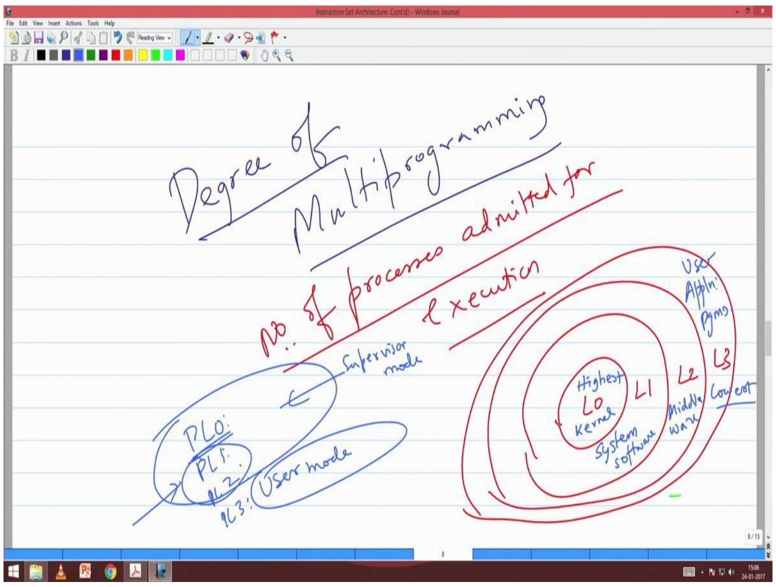
Computer Organization and Architecture

Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

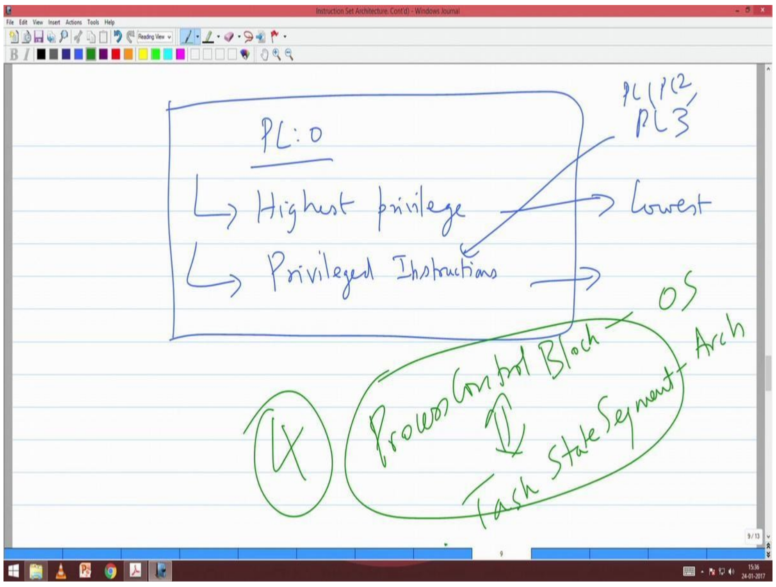
Lecture – 11
(Part – III)
Lab 2: Segmentation

(Refer Slide Time: 00:19)



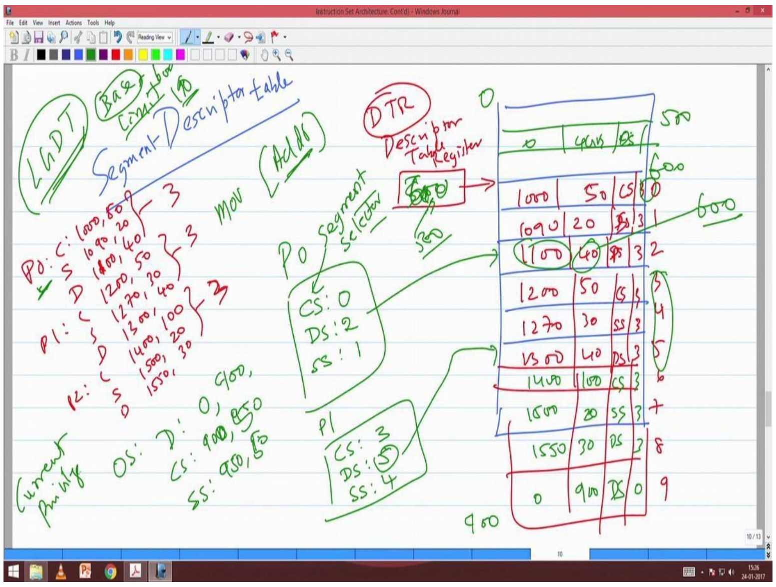
Now P L 0 has the highest privilege, P L 1, P L 2, P L 3 this is the user. So, many of the operating system that runs on x86 architecture will take all these 3 privilege levels are supervisor mode, and this as use some mode right, but the actually you can do much more classification that it does not exploit this 3 modes there, but anywhere P L 0 is it will actually go to P L 0 for supervisor, and P L 3. For user it may it will not to use these many I am not seeing any unique code which uses P L 1 and P L 2. So, the operating the processor itself will work in 4 privilege levels, privilege level 0 privilege level 1, privilege level 2, privilege level 3 you understand right.

(Refer Slide Time: 01:26)



Let us talk about P L 0 this is highest privilege there are certain instructions that could be executed only by privilege level 0 code. Those are called privileged instructions. This is be executed only by a privilege level 0 code on other fellow can execute this if I am a privilege 3 code I cannot do it you getting. So, this is privilege instructions. So, these 2 are very important for us to understand P L 0 these 2 are very important. Well P L 3 is lowest privilege and it cannot execute any of the privilege instruction only. P L 0 can execute P L 1, P L 2, P L 3 none of these can execute privilege instructions right.

(Refer Slide Time: 02:49)



Instruction Set Architecture. Cont'd) - Windows Journal
File Edit View Insert Actions Tools Help
Reading View

LGDT

Base

Limit 190

Segment Descriptor table

MOV [Addr]

P0: C: 1000, 80
S: 1090, 20
D: 1100, 40 } 3

P1: C: 1200, 50
S: 1270, 30
D: 1300, 100 } 3

P2: C: 1400, 20
S: 1500, 30
D: 1550, ... } 3

Current Activity

OS: D: 0, 900,
CS: 900, 950
SS: 950, 50

DTR
Descriptor Table Register

500

P0 Segment Selector
CS: 0
DS: 2
SS: 1

P1
CS: 3
DS: 5
SS: 4

	0	40NS	DS	
0	1000	50	CS	3
1	1090	20	DS	3
2	1100	40	DS	3
3	1200	50	CS	3
4	1270	30	SS	3
5	1300	40	DS	3
6	1400	100	CS	3
7	1500	20	SS	3
8	1550	30	DS	3
9	0	900	DS	0

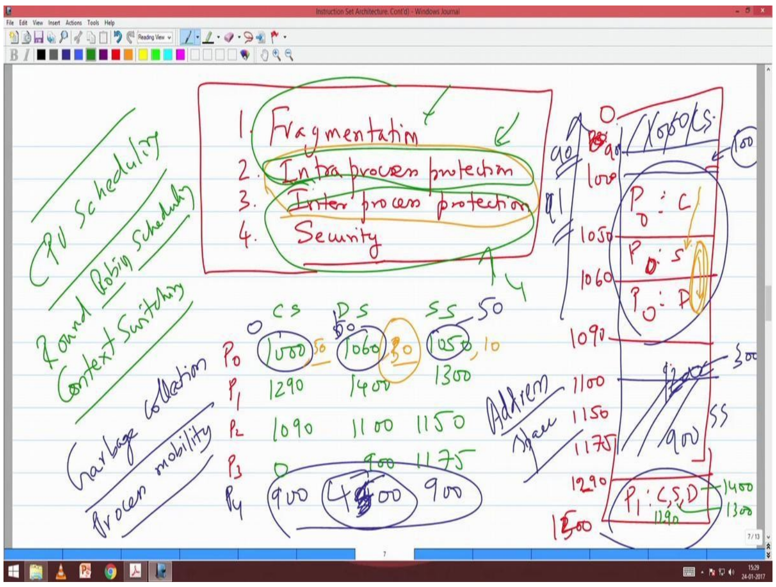
0
500
600
600
900

10
10 / 13
15:26
24-01-2017

So, let us let us have the notion of what a privilege instruction is right. Now what happens is as follows now when the system comes up operating system comes up in your memory it creates a table called a descriptor segment, descriptor table. This segment descriptor when the operating system comes into existence it creates a seg descriptor table is called a segment descriptor table. Please note this carefully because I am going to talk about security here and this is very important. Now this descriptor table will be starting from some point and that point is pointed to by a descriptor table register DTR. Where it starts that will be stored in a descriptor table register able to understand what I am saying.

Now, in the descriptor table register what I will store is for every segment that I am creating and the operating system right. In the descriptor table I what I will do is for every segment I am creating. So, let us say P 0 had code stack data P 1 had code stack data P 2 had code stack data P 3 had code stack data let us go a little back and see where were they.

(Refer Slide Time: 04:42)



So, 1000, 1290, something I will just start 1000 1000 comma 50 1090 comma 20100 comma 40, this is 1200 not cross right 1200 comma 50270 comma 30300 comma 40. So, 1400 comma 100 1500 comma 20050 comma 30600 comma

100 10700 comma 10750 comma 40 something like this.

So, we are all the segments. So, whenever the process P naught came into existence immediately your sigma at descriptor table the operating system, what it does is it allocates that memory what it means by allocating with memory it will just put. So, let us in the. So, this descriptor table has say several descriptor. So, let me say 0 1 2 3 4 5 6 7 8

9. So, 9 descriptors, when P 0 came into existence we will have put thousand here and 50 here. Limit and it will say this is a code segment and it will give a privilege level. So, this is a 3 process operating system is executing it will give you a privilege level 3 this is user process.

Let me say this is user process privilege 3, this is also user level this is 2 and this is all are 3 all are 3 each. Then it will create another 1090 20 and this will say this is a data segment and privilege 3 then it will say 1140, it will say stack and then it is 3 stack segment. So, let us just fill it very fast 1270 1350 30 40 c d SS this is SS and DS sorry c

SS s DS 3 3 3 1400 1500 1000 550 now 100 20 30 c SS s DS 3 3 3 and so on. Now this will be stored in this now there will be some one thing for the operating system itself.

So, let me say this start at 0 and it goes on till you know 900 because your 1000 all your. So, first part is loaded by the operating system there will be one data segment for the operating system which will say I will start from 0 and it will go till 900 and say this is data segment and there is privilege and will be privilege. So, the operating system when the operating is executing let me say forget this P 3, when the operating system is executing is data will be some 0 to 900. So, let us say limit is 900 and code will be from 900 1 to say see the 900 to 9 50 or something (Refer Time: 09:08) stack will be some 950 to 1000. And your remaining let us assume that this is how the day operating system very simple small operating system.

So, when the OS is running suddenly one process comes up a I want execute a dot out comes. So, the OS gets initiated now it will go and allocate thousand 1090 1100 in the descriptor table now. So, what will be stored in CS, it will not stored thousand 1050 rather it will store 0 in DS you will store one sorry 2 in SS you will store one when P 0 is executing right. I will not store the base limit etcetera what I am going to store there is the selector. So, this action shelly rather than calling this is as a segment register, I will call it as a segment selector because it is selects from the descriptor table what segment I need your getting. This I will able to follow I select from the descriptor table what I need.

So, here please note that 1050 CS 3 is stored where in the descriptor table. So, when I start executing in CS, I will put 0 in DS I will 2 in SS I will put 1. So, the moment somebody wants access data it goes to 2 the entry to here there it form a finds the base address one 1100, then it will also check if the offset is more than 40 if it is more than 40 it will say this fellow is doing something wrong is raise an exception, and then it will start execute. So, instead of storing the value 1100 in DS, I store it in a table and I make this point to a table. Now what happens and P 1 comes to execution you as CS now will become 3 DS will become 5 SS will become 4 right basically these 3 segments 3 4 and 5.

So, when P 1 wants to access data what will happen it will go to 5 in 5 what is the entry 1300 and that will be treated as a base and there it will go and check if I am going to access anything more than 40 bytes from that price if it is going to be it will rise and exception. So, what happens is right I can basically go and access these values now other than this value other than this value. So, what the most important thing is that I cannot go and change anything here as a user process like P 0 right, as a user process P 0 can I go and change anything in this segment, if I want to go and read or write from the segment I have to put some offset address and that will get added automatically to my segment base.

So, even if I want access memory, I will put move some address here I have to put that this address automatically will get added to your data memory if as P 0 right my data memory is 1100 it will get access to 100 and if it is greater than or this onem 40 bytes then automatically it will say sorry it will not even allow u right. So, if I want to go and change any of these values suppose I want to go and play with this table suppose I want to go and access say P 3 is P 2s data right. So, essentially I have to go and make this as this limit as sum 600 if I make this limit as 600 then I can basically go and play with the data, but if I want to change this to 600, then what should I do I have to go and access this and change it I cannot access this right because I am restricted to access in this.

Student: (Refer Time: 13:38).

I cannot even go and see I have to go and change this no it depends upon whether it is a read only segment or a right if it is a read only, if it is 0 if this is privilege 0, first and form us I cannot come to this at all after that only I can go and change. The other thing is what I can do, I can go and change this descriptor table register. So, it this is descriptor is

stored at sum 600 right, this is 600 from 600 onwards this descriptions are stored. Suppose this 600 I will make it as 500 and 9 500 I will get some space here 500 and there I will add my own descriptor which is start at some 0 and go to 4 g b with DS, and I can go and play with it correct. So, for doing that what should I do I should go and change the descriptor table register. So, there is yeah there is a command called LGDT load global descriptor table register. This LGDT is and you give some arguments to this P k will teach this and great depth there right.

What will happen is this LGDT will go and update this register. So, initially I had this as 600, and from there this entire table was stored. I can for me to change it to 500 I have to execute a command call LGDT right. And if process 0 wants to go and execute a command of you know this fellow LGDT, process 0 is at which privilege level 3 right now LGDT is a privileged instruction, privileged instruction means who can execute LGDT only a P L 0 code get execute right. So, the process cannot go and change the P 0 cannot go and change the value of the base of the descriptor table, and add it is own descriptor. Because this LGDT is a privilege instruction and only privilege 0 code in our case the kernel can go and touch it. So, when I descriptor table is fixed with privilege 0 then what happens nobody can change it only the kernel can go and change it right. And for me to go and change anything within the descriptor is not possible because I will not be permitted as P 0 I can only access 1100 to 1140 39 anything more than that I will catch.

Permit to access anything more than that 39 I have to go and change, the entry in the descriptor table. I cannot go and change the entry in the descriptor table right. It is a chicken and egg problem. So, to change the entry in the descriptor table I need permission to with have the permission, I have to change the entry in the descriptor and I can go and add my own fancy descriptor because that LGDT basically tells me where the descriptor starts and I cannot go and change it. And also there is a limit here LGDT will give you the base of the descriptor table and also the limit tells how many segments currently I have. I cannot go and add more segment. So, currently I have only 9 segments. So, this base will be some 600 and the limit will be some 9 or sum multiple of 9.

Now, I cannot go and add more than 9 segments 9 or ten. So, 10 I cannot add more than 10 segments. So, I will not be in a position to create a new segment descriptor (Refer

Time: 17:33) you are able to get this. So, this is one part of the story so. So, now, the first thing that, we will understand there are lot of doubts I will come to those doubts, but please understand here that my segment register essentially stores as selector which is nothing, but an index into your descriptor table and my descriptor table basically stores the descriptor table register basically stores the start of the descriptor table. And the value of that start that register can be changed only by a P L 0 code. And what is stored there would be 4 things for every descriptor it is base it is limit what type it is and then what privilege level it is.

Basically it is describing the segment it is giving the attributes like your name is, and so your roll number is. So, this here this goes it is basically giving you the attribute that is why we call it as a descriptor table. Now as a process as a process, I can access only any data. So, what is a privilege level of a process for every. So, I am executing from some. So, I am executing code segment right code segment 0, when P 0 is executing code segment 0 that code segment 0 as a privilege level 3. So, there is a privilege level. So, whenever I am executing my code segment register CS register has a selector that will be pointing to some entry in my descriptor table. That entry would be basically a code segment type code segment and it will have a privilege level and that is the privilege level.

So, that is also called as CPL current privilege level. So, initially your operating system will be executing it is P L it is privilege level will be 0, when the contact switches took; that means, when the contact switches this c c CS register will be loaded with this segment DS with this DS then what will be the privilege level, now CS segment points to 0 the privilege level now becomes 3 when P 1 comes into existence, now the CS register points to 3 select selector 3 and that is that as privilege level 3. So, that is how right. So, as you go from one process to another each process when it is executing will have some selector in the code segment otherwise it cannot execute.

And that code segment as a selector and that selector will have automatically a privilege level. It will actually point to some descriptor in a descriptor table and that will have a privilege level and that is the privilege level of your process. So, what we will. So, there is there is there is one more very important question that we need to answer, but that we will reserve it for next class, but today what we do in assignment one is that we will now work with privilege level 0 code you are the kernel. So, you execute everything as a

kernel mode right. So, we will only right privilege level 0 code, but in privilege level 0, I want you to set up at descriptor table. In the descriptor table I want you to add lot of descriptors DS CS SS ES FS and all and we will give you a program in which the data will be stored there code will be stored in some segment data will be here in there. You have to set up all those you have to set up this GDT you have to setup your GDT here and then you load your program and start executing.

That is what you will be doing here say essentially what you are learning is that will be given a problem, for which you write an assembly you write a c code and then translate it assembly code that will be easy for you. You understand some amount of translation from a programming language to a to assembly language. By this you get a good grip on what a good grip on assembly language, and translation basically. Once you get this we also when the program has to go for execution when it goes and says the operator is the way allow it to execute what are the steps done by the operating system you are going to do that manually. You are going to create a code segment you are going to tell story a data statement stack statement that you are going to loaded you are going to already will have here descriptor table registered pointing somewhere you have going to load these values, and then execute the program and then show the results.

So, essentially when a program wants to go and execute what is the architecture support we are not writing operating system here, we are just write trying to understand how the architecture will support this and that part we will do it. So, as you see we will do it in the first assignment. So, there are lot of benefit is of segmentation. So, these 2 fragmentation of course, I am explain what is fragmentation intra process protection we will do in the first assignment itself while inter process protection, and security we will do it in the fourth assignment.

So, the first assignment would be on this segmentation the second assignment would be on interrupt service routine or exception handling, the third would be on virtual memory and then the fourth will be on task scheduling. So, we will we will do all these 4 assignments and the next thing is that you will never be given if you are continuing on system building next generation is going to be on mobile phones and the IOT devices you will never have such a fancy debug environment, understand.

So, lot of time you have to do from scratch. So, imagine writing and operating system without a debugger. Operating systems have to be written without a debugger, why? Because debugger runs on top of the operating system, you do not even have an operating system, I will have a debugger. So, imagine writing everything by just seeing some 7 segment display and trying out something is working or not see look at that. So, this type of a very careful development, lot of time we will fail because you do some silly errors. So, this will make you very attentive when you do program. So, lot of benefit is do come from this. So, please take it in the spirit that and I want you to work out some.

Now doubts somebody had a doubt yes Jnanabik, yeah.

Student: (Refer Time: 24:10).

Exactly. So, the operating system will maintain this book keeping. The operating system will maintain this information for which process what is. So, it maintains something called a process controlled block, you will be learning that in assignment number 4.

Student: (Refer Time: 24:41).

For every process yes unfortunately lot of book 104 different entities have to be stored 104, 1 0 4 and at least 3 or 4 of them or pointing to books, book in the sense this type of information management. So, that is what it means to write an operating system. So, yes this is a very good query, I need to maintain information about processes. So, there is something called a process controlled block, in operating system we will call it process controlled block in in architecture we call it tasks state segment process control block is the language of operating system. This is equivalent to what we call as task state segment in architecture now these 2.

Now, about you will learn in assignment number 4. How we can go do this book keeping etcetera we have we have some doubts jnabik.

Student: Is there any base the user process can.

Is there any way by which the user process can manipulate the selector, yes, user process can load selectors we are going to come to that in great detail that is one question that is unanswered, but then we will do it in as a assign next doubt?

Student: (Refer Time: 26:14).

There is a virtual no as of now I am not talked about. Virtual memory at all virtual machine, there virtual machine does know it is a here now working at the hardware levels. We have across the virtual machine you are now taking assembly code and executing on the processor. So, essentially coming back the difference between what you did as an assembly language that time right, in your third semester and this is there we were concentrating on things that are important for executing your program correct, we never talked of anything that is important for the operating system. So, lot of operating system operating system our view us that it give us a library of functions, and we used it. You remember the assignment right do not forget it man you all told it is tough means it should be in your head that is out.

So, right now when we go in what you are going to see that the compiler path that is there already there. So, we just gave you some tutorials and then you can follow that. Now I know how you can translate it translate a normal program into you know assembly. Lot of concentration now will be on those type of machine language instructions which are going to do functionalities related to operating system means. So, that is that is the new coloring, for a there is no need for me to come and sit in a B.Tech priviliar premier institute fourth semester and say a x comma b x will multiply a x into b x and store the answer in a x, there should crop, I do not want to do that, what that anybody can understand you can understand.

What we need to understand here is there how this assembly instructions will help lot of things on the operating system saying that is. So, as you see the coloring is going in that way right we are talking more about OS, fund as rather than co general compiler fund as right, that is very important here.