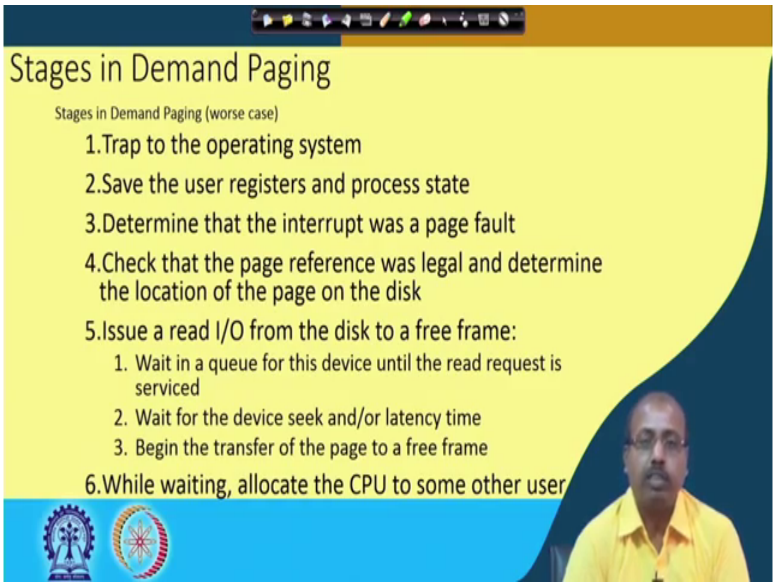
Operating System Fundamentals
Prof. Santanu Chattopadyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 51
Virtual Memory (Contd.)

So, in our last class, we have been discussing about stages in demand paging.

(Refer Slide Time: 00:33)



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, just to recapitulate the thing and to understand it in a better fashion; so, when a page fault occurs. So, when a processor tries to access one memory location and that particular page is absent in the memory. So, what happens is that it generates a trap to the operating system. So, this trap is defined by the processor designers. So, when some exceptional situation occurs. So, that is actually inform to the operating system that ok, exceptional case situation has occurred the page is not available.

So, there may be trap generated for different situations, like page fault is one of them then divide by 0 is another one. So, like that there are many cases in which the trap may be generated. So, first thing is that whenever a trap occurs, when it is an exceptional situation. So, some exceptions service has to be initiated. So, to do that first the user registers and the process state they are saved in the stack. And, then the system decides that that the interrupt that or the exception that it got is a page fault.

So, after that after it has been decided that it is a page fault, then the processor the operating system will check that the page reference was legal and determine the location of the page on the disk. Now, when a page is absent or a memory location is absent in the physical memory. So, there may be two situations; one case is that the page the address generated by the process is wrong. So, it is outside the address space of the process. So, as a result it is an error.

Other possibility is that the if the reference is valid, but the page is not present in the memory. So, first thing that has to be checked is whether the address belongs to the address range of the process. So, that is done and it check whether the reference was legal or not. If, the difference is legal, then it tries to determine the location of the page on the disk, because now the page will be available in the swap space.

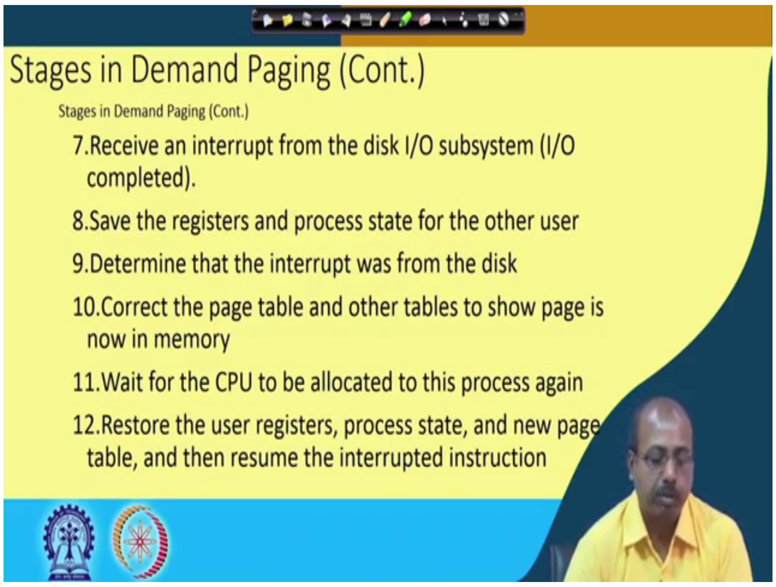
So, the operating system tries to find out the location or find out the swap space sector number where the particular page is located. And, once it has been determined that sector address has been determined, then it issues a read I O from the disc to a free frame. Now, there may be a situation like, so in the memory the page may there may be some frames free. So, accordingly it will determine a free frame and then it will after determining the free frame. So, it will wait for the disk to complete the I O operation. So, that the page is transferred from the swap space to the free frame.

So, in the process what happens is that the process that was executing so, that cannot proceed anymore. So, it will wait in a queue for this device until the read request is serviced. So, and the in this so, it will wait for the read request to be over and as we know that, this data transfer is done by means of some direct memory access sort of controller DMA controller. So, the process cannot proceed further. So, it has to wait till the read request has been serviced.

It wait for the device seek and or latency time. So, the that the disk head has to go to proper sector address and then only for that it will required some seek time and latency time. So, we will see them while discussing on the disk. And, after that it starts transferring the page to a free frame. So, this whole thing is normally done by the DMA controller and once the DMA controller is done with the transfers. So, it informs the CPU that the transfer is over.

So, while waiting the CPU is ideal. So, CPU can better be given to some other users or some other process. So, that is what is done the short term scheduler comes into existence and it schedules the next ready job to be to run in the CPU. And, the process we generated the page fault is now in the blocked state.

(Refer Slide Time: 04:47)



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, after sometime the disk transfer will be over as a result it will receive an interrupt from the disk I O subsystem that the I O operation is over. And, then thus a now this process has to be this is again interrupt so, whichever process is running now. So, for that it has to save the registers and process state. And, then again the interrupt service for the disk has to be initiated. So, the waste determines that the interest received is from the disk and accordingly it has to the initiate the disk transfer, initiate the corresponding action.

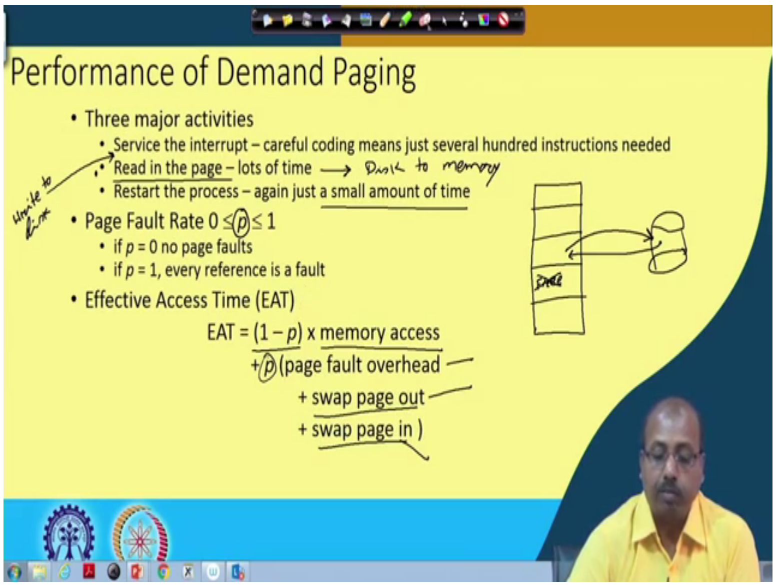
So, what is the action now? Now page has been transferred. So, now, my page table has to be updated. So, previously the page table corresponding to this particular page the entry was marked invalid now that frame number to which the page has been copied. So, that is noted down in the page table and the page table entry is marked as valid. So, this that is the correct the page table another table to show that the page is now in memory and then the process is in the ready state. So, it was in a blocked state, when the page table has been corrected in the process makes a transition to the ready state.

Now, again after sometime the short term scheduler will possibly pick up this process for running. So, the when it will be CPU it will be waiting for the CPU to be allocated to the process again and then when this process restarts. So, before that it has to restore user registers process state and new page table. And, then only it can resume interrupted instruction. So, you see there is a big piece of job that has to be done when a page fault occurs.

So, naturally the process that that created that face the page fault has to be withdrawn from the CPU and it will go to the blocked state and it will start after sometime. So, the prediction like how much time a process may take in a demand in this virtual memory environment so, that is the question ok. So, you cannot we cannot have a straight forward answer, because when the disk I O will be complete. So, that is not deterministically determinable. So, the naturally we cannot say like when the process will finish. So, that is a big problem with this virtual memory type of environment.

But, at the same time it gives us the flexibility that that the user the logical address space of a program can be huge ok. So, that is the advantage it provides; so, though we compromise on the performance side, but the other flexibility like size and all. So, that makes it very much important one for these memory management policies and most of the modern operating systems. So, they will support this virtual memory and perhaps the more complex versions of that.

(Refer Slide Time: 07:49)



Performance of Demand Paging

- Three major activities
  - Service the interrupt – careful coding means just several hundred instructions needed
  - Read in the page – lots of time → Disk to memory
  - Restart the process – again just a small amount of time
- Page Fault Rate $0 \leq p \leq 1$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault
- Effective Access Time (EAT)

$$\begin{aligned}\text{EAT} = {} & (1 - p) \times \text{memory access} \\ & + p\,(\text{page fault overhead} \\ & \quad + \text{swap page out} \\ & \quad + \text{swap page in})\end{aligned}$$

Now, what is the performance of demand paging? So, there are 3 major activities; one is the to service the interrupts. So, careful coding means just several 100 instructions needed so, for servicing. So, if we are writing that interrupt service routine carefully, then also it will take about the 100s of instructions, then the next part is reading the page. So, this is actually the time that that is going to have very high effect on the that is going to have a very high effect on this overall timing.

Because, this is the disk to memory transfer and there are 2 issues; first of all that the disk controller maybe busy doing many other operation. So, that is many other request it was servicing so, that request whatever generated by this process. So, that is in a queue. So, that is one problem and the second problem is that disk access by itself is much slower than the memory access or the CPU operation, because disk is a electromechanical device. So, that takes lot of time.

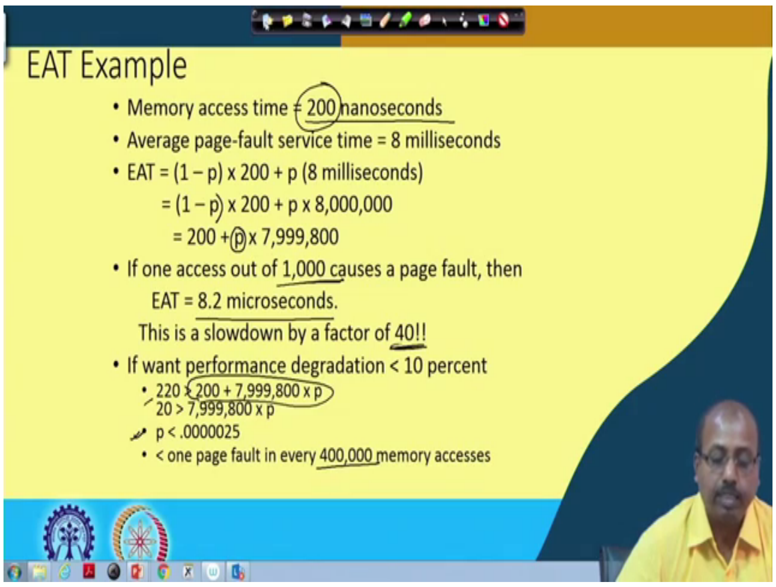
So, reading the page. So, this is the most time consuming part in the whole operation and after that after the page has been transferred restarting the process. So, this again a small amount of time ok. So, this is not that large. So, if we assume that a page fault rate is p, then p can be a value between 0 and 1. So, p equal to 0; that means there is no page fault and p equal to 1 means every reference is a page fault. So, this is the page fault rate let is per how many pages you have how many page faults occur per page per reference. So, p equal to 1 then per reference per memory reference there is a page fault.

Now, what is the effective access time? So, if the if there is no page fault; so, this 1 minus p in that case the page is present in the memory. So, that is memory access time plus if there is a page fault and whose probability is p. So, this is page fault overhead swap page out plus swap page in. So, there are 3 cases in fact, in the previous discussion that we were doing. So, basically we said that there are memory frames and when a page fault has occurred we assume that some frame is free maybe this was a free frame.

But, in reality what can happen is that what is maybe while looking for the free frame, we may find that no frame is free all the frames are occupied by some valid pages. In that case we have to make some frame free ok. So, what we can do so, if all the pages are occupied maybe we have to select one particular page. And, that page we swap out and write it or write onto the disk space, on to the swap space we write it and then the then this page because free and then we swap in the actual the needed page here. So, that way we have got a page swap out time and a page swap in time. So, both the times may be there.

So, the instead of doing only a disk read only say this operation. So, it may be there is a disk write also. So, there may be another operation that is that it will try to write on to write to disk. So, this operation may also be required when the when we have got this a none of the memory frames are free. So, this is there so, that way it takes this demand paging performance effective access time becomes governed by this particular formula.

(Refer Slide Time: 11:37)



EAT Example

- Memory access time = 200 nanoseconds
- Average page-fault service time = 8 milliseconds
- EAT = (1 – p) x 200 + p (8 milliseconds)
  = (1 – p) x 200 + p x 8,000,000
  = 200 + p x 7,999,800
- If one access out of 1,000 causes a page fault, then
  EAT = 8.2 microseconds.
  This is a slowdown by a factor of 40!!
- If want performance degradation < 10 percent
  - 220 > 200 + 7,999,800 x p
    20 > 7,999,800 x p
  - p < .0000025
  - < one page fault in every 400,000 memory accesses

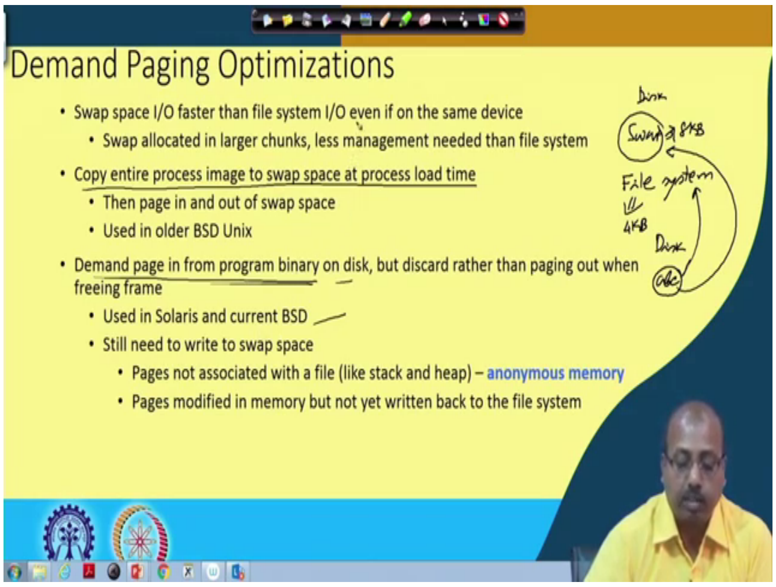
So, next we will try to look into an example, that will tell us how this EAT can grow. Suppose, the memory access time is 200 nanoseconds and average page fault service time is 8 milliseconds, then effective access time is 1 minus p into 200 plus into 8 millisecond. So, we so, that is there should be a bracket here bracket is missing. So, basically in this formula so, if you substitute. So, it becomes 200 plus 7 p into 7,999, 800.

Now, if we so, it all depends on the page fault rate p. So, if we assume that this one out of 1000 memory accesses cause a page fault on an average. So, if that is the scenario, then the effective access time becomes 8.2 microseconds. So, the while the memory access time was only 200 nanosecond, in the in the virtual memory environment, the effective memory access time is becoming 8.2 microsecond. So, that is a significant increase in the memory access time. So, this is a slow down by a factor of 40.

So, naturally this is a very high page fault rate ok. So, if we want that the performance degradation should be less than 10 percent. So, so, that in that case this 10 200 milli 200 nanosecond is the actual time. So, if I allow 10 percent degradation. So, that is 220. So, this whole expression should be less than 220 or it gives rise to the equation that p should be less than 0.5 000025. So, one page fault per 400 1000 memory access. So, that is a very low rate of page fault. So, page fault rate should be very very low. So, if we want that the performance degradation will be very small.

So, that is that example shows that we have to do something. So, that this page fault rate becomes pretty low. So, that is actually we have to think about like the how many pages of a process be loaded, then how which pages do we load into memory. So, all this decision so, they are going to affect this EAT competition and if we really want that the EAT should be EAT degradation should be low then the page fault rate should also be very very low. So, we will look into that how this is address to in this demand paging environment.

(Refer Slide Time: 14:09)


Demand Paging Optimizations

- Swap space I/O faster than file system I/O even if on the same device
  - Swap allocated in larger chunks, less management needed than file system
- Copy entire process image to swap space at process load time
  - Then page in and out of swap space
  - Used in older BSD Unix
- Demand page in from program binary on disk, but discard rather than paging out when freeing frame
  - Used in Solaris and current BSD
  - Still need to write to swap space
    - Pages not associated with a file (like stack and heap) – anonymous memory
    - Pages modified in memory but not yet written back to the file system

So, demand paging. So, this is the optimizations that we can do in demand paging environment we all like this. First of all the swap space I O faster than file system I O even if on the same device. So, we have to so, is we said that the swap space where this process pages are loaded. So, they are in the disk. Now, this particular disk is known as the swap space or swap disk.

Now, this disk even if the it is so, in the disc we have got some portion defined as swap space and we have got the other portion which holds the normal file system, now they are held on the same disk. So, raw speed if you try to compare, then both of them access to both the types of region so, they will have the same timing.

However, what we can do we can make this block size of this swap space to be larger than the block size of the file system. So, block size means in one transfer how many bytes are transferred to the main memory or when you are transferring from memory to disk how many bytes are transferred per access. So, normally may maybe my for my file system that block size maybe say 4 kilo byte and while for the swap space this is may 8 kilo byte ok.

And, as you will see while going to this disk I O so, you will see that for accessing every block. So, there is a seek time, when there is a latency time, that is that will come and that is actually the time needed for the mechanical movement of the read write head. So, that is one of the major times. So, after the disk has been the disk head has been aligned to the proper sector, the data transfer starts and that is pretty fast.

So, if I if my block size is small; that means, I have to do that the disk head alignment several times. So, if the block size of swap space is double of that of file system. Then, naturally number of, that read write head alignment will be half of the case that we have in the file system.

So, swap space is allocated in larger chance. So, that this disk read write head alignment is less and less management needed than file system, because we do not so, we make the system much more simpler, for file system one important requirement is that the file should be easily accessible. So, accessing a particular file searching for a particular files. So, that makes it complex and this file system is organized in such a fashion, that this that these access or the searching becomes easy.

Whereas, in case of swap, swap space that is not the case. So, we have to just locate the corresponding blocks for the process. So, we can just keep it like this. And, this is temporary once the process is over all the blocks will be released ok. So, that we do not need to remember it forever, unlike file system where a file is resides there for some for quite good amount of time, or maybe the for the entire lifetime of the system. So, that way the managing the file system is more tricky and it has to be more efficient than managing the swap space.

So, naturally this swap space management being simpler, so the access is going to be faster. So, swap space I O is faster than fine system I O, even if they are located on the same disk. Then, the second thing is that copy entire process image to swap space at process load time. So, what is done, like if this is my if this is the disc then in my in the disk I have got the process some program is there say program name is abc. So, this abc the executable version of this abc. So, this is there in the file system of the disk.

Now, what we do is that at the very beginning this entire abc is copied onto the swap space also. So, that after initiating whatever copy or read write is made. So, they are made from the swap space only. So, we do not need to refer to the original file abc in the disk so, in the file system. So, this copy of copy entire page process image to swap space at process load time. So, this is very important. So, that makes the situation that in future, we can access directly from the swap space only we do not need to refer to the file system.

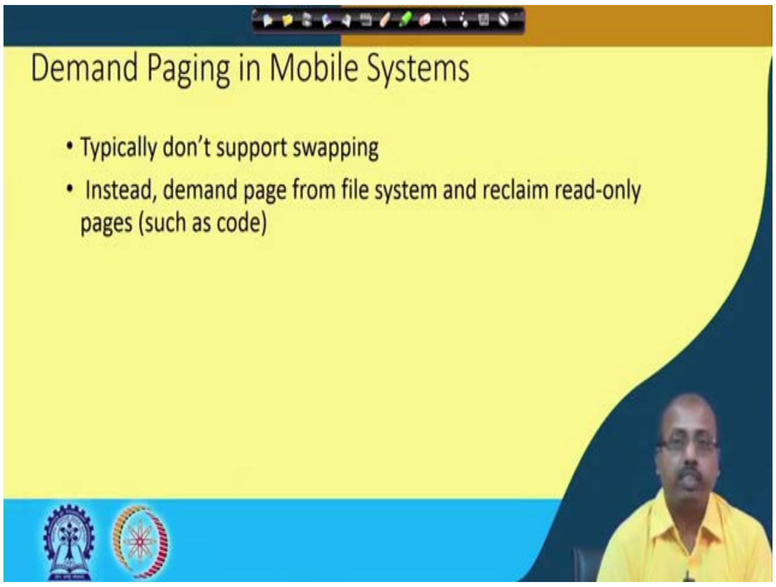
So, page in and out of swap space. So, wherever we need this need to write a page back to the disk. So, we write to the swap space and whenever we need to load a page from the so, from the disk so, we will actually load it from the swap space. And, this was this is used in older version of BSD Unix. Another, another other point that we have to note is the demand page; demand paging from program binary on disk, but discard rather than paging out when freeing the frame. So, demand page means that we do not copy the all the pages of the program. So, only the a few pages are loaded and as and when it is demanded. So, it is copied from the disc.

So, that way so, this is from program binary on disk so, they are copied, but after that if the page is not no more required. So, you do not write it back. So, until and unless it is absolutely necessary, we do not write back the memory frame to the corresponding page in the secondary storage. So, this is used in Solaris and current version of BSD, still need to write to swap space, because page is not associated with a file like stack or heap. So, that is as you know that when we take the executable version of a program in the disk, then we do not have this two spaces likes the local variables and this dynamically created variable. So, they are nonexistent at that time. So, they do not have any corresponding memory location.

So, but when the process starts running this stack and heap. So, they are also allocated space. So, that way it is useful. So, that they those are those blocks they are also to be written on to the onto the swap space ok. So, the page is not associated with a file. So, they need to be written and they we call it anonymous memory because in the in the executable version of the file.

So, they are nonexistent then page is modified in memory, but not yet written back to the file system. So, some values have been modified, but it is not yet written onto the disk. So, those pages so, if we want to swap out, then those pages are to be written to the swap space. So, that way we still need to do still need to write to this disk space ok. Some so, that has to be there; so, we cannot omit this swap space all together.

(Refer Slide Time: 21:25)

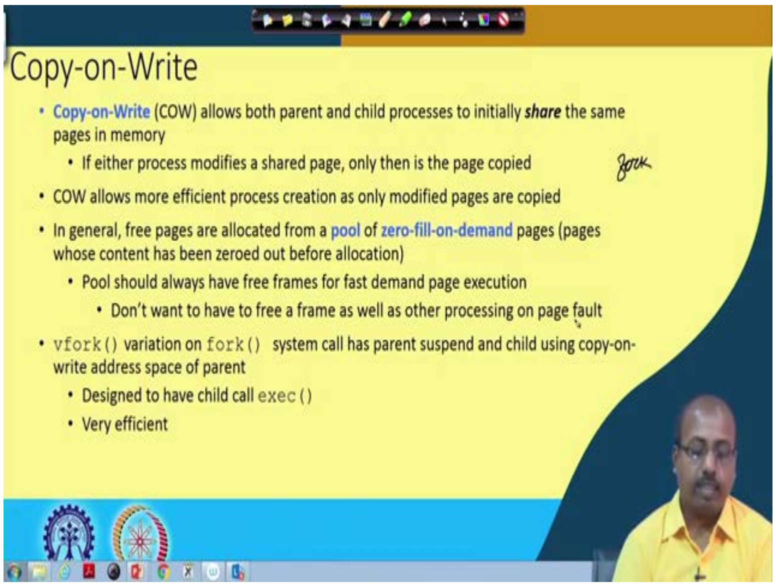


Demand Paging in Mobile Systems

- Typically don't support swapping
- Instead, demand page from file system and reclaim read-only pages (such as code)

So, the type of approach that we follow; so, similarly this demand paging in mobile system so, in mobile systems to they typically do not support swapping, instead demand page from the file system and reclaim read only pages such as code. So, this is in mobile devices we do not have very large disk. So, that is the so, we cannot we do not support swapping. So, it is directly made from the file system. And, the read only pages such as scored so, they are over written, because they need not be written back onto the disk.

(Refer Slide Time: 22:01)



Copy-on-Write

- Copy-on-Write (COW) allows both parent and child processes to initially ***share*** the same pages in memory
  - If either process modifies a shared page, only then is the page copied
- COW allows more efficient process creation as only modified pages are copied
- In general, free pages are allocated from a pool of zero-fill-on-demand pages (pages whose content has been zeroed out before allocation)
  - Pool should always have free frames for fast demand page execution
    - Don't want to have to free a frame as well as other processing on page fault
- `vfork()` variation on `fork()` system call has parent suspend and child using copy-on-write address space of parent
  - Designed to have child call `exec()`
  - Very efficient

Then, another very important concept that we have is the copy on write. So, copy on write it allows both parent and child processes to initially share the same pages in memory. So, as we know that whenever the Unix or Linux system this type of operating systems. So, so there the fork system call is executed, then this parent and child they share the same code ok.

So, they initially share the same pages and in the memory. If, either process modifies the shared page only then the is the page copied. So, whenever there is a right operation on the on the page. So, then a copy is made. So, this copy on write type of principle so, this is useful. So, if both the processes they are only doing some read operation, they are not doing any write operation, then naturally we do not have to do anything. So, the both the processes can continue.

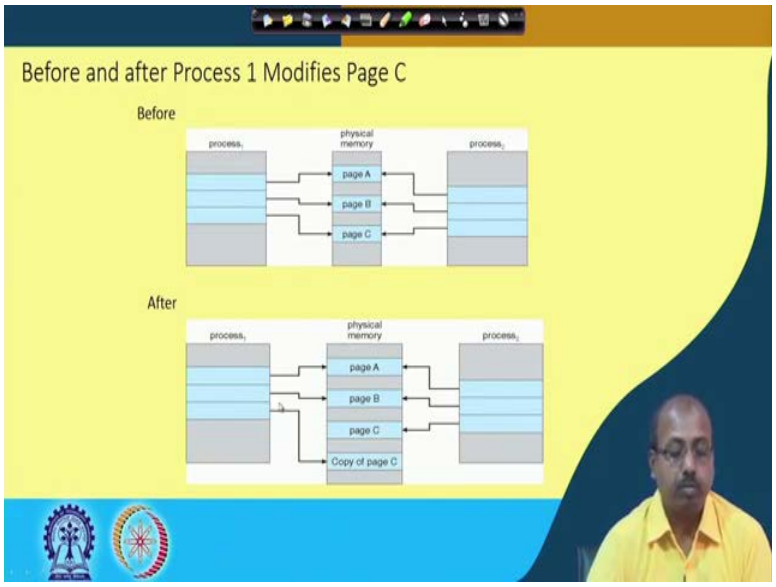
But, if one process wants to modify something, then the there is a problem and in that case we have to create a new page and allocated to the process. So, that way the shared pages are modified. So, only then the page will be copied secondary storage. This copy on write allows more efficient process creation as only modified pages are copied. So, other pages will not be there will not be copied. So, they will be shared.

In general free pages are allocated from a pool of 0 feel on demand pages. So, pages whose content has been 0 doubt before allocation. So, free pages will be allocated so, they so, we have to have some pool of free pages and then whenever a demand comes for this free page. So, it is allocated from that pool. Pool should always have free frames for fast demand page execution and do not want to have to free a frame as well as other processing on page fault. So, the since they are coming from a free pool. So, we do not have to we do not have to written that particular frame particular frame. So, pool should always have free frames for fast demand page execution. So, if we have free frame then naturally I do not have to write back anything.

And, if we do not want to have a have to we do not want to have to free a frame as well as well as other processing on page fault. So, we do not have to write back. Then there is a vfork which a variation of this fork system call has parents suspend and child using copy on write address space of parent. So, this parent process will get suspended and the child process, it will be using a copy on write address space. So, so basically if there is a if there is a write in that case only, this parent address space will be copied for to a new page and child will modify it, otherwise child will modify the parent address space only.

So, designed to have child call exec so, the so, the child process can call exec also and if it calls exec then this entire address space gets over written for by the child. So, naturally I have we do not have to create separate address space for the child. And, it is very efficient because this memory copy and all that are not required.

(Refer Slide Time: 25:31)

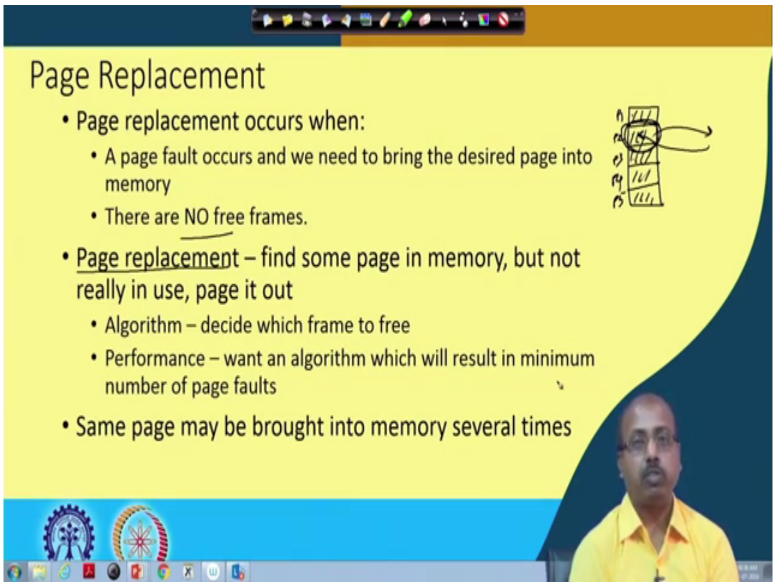


Before and after Process 1 Modifies Page C

So, next we will see like how this can be utilized? So, suppose be suppose before process before the process one modifies page C. So, process one and process 2. So, they were sharing the pages page A, page B and page C.

So, they are page tables they are pointing to the same on same like this. And, after process one modifies page C, then you we have to give it a copy of page C. So, copy of page C is given and now this page table pointer points to the copy whereas, the original process to so, it is pointing to the original page. So, this is the copy on write type of principle. So, once the copy is made in that case we have to once we have to do we write operation or modifying a page. So, we have to do a copy of that.

(Refer Slide Time: 26:19)



Page Replacement

- Page replacement occurs when:
  - A page fault occurs and we need to bring the desired page into memory
  - There are NO free frames.
- Page replacement – find some page in memory, but not really in use, page it out
  - Algorithm – decide which frame to free
  - Performance – want an algorithm which will result in minimum number of page faults
- Same page may be brought into memory several times

Next very important issue in this virtual page virtual paging environment or demand paging environment is the concept of page replacement. So, page replacement means when we want to load a new page into a free frame. So, it may so, happen that there is no frame free. So, in that case we have to select one of the pages, which is currently occupying some memory frame to be taken out of the memory and some new frame a new page should be allocated that particular frame.

So, this page replacement occurs when a page fault occurs and we need to bring the desired page into the memory. So, that is the page fault. So, this is the page replacement at that time it will be required and there are no free frames. So, under this situation so, we have to have we have to have this page replacement. So, the policy of page replacement is to find some page in memory, but not, but not really in use. So, page it out. So, basically the situation is like this that in my memory. So, I have to I a process has asked for a new page, but this all this memory frame so, they are occupied, they are occupied by valid pages of other processes, that is there are no free frame.

Now, I have to select one of these pages as the victim page. And, whichever page we select then that page has to be swapped out and as a result this frame will become free and there I will be able to load the new page; so, this that is the policy of page replacement. So, find some page in memory, but which is ideally that is not really in use. So, it is there in the memory, but it is not being used by the process by the processes. If we can find such a page, then we will put it back onto the swap space and free that frame and we can load that load the new page that is required into that particular frame.

So, we need some algorithm to decide which frame to free. So, that is a very important issue like which cream we want to free. So, that has to be seen and the performance so, want an algorithm which will result in minimum number of page fault. So, as you can as we have seen that this number of page fault so, it is going to affect the performance of the system significantly. So, that has to be done. So, this the effective access time should not increase very much. So, for that this page fault rate should be low.

So, this page replacement should be such that, we are not replacing a page which will be required in near future. So, ideally I should be able to look into the future and understand like which page is not going to be referred to again in the near future. So, that page that particular page can be removed from the memory and the corresponding frame made available for loading the new page, but definitely looking into the future is not possible.

So, we have to have some more realistic algorithm and naturally that will affect the system performance. Some page may be brought into memory several same page may be brought into memory several times. So, it may so, happen that so, at this point of time we have seen suppose this was a page 1, this was a page 1. So, this is page 2, page 3, page 4 like that.

Now, this page 2 we have the swapped out now it may so, happen that at the very next time instance. So, again there is a reference to a memory address which is actually in page 2. So, again there is a reference to page 2. So, that so, since page 2 is not present now in the memory so, there will be a page fault. So, again that page fault has to be serviced and it will take time for getting that page into memory again another free frame another frame has to be determined whose page can be written back into swap space and there we can load page 2.

So, this way the same page may be going out of which has which had gone out of memory may be coming back to memory after sometime. So, this may go on several times. So, same page may be swapped out and swapped in several time. So, if that thing happens. So, if we are not very I should say judicious about selecting the page that is going out of this main memory, then what can happen is that it will very soon we will be generate a lots of page faults ok.

So, there are several algorithms that we are going to discuss that will be that are being used for these selecting the victim page, but whatever we do the our objective is to minimize the number of page faults.

So, we will continue with this discussion on these page replacement algorithms in the next class.