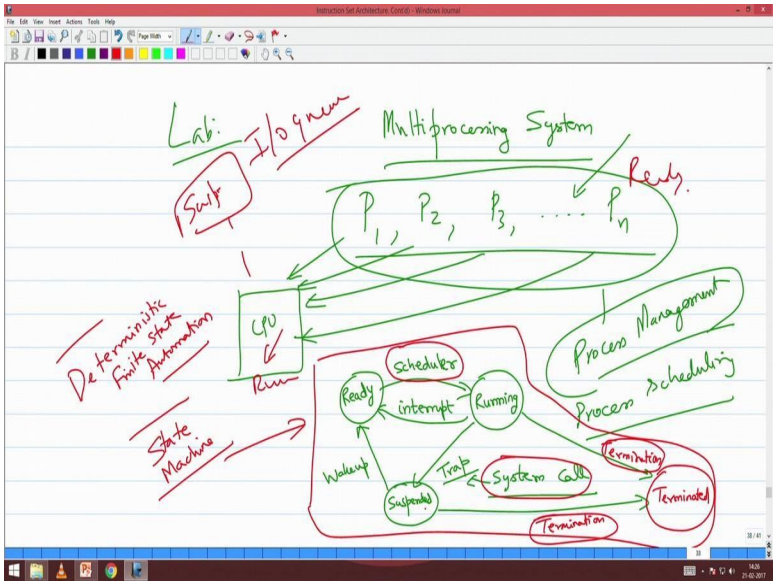
Computer Organization and Architecture
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 22
Process Management

The main important aspect is that in a system today in a multi processing system, there are multiple processes all of them need to execute and we have say one CPU right on which this these processes can basically execute one after the another.

(Refer Slide Time: 00:22)



And there is an operating system routine or part of the operating system routine which is called process management, which will be responsible for assigning the process to the CPU. The process management is actually responsible for taking say this process will execute on the CPU at this point of time right. So, as we had instruction scheduling we also have something called process scheduling right and importantly.

So, essentially the processes can be in three stages a processes, a process the moment I type dot slash a dot out it comes and sits in a cube it is called the queue. So, the process the process in what is in a state called ready state right it is ready for execution, right

then your scheduling algorithm or what we call as a scheduler will schedule it then it goes to a running state right, if there is an interrupt it comes back to the. So, what is an interrupt? Exceptions are of two types, interruption traps interrupts are those that are given by an external input while trap is something that is given by the process itself.

So, it goes into an if it gets an interrupt it goes back to the ready state, while if it executes a trap then it goes to as suspended state and then on wake up it basically comes back to the ready state. For example, if I want to do a print f I go to a suspended state because the printf is going to be executed by the operating system. Printf is a routine which is to be executed by the operating system right or malloc it is a routine that is going to be executed by the operating system. So, that type a thing this trap we call it as a system call. If I execute a system call then what happens I as a process will go to a suspended state the operating system will start executing it will finish it after finishing it will wake me up and I again go to a ready state, and again it gets scheduled to start running are you able to understand this.

And at some point of time on running or in suspended state I may have what we call as termination, I will go to a terminated state. So, this is basically. So, the process can be in four states it can be in a ready state, it can be in a running state, be suspended, it can be terminated right are you able to follow right. So, scheduler is an operating system path when I do a trap this system call is also part of the operating system right then the process completes again this termination is done by the operating system. So, a termination is also is done by the operating system, what do you mean by a terminating a process. Now that you have understood lot more computer organization what are the steps involved in terminating a process I have to free the memory I have to free all the resources allocated etc etc basically freeing of the memory and then the all the information that are necessary for this.

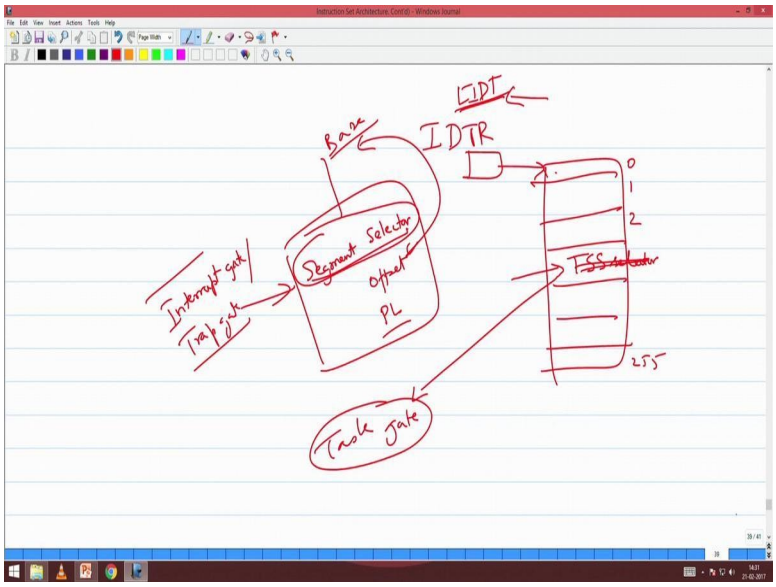
So, this is how a multi process system. So, this is called what, this is called what we call as a state machine. So, I am using a state machine to describe the difference states of a process. You see state several state machines when you want to describe different aspects of your architecture operating system software, database everywhere you see state machine. So, what is the state machine you have learned no automated theory right to

some extent. So, what is the state machine called? Deterministic finite state automation.

So, in the operating system course that you are going to do next semester, you will go more in depth in to how the state machine is you know processed right the role of the. So, the one of the important functionality of the operating system is process management and what do you mean by process management? For every process I should go an fix which state it should be; every process has its own state machine it has its own state information. So, all the process in this queue will be in ready state all the process that is executing on the CPU will be in running state and some process waiting on the I o queue or something like that right for doing the I o that will be in suspended state and so, on and so forth.

So, how do I manage the state transition of each of the processes essentially constitute what we call as process management. So, when you start your operating system course, you will learn about process man in great depth right. What you will do now is, what are the underlying architecture features that will enable you to do process management. Are you able to get this? What are the underlying architecture features that will enable you to do process management and that is what we are trying to do. So, already you have seen segmentation, you have also seen you have been exposed to inter service routine.

(Refer Slide Time: 07:37)



So, now, with since you have understood how interrupt service routine works interrupt descriptor table register, that will point to a start of an I DT and there are entries 0 1 2 till 255 in this, now every hardware trap will be assigned a number. So, you go to this number and in this number what will be stored? There will be a segment, segment selector right and then off set then there will also be a privilege level.

So this is called basically this is called the interrupt gate there is some small difference interrupt gates slash trap gate. In this I can also put something called a task take segment selector, TSS selector or what you call as sorry I can call it as task gate we will deal about it later there is also a thing that we can put at task gate here. So, currently what you are going to in your GDT what did you put? You put normal segment selector for data and code here you are going to put segment selector or segment descriptor for interrupt and that will basically have a segment selector, now you where this segment selector stored in can be stored in GDT where it can be stored in GDT or LDT now.

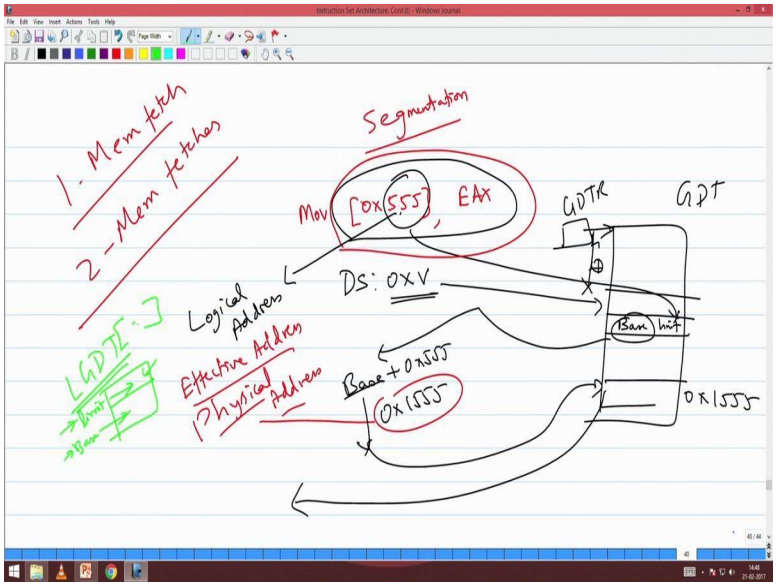
So, basically now from there you go there and there what will you get.

Student: (Refer Time: 09:20)

The base address, to that base you will add this offset check for the limit and start executing from there this is what the hardware supposed to do. Did not you need this are you in the class or not. So, how do you fill up this IDTR? Using the instruction LIDT; and this LIDT is what it is a privilege instruction. So, only the 0 level can set up an interrupt descriptor table right. So, this is the entire set up about I DTS. So, this is the set up about interrupt I hope you all know everything about segmentation segment descriptor privilege level limit etc ok.

Now, let me talk of some couple of scenarios. So, architectural you know enhancements to improve performance right. So, this is one that we need to look at from segmentation. So, this is something more than what you have studied, maybe you have studied this let me just find out.

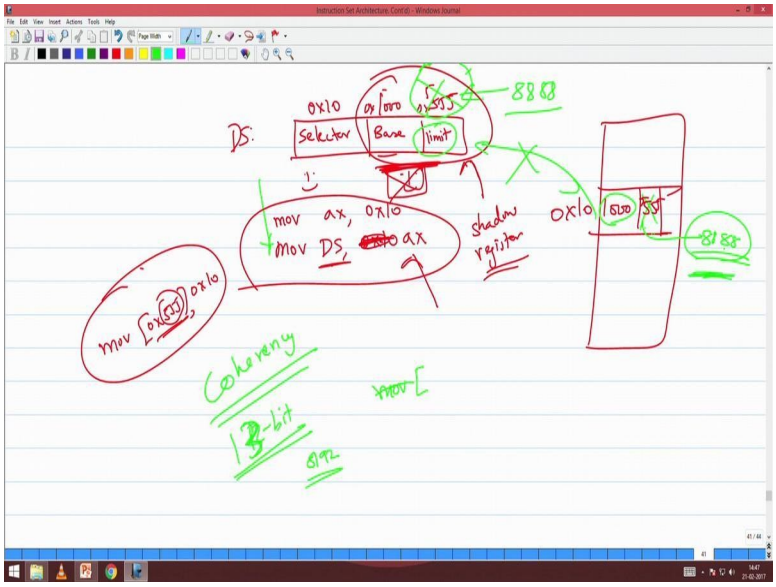
(Refer Slide Time: 10:35)



So, what happens if I say mov sum 0 x 555 comma EAX, what should happen now, what are the steps involved in doing this. First this is there is a there is a DS register, the DS register will have some value let me say 0 x v, I go to that corresponding entry in the GDT, there is GDTR which points to the basis of this I add this and basically get to this entry here what I am going to see I am going to see the base and limit, I take the base from here and I do base plus 0 x 5 5 5. I also check the limit 5 5 5 with this limit and if it more then I say segmentation fault otherwise I continue here.

So, then what this gives me let the base be some 0 x 1000 then it gives me 0 x 1 5 5 5, this again go to the memory 0 x 1 triple 5 I access and bring it back. So, what one memory access essentially means going to the GDT which is where is the GDT stored also in the memory, and then getting the base and limits checking it and then calculating. So, this is this address is what this is logical address and this address is actually physical address or what you call as effective address correct. Once I calculate the effective address again I go again I get. So, one memory fetch actually became what today now it has become two memory fetch one memory fetch is essentially became two memory fetch correct now how do you solve this problem.

(Refer Slide Time: 12:59)



Now, is a DS register there will be selector then this is visible, then there is some invisible path which will be the base and limit this is the invisible part. So, nobody can actually go the user cannot go and change. Now what happens let us say this is the GDT t and there is some say 0 x 1 0 entry, now when I say mov and let it as base as 1000, limit as some 55 and some. When I say mov DS comma 0 x 10, what happens actually I cannot say 0 move comma a x, mov a x comma 0 x 10 always you do like this right. So, when I say a x comma 0 x 10, what will happen and mov d x comma a x what will happen? 0 x 10 gets stored here 1000 gets stored here, and 55 gets stored here, but this is a shadow path this is called the shadow register. So, when I say mov 0 x 555 comma say some 0 x 10 or whatever ,this 555 is added to the base that is stored in the shadow register I need not go to the memory. So, I will just add to this base whatever this will be 5555 also I just add to this base in the shadow path right instead of going to the memory.

So, whenever I am updating the DS register, the selector gets stored in the DS register in addition my base and limit as stored in the GDT or LDT stored here. So, whenever I want to calculate 0 say like instructions like this, I need not actually go to the memory to find out that the base is 1000 that will be available in the shadow register itself. So, the two memory access actually became one memory access correct now what is the draw

back of this. So, suppose I want to you know increase the size of this segment this

segment as grown I want to increase it happen more memory. So, I want to raise the limit from 555 to some 8888. Now what I will go I will go and change in this location, how will I change in this location? I use normal mov mov whatever GDT are let it be some 1000. So, I will use normal mov to basically go and change this location change. So, I can go and change this as 8 8 8 8 this is I am only touching the memory.

That 8 8 8 8 will not get how will it get reflected here? It will not because I am only changing the memory I can go and change memory in way I want right this like normal memory operations like mov I used to go and change this, when it gets updated here in the GDT how will it immediately updated it will not get updated. So, even though this has become 8 8 8 8, if I take selector pen and I will look at the base limit it is still 8 8 8 8, but the shadow path will still retain 0 x 5 5 5 right. So, the moment I go and change the limit again I should re execute these two instructions to see that this 5 5 5 is replaced by 8 8 8 8. So, whenever I go and change the base or limit of my descriptor, I now go and do mov a x comma the descriptor mov d x comma a x I go and do it so that whatever change I have done to the base and limit or privilege level essentially gets updated here also ok.

So, what this problem called? Some value in the shadow register and some other value in the GDT, what is this problem called I have talked about this. That is coherency issue its not cache anyway. So, that is a coherency issue right the data is not consistent data is not coherent. So, there is a coherency issue for all particle purpose this shadow register is basically a cache of your right cache of what cache of your GDT again. So, this basically leads to a cache coherency issue. So, every time you go and expand a segment you need to go and reload your DS, unless it is documented you will not know what is happening correct? You always feel that if I go and change in the you know segment descriptor table in the GDT r if I go change the limit or anything this will work, but it will not work right.

Are you able to follow any other doubts? Operating system has to check for all operating system has to check for all who are all it has it will know who were all the segments for the current process it will know, who were all the segments who were all the follows that are using it. So, it has to do it for all the ones in the case of task switching it is automatically taken care of, I will tell you how it is happening in the fourth programming

assignment post. So, after I S R we have to paging and then you have to do task switching. So, two more assignments are there two more solid assignments are there next yeah.

Student: Total number of entries in a GDT. You were saying that (Refer Time:19:18) of that it could be 256.

250 to 255 I thought yeah tell me.

Student: But since the segment selector we have 30 index for the entry in the segment descriptor for in the GDT. So, there can be 8 to 9 entries.

There can be 9 to 9 entries, but somebody has to map it no.

See this is not see when an interrupt comes it is the hardware that is mapping that to a entry in the I DT, the maximum interrupt that the hardware see who is doing this? There is an hardware which realizes that there is an there is an what do you say there is interrupt, it takes that number and then it goes and goes to a interrupt descriptor table and say start executing this. So, that hardware has to realize this number and then take that number to the I DT and index there, and that realization it can do only up to 256 entries you are getting this it is not like normal addressing.

Student: Sir I am asking for GDT.

What GDT?

Student: Global descriptor. The total the total number of descriptors in the GDT. How much they can be.

Number of descriptors in a GDT. So, this is 16 bit and in the 16 bit we are removing the. So, 16 in a twelve bit.

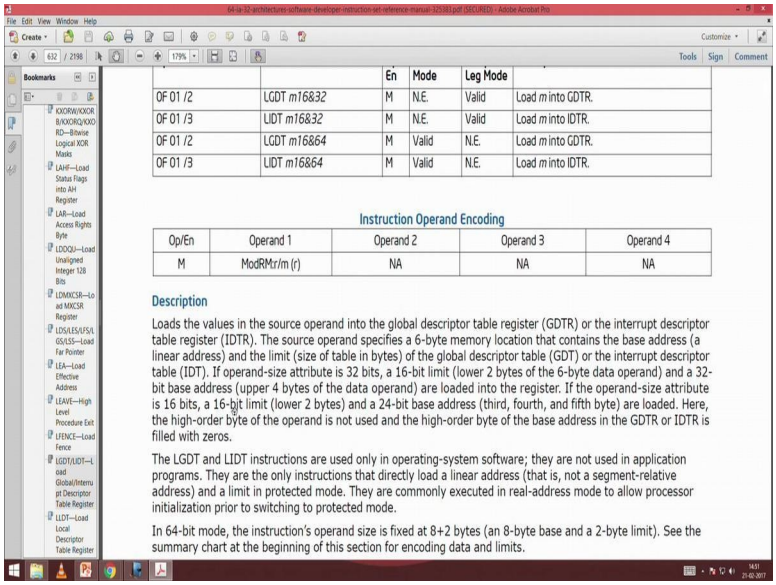
So, what is two power thirteen we can have up to 8 1 9 2 descriptors I said 255 only for the interrupt descriptor table not GDTR right. So, yeah this is 255 only for the interrupt descriptor.

Student: Ok.

Not for the GDT.

G DT I can go whatever I want. So, the only one thing that you need to confirm in the I GDT statement you basically store the base and the limit of the descriptor, when I do the LGDT. L GDT of some address and in that address I will store the limit and base of the descriptor, I do not know what is the size of this limit I think that is 68 bit no maybe 8 bits I will go to Intel manual yeah.

(Refer Slide Time: 22:07)



		En	Mode	Leg Mode	
0F 01 /2	LGDT *m16&32*	M	N.E.	Valid	Load *m* into GDTR.
0F 01 /3	LIDT *m16&32*	M	N.E.	Valid	Load *m* into IDTR.
0F 01 /2	LGDT *m16&64*	M	Valid	N.E.	Load *m* into GDTR.
0F 01 /3	LIDT *m16&64*	M	Valid	N.E.	Load *m* into IDTR.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM:r/m (r)	NA	NA	NA

Description

Loads the values in the source operand into the global descriptor table register (GDTR) or the interrupt descriptor table register (IDTR). The source operand specifies a 6-byte memory location that contains the base address (a linear address) and the limit (size of table in bytes) of the global descriptor table (GDT) or the interrupt descriptor table (IDT). If operand-size attribute is 32 bits, a 16-bit limit (lower 2 bytes of the 6-byte data operand) and a 32-bit base address (upper 4 bytes of the data operand) are loaded into the register. If the operand-size attribute is 16 bits, a 16-bit limit (lower 2 bytes) and a 24-bit base address (third, fourth, and fifth byte) are loaded. Here, the high-order byte of the operand is not used and the high-order byte of the base address in the GDTR or IDTR is filled with zeros.

The LGDT and LIDT instructions are used only in operating-system software; they are not used in application programs. They are the only instructions that directly load a linear address (that is, not a segment-relative address) and a limit in protected mode. They are commonly executed in real-address mode to allow processor initialization prior to switching to protected mode.

In 64-bit mode, the instruction's operand size is fixed at 8+2 bytes (an 8-byte base and a 2-byte limit). See the summary chart at the beginning of this section for encoding data and limits.

So, this basically yeah as you see here I am just moving the cursor my hand oh shit.

Student: What is the matter?

This has a 16 bit limit. So, it has a 16 bit limit, if operand size attribute is (Refer Time: 22:36) 16 is 6 byte memory location that contains the base address and the limit yeah, it

is a 6 byte. So, 2 bytes are available. So, this is 16 bits and the limit size a linear address and the limit that is 4 plus 2 6 bytes. So, two bytes is 16 bit. 16 bit means I can have 8 1 9 2 entries yes because I can have 13 8 bit no. So, that is a limit you can have 8 1 9 2 entries yes except that you may go mad if you have such.