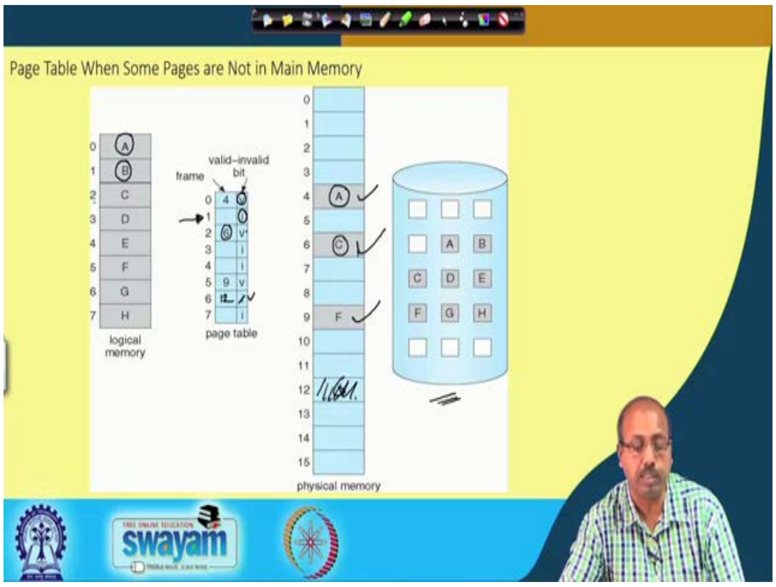
Operating System Fundamentals

Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 50
Virtual Memory (Contd.)

(Refer Slide Time: 00:30)



Page Table When Some Pages are Not in Main Memory

So next we will be looking into some example in which we have got a set of pages, and the page table shows here that we have got these pages this logical memory. So it is divided into number of pages 0 to 7, 8 pages. Now some of these pages so, we have got this page table. So, in this page table so, the corresponding pages some of them are valid some of them are invalid.

Like say page number 0, it has been loaded into memory frame 4. So, this is 4 so, this is valid. So, it is there. So, if the content of this particular page is A. So, it is available here then a frame number sub page number 1 so, it is not yet loaded into main memory. So, it is. So, page number 1. So, the corresponding frame number is not written here and it has been marked that it is invalid.

So if the processor generates an address which is referring to say A to access this B so, it will be generating a page fault. Because when it comes to this particular page table entry, it will find that the corresponding bit is invalid. And if it is invalid then a page fault will be generated.

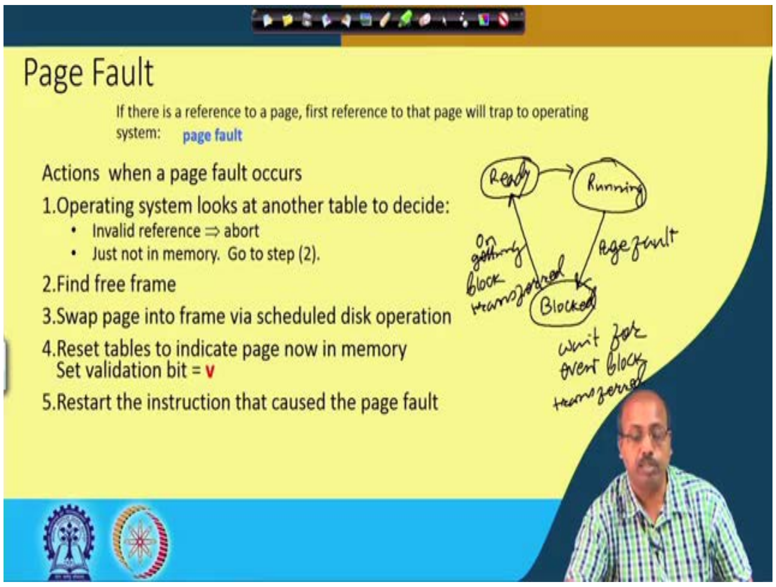
Then the again this page number 2 so the corresponding frame number given is 6 and it is available here. So, and the corresponding bit is set to valid. So this way, in the this is the secondary storage or disk. In the secondary storage we have got all these pages ok. So, A B C D E F G S all these pages are here, but out of that only a few pages have been loaded into main memory for execution.

So, if we take a snapshot of the system may be at present the system is using these pages A C and F so they are present in the memory. And if while executing this the program code, maybe it will generate some reference to some other pages and if that reference occurs then that will give rise to a page fault, because the corresponding in a bit is invalid so, it will give rise to a page fault.

And when a page fault will occur so, it will load the corresponding page from this disk into this one. Like for example if it generates a reference for page G, then the system has to figure out one particular free page suppose these are free frames, suppose this frame is free and it will load the page G here. And accordingly it will update this particular entry number 6 as 12 and make it valid. So, that way the modification will be done.

So, page table when some pages are not in main memory. So it will be the situation will be like this and then if some page fault occurs then it has to be handled and it is handled in this fashion.

(Refer Slide Time: 03:19)



Page Fault

If there is a reference to a page, first reference to that page will trap to operating system: page fault

Actions when a page fault occurs

1. Operating system looks at another table to decide:
   - Invalid reference ⇒ abort
   - Just not in memory. Go to step (2).
2. Find free frame
3. Swap page into frame via scheduled disk operation
4. Reset tables to indicate page now in memory
   Set validation bit = v
5. Restart the instruction that caused the page fault

$$\begin{tikzcd}
\text{Ready} \arrow[r] & \text{Running} \arrow[dl, "\text{Page fault}"] \\
\text{Blocked} \arrow[u, "\text{On getting block transferred}"] &
\end{tikzcd}$$

wait for event block transferred

So, what is a page fault? So if there is a reference to a page and then the first reference to that page will trap to the operating system, it is called page fault. So, a page consists of a particular number of bytes typically we have seen that the pages are of size say 1 k, 2 k, 4 k, 8 k like that.

So, when the first reference for that page comes and that page is not there in the memory then the page fault occurs, and then what the system does is that it transfers the page from the secondary storage to the main memory. So and that transfer is not for a single byte; so, it is for the entire page, because from secondary storage we access in terms of blocks not in terms of bytes.

So, one block is equal to this memory frame size. So, that way the 1 block will be accessed and this entire content will be brought into the main memory. So, the page fault is occurring when the first reference to the page occurs that gives a trap to the operating system and that is a page fault. Now what the operating system is going to do when a page fault has occurred? So, operating system it will look for are at another table to decide the invalid reference. So if the reference is invalid, then it will be about it because it is not a valid reference. And if it is that the reference is within the address space of the process, but the page is just not there in the memory it is there in the secondary storage.

In the in that case it has to go to step 2. In step 2 the first thing that it does is that it figures out a free frame. Like here you see that as I was telling if it is there is a reference to G, then first thing that it has to do is to find out a free frame in this physical memory.

So, after that frame has been found; after the free frame has been found then we will copy this second copy this page from the secondary storage into this particular frame. So, swap page into frame via scheduled disk operation. Now one thing we must understand that when this thing occurs so, CPU does not go into doing the copy from the secondary storage into primary memory. Because there will be a huge speed gap between the operation of the CPU and the operation of the disk.

So, what is done instead is that, CPU will tell the disk scheduler that this particular transfer has to be done and that way the process gets blocked ok. So, if you remember that process state transition diagram. So it was in the running state, it was in a running state and a page fault has occurred. So when this page fault occurs, the process goes into the blocked state so on page fault, it goes into a blocked state and in this blocked state the process will be waiting.

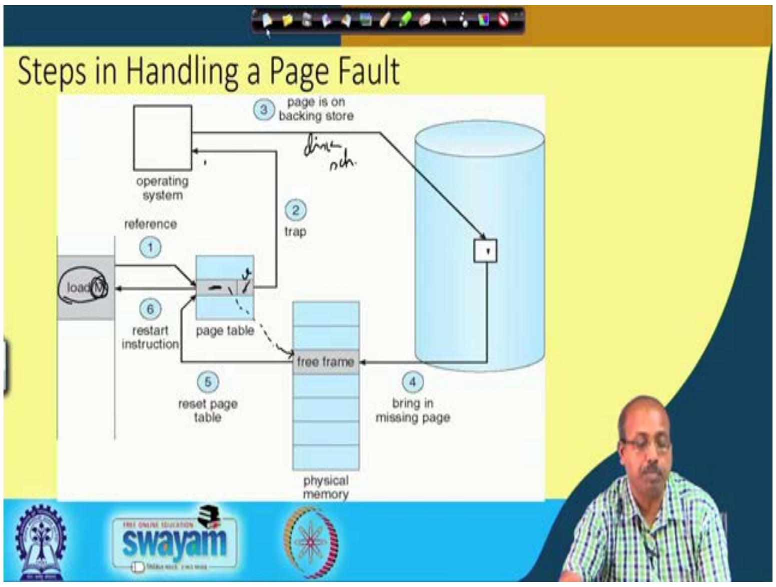
And this is actually waiting for wait for event that block has been transferred it will be waiting for this particular event. Now, when it is waiting for that, so it has told that their waste disk scheduler that I need this particular block and then it goes into a blocked state. So, after some time that disk scheduler will schedule that disk transfer from the secondary storage to the main memory and the when that transfer is over, then this process will be informed that that transfer is over, now this process will make a transition from running to ready on completion of the event. So, on getting the block transferred. On getting the block transferred so, it will be going to the ready state.

Again after some time the CPU scheduler will schedule the process and it will go to the running state. Now, the process when it tries to execute that particular instruction it finds that, the corresponding page is there in the main memory so, it will continue executing that. So, this is the whole thing that when the page is not found, then it will be; then the system will be will go to find a free frame and it will be swapping the page via that schedule disk operation.

And from this step 2 onwards so, you can understand that the process for which this thing has happened the page fault has happened, the process is put into a blocked state and the CPU will be doing something else. And when the data transfer will be over then the process will be informed and then it will be coming back. So, this reset tables to indicate page now in memory and said validation bit to be equal to v and then restart the instruction that caused the page fault ok.

So, at this point when the process it is not written here explicitly that the process when it goes from blocked to ready state, the next time it is getting scheduled. So it will start from that point the instruction that it had left before occurrence of this page fault. So this is how this page fault processing is done.

(Refer Slide Time: 09:01)



Steps in Handling a Page Fault

So, this is a diagrammatic view of how this page fault is handled. Suppose I have got an instruction that is being executed which is load M ok. So, it finds that this load M. So, this M is a memory address and this particular memory address is; this is not there the where the location at which we want to write that particular location is not there in the main memory

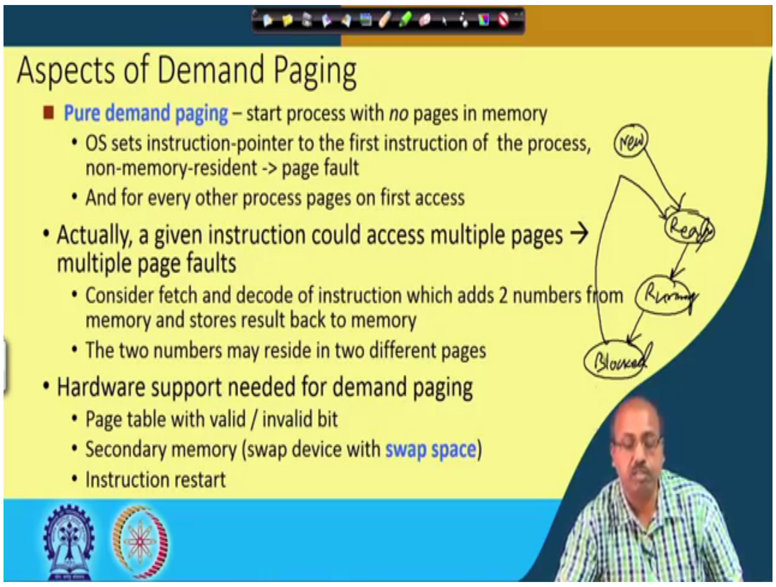
So, it tries to access so and it finds that the corresponding page number corresponding to this address M so, that is a invalid ok. So, this gives a trap to the operating system that ok, this particular page is missing. So, that page has to be loaded for this particular process this page is missing so, that has to be loaded.

So, that is a trap to the operating system and in term the operating system will find the page on the backing store. So, this is done by the disk scheduler. So, where exactly it is loaded etcetera so, that is found by the disk scheduler. So, it will find that this M is located in this particular block and then this page has to be brought into the main memory. Say assuming that there is a free frame available so, it will find, it will copy this particular block into this free frame and after that this page table will be reset.

So, this will be made to point this particular entry will be made to point 2, this particular frame it will now point to this particular frame and this I bit will be made equal to v it will be made equal to valid and then the instruction will be restarted.

So, this load M instruction will execute now and now the page is there so, naturally it will be executing it properly without any problem. So these are the steps in handling this page fault by the operating system.

(Refer Slide Time: 11:08)



Aspects of Demand Paging

- **Pure demand paging** – start process with *no* pages in memory
  - OS sets instruction-pointer to the first instruction of the process, non-memory-resident -> page fault
  - And for every other process pages on first access
- Actually, a given instruction could access multiple pages → multiple page faults
  - Consider fetch and decode of instruction which adds 2 numbers from memory and stores result back to memory
  - The two numbers may reside in two different pages
- Hardware support needed for demand paging
  - Page table with valid / invalid bit
  - Secondary memory (swap device with swap space)
  - Instruction restart

Now there are different aspects of this demand paging; one of them is called pure demand paging. So pure demand paging it says that does to start process with no pages in memory. So, initially we do not load any page of the memory into the process. So, into the memory; any page of the process into the memory.

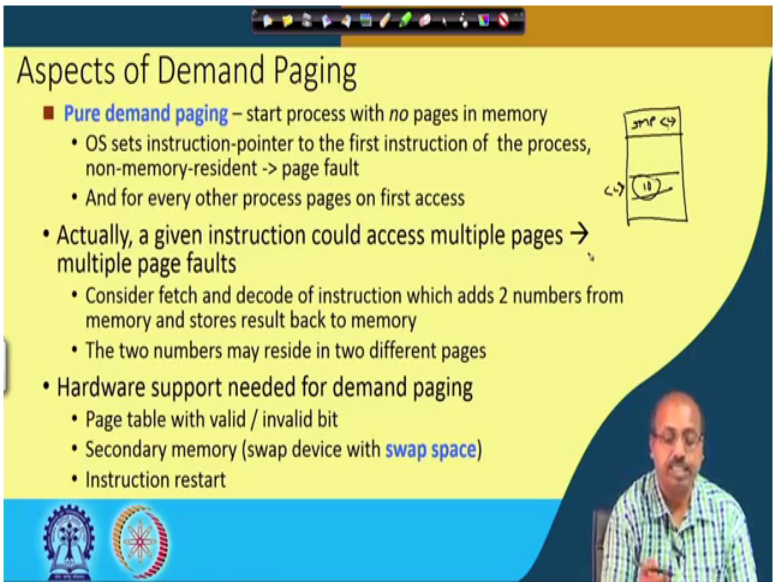
So how is how is it happening? Like a process has made a transition from a process has been given for execution so, it is in the new state. So, the system just notes down the state of the process, it does not do any copy of these pages from disk into main memory and the process now makes a transition to the ready state.

Now, after sometime the process will be scheduled. So, it will come to the running state; and when it comes to the running state at the very first instruction it will generate a page fault. And when it generates the page fault, then only this process the corresponding page will be loaded. So, this process will go to the blocked state with an information to the OS that it needs the first page the corresponding block number etcetera, and OS will load the corresponding the first page will be loaded and then it will be informed then it will again go back to the ready state.

So, this is a pure demand paging whether it is advisable or not that is a different question. That is without loading any page should we start the process at all for execution. So, those questions are there, but in a pure demand paging form. So, that none of the pages are loaded.

So, we start process with no pages in memory, OS sets instruction pointer to the first instruction of the process which is so, non memory resident. So, naturally this page is not there in the memory. So, it will generate a page fault. For every other process; pages on first axis so, for every other processor pages also. So, it will process the page only when the first access occurs. Again the first page has been loaded so, that way starts executing

(Refer Slide Time: 13:18)



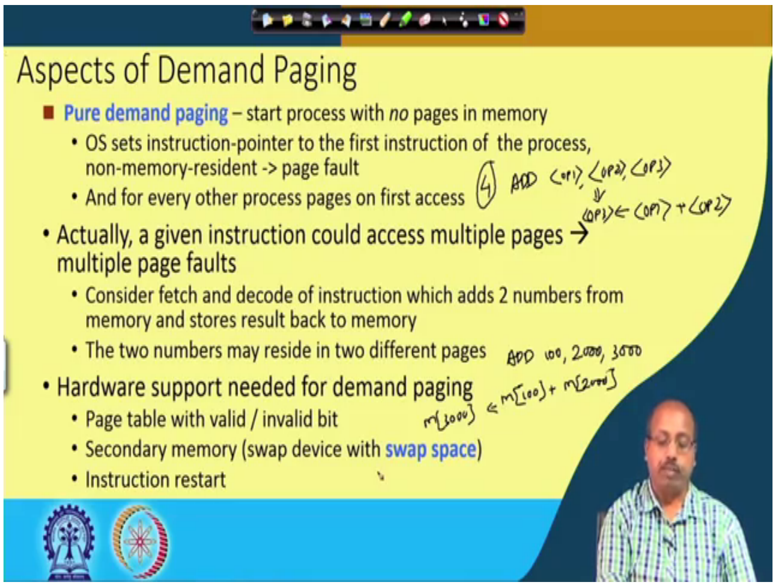
Aspects of Demand Paging

- Pure demand paging – start process with *no* pages in memory
  - OS sets instruction-pointer to the first instruction of the process, non-memory-resident -> page fault
  - And for every other process pages on first access
- Actually, a given instruction could access multiple pages → multiple page faults
  - Consider fetch and decode of instruction which adds 2 numbers from memory and stores result back to memory
  - The two numbers may reside in two different pages
- Hardware support needed for demand paging
  - Page table with valid / invalid bit
  - Secondary memory (swap device with swap space)
  - Instruction restart

Now suppose the first page has if this is the. So, if suppose that this is the first page, here there is a jump instruction and this jump is to some address L, which L is actually in this particular page. So, this is say page number 10 ok; so, it is in page number 10. So, at this point it will generate another page fault and as a result.

So, this is the first reference to page 10 and then this page 10 will be loaded into some memory frame and then it will continue. So, for every other press process pages that page will be brought into the memory on the first axis. Actually a given instruction could access multiple pages. So, naturally there can be multiple page faults.

(Refer Slide Time: 14:06)



Aspects of Demand Paging

- **Pure demand paging** – start process with *no* pages in memory
  - OS sets instruction-pointer to the first instruction of the process, non-memory-resident -> page fault
  - And for every other process pages on first access
- Actually, a given instruction could access multiple pages → multiple page faults
  - Consider fetch and decode of instruction which adds 2 numbers from memory and stores result back to memory
  - The two numbers may reside in two different pages
- Hardware support needed for demand paging
  - Page table with valid / invalid bit
  - Secondary memory (swap device with **swap space**)
  - Instruction restart

$\textcircled{4}$ ADD <OP1>, <OP2>, <OP3>

$\Downarrow$

<OP3> $\leftarrow$ <OP1> + <OP2>

ADD 100, 2000, 3000

$m[3000] \leftarrow m[100] + m[2000]$

Like a typical example is like this, suppose I have got an instruction which is say addition instruction, ADD op 1 operand 1, operand 2, operand 3. So, this is a most generic form of this instruction and maybe the meaning is that operand 3 gets the value of operand 1 plus operand 2. Now, in the most generalized form of this operand so, this all these operands they can be memory addresses.

So, I can say like ADD 100, 2000, 3000, meaning that the content of memory location 100 and content of memory location 100 and 2000 so, they will be added and the value will be stored in memory location 3000. So, this may be the meaning of the statement.

Now, you see that since they are all three different memory addresses. So, they can give rise to a number of so, all of them can potentially give rise to page faults because none of the pages are loaded so, it can give rise to 3 page fault. So, as a result so, each instruction can give a number of page faults. So, if I consider the fetch and decode instruction which ADDs 2 numbers from memory and stores the result back into memory, now these two numbers may reside in two different pages and the address may be there in another and the destination may be there on a third different page.

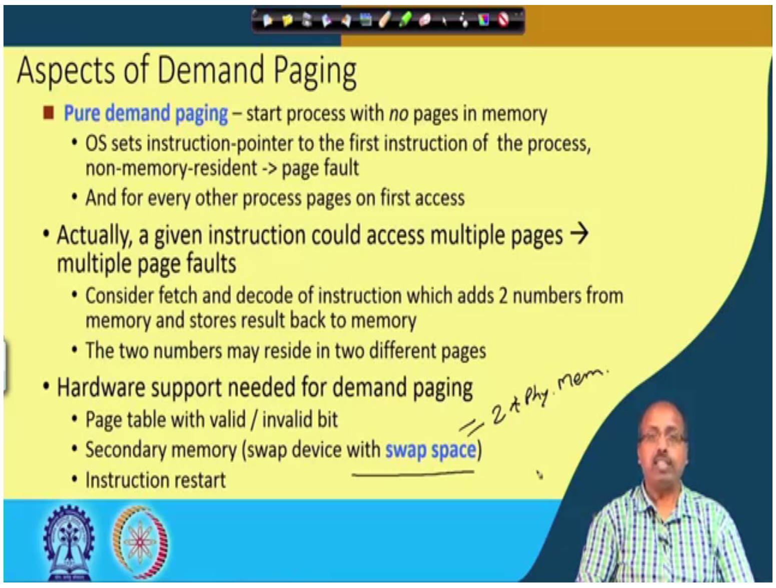
So, as a result there can be three number of page faults that can occur just to do this addition. And of course, one more page fault can occur in doing the fetch is this instruction access itself may be a page fault ok. So, that way total number of page faults for this instruction may be equal to 4; at most it is equal to 4. So, that is a single instruction execution. So, it can give rise to 4 number of page faults.

So, definitely so, this is going to going to be a huge overhead over the system and it's the hardware support must be there to handle this intelligently. So, hardware support is needed for demand paging. So, these are the things that we will need like to have like this page table should have this valid invalid bits, then the secondary memory should be provided which he called the swap device or swap space.

So, apart from the disk that stores the programs or the processes their compiled versions, we also need to have some space for holding these processes under execution their image. So, many times we will be writing back some memory frames to make space for the currently needed pages of a process. So, these frames that are removed from this main memory so, they need to be stored in some secondary space so, that is know that has they have to be copied onto swap space.

So, this way that swap space is coming as a very important issue and if the size of the swap space is small then definitely that difficulty is that, you cannot copy a large number of memory frames there. So, your process size will be restricted by the size of the swap space. On the other hand if the swap space is very very large then definitely so, there that will eat up a good amount of disk storage that also may not be desirable.

(Refer Slide Time: 17:51)



Aspects of Demand Paging

- **Pure demand paging** – start process with *no* pages in memory
  - OS sets instruction-pointer to the first instruction of the process, non-memory-resident -> page fault
  - And for every other process pages on first access
- Actually, a given instruction could access multiple pages → multiple page faults
  - Consider fetch and decode of instruction which adds 2 numbers from memory and stores result back to memory
  - The two numbers may reside in two different pages
- Hardware support needed for demand paging
  - Page table with valid / invalid bit
  - Secondary memory (swap device with swap space) $= 2 \times$ Phy. mem.
  - Instruction restart

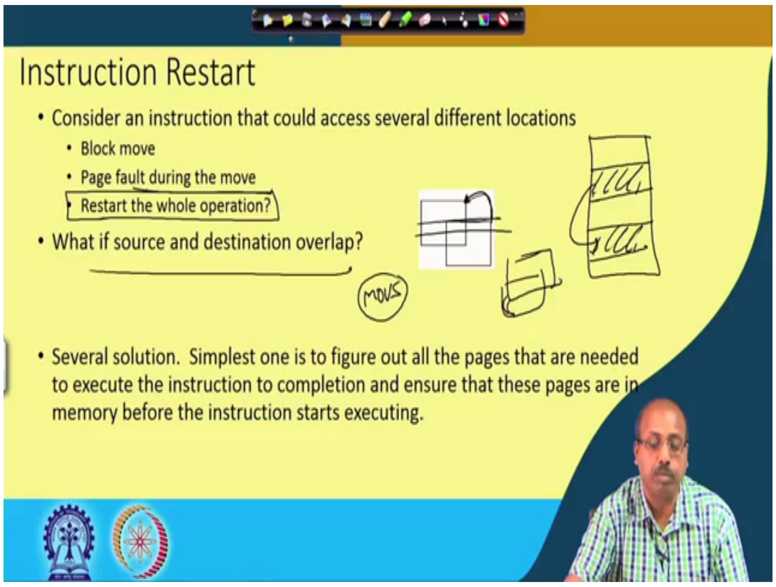
So, we have to so, many operating systems. So, they say to have this swap space of different sizes. A typical guideline is the swap space should be two times then the size of the physical memory. So, this is a typical guideline. So, if you have got 1 GB of main memory so, you should have at about 2 GB of this disk space marked as swap space. So, we will see how this swap space is going to help.

So, we should have the facility for this swap space in the secondary storage and we should have the facility to do an instruction restart. That is one instruction by generated page fault. So, with that the instruction was halted at that point of time. So, the process got blocked and after some time when the process comes back, the page being loaded into the memory frame then we have to restart the instruction.

So, when we restart instruction maybe the previous to the instruction was half calculated like this addition operation maybe you have done the addition added the two operands, but before storing it in a third value so, we generated a page fault. Now when the instruction starts again are we going to do the addition again so, or starting from the middle so, that is a big a big question.

And sometimes it may be that if I restart so, restarting the instruction may not be very easy because it has modified some of the values of the variable so, that can happen. So, from the processor side I need to have some scheme by which I can restart the instruction without any worry about the data items that the instruction is going to modify. So, that is another problem. So, these are supported by this modern designers so of this CPU designers the processor designers. So, they take care of that.

(Refer Slide Time: 19:30)



Instruction Restart

- Consider an instruction that could access several different locations
  - Block move
  - Page fault during the move
  - Restart the whole operation?
- What if source and destination overlap?
- Several solution. Simplest one is to figure out all the pages that are needed to execute the instruction to completion and ensure that these pages are in memory before the instruction starts executing.

So, the instruction restart issue another that typical example can be when you are trying to do a block movement. So, suppose I have got 1 in the memory so, I have got 1 chunk of data that I need to copy to another block of memory. So, I need to copy it to this space ok. Now apparently it seems that it is very simple because you can just start copying from the first element and go on copying it. But the difficulty that can come is that, I can have overlapping say like say this block and this block. So, they are overlapped in such a fashion.

So, in that case I have to start copying from the beginning and or if the overlapping is on the other side like this, then I should start copying from the bottom. So, this is there, but this takes quite some time and this if these blocks they span over more than one memory frame or more than one page, then there is a possibility that there is a possibility that a page fault occurs in between.

Now, the as per our understanding when this page fault occurs then this page has to be brought in to some main memory frame and the instruction has to be restarted. Now when we restart the instruction do we again begin the copying from the from the first element or we start copying from the point at which we had stopped. So, if there is a so, the I have to answer this question that if there is a page fault during this movement like this.

So, then which do you want to restart the whole operation or we want to do a part of it only. And the situation is more severe like if the source and destination blocks they do and they have got an overlapping.

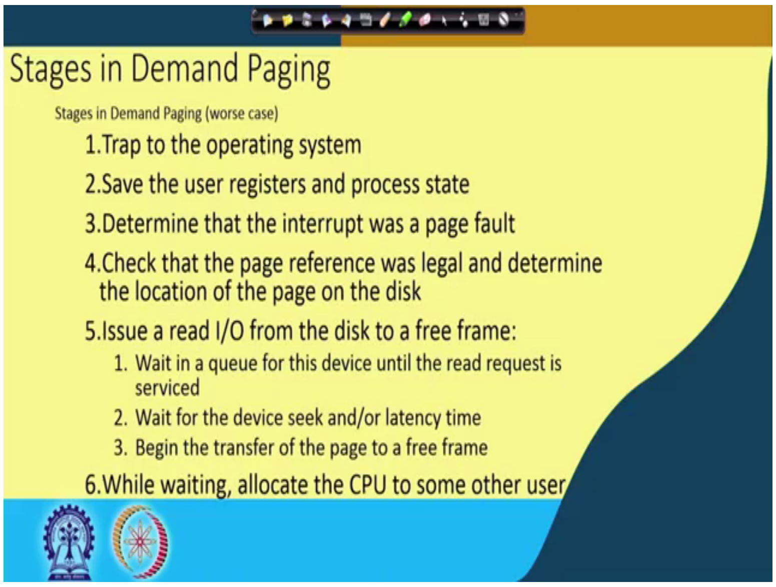
Then whenever we are doing a swap out maybe we have selected some portion which is important for us and that page has been swapped out or that frame has been swapped out. Then it will create further problem. So, we have to be very careful about this block movement type of instruction. And this block movement type of instructions are pretty common in many of the processors like this Intel architecture so, there is an instruction called MOVS ok. So, there is a machine instruction called M O V S, MOVS.

So, this MOVS instruction is one of this block movement type of instruction. So, naturally if it is working in a demand paging environment, then it may so, happen that this block transfer is becoming inconsistent due to the page fault occurring in between so, that has to be taken care of.

So, there can be several solution to the problem; the simplest one is to figure out all the pages that are needed to execute the instruction to completion and ensure that these pages are in memory before the instruction starts executing. So, this is the additional thing that I need to do.

So, I have to see that if I want to do this MOVS instruction execution, what are the pages that should be there in the memory. And I make sure that during the entire execution of this instruction, those pages are locked to the corresponding frames. So, that they are not moving so, they are they are not swapped out from the system. So, that has to be there. So, this way whenever there is instruction restart so, that has to be careful when the either system is going to support this demand paging environment.

(Refer Slide Time: 23:20)



Stages in Demand Paging

Stages in Demand Paging (worse case)

1. Trap to the operating system
2. Save the user registers and process state
3. Determine that the interrupt was a page fault
4. Check that the page reference was legal and determine the location of the page on the disk
5. Issue a read I/O from the disk to a free frame:
    1. Wait in a queue for this device until the read request is serviced
    2. Wait for the device seek and/or latency time
    3. Begin the transfer of the page to a free frame
6. While waiting, allocate the CPU to some other user

So, after that we have will look into the stages in the demand paging in the worst case what can happen? The first thing that, the first stage is trapped to the operating system So, operating system when the page is not there, so, the processor will generate a page fault interrupt and that interrupt is given to the operating system telling that there is a page fault. So, when this page fault occurs, then the first save the user registers and process state because what is happening is that this process cannot continue anymore ok.

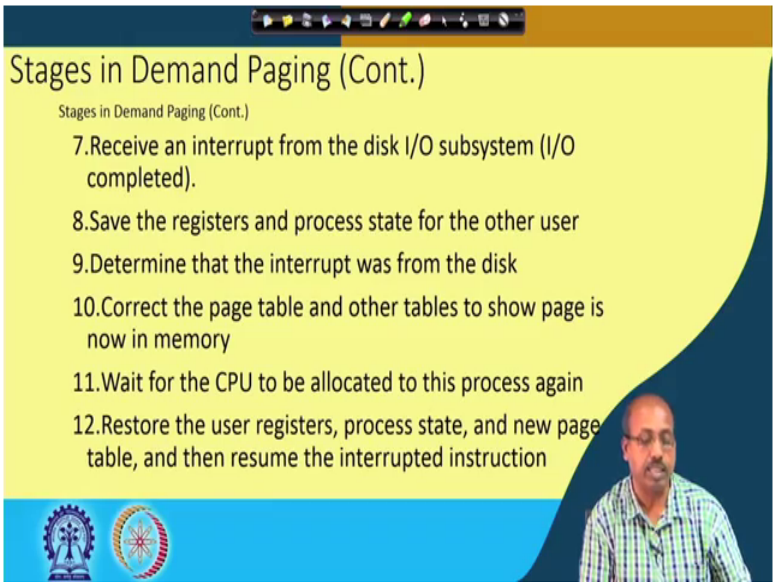
The process has to be swapped out and from the CPU and some other process will get in. So, it has to be; so, we have to save the status of all these registers and the process state. Determine that the interrupt was a page fault so, by doing the when this trap comes to the OS. So, OS has to see which trap it is so, it has to analyze the trap. And understand that it is a page fault and check that the page reference was legal and determine the location of the page on that disk.

So, whenever a page fault occurs so, it; so, a process has requested to access a location and it has been found that the corresponding page is not there. So, does it mean that whether it is valid for this particular process or not, that is the first thing to be checked. So, it checks whether the page is a legal one; whether it was a legal one to refer to that particular page and determine the location of the page on that disk. Then issue a read IO from the disk to a free frame. So, it has to find out a free frame onto which it can be copied and then it has to issue a disk read operation so, that the frame that the page will be read from that disk and put into the frame.

Wait in a queue for this device until the read request is serviced. So, as I was telling the disk scheduler will be given the job of finding this operation like who is doing like who, where should who will do this operation like when the job is over, the copying from the disk to the main memory; when it is over so, that has to be done. So, it will wait for the device seek and or latency time. So, the disk will do go to a particular block. So, it will do a seek and latency time so, that we will see later when you go to the disk scheduling policies and then it will begin transfer of the page to a free frame.

So, this is the; these are the things to be done and while waiting it will the system will allocate the CPU to some other user. As I because this process cannot continue because its page is not there so, the process is put into the blocked state and some other process is allowed to proceed. So, this is the thing that are going to happen in a demand paging environment. So, after the transfer has been done then only this process can continue.

(Refer Slide Time: 26:18)



Stages in Demand Paging (Cont.)

Stages in Demand Paging (Cont.)

7.Receive an interrupt from the disk I/O subsystem (I/O completed).

8.Save the registers and process state for the other user

9.Determine that the interrupt was from the disk

10.Correct the page table and other tables to show page is now in memory

11.Wait for the CPU to be allocated to this process again

12.Restore the user registers, process state, and new page table, and then resume the interrupted instruction

So, the next point is next stage is that receive an interrupt from the disk system that IO is complete. So, here at this point number 5, we have seen that a disk IO request has been put by the OS to the disk scheduler. So, disk scheduler will do the operation and when the operation will be over so, it will come to a step number 7, it will send an interrupt from the disk IO subsystem that the job is over.

Save the registers and process state for the other user. So, some other process was executing. So, if we decide that it has to retail the OS has to determine that the interrupt has come.

So, which interrupt it is and all. So, some other process which was there which was executing so, that has to be suspended and the OS process will be; the OS will come into picture. It will determine that the interrupt was from the disk, it will correct the page table and other tables to show the pages now in memory and it will wait for the CPU to be allocated to this process again. And after that it will restore the user registers process state new page table and then resume the interrupted process.

So, the whole situation is like this that when a process tries to access and finds that the page is not there it is a page fault, then it gives a trap to the operating system. So, operating system before going to that it has to determine what type of interrupt it is. So, for as part of the interrupt service policy, the processor will save the return address and this, all these values the processor status register the processor registers and all into this PCB of the process, and then the OS interrupt handler will be invoked.

An interrupt handler finds that it is a paging interrupt so, it will be initiating a disk operation. And the process which was which got this we generated this interrupt so, that is now that is taken of the CPU and some other process is put into the CPU. This there the previous process is blocked; after some time the disk will complete the IO operation and it will again interrupt the operating system telling that the IO is over.

So, again for going to this interrupt service routine, the process the currently running processes information has to be saved in the corresponding PCB and once the this interrupt service routine comes, it will determine the type of the interrupt and then it will correct the page table that this page has come. So, it has to correct the page table entry and all, and all other tables indicating that the process is the page is now available in the memory frame.

Till you wait for the CPU after and then after some time the CPU scheduler will again pick up the process which actually got interrupted which actually got the page fault. So, that process will be informed that that process will be scheduled. And when this process is scheduled so, before starting the process the system has to restore all that CPU user register process state, and this new page table and all that and then only it can resume the process.

So, this is the whole set of operations that are taking place when an page fault interrupt occurs. So, what we must understand that, it is not a very simple thing to handle the page fault there many operations are involved. As a result this page fault handling mechanism it should be very very efficient for the system to be successful; otherwise it will be putting a burden on the system and the system performance will only go down ok.

So, we will see in successive classes like how they are taken care of in this in different operating systems to make this page fault handling mechanism very very efficient, we will continue with this in the next class.