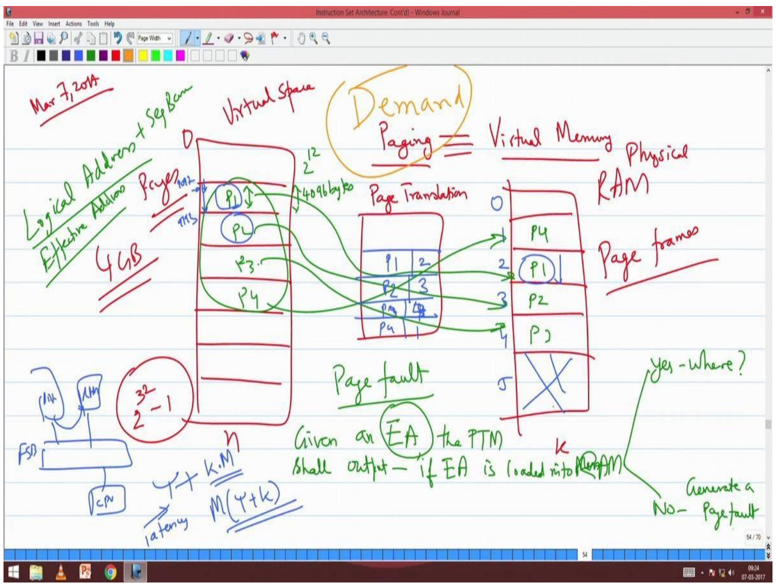
Computer Organization and Architecture
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 27
Part 1
Locality of Reference, Demand paging

(Refer Slide Time: 00:28)



So, we will continue with paging. So, last class we stopped with the understanding that there is a logical address space which can go from say in the Intel architecture or any 32 bit architecture can go from 0 to 2 power 32 minus 1. So, this is 4 GB address space and this is what you offer to a program to the program you will say have this virtual space. Un this you do all your segmentation everything right you create your code segment, you create your data segment, stack segment, have multiple segments put your detail everything you do here.

Now, subsequently there is going to be a layer. So, this is a virtual address space; for the time being, we assume that it is going to be on the disk it is going to be in the disk how it is organised in the disk the operating system course will teach you. So, somewhere it is on the disk this entire 0 to 4 GB can be in contiguous location or it can be in distributed location, but those things the operating system will teach you how this 4 GB is organized on the disk.

For the purpose of computer organisation architecture course, we will assume that it is going to be available in the disk and when we want we can go and access it. Now, then there is a this is the real RAM slightly less in size. So, what happens is that this entire virtual space is mapped on to this RAM as and when I move some instruction or data from the virtual space to the physical space. So, we call this as a physical RAM. So, what we do is we split this virtual space into pages and then we split this virtual memory the physical memory into what we call as page frames like how a photo will fit into a photo frame page will fit into a page frame. And then what happens there is a page translation mechanism.

Please note that there can be n pages here, there will be k pages here. All the n cannot be loaded here. So, we load only from pages that are necessary and that is where we came out with the statements saying that for me to successfully complete the execution of a program, if I ensure that at any point of time the next instruction to be executed and the data necessary for the next instruction to be executed, if it is present in the RAM. And this we ensure through the entire life time of the process from the start till it is getting killed. So, if that is ensured then we can basically execute the program.

So, if my program is say spawning over say page 2, page 2, page 3 and page 4. So, four pages this is my entire program not all these four pages need to be here you understand it can. So, I start executing page one. Now, I will load page one alone here then I will start executing page 2, I will load page 2 here. Then when I start executing page 3, I will execute I will move page 3 here and then I will also move page 4 here it can be mapped. So, page 1 is mapped here page 2 is mapped here page 3 is mapped here page 4 can be mapped anywhere. So, when I generate an so what is this page translation mechanism when I generate a logical address then I go to this page translation mechanism and ask [FL] this logical address is it loaded into the memory or not?

The first question I ask this page translation mechanism is this logical address loaded into the memory or not? If it is loaded into the memory, this page translation mechanism should tell me where it is loaded in the memory. If it is not loaded in the memory then the page translation mechanism should give me a page fault, it is called a page fault. Saying, hey it is not loaded then I will go the operating system has to basically go and load that into memory. Are you getting this?

So, let me just write what I have told, given a logical address I will call it as LA given a logical address the page translation mechanism I will call it as PTM shall output if LA is loaded into memory into RAM. Here yes means where, it should tell me where; no means generate a page fault. Already you have done interrupts yes or no? Interrupt service routine you remember? Very good, now page fault is another interrupt if you look into the interrupt table page fault will be interrupt number 11 or 12 something of that sort. So, we will lab assignment we will do a page fault.

So, this page translation mechanism is handled by the hardware; it is set up by the operating system handled by the hardware. What do you mean by handling? This entire whatever I have written in green is basically done by the hardware. So, once logical address is generated, the hardware takes care of taking that logical address going to the page translation mechanism asking it whether it is already loaded into the RAM, if yes it will tell you where it is; if no, it will tell you it will generate a page fault.

The moment I generate a page fault please note that it is going to be a trap. So, essentially there is so you go into that interrupt descriptor table, corresponding to page fault 11 or 12 which ever this page fault is you start executing that interrupt service routine. That interrupt service routine is called the page fault handler. And that interrupt service routine is a part of your operating system. Are you able to follow all these things?

Now, each of these page in the Intel architecture is 4096 bytes, so that is 2 power 12 bytes. So, I will start. So, let me I will start executing let me put that in. I will start executing when I reach this point I will not find this p 1, because I will generate some address say some thousand 4097 I will. The moment I generate 4097 this fellow say no, no it is not there. So, immediately page fault will happen. And then what it will do it will load the entire 4096 into one of these vacant frames here whichever frame is vacant it will go and clear it. Who will load it, the page fault handler will do it right, the page fault handler which is a part of your operating system will take this p 1 from the disk and [FL] load into this memory and now say execute. So, it will start executing.

So, what it means I start executing like this means I will start executing like this here. When I start executing this instruction means what I will only take the instruction and memory from the physical space. So, I will start executing this program loaded here. The moment I reach this boundary right then I will start generating some address now 4096 9000 some 192, so 90193 will come there. Now, it says no it is not there then immediately what I will do again there will be a page fault, then it will load this p 2 into one of these available frames here may be here or here whatever available, and it will start executing. So, when it is loading, it will also say let us call this frames as 0, 1, 2, 3, 4, 5 like that.

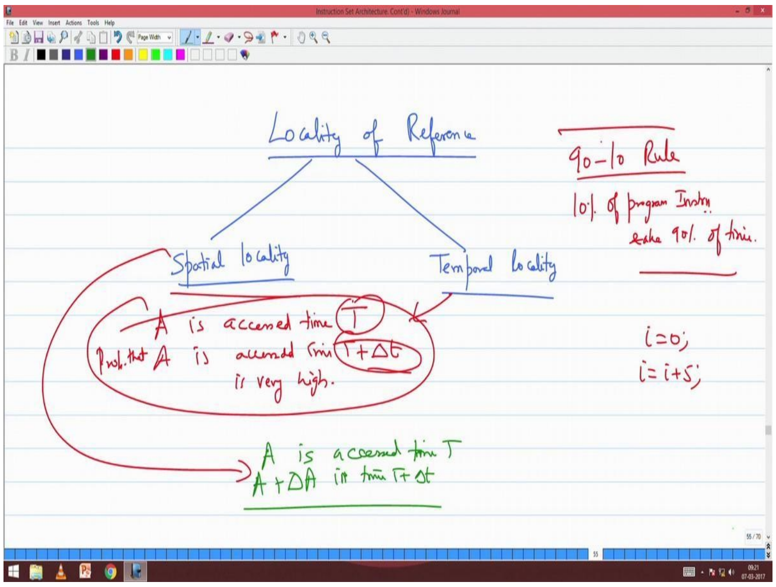
So, when I am loading I will just say p 1 is loaded at 2. When I loaded this I will say p 2 is loaded at 3; and similarly when I come p 4 is loaded at 4 then assume these are all full. Then I can say p 3 is loaded at 3, sorry p 3 is loaded at 4 then I will go and say p 4 is loaded at 1. So, so I will also make a mark here. So, what happens is first I generate 4097 to start with, it is not there I will load the entire p 1 here then 4098, 99 everything will be available here. Then I go back when I cross that 90193 again I come here again I will load then the subsequent 4096 bytes are available here.

So, why it so these type of a movement is called block moment, why do I move in block, already I told you when I want to access disk there is a common bus front side bus from side bus. This is called the front side, but there is a common bus my CPU has to request this bus there is bus master hey give me this bus I want to transfer data over the bus because your disk and the memory the RAM are both connected to this bus. So, when I want to request this bus, hey give me this access. So, the bus master will take some amount of time to give you that access. So, there is a latency which we call it as tau involved in getting access to the RAM in getting access to the disk. After that I am going to transfer data from disk to the memory I can go and transfer data from disk to the memory.

But first the CPU when it is desires to basically take some data from disk to RAM who will be deciding here the interrupt service routine which is executing on the CPU it is part of the operating system will be desiring to move some data from disk to RAM. Then it has to wait for some time that is called the latency. So, then it has to transfer. So, there is so byte if it is going to take some K units of time and I am transferring M bytes K into m would be the time tie, but if I am going to do byte by byte oh this fellow wants 4096 now I will fetch only 4096 then I will fetch only 4097. Then this multiplicative factor M this will become M into tau plus K and that will be extremely prohibitively time consuming. Because now it is a very significant amount of time, you understand this. So, that is why the moment I get bus instead of getting one I will get some 4096 bytes rather than just getting one or two or three bytes.

If I just one byte and then again I repeatedly I keep on doing every time I will be spending the tau time and that is going to be extremely prohibitively time consuming. So, this is one side of the story where we are talking from an architecture perspective from a performance perspective.

(Refer Slide Time: 12:08)


Instruction Set Architecture, Cont'd) - Windows Journal
File Edit View Insert Actions Tools Help
Page Width

Locality of Reference

Spatial locality

Temporal locality

A is accessed time T
Prob. that A is accessed time T+Δt
is very high.

A is accessed time T
A+ΔA in time T+Δt

90-10 Rule
10% of program Instrn
take 90% of time.

i=0;
i=i+5;

55/70
55
09:21
07-03-2017

The other side of the story is every program now follow something called locality of reference. We will deal about this in more detail when we go to cache, but is is also true in the case of memory. So, in the case of virtual memory what do you mean by locality of reference. There are two locality of reference which we call which we basically reported in the literature and which we see in practice spatial locality and temporal locality. Let us go back to the original statement of 90-10 rule where in I say 10 percent of the program instructions take 90 percent of time, 10 percent of program instructions take 90 percent of time. This is a well established empirical experimental rule right if you have a large program you will have one thug for loop or while loop which will keep on executing which will be just 10 of the instructions; all the other things would be initialisation, malloc all these things will do once so that will be 90 percent of instruction. So, if we have 100 instructions there will be only 10 instructions which will take 90 percent of the time.

So, what does it imply? So, when this program is loaded into the memory when this program is loaded into the I mean the corresponding data is loaded into the memory there is chance that the same address is executed fetched again and again and again and again and again and again. There is also chance that the large probability that the same data is accessed again and again. If you have a for loop the variables there will be accessed again and again, are you able to get that. So, the same address will be accessed multiple number of times and that is what we call as spatial locality; spatial means in the address space.

So, if an address A is accessed at time T accessed means read or return read or written at time T then the probability that A is accessed at time delta T plus delta T is very high, this is what spatial locality is essentially sorry if A is accessed at time T the probability that A is accessed at time T plus delta T is very high this is what we name by temporal locality. I have one address I access it at time T within the next few cycles the probability that I will access the same address is quite high. For example, I want to do you some i equal to i plus 5 or something like that I will initialise i equal to 0 then I am immediately go and do i equal to i plus 1. So, so the way we think we start using a variable we initialise it just before using that variable; that means, your locality of reference this is what we mean by locality of reference.

So, temporal locality is in terms of time. So, if I access an address A at time T, the chance that I will access the same address A at time T plus delta T is very, very low. What is spatial locality say the spatial locality essentially says if a access address at time T the probability that I will access A plus delta A in time T plus delta T is very high that is what I call by spatial locality. If I access thousand the problem now the probability that in the next few next seconds, I will access 1001 which is going to be extremely high. So, what does this do you get the difference between spatial and temporal locality. Temporal locality is in terms of time spatial locality is in terms of the address space as you see in the red whatever I marked in red in the screen it is temporal locality. Because I am accessing the same address am not talking about locality in the address I am talk accessing the same address, but in neighbouring time intervals. But when you come to spatial locality I am looking at the neighbourhood of that particular address in the next few cycle.

So, the probability that A is accessed at time if A is accessed at time T, probability that A is accessed at time T plus delta T is very high this is what temporal locality teaches us. If A is accessed at time the probability that A plus delta A will be accessed in time T plus delta T this is what spatial locality teaches us. So, these two locality of references basically you know supports our point that I will bring 4096 bytes and instruction because the probability that I will be using lot of those 4096 bytes in the next few cycles is very very high because of what because of the locality of the fetch got it?

Now, the question is what are the questions? So, please try and ask me some questions. Now so idea is sort of clear. So, let me just make it a little more formal from an Intel x86 perspective because Intel is one architecture where this paging has been explicitly done in a very interesting way. So, now we have a logical address what is a logical address the logical address is that which generated by the compiler. So, the compiler says some address K then what happens when you actually execute it is getting added to the segment base.

So, the logical address plus segment base will give you what you call as effective address. This effective address is what is fed into the page translation mechanism and that will give you that will answer this thing that I am marked in green here. If this effective address instead of LA, I will call it as EA please not here I am circling it. This given an effective address page translation mechanism shall output if that effective address is loaded into RAM yes means where, no means it will generate page fault. So, this is what happens.

So, what you do, when you are generating the program you have a logical address space in that logical address space you load that program your compiler sees only the logical address space. So, the architecture and operating system together makes a you know together forms an alliance, and basically tells the program saying hey you take compiler it says take 4 GB, 4 GB is available to you. You put all your segmentation whatever you want there and finally, you generate something like an affective address which you want to access in the virtual space, give it to the paging mechanism, I will tell you whether it is loaded into the RAM. So, yes, where it is loaded; if no I will tell I will create a page fault and again operating system can take the necessary actions, are you able to get this very clearly. So, this is also called demand paging.

Why we call it demand paging. What is on demand radio, what happens in your what d 3 is equal to d c plus what is this you do not know? Yeah I know it. So, on demand video, so like. So, when I want the page I take it. So, I do not go and load anything first I wanted p 1, I went and loaded it; I want p 2 so I went and loaded it; I want p 3, so I went and loaded it. So, whenever I need the page, I load it into the memory. So, this is why this is called as demand paging. I demand the operating system, hey I need it give it operating system goes and fetches for me. So, this is why this particular mechanism is called demand paging.