

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

Lecture - 13
Architectural Aid to Secure Systems Engineering
Session - 12: Process isolation Through Segmentation

Now, we go into the real crux of this course. Yesterday, I told you the basic fundamental thing about security is that I need isolation. Isolation from what? Isolation from memory, isolation from wherever I store some data, 2 processes this is all data processing and 2 processes communicate by sharing data. So, I need a complete isolation between processes and how will segmentation. The concepts that you have discussed so far in this session. How will it help us in achieving process isolation and that is what we are going to do. Already, we have talked about intra process isolation. Now, we will talk about inter process isolation.

What is intra process isolation? I isolated my data, I isolated my stack, I isolated my code. Now, today I am going to talk about isolating 2 different processes. There are 2 processes, process 1 and process 2, that has CS, DS and SS. That process 2 CS or SS or DS should not be accessible to process 1 and vice versa. How are we going to achieve this? I will go very slowly on this session. I will go step by step and I hope at the end, we will get some very good understanding.

(Refer Slide Time: 01:44)

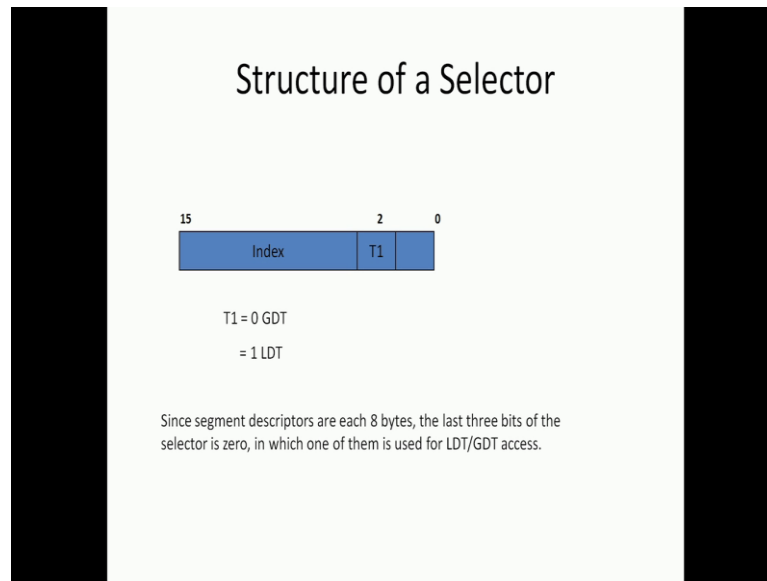


Descriptor Tables

- There are two descriptor tables
 - Global Descriptor Tables
 - Local Descriptor Tables
- The global descriptor table's base address is stored in GDTR
- The local descriptor table's base address is stored in LDTR
- The two *privileged instructions* LGDT and LLDT loads the GDTR and LDTR.

So, let us start from this point. First there are 2 descriptor tables, the Global Descriptor Table and the Local Descriptor Table. The global descriptor tables, base address is stored in GDTR. The local descriptor tables, base address is stored in LDTR. The way I could change that address is using 2 instructions, namely LGDT and LLDT and these 2 instructions are privileged instructions and it cannot be done by any Tom, Dick and Harry. It has to be done by OS, it has to be done by a privileged level 0 code. Now, I was expecting this question, somehow may be it is slightly obvious question.

(Refer Slide Time: 02:46)



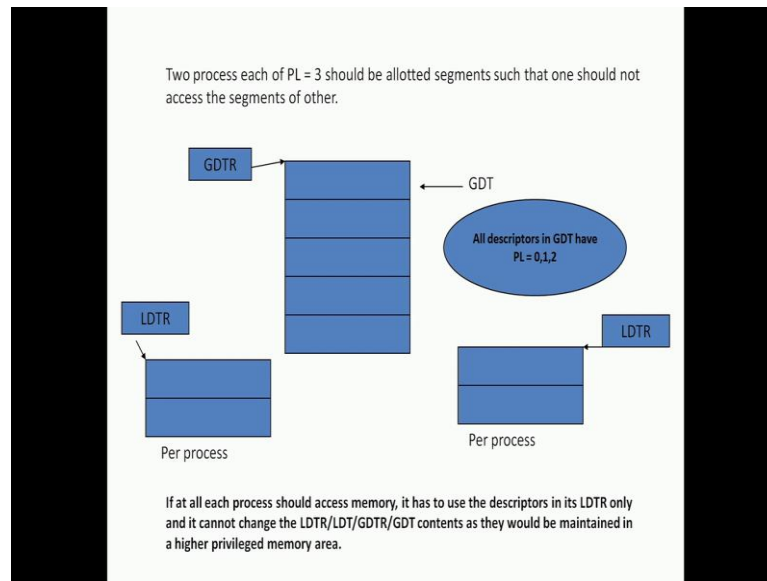
We said, jump 0x20 colon 1000, then I said go to the 32nd entry in the descriptor table. Which descriptor table? Is it the local descriptor table or the global descriptor table? Have you told there, how will you decide that and this is where some intricacies of the architecture have to be understood. Now, what is the size of your descriptor? 8 bytes. So when I start storing descriptors one after another, suppose I stored the first descriptor at 0 at 1000 or something then next descriptor will be stored 1008, so next will be stored where 60. So, the last 3 bits are always 0 because these are all 8 bits, in that and your descriptor table actually starts in 8 byte boundary. It starts in an address where the last 3 bits are 0, if it is divisible by 8 that means the last 3 bits should be always 0. So, I start with 0 with some address in which the last 3 bits are 0, then the next descriptor will be stored in another address which is also a multiple of 8. Again the last 3 bits will be 0 and so on so forth. When I am giving the actual offset address within the descriptor table, the last 3 bits will always be 0 and those 3 bits I can utilize it for some other purpose because they are not useful for me to find the offset. Anyway, I can append three zeros and go and get the offset.

In those 0 bits, the first 3 bits I will be using it for some other purpose. I am going to talk about it later. The third bit that is 0, 1, 2. The third bit which is the second number 2, 0 is the first bit, first is the second bit and second is the third bit. The third bit, if I make T1 is

equal to 0 that means it is GDT. If T1 is 1, that means it is LDT. If I say 0x20 that is 0 0 0 0 0 1 0 0. Even if I say 0x20 that is the first 4 bits are 0 then the next 4 bits are 0x1, that is how you convert hexadecimal to binary. Now, that essentially means second bit is 0, so it is a GDT. If I want to access this same 0x20 descriptor in the LDT, I should put 0 0 1 0 0 1 0 0. What is 0 1 0 0 4?. If I say 0x24, then essentially it means what the 20th descriptor in the LDT. You are getting this.

So, this is some architectural conventions that this is the way the architecture maintains. Otherwise, you know, you will have one separate bit for LDT, GDT becomes much complex. So, these are all some of the optimizations architecture as done. So, what will the architecture do? It will start loading your descriptor table in an address, which is a multiple of 8. We will see how it is going to be and that address will be pointed to by whom the GDTR and every offset within that address will also be multiple of 8. If I am a multiple of 8 then the last 3 bits are 0's. That last 3 bits can be used for something else and in which one of the 3 bits, the most significant of those 3 bits is used to indicate whether I am trying to access that descriptor in the GDT or whether I am going to access that descriptor in the LDT. It is very very important. Since, segment descriptors are each 8 bytes, the last 3 bits of the selector is always 0 in which one of them is used for indicating whether I am going to get the descriptor from the LDT or get that descriptor from the GDT. This is clear when we do the lab exercise, you will actually do this, so we will tell you how to do this.

(Refer Slide Time: 07:58)



Now, this is the most important slide which will try and answer you many questions and we will spend at least 10 minutes in the slide to understand what it means. This is the slide, which is talking about intra process protection in your setup. Let us say, there are 2 processes is each of PL equal to 3, privilege level equal to 3 that means, they are executing through a code segment whose privilege level is 3. Each of these processes will have its own, so when process 1 is created like I say, a dot out and press enter key, then the operating system will create a process execution environment for me and allow you to execute. When a process 1 is created by the operating system, it creates an LDT for it. When the process 2 is created by an operating system, it creates another LDT for it.

So, when process 1 is executing, it will load the LDTR with that LDT start address. When process 2 is executing, it will load the LDTR with the process two's LDT address. Essentially, this LDT is per process, for every process and now note that process 1 and process 2 are all privilege level 3 code. So, they cannot go and change the value of the LDTR, why? Because the value of the LDTR can be changed only by an instruction, namely LLDT and the privilege level for that is 0. So, it is a privilege restriction. Now, I have a GDTR and the process cannot even change that GDTR. All the segments in the GDTR will have privilege level 2 and up 2 and below. As a process 3, I cannot go and

use anybody in the GDT because their privilege level will be less than 3, it will be 2, 1, 0. So, I cannot use any segment in the GDT. Since, I cannot change my LDTR, I can use only the segments within my LDT, so whatever I can use is only within my LDT and in that LDT the operating system will populate only those segments which it has allocated to you. It will not go and allocate anything else. I can use only the segments within my LDT and those segments are filled up by the operating system. Similarly, I cannot also go and access that fellow's LDT because that is separated from me. If I want to access that fellow's LDT, I have to change the value of the LDTR to point to this LDT and I cannot change it because I am a privilege level 3 code, only a privilege level 0 code can change it. Now, I am restricted to use my LDT alone. I can use the segments that are described only by the descriptor inside my LDT and that is also populated by the operating system.

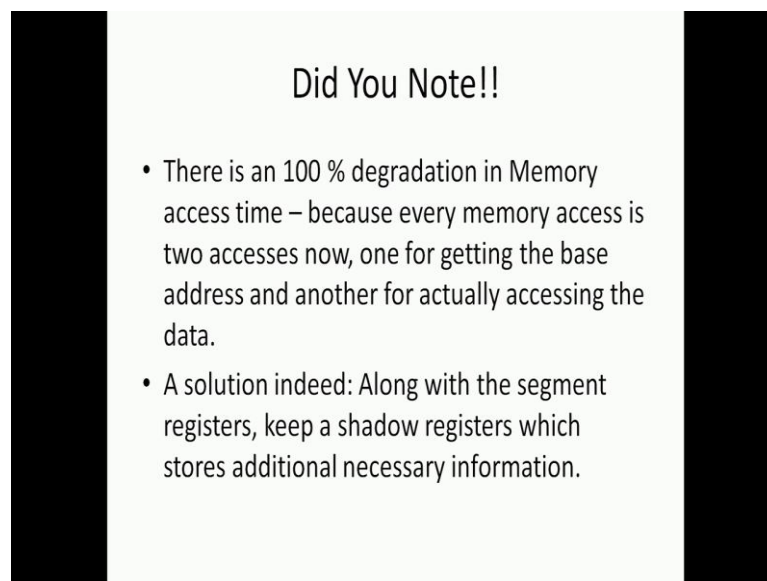
Now, the GDTR. The GDT and the LDT, all these LDT's are also in the memory. They are all inside a segment which is privilege 0, that is the operating system will have some memory reserved for it. In that memory, in the operating system memory these GDT's and LDT's will be loaded there and that segment will also be a 0 segment. So, I cannot go and change the entries in the GDT or LDT because they are in a 0 level segment. By this what I am doing, I can use only the code segment, data segment and stack segment that are part of my LDT. I cannot go and use the other fellow's code segment, data segment because they are not part of it. If I want to use that, I have to go and change the LDT value and that is not possible because I am privilege level 3 code and all the GDT is also in privilege level 0 segments, there I cannot use any segment in GDT. Then I cannot go and add anything to my LDT or add anything to my GDT because these are stored in a memory which is also at privilege level 0 in the operating system memory. So, I cannot go and add or delete anything into my LDT also. So, by this there is a complete process protection between process 1 and process 2, both are executing at privilege level 3, but this cannot touch that fellow's code stack data and that fellow cannot touch this fellow and they cannot do anything except. What it is assigned to them by the operating system? This is a model by which we can get isolation between processes.

Now, I will read the last block statement, if at all each process should access memory it has to use the descriptors in LDTR only and it cannot change the LDTR or the LDT or

the GDTR or the GDT contents as they would be maintained in a higher privileged memory area. They are all memory locations only. They are all inside the memory but if I want to go and change a code then I need access to that segment and that segment would be a privilege 0 segment. So, I cannot go into it. You all got this.

This is how a combination of the GDT and LDT plus privileged instructions put together and the privileged level associated with every descriptor. All these things put together can give us a perfect inter process isolation. But many of the operating systems, even if you go through the thing, does not implement it as rigorous as this and that is why we have some issues. We will look at that when you go into the operating system course, but let us understand this in this prospective.

(Refer Slide Time: 14:39)



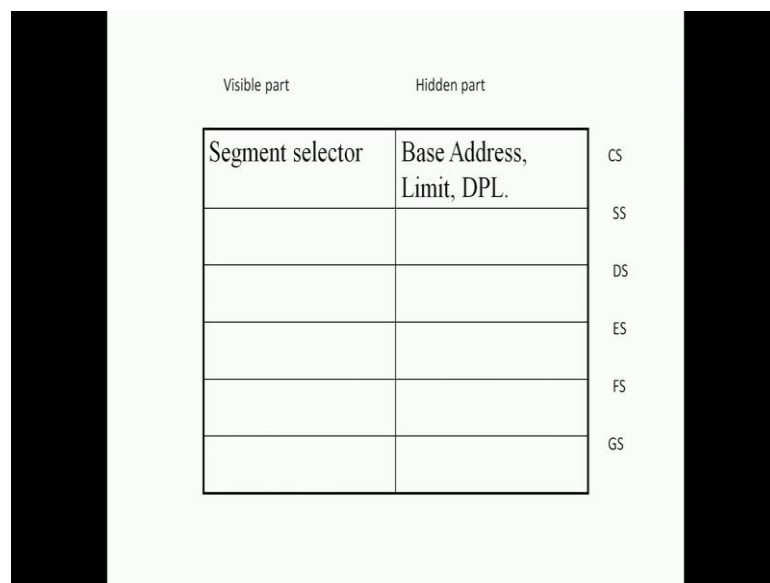
Did You Note!!

- There is an 100 % degradation in Memory access time – because every memory access is two accesses now, one for getting the base address and another for actually accessing the data.
- A solution indeed: Along with the segment registers, keep a shadow registers which stores additional necessary information.

Now, these are some architectural issues accessing memory itself is very very slow and that is why we bought in cash memory etcetera. We will not deal about cash memory etcetera, they are all topics of computer architecture. But we introduced a notion of cash because memory access is slow. Memory access is at least 10 times slower than the processor speed. It is a very very optimistic assumption. Memory can be much lower than the process. Now, what we have done by introducing this security issue when I want to access memory, first I have to go fetch the instruction. Now, I want to access memory,

first thing is I have to go to the descriptor table from there I have to collect the base and then add it to my ring, again now I have to go and access memory. So, 1 memory access has now become 2 memory accesses. Do you understand this? Already memory access from a performance point of you is like a monkey. Now, this monkey has become drunkard, mad, it kills your performance, now it becomes extremely slow. So, this is cost of security. How do you solve this problem?

(Refer Slide Time: 16:20)



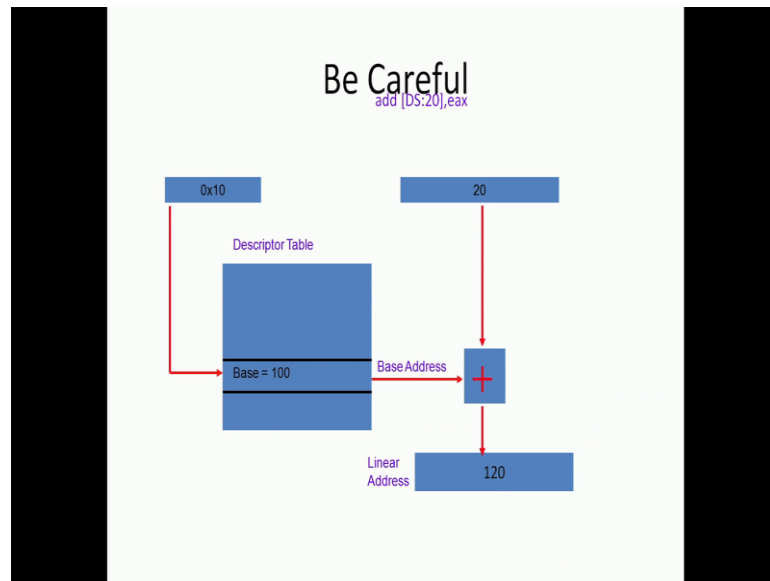
The problem is solved as follows. When I say CS, there is a visible part of that CS which is visible to the programmer, and what you see in that visible part? The offset into that table but there is a hidden part which you call as a shadow part is not visible to the programmer visible, means I cannot go and manipulate it. In that part, I will store the Base Address, I will store the Limit, I will store the DPL, the privilege levels. So, the moment I say move DS comma 0x10, I move the base address, the limit and the privilege level of that 16th descriptor also along with that. When I am updating the DS visible part with the value 0x10 that is 16 for the corresponding descriptor whatever the base address limit and DPL, I store it in the shadow or the hidden part.

So, whenever I am accessing DS what are the checks I need to do? What are the first things? I need the base address, if I say move EAX comma DS colon 0x1000 or

whatever. So, what I need? I need the base address, that base address will be available in the hidden part. So, I need not go to the memory, I will get it from the hidden part. Then what I do, I have to check the offset with the limit, whether the offset is exceeding the limit that also I will have it in the hidden part, and then what it is that I need to do? I need to check the privilege level that also in the hidden part but the architecture has actually made it hidden, so that the programmer cannot manipulate it, this is not manipulating. Still my notion of security remains same, but my performance now is addressed because this is completely a hidden part. For me to get the base, for me to get the limit for checking and for me to get the privilege level for checking, I need not go to memory every time. I will look at the hidden part, you are getting this. So, by this I am addressing the performance issue. There was performance degradation, there would be major performance degradation, if at all it is within this hidden part.

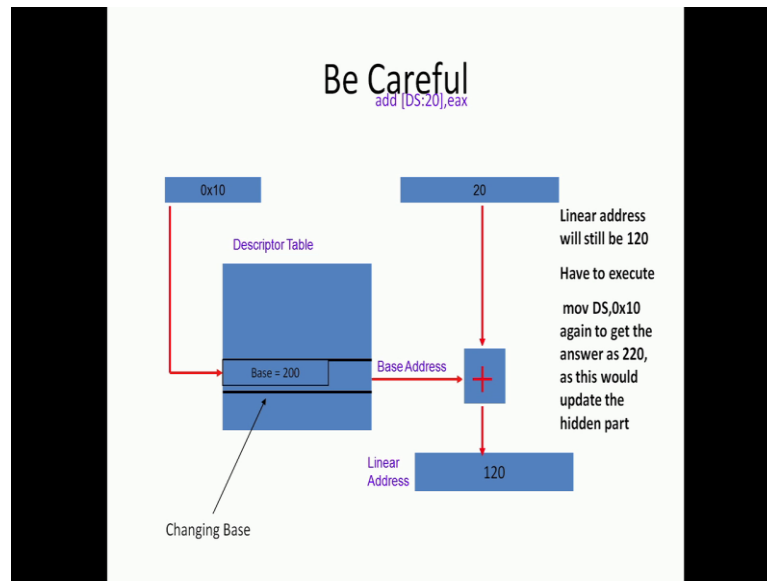
What lesson do we learn? Some point of time when we develop some secure architecture for specific purposes, we may do general purpose securities. One of the things is that more and more we start putting security checks, there will be a direct correlation in terms of degradation in performance and that every point of time we need to come out, which such type of innovative solutions that will address this performance and in the act of innovating a solution. We should also see to it that the security is not again compromised in this case. Yes, the security is not compromised to a large extent because we know the base address limit, DPL is we know that it is stored in a shadow part, which is not accessible to the programmer, it is isolated from the programmer.

(Refer Slide Time: 20:13)



Now, this is where you should be extremely careful. Let us say that this is `mov 0x10,` and that 0x10 is in DS the 16th descriptor essentially source the base address 100. Suppose, I say `add DS colon 20 comma EX,` what will happen? DS stores the values 0x10. So, I go to the 0x10 to the descriptor I get a value 100. That 100, I add with this 20 and I get an address 120. In that 120, whatever value is stored I take that, I add it with the value of EX and store it back there itself. So, this is destination source. I take the value at 120 and add it with the EX and store it back to 120 memory location itself. So, this is how this happens.

(Refer Slide Time: 21:32)



Now, what has happened is I go and change the base from 100 to 200. I just go to the memory location of operating system and make that value as 200. Now again I execute, add DS colon 20 comma EX, what will the new value be? The new value will still be 120. Just because I changed it to 200, the hidden part will not change because this is in memory location. There is no way by which I go and say that this memory location corresponds to that hidden part because that type of a link I do not have. As far as I am concerned, I am going to some memory location and making 200 but this has to get reflected back in the hidden part of your DS. So, after I change this 200, immediately if I go and execute, add DS colon 20 comma EX, still it will give me the answer 120. So, whenever I go and change the base, immediately I have to execute, move DS comma zero x 10 again, that time what will happen, zero x 10 will be there but then the new base and the limit and the address will go and get updated into the hidden part. Now, you again do, add DS colon 20 comma EX then it will give you 220. So, these are something that happens, when you start.

So, why I am giving you this case study is because some point of time, you start porting separation kernel on to your x86 or ARM platform. Then there are lot of issues that will come and many of the things you may have to do it in assembly and when you start doing this, these are some of the issues that will come very very important there. So,

what did we do? We wanted to introduce security because of introducing security, there was performance degradation. To remove that performance degradation, what did we do? We added a shadow register and because of adding that shadow register, we ended up with some other problem and that is what we are solving.

So please note that every problem has a solution which leads to another problem, which again has a solution, which leads to somewhere, it stops. In this case, fortunately is stopped here. So, what I am trying to tell here is to express this feeling here that whenever we try to solve a problem, where we miss is that we do not go through the grass route. So, this is one example, where we have gone fully to the end and saw that everything is fixe and this is also a case study, where there is a problem, let us have this fixed and what is the implication of that fixed, is very pretty obvious here right, but it is an enemy of correctness. So, you lose that correctness, if you think it is obvious, this is how we proceed.

So, please have these case studies in mind because when you build a security architecture in whatever form, these types of small, small things actually makes a big difference because ultimately if I give yesterday as I told you, I have a very, very good security in my financial institution but I say every one week you change your password and no fellow will come to your institution. They will close your bank account and go to some other bank. So, you should get a security solution that is also user friendly. Suppose, I say, if you enter this hall, I will put some 5five finger print and each combination of fingers. Usually, I do not want to attend this course. So, it is very very important that you get some practical solutions and here just introducing that privilege level, what has happened? We landed up in a mess with respect to performance and then we say I am going to market this processor without this type of fix then your code will be very very slow. Nobody will buy it, fine if I lose data, it is fine, I cannot undergo this torture in case then. So, you have to come out with fix and that fix will have some other issues.

This is one example of that.