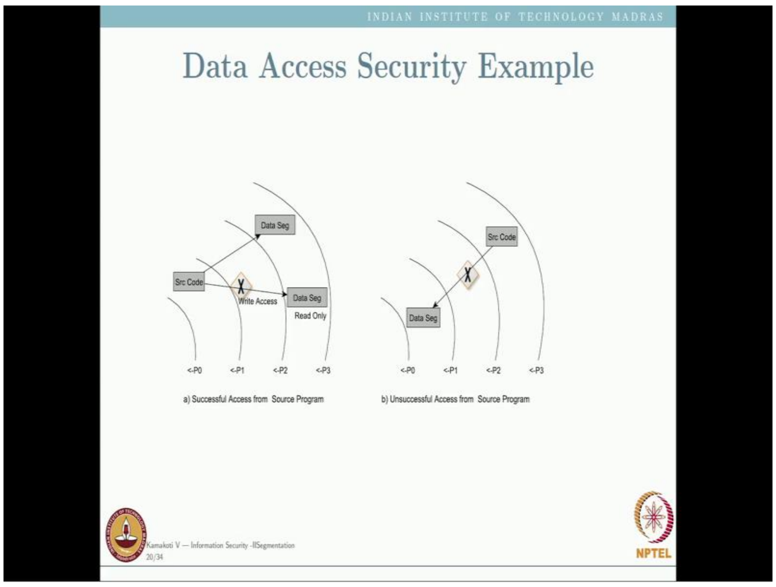
b) Unsuccessful Access from Source Program

Kamakoti V — Information Security -IISegmentation
19/34

NPTEL

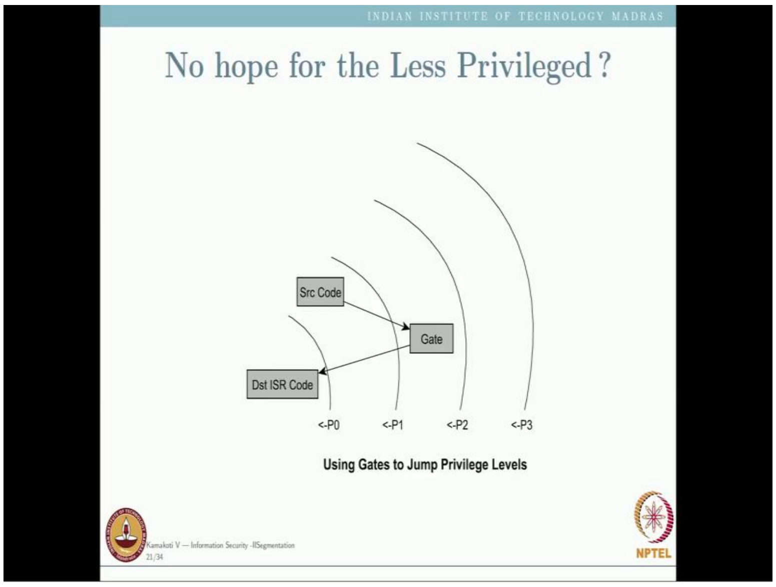
This is just a very quick summary of what we discussed yesterday. So, each descriptor there could be a privilege level for each descriptor, if your source code is at privilege 0; it can access a destination segment or a destination code or data segment which is at privilege 3 directly, but it cannot go and read a destination code even though it is in privilege 3, but it is execute only I will not be in a position to read it. So, I have a privilege 0 source code, it can jump to a destination code at privilege 3, but it cannot go and read destination code segment which is execute only, even though it is privilege 3. Similarly a source code at privilege 3 cannot jump or you know read a destination code or data at which I said the DPL of privilege 0.

(Refer Slide Time: 18:18)


INDIAN INSTITUTE OF TECHNOLOGY MADRAS
Data Access Security Example
Data Seg
Src Code
X
Write Access
Data Seg
Read Only
<-P0
<-P1
<-P2
<-P3
a) Successful Access from Source Program
Src Code
X
Data Seg
<-P0
<-P1
<-P2
<-P3
b) Unsuccessful Access from Source Program
Kamakoti V — Information Security -IISegmentation
20/34
NPTEL

Similarly, a source code at privilege 0, can read or write at privilege 3, but cannot write into a data segment which is at privilege 3, read only data segment at privilege 3. A source code at privilege 3 cannot read or write any data in privilege 0.

(Refer Slide Time: 18:42)


INDIAN INSTITUTE OF TECHNOLOGY MADRAS

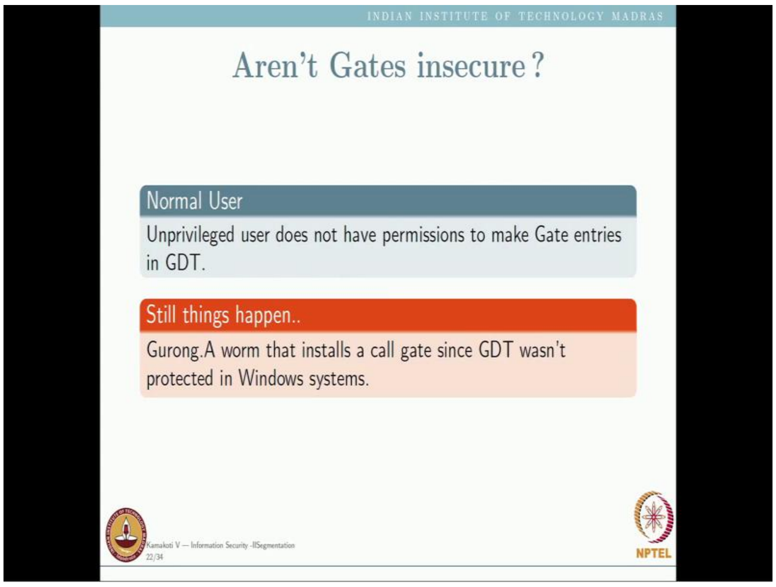
No hope for the Less Privileged ?

Using Gates to Jump Privilege Levels

Kamakoti V — Information Security -IISegmentation
21/34

For the less privileged code, we introduced a notion of call gate through which you know a code can access a numerically lesser privilege code. So, a source code in privilege 1, using a gate can go and access inter of service routine or any code at privilege 0 using the call gate or the trap gate.

(Refer Slide Time: 19:15)


INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Aren't Gates insecure ?

Normal User

Unprivileged user does not have permissions to make Gate entries in GDT.

Still things happen..

Gurong.A worm that installs a call gate since GDT wasn't protected in Windows systems.

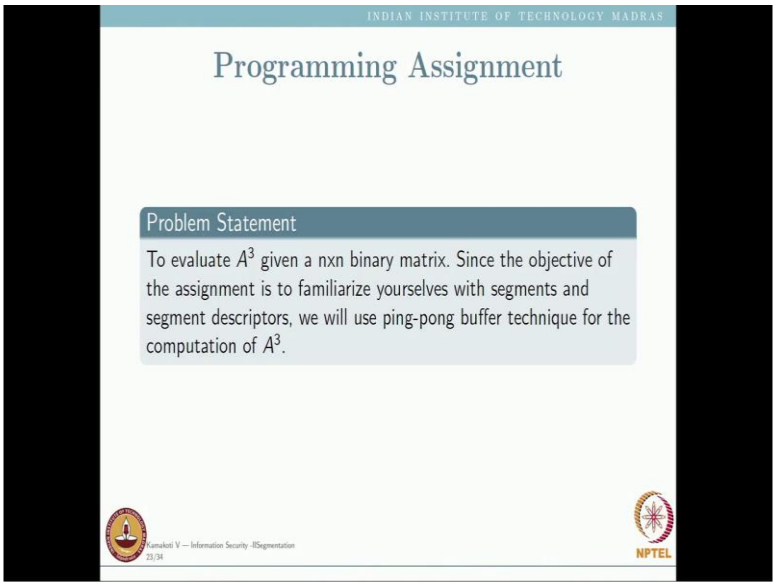
Kamakoti V — Information Security -IISegmentation
22/34

I will quickly go through again a source code at privilege 0 can jump to your destination code at privilege 3, a pro's at privilege 0 cannot jump to a destination code at privilege 3. If that cannot read at destination code segment at privilege 3, if it is execute only segment. Similarly a source code at privilege 3 cannot jump to a destination code at a lesser privilege, a source code at privilege 0 can read or write any data in the data segment at privilege 3 or above privilege 0 and similarly, but it cannot write into a data segment which is read only, even though the privilege is numerically higher.

A source code at privilege 3 cannot access, cannot read or nor write into a data segment visit at privilege 0. But a source code at any privilege can go and access something at a privilege level lesser than that using the notion of call gates and interrupt gates, we have already seen this. Now all these gates are the insecure, if you actually have some you know if you actually have if you are not careful about coding then it is a problem. So, this is one case study, Gurong is actually a worm that installs actually a gate call gate, so that it can get a higher privilege because in some earlier versions of windows the GDT was not protected. So, it went and installed a call gate there and once I install a call gate, I can do whatever I want, I can get my privilege escalations very quickly.

If I can go and define what the call gate is that call gate descriptor then I can do wonders please understand that it is a major vulnerability.

(Refer Slide Time: 21:22)


INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Programming Assignment

Problem Statement

To evaluate A^3 given a nxn binary matrix. Since the objective of the assignment is to familiarize yourselves with segments and segment descriptors, we will use ping-pong buffer technique for the computation of A^3.

Kamakoti V — Information Security -IISegmentation
23/34

NPTEL

Now, this is all about segmentation, this is a very, very quick summary about segmentation and now what we will try and do, I will answer some questions immediately and then we will go into the assignment part. Now, we will start looking at the assignment and hopefully many of you or all of you can run that assignment in your place and see that it is all working.