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

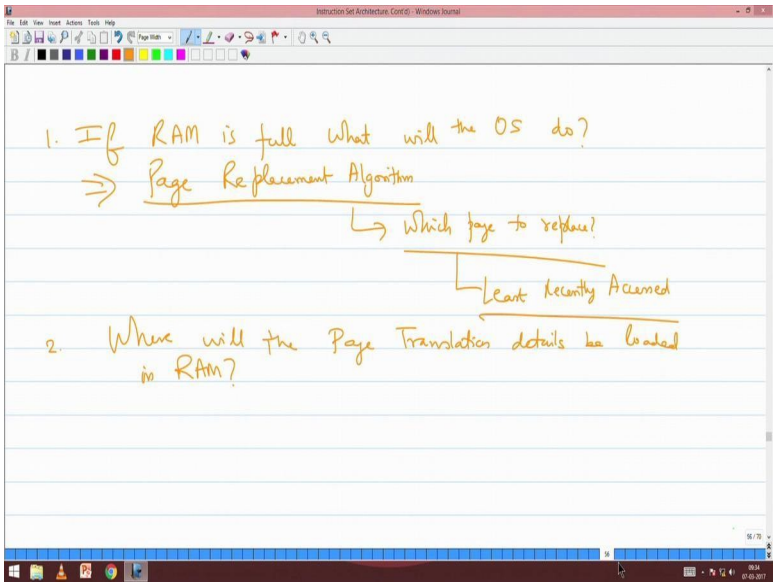
**Lecture - 27**
**Part 2**
**Page Replacement Algorithm**

Now, I want you to ask me many questions on implementation of this. So, we have been orienting this course even the third semester course and this course where in we are talking about implementation. Now, what are all the questions you have in mind? Now, I have given a very colourful slide get out some questions from this, how do you go ahead and implement this.

Student: If the RAM is full which page will?

That is a bigger question right if the RAM is full which page will do, if the RAM is full which page will I do. So, what is your answer?

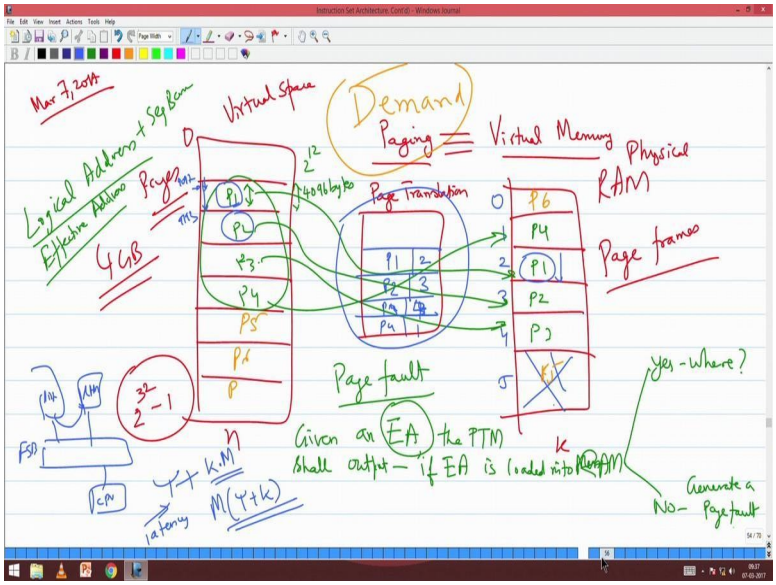
(Refer Slide Time: 00:52)



So, question number one, if RAM is full what will you do, what will you means what

will the OS, what will it do, what can it do. So, you got the question, very interesting question we will we will as your thought process goes we will orient the class, I do not want it to go by my thought process.

(Refer Slide Time: 01:22)




So, imagine there are only 6 pages here 0 to 5. Now this program has 7 pages. Now, he has loaded p 4, p 1, p 2, p 3, p 4, p 5, yeah p 6. Now, seven wants to enter where will I load it. This is very important question, very excellent question. Now, you tell me what will you do any way.

Student: You have to remove, when you remove the recent.

We have to remove something and put this. So, I have to replace something. So, we follow what we call as a page replacement algorithm correct. I have to replace something. So, I do a page replacement algorithm. So, there are lot of page replacement algorithms. Now, I will teach you some page replacement algorithms also, but the os course is expected you will learn many more page replacement algorithms, but I will give you a list of many things about page replacement algorithm. Now, what should be what should be the so I replace some

page so, so the next question is immediate obvious question is which page to replace correct that is the image. So, this is the only question

that this fellow answers. So, this which page to replace, I replace that page, I replace this page everybody everything became a paper so that is the thing. So, which page to replace?

Which page to replace there are hundreds paper hundreds of papers. Now, there is one sensible thing which they call it as least recently they call it used, but I call it accessed that makes sense. They say LRU I say LRA least recently accessed, some page I did not touch in the past at all, I neglect it, you can replace that because there is no chance that you will go and touch it. Who is telling this which property is telling you this LRA will work nicely temporal and spatial locality, temporal locality in particular. Because in the recent part I have not touched that page, so my chance that I will not touch that page is going to be less. So, this will all so you ask the programmer to write his own program see I can write a program that will go and kill this completely kill this page replace policy. We can write program.

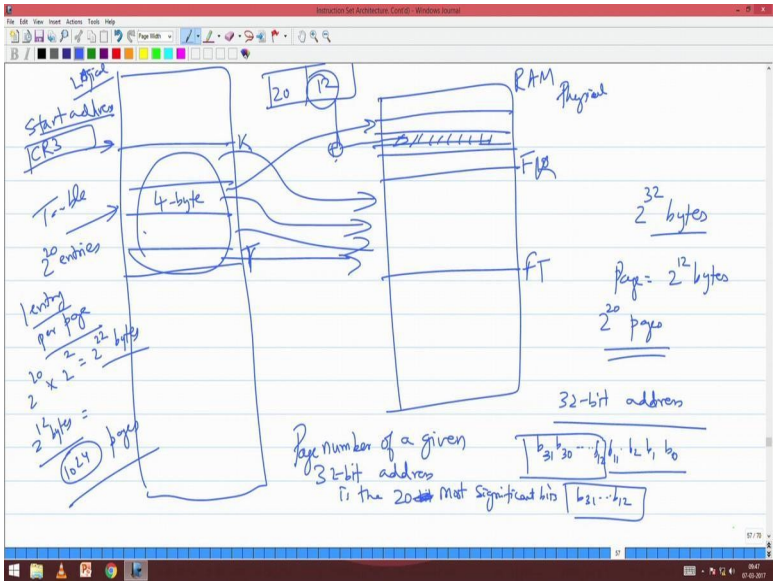
We can we will give some example we also ask you some [FL] questions also in that. But on the other hand if you write normal fellow writes a program with mind then this least recently accessed will work perfectly well, there no necessity for anymore research in this so that is the thing. Now, we will; so how do you implement least recently accessed. So, we will the same thing will happen whatever I am going to teach here why I started with memory is that whatever I am going to teach here will also be applicable in the case of cache memory. We just going to be close.

Now, we are talking about RAM right this all these things happening RAM. We will now talk about same thing is cache memory there I will tell you how to implement this LRU etcetera at the cache level. But this is a which ways to replace is something. And who will execute this page replace algorithm operating system that is a general the interrupt service routine which is the page fault handler (Refer Time: 04:58) explicit page fault handler, operating system covers everything right. So, the page fault handler will be responsible for doing this page replacement.

Next two, there is a much more fundamental question right what is a much more obvious fundamental question, hey common, obvious question. Where will this page translation

table be loaded correct? Is it going to be separate hardware or is it going to be so where was the segment table loaded in the RAM, the segment table which segmented the memory was itself in the memory. You got this right? We have done segmentation. So, if you are done your first assignment very very well understood and did it then you will appreciate the fact that the segment table which was describing the different segments in the memory itself is inside a segment. You got it? So, that is how first the operating system will fix it like that the entire page translation mechanism will be there in the RAM. So, now, we will go into step by step delineation. So, this is the question. So, the second question what we want to load is where will the page translation details be loaded in the memory where will it be loaded in RAM.

(Refer Slide Time: 06:54)




So, yes, this is the RAM. This is the virtual address space - big one. This is the logical address space. This is the physical RAM. The operating system will load the entire table here, this is basically a table right the entire table in this RAM. And then such that say let this be some K to some T. So, let there be some pages here. So, for these pages, so if I generate a address K, it will look somewhere inside this table. If I generate the address K, it will look somewhere in this table that itself will be. So, all these pages like K to T

will be permanently loaded into your physical memory. So, these things will be permanently loaded in some say some page frames say frame K to frame T.

So, I will create a space in the so when you do the assignment this becomes extremely clear, but till you do the assignment I will try and make the best to. So, the entire page translation mechanism will be loaded into the memory and we will be allocated a space in their virtual memory and that will spawn some several pages and all those pages will also be permanently loaded in your physical memory. Why should it be permanently loaded in your physical memory, otherwise I cannot do the translation. The table needed for the translation is not there in the physical memory then I am gone so that is that is this is the crux. So, this is something which none of these books teach you. So, these are something that you need to keep in mind and apply and that only comes when you do the lab in much more detail.

But please understand that my page translation mechanism that we have seen here this is basically a table like you descriptor table etcetera. And this table will have a space in logical address space, which will spawn across multiple page frames and all these pages which are there which will contain the information of this page translation should be permanently residing in the physical memory the moment I enable page. You understand this, do you follow or not? Otherwise what will happen I cannot do the page translation.

Now, let us go and see we will do a very simple thing today now I have 2 power 32 bytes in my virtual address space. Now, there are each page I have made it as 2 power 12 bytes, each page I have made it as 2 power 12 bytes. So, I have 2 power 20 pages. So, we got it. So, given a 32 bit address how will I find which page it belongs to. Given a 32 bit address let me say b 31 b 30 to b 0, b 1, b 2, b 11, b 12 and all which page does it belong to.

Student: b 31 to b 2.

b 31 to b 12 because each sorry please understand that each page is 2 power 12 bytes. So, the page number of a given address is given by the most significant 20 bits. Are you able to find out, are you able to follow? Page number of a given 32 bit address is the 20 bit 20 most significant bits namely b 31 to b 12 correct. So, I have a table this table will have 2 power 20 entries, it will have 2 power 20 entries. So, in this 2 power 20 entries each entry will so one entry per page. So, this fellow will have 2 power 20 entries one entry

per page so entry per page.

So, what will be that entry, it will tell you if at all the page is loaded in the RAM where it is loaded. So, what I do when I generate an effective address, I take the first 20 bits and I go into this table index into this table. And when I index into this table, what it will tell me if at all that particular page is loaded into the RAM where it is loaded, the address to which it is loaded. So, it will store a 4 byte address because it is a 32 bit RAM. So, I have to store a 4 byte address. So, totally how many things I need I need 2 power 20 entries into 4 byte is 2 power 2 right. So, I need 2 power 22 bytes to store this entire table.

Each page frame is what, 2 power 12 bytes, each page is 2 power 12 bytes. So, I need 1024 pages to store this are you getting this. So, what I do when I generate a 32 bit address, 32 bit logical address or 32 bit effective address, I add to the segment base etcetera I generate a 32 bit effective address in which the first 20 bits will tell me which page I belong to, the remaining 12 bits is an index into the page. So, I take the first 12 bytes first 20 bits and a index into this table, this fellow will tell if at all I am loaded into the memory it will say where I will load it for each of these page it will tell. So, I go to that address so that will be a 32 bit address.

So, let us say this page is loaded here. So, I have a 32 bit address where this is the 20 bit and 12 bits. So, this will tell me the base of where it is added, and I will take this 12 bits as an offset here and I will index. So, this is the actual address which I need to access. Is this clear? So, it is a very, very simple thing. So, I take the 20 bits I index into this table there are 2 power 20 entries. So, for each of the combination of this 20 bits, I will have an entry. I will go to that entry in that entry, it will tell if at all the page is loaded where it is loaded, it will tell me the starting address in the RAM. To that I will add this 12 bits whatever that offset is given by the first 12 bits and that will tell me exactly what that address is.

Student: Sir, why it is each entry 4 bytes?

Because this is a 32 bit address know RAM is 32 bit. So, I can essentially I have 4 gb

also fully because it is a 32 bit address space, I could have 4 gb, now I have 2 gb.

Student: Sir.

Yes.

Student: You mentioned that the table cannot be there in the test should be on the RAM.

Table should be wait one minute the table should be in the disk and all also in the RAM.

Student: Why should it be load it in the RAM?

Because when I want to access the architecture can see only the RAM it cannot see the disk. When I want to do a translation right see you have an instruction when you are executing an instruction at least three times you are touching memory to fetch the instruction, to fetch the data and store back the results. So, in the real worst case you will be touching the memory three times; every time when I touch the whatever the CPU is generating is only an affective address that affective address has to get translated into a physical address.

So, three times when I go and access a disk for every instruction execution, you go mad. First it will becomes extremely slow, second thing is accessing a disk you have to execute a program to access a disk which is called a disk driver. So, as a CPU I cannot access some byte in the disk directly I have to go to a devise driver of the disk which will know how the data is organised in the disk and it will fetch for me and that disk that devise driver or the disk driver itself is a program. So, for me to execute one instruction if all these disk table is there in the disk, I have to know how I am going to translate. So, at least for every time when I am accessing and when I am executing an instruction, three addresses I need to translate in the worst case. For just translating these three things, if I am going to execute that devise driver program three times and go to that disk and get it then it is going to become extremely complex.

Are you able to appreciate that? So, my entire page translation has to be loaded into the RAM and I will look at the RAM and basically do the translation I can access the RAM

as a CPU without executing another program, but I cannot access the disk as it is CPU without executing the devise driver right. So, to access the RAM from the CPU, I need not execute another instruction; the same instruction itself can go and fetch the data from the RAM, but for me to access data in the disk, I have to execute the another set of instructions to get it. So, that is why when I want to execute something that has to be available in the RAM, it cannot be from the disk I cannot directly take and execute. Of course, you can do some raw mapping and all those are all crazy things that will kill your performance. Are you able to follow?

Student: Yes sir.

Good, now just finishing this.

I have this in the RAM. So, I generate. So, as a CPU I generate a 32 bit address, I take that 20 bit index into this table. So, somewhere some tables should store these start address of this. So, in the Intel architecture there is a register control register called CR 3 right there is a control register called CR 3 that CR 3 will load these start address of your memory are you able to follow? So, in that CR 3, so what I do when I generate a logical address of affective address of the 32 bits I take the first 20 bits and I index into the table starting at the address given by CR 3. So, I come to that entry, there I find out if it is loaded then I get that 4 byte address. So, I go to that address I add these 12 bits which is the offset and that will give me exactly in the RAM which memory location any times. To access these table this table cannot be accessed if it is in the disk. So, this entire table that is the set of 1024 pages that 1024 pages should be available on the memory and the start of that table will be given by CR 3.

So, when I generate an affective address I go to the CR 3 I get the index is 20 bits into that table and I access the RAM to get that 4 byte, that 4 by I add that 12 bits offset and that I again to the RAM and fetch the data. You are getting this? So, this is the overall. So, I said if the page is loaded then I am taking 4 bits. How will I know whether the page is loaded or not that is your question right think [FL] a bit. You have already done something similar in segmentation right. A segment selector there is some last three bits

right what are those three bits used for some privilege level, low ldt, gdt right you do not

need to specify those last three bits are anyways zero, and you are using it for some other purpose like this can I use something here that is a clue?

So, we sign off now with saying that there is a base address CR 3, the entire 1024 pages are available in the memory and then if the page is loaded it will give me a 4 byte address to which I will add the 12 bits to find the exact location. Now, what has happened now memory access RAM itself is a killer, now to first time when I have to go to a table, from that table I have to get some address, and then from that address I have to add something again I have to go and access. So, one access has now become two access. More interestingly, we saw right in the case of segmentation what has happened, so let us say so when I generate a logical address it goes to the gdt gets that segment base add it it generates affective address. Now, this affective address you take the 20 bits go somewhere get a base get the base of the table add the 12 bits then you go and access.

So, by time I see that whatever I want to access, I have gone through many multiple memory access. So, to solve all these problem, we are the notion of a shadow register. So, you all understood what is shadow register in segmentation correct. So, the shadow register reduce that access and because of shadow register, we had some issues. What was the issue? If I go and change something in the g d t it will not reflect again I have to move. So, now like shadow register we will also have something here right page translation because so many things so some of the architecture details.

If you do not have those shadow register or anything the simple one memory access which itself we consider as a monkey now becomes free memory access. So, one memory I can say it became mad, second memory got hurt, third memory it got drunk or bit by a scorpion. So, this monkey becomes extremely crazy. Now, two things that we need to keep in mind from your implementation point of view, we need to say should I have all these 1024 pages, can I do a demand of the demand paging something like that? What we what do we talked sort of demand paging, we said demand paging is that whatever I need I want to load it whenever I need it.

So, can I do something more on reducing this number of 1024 pages that is another the important question is how do I store information whether a pages is loaded or not then

we will see how these things are managed. So, we will get it. So, at the end of this lab 3, you will be setting up a page table, you will be accessing one by one, you will generate a page fault you will do several things as a part of that third exercise.