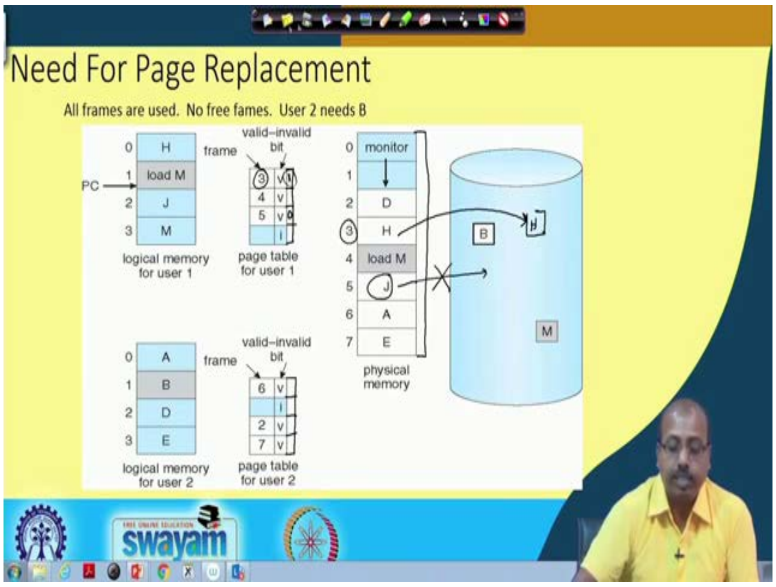
Operating System Fundamentals
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 52
Virtual Memory (Contd.)

Next we will look into an example where we tried to explain this need for page replacement.

(Refer Slide Time: 00:35)



Need For Page Replacement

All frames are used. No free fames. User 2 needs B

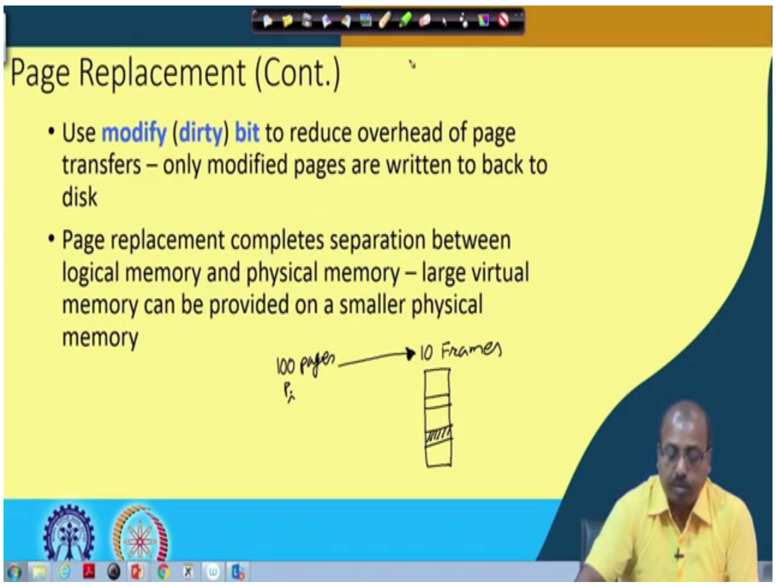
So, all frames are used no free frame and user two needs the page B. So, this is the logical memory for user 1. So, this is the corresponding page table for user 1 and this is the logical memory for page 2 for user 2, then this is the corresponding page table. So, you will see that in case of this user one so, this page number 0 1 and 2. So, they have been loaded into corresponding memory frames 3 4 and 5 and page number 3 that is still invalid.

So, this is the page table for user 1 and currently user 1 is executing here. So, this load M; so, load M. So, this is there; so, this is it is accessing this load M instruction. So, this is this will be fine. So, this is the page is there in the memory. So, it is trying to access page number 3; frame number 3.

Now on the other hand, suppose this user 2 needs to access this particular page B. So, if it wants to access this page B, then if see the corresponding page table entry is invalid; meaning that the page is not present in the memory. So, naturally it will generate a page fault.

Now when it generates a page fault so, you see that this physical memory. So, none of the frames are free. So, all the frames are occupied so, we need to have a page replacement. So, we can select one particular page; suppose we select say page number 1 so, this frame number 4 or page 1 of process 1 to be replaced. So, what we can do? We can take it out and load the page B here. But the point is if we do that that at the after some time when user1 process we will come, then it will generate another page fault when it is trying to execute this load M instruction; so, that is there.

(Refer Slide Time: 02:32)



Page Replacement (Cont.)

- Use modify (dirty) bit to reduce overhead of page transfers – only modified pages are written to back to disk
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory

So, for this page replacement to decide like whether we want to write back the page on to the disk; so, whether it is necessary or not so, we can use a modify bit or dirty bit to reduce the overhead of page transfers, only modified pages will be written to written back to the disk. So, in the previous example suppose in this situation, I have to select a victim page where which, whose frame has to be made free and where I will be loading this page B.

Now, if we have got another bit here that tells like whether the page was modified or not. So, there is a dirty bit and this so, basically with this dirty bit we can think that it is associated with this page table. So, there is a dirty bit and that dirty bit actually identifies whether the corresponding page was written by the process or not; after it was loaded. So, if the dirty bit is made equal to 1; that means, the page was modified.

So, if you decide that I will replace page this particular page that is the page this particular frame 3, then we find that the corresponding bit is 1; that means, the page was modified. So, in that case you need to write H back on to the disk, the swap space first and then only you can make use of this frame 3.

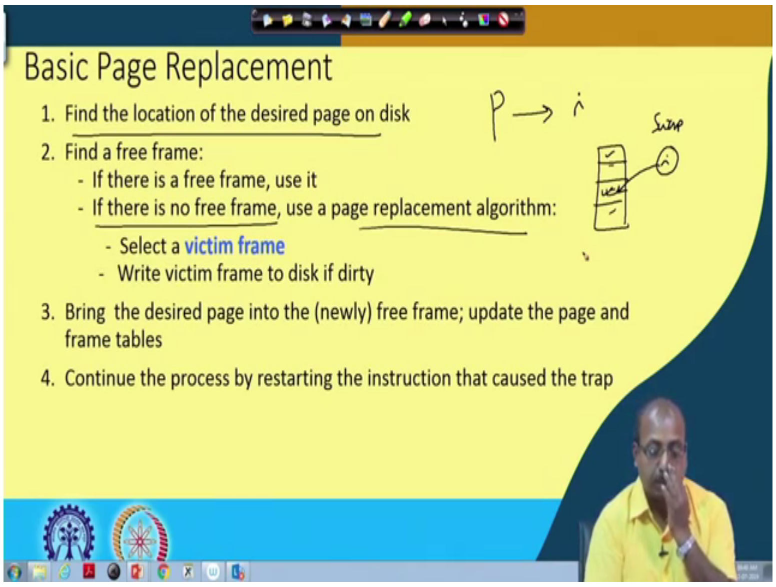
On the other hand so, if you find that say for this one this 5, this bit is 9; then if we select this then since this was not modified. So, I do not have to write back this particular page to memory. So, that writing is not necessary because it is already it is not modified after it was loaded. So, this way we can make you of this dirty bit or the modified bit to take care of this situation that is whether I need to write back or not. So, if the modified bit is 0, I do not need to write back. So, that will save my time ok.

So, this way this operation can be done this using this modified bit and this page replacement it completes separation between logical memory and physical memory. So, you can have large memory that can be provided on a smaller physical memory. So, physical memory is small, but at one point of time maybe to going to an extreme example. Suppose my physical memory can has got only 10 frames. So, it does not mean that my program should be limited to 10 pages only, my program can have 100 pages ok.

And as you know this page size and frame size are same so, this 100 page program may be mapped on to this 10 frame memory by the way that initially you load only say few pages of the memory of the program. And then as the process is executing so, whenever it requires to access to other pages so, they will be loaded so, on the basis of this demand paging. So, this page replacement so, if all the frames are occupied also; if all the frames are occupied also, then this particular process suppose process P i; so, it might have been allocated a few frames here. So, these may be only that two frames allocated to process P i.

So, at least I should be able to replace this particular page by the new one of the process P i so, that can be done. So, I can continue like this. So, even if there are only a few pages available for a process so, theoretically there is no limit on the size of the process ok; the process size can be huge so, but of course, there are other issues. So, we will come to that, but theoretically there is no limit on the size of the program.

(Refer Slide Time: 06:37)



Basic Page Replacement

1. Find the location of the desired page on disk
2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm:
     - Select a victim frame
     - Write victim frame to disk if dirty
3. Bring the desired page into the (newly) free frame; update the page and frame tables
4. Continue the process by restarting the instruction that caused the trap

So, that is the advantage, but what we have to concentrate on is the page replacement policy. So, page replacement policy this is going to be very serious because that will determine that we are not doing, we are not replacing a page which is going to going to be accessed again in the near future. So, the operations that we do in the page replacement algorithm is to find the location of the desired page on disk, then find a free frame. If there is a free frame use it and if there is no free frame, use a page replacement algorithm.

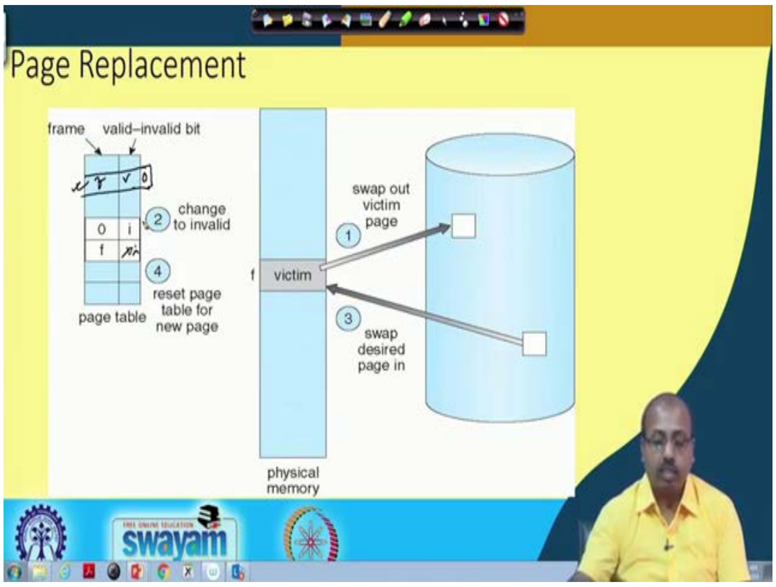
So, we want to load a particular page so, process P. So, it has generated a request; process P has generated a generated a request for page i. So, first thing is that in the swap space, we find out where is this particular page i. So, once we have found that it is there in this particular block in the swap space so that is the first step. So, find the location of the desired page on disk. After that we have to look into the memory to in search of a free frame. So, if we can find a free frame; suppose this frame was free then our job is simple.

So, we can just copy this i onto this free frame and accordingly we update the page table. But the second thing is that if there is no free frame so, all are occupied then we have to use a page replacement algorithm for page replacement algorithm. So, it again consists of two parts one is to select a victim frame. So, if all frames are occupied, then we have to select one frame which can be a good victim ok. So, now, there can be naturally the criteria for selecting a victim page is the page which is not likely to be referred to in the near future. So, that way we select a frame select a page to be swapped out from the memory to the disk. So, we select a victim frame and write victim frame to disk if dirty.

So, we look into the corresponding modified bit of the page and if the page has been modified; in that case so, we have to write that page back on to disk first before take claiming that particular frame. And after it has been written, now this new page there can be loaded on to that. So, number 3; step number 3 is to bring the desired page into the newly free frame and update the page table and frame table. So, they have to be updated and then the process continues by restarting the instruction that caused the trap.

So, basically the process goes from this ready state to running state again and then when the process gets scheduled and then the process will restart its operation. So, that is the basic philosophy of this page replacement policy. So, we will see several algorithms for this page replacement in our next few slides.

(Refer Slide Time: 09:40)



Page Replacement

So, this is the diagrammatic representation of whatever we were talking about. So, this is the page table giving the frame number and the valid invalid bit. So, this is the frame number. So, o and this is invalid and this is frame number f and this is valid. So, at first; the first step is to find the victim frame. So, for after analyzing all the pages that are there in different frames in memory; so, suppose my page replacement algorithm selects page number f as the victim page.

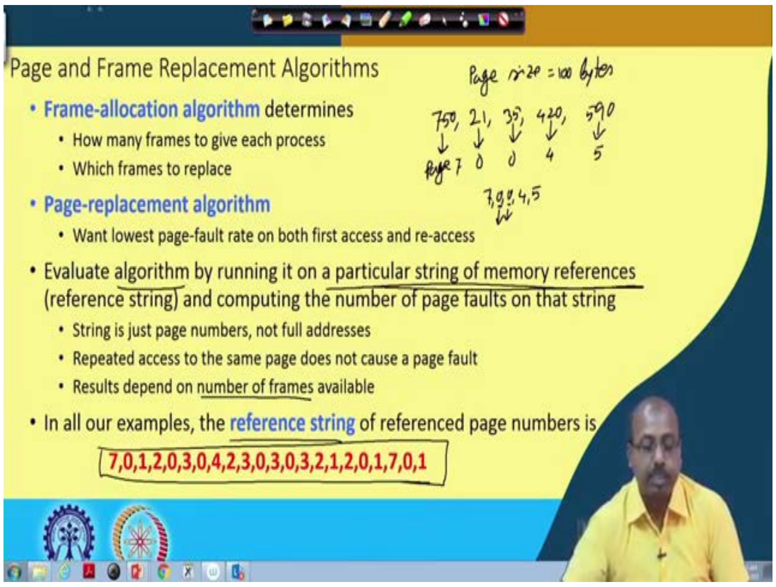
So, the first step is to write this back onto this disk because this frame number f it is assumed that this is a valid page and then it is also assumed that the modified bit was set to 1.

So, it is first written back onto the swap space the page number or the corresponding frame and then we change this bit to invalid bit ok. So, this f becomes invalid bit. Now this page will be swapped in. So, this page will be swapped in and then after that we have to reset the page table for the new page. So, this whatever be the corresponding page number. So, after this f has been swapped out so, this entry will turn to invalid telling that this is invalid.

Now, suppose this is my page this page is loaded here. Now so, corresponding entry suppose this is the page that has been loaded now and that that will be made equal to f and this valid bit should be turned to v. And that modified dirty bit so, that should be made equal to zero telling that the page is not yet modified. So, like that this page replacement should proceed ok.

So, the first step in page replacement is to determine a victim page that can be swapped out. And depending upon the characteristic of that page whether it is modified or not, either you need to write back or we can continue copying the newly the new page that is needed.

(Refer Slide Time: 11:57)



Page and Frame Replacement Algorithms

- **Frame-allocation algorithm** determines
  - How many frames to give each process
  - Which frames to replace
- **Page-replacement algorithm**
  - Want lowest page-fault rate on both first access and re-access
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
  - String is just page numbers, not full addresses
  - Repeated access to the same page does not cause a page fault
  - Results depend on number of frames available
- In all our examples, the **reference string** of referenced page numbers is
  7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

So, this is the way this any page replacement algorithm should work. So, the first thing is that this page and frame replacement algorithms so, we have to talk about this frame allocation algorithm. So, frame allocation algorithm will determine how many frames to give to each process and which frames to replace. So, like a process when it is going to execute, then how many frames should be allocated. So, that is the frame allocation like a process may a process may have say 100 logical pages ok

Now, how many physical frames will be allocated at the beginning? So, ideally I should not have I can have say 0 page or 1 page allocated or maybe 5 page is allocated, but what is the number ok? So, that there depending upon some policy, we will see that we can have some frame allocation policy that will tell us like what is the minimum number of frames that should be allocated to any process. And that is one thing and the second thing is that which frames to replace like if you are if it is going to a page replacement or frame replacement algorithm, then which pages are to be replaced.

Then this page replacement algorithm so, as I was telling; we want lowest page fault rate on both first access and re-access. So, that is the thing that we want that the number of page faults will be pretty low.

We evaluate the algorithm by running it on a particular string of memory references and computing the number of page faults on that string. So, you can know as we will see very shortly so, there can be many such page replacement algorithms. But what is the performance of any page replacement algorithm? How do we judge it? So, they are normally judged by means of using some; by means of using some sort of string of memory references so, in terms of the pages.

So, for example, suppose my page size is say 100 bytes; suppose my page size is say 100 bytes and a process it accesses the memory locations say 750, then say 21, then say 35, then 4 say 420 then 5 9 0. So, like that suppose these are the successive addresses generated by a process; successive logical addresses generated by a process. Now since this is 750 so, this goes to page 7. Similarly this goes to page 0, this goes to page 0 so, this is in page 4 so, this is in page 5 so, like that.

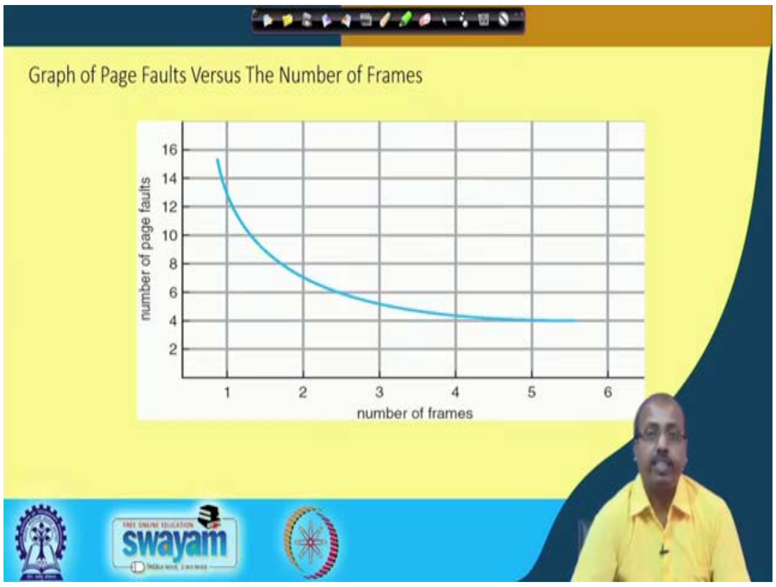
So, this particular string that we are getting that is 70045. So, this is sufficient to judge or the whether there are page faults occurring. And if there is a page fault, then how many page faults are occurring in the total lifetime of the process. So, in this way we will be defining a string of just page numbers not full addresses for discussing on this page replacement policies. So, repeated access to the same page does not cause page fault.

For example, here I have got an reference to 0, then the next or next one is also 0. So, the repeatedly we are accessing the same page though maybe at different addresses, but they are not generating a new page faults ok. So, assuming that when this page was accessed so, page 0 was present in the memory. So, then the next is 0 so, that they are also the page must be present in the memory frame and result depends on the number of frames available also.

Like if the number of frames is say, in the number of memory frames is say 10 in one case and it is 20 in another case, then naturally the performance of this page replacement algorithm in terms of number of page faults will differ. So, that is another important thing. So, number of frames. So, one thing is that the algorithm that we are talking about the page replacement algorithm and second thing is the number of frames. So, these two so, they are going to determine the performance of this page replacement algorithms.

So, the reference string or referred page number; so, in all our example so, we will be using this sequence of page number to page numbers to be accessed by a process ok so, that we will use as a benchmark ok. So, depending on the address is generated so, you can very easily I have the corresponding page reference string and we can do it like this.

(Refer Slide Time: 16:36)

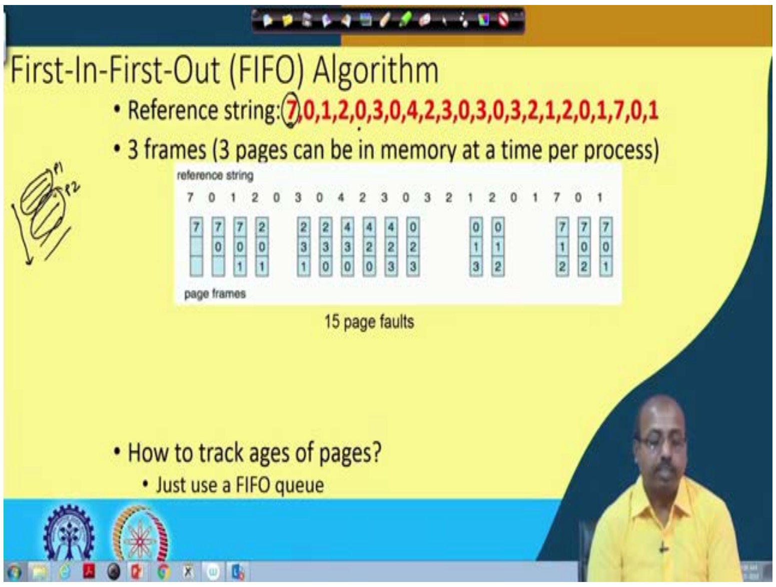


Graph of Page Faults Versus The Number of Frames

So, as you are increasing number of frames; so, if the number of frames is pretty low, so, if it is only 1, then you can see that the number of page faults will be pretty high because whenever since there is only one page available one frame available. So, first the only the first page of the process will be loaded.

Now, when the program start executing very quickly it will refer to some other page as a result some page fault will occur. And while in that page so, it may go back to page number 1. So, again there will be a page fault. So, that way number of page fault will be pretty high when the number of memory frames are is low. And this way if you are increasing number of frames, then this page fault number of page faults will start reducing and after some time it shows a saturation. So, even if you increase number of memory frames beyond some value, the page fault rate does not decrease further. So, this is there.

(Refer Slide Time: 17:37)



First-In-First-Out (FIFO) Algorithm

- Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1
- 3 frames (3 pages can be in memory at a time per process)

15 page faults

- How to track ages of pages?
  - Just use a FIFO queue

So, first policy that will be looking into is known as the First In First Out or FIFO. So, it says that whenever you have to select a victim page, you select the page which came into the system earliest. And the logic behind this is that since this page came to the system quite early. So, and it stayed in the system for so much of time, then it is very much likely that this page will not be referred to in the near future compared to others, like say this particular page 7 so, this page 7.

So, it is when this so, this page 7 so, when we are say at this point. So, you can you can between these pages 0 1 and 2. So, you can expect that this page 7 will be referred relatively less number of times compared to other pages as 0 1 and 2; so, because this page 7 came long back into the system.

So, what happens is that there is a whenever we run a program; whenever we run a program so, it follows if these are the program statements. So, it starts from this point and proceeds in this direction. Maybe so, these instructions were in page 1. So, these instructions are in page 2 so, like that. So, it is very much likely that at the beginning of the program page 1 was referred to, but as we go later in the program so, page 1 is not being referred to that much.

So, naturally if we have to select a victim so, page 1 may be a good victim for selection. So, whichever page came earliest in the system so, that may constitute a good victim. So, for this particular reference string that we are looking into; suppose there are 3 frames. So, the 3 pages can be in memory at a time per process. So, this 7 so, initially if we assume that frames are all empty then the first reference of 7. So, this 7 is loaded into one particular frame.

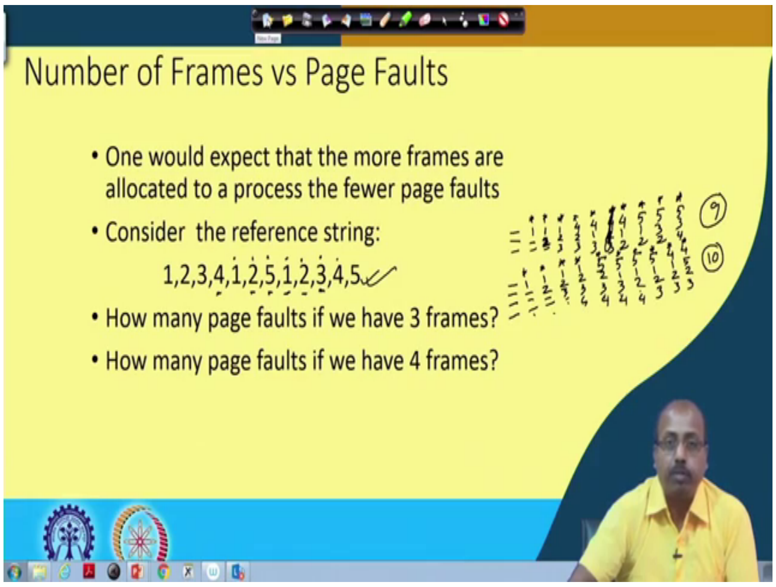
So, there is a page fault here because the page 7 was not there. Then suppose there is a reference to page 0 and again page 0 is not here. So, again a page fault will occur and the page 0 will be loaded now the 2 frames are occupied. So, this is also a page fault then at 1 so, that is again 1 is not present in memory. So, it is another page fault and 1 is loaded into memory. Now I have to so, now, there is a reference to page number 2. Now I have to since all the 3 frames are occupied and page number 2 is not there in the memory.

So, it has to figure out one page for replacement. And it based on the FIFO policy so, whichever page came into the system earliest. So, that will be selected as victim. So, this 7 is selected as victim and it is replaced by 2 ok. Then comes this memory reference 0 and the page 0 is already there in some memory frame so, there is no page fault. Then 3 so, when this 3 comes so, page 0 came into the; page into the system the oldest. So, this is the oldest page. So, that gets replaced by 3. Then again there is a reference to page 0. So, you see page 0 is just replaced and after that again there is a reference to page 0.

So, again the oldest page that is 1 so, that goes out of this frame and the corresponding frame is freed and 0 is put here. So, this way if we see that the entire sequence of this page references, then altogether there are 15 page faults that has occurred for this particular string.

Now naturally so, this may be a good policy with the understanding or expectation that if a page came into the system quite early compared to others. So, it is very much likely that old page is not going to be accessed in the near future. How to track ages of pages? So, we can just use a FIFO queue and then if we keep the page numbers there. So, based on that we can very easily find out which page to be replaced. So, we can have a FIFO queue implemented and based on that we can do this replacement.

(Refer Slide Time: 21:59)



Number of Frames vs Page Faults

- One would expect that the more frames are allocated to a process the fewer page faults
- Consider the reference string:

  1,2,3,4,1,2,5,1,2,3,4,5
- How many page faults if we have 3 frames?
- How many page faults if we have 4 frames?

Next so, if we try to see number of frames versus page faults. So, it is expected that one would that the more frames are allocated to a process, number of page faults will be reduced. So, we have seen the graph previously also there we have seen as we are increasing the number of frames the expected number of page faults also decrease. But is it always the case ok? So, we consider one example where we have got a reference string like this 1, 2, 3, 4, 1, 2, 5, 1 like that. And if we have in one case, we have got 3 memory frames; in another case, we have got 4 memory frames and we try to see the effect on the number of page faults ok

So, we will see this one like if you have got 3 memory frames, then initially all of them are so, all the three frames are empty ok. Now this one comes so that is a page fault; then for 3, I have got 1 3. So, that is a page fault then comes sorry 1 2 1 2 then 3 comes. So, that is again a page fault 1 2 3 like that then there is a 4; so, 4 is there. So, 1 is the oldest so, 1 goes out. So, it becomes 4 2 3 and that is a page fault. Then comes 1 again so, this 2 goes out. So, it becomes 4 1 3 that is a page fault, then this 2 comes. So, this 2 is again.

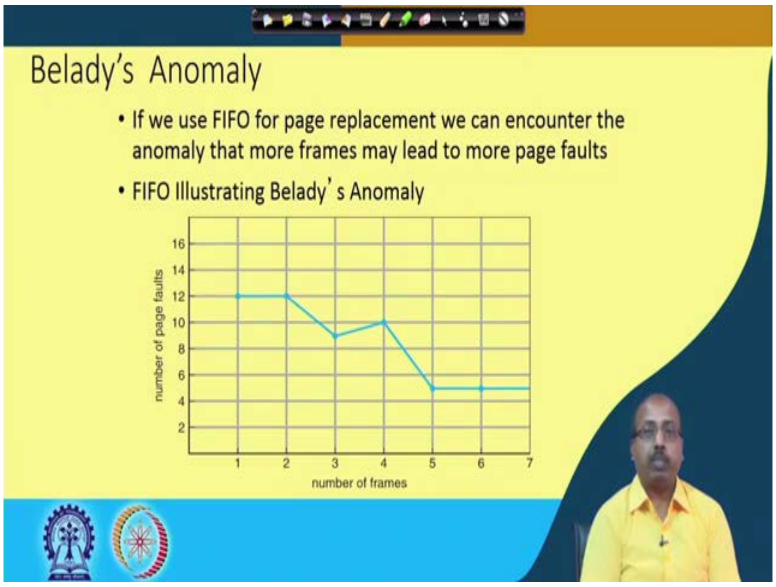
So, if this is this will be 4 2 3 there will be a page fault here sorry 4 4 1 2 there will be a page fault here now 5 comes. So, now, 4 will go out. So, it will become 5 1 2 and there is a page fault now comes 1. So, 1 is already there in the memory, 2 is also there in the memory; now comes 3. So, when 3 comes so, you see that 1 is the oldest one. So, 5 3 2 so, that thing comes; now there is 4. So, this 2 goes out. So, it becomes 5 3 4 it is a page fault and then 5 that is already there. So, number of page fault is 1 2 3 4 5 6 7 8 nine. So, number of page faults is equal to 9.

Now, we consider the case where we have got 4 memory frames. So, initially all 4 are empty. Now it is referring to page number 1. So, there is a page fault, then page number 2 that is a page fault, then 3, 1 2 3 that is a page fault then 4, 1 2 3 4 that is also a page fault. Now comes 1; this one 1 is already there, 2 is also there; now comes 5. So, when 5 comes so, this 1 will go out. So, it will become 5 2 3 4. So, it is a page fault. Then again 1 comes so, this 2 goes out. So, 5 1 3 4, then again 2 comes; so, this 3 will go out so, 5 1 2 4.

So, these are all page faults. So, this 2 has come now this 3 comes when this 3 comes. So, this will become this 4 will go out 5 1 2 3 ok, then again this 4 comes. So, this 5 will now go out. So, it will become 4 1 2 3 and then again 5 comes. So, this 1 will go out. So, this will become 4 5 2 3 and this is also a page fault. So, what is the number of page fault ? 1 2 3 4 5 6 7 8 9 10.

So, what has happened, we have increased number of frames from 3 to 4. But effectively for this particular reference string, number of page faults has increased from 5 from 9 to 10. So, that is a big problem ok. So, I this is counter intuitive. So, this actually can happen in case of this first come first serve based policy that even if you increase the number of memory frames, the page fault may increase.

(Refer Slide Time: 26:55)



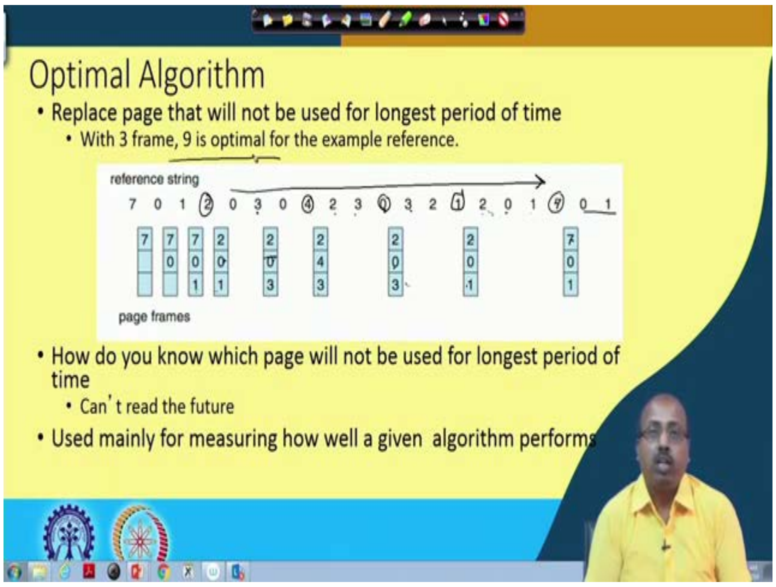
Belady's Anomaly

- If we use FIFO for page replacement we can encounter the anomaly that more frames may lead to more page faults
- FIFO Illustrating Belady's Anomaly

So, this particular phenomena is known as Belady's anomaly. So, if we use this FIFO for page replacement, we can encounter the anomaly that more frames may lead to more page faults. So, this is the thing that if we for that particular example. So, if we have got one frame that the previous string. So, that was giving us twelve page faults. If you user 2 frames, then also it will remain at 12. So, if you use 3 frames, then this number of page faults drops to 9 that we have seen. If you use 4 frames then again number of page faults, it goes up it becomes 10 so, but after that it again decreases.

So, if you increase the number of frames to 5, then the page fault rate drops and then it remains like that. So, this particular situation when after it change increasing the number of memory frames, the page fault rate is increasing. So, this is known as Belady's anomaly. So, this FIFO algorithm it has the potential to show Belady's anomaly. So, we have to be careful. So, we should not just expect that my page fault rate will drop just because I have increased the number of memory frames.

(Refer Slide Time: 28:07)



Optimal Algorithm

- Replace page that will not be used for longest period of time
  - With 3 frame, 9 is optimal for the example reference.

- How do you know which page will not be used for longest period of time
  - Can't read the future
- Used mainly for measuring how well a given algorithm performs

Next we will be looking into another algorithm which is known as optimal algorithm. So, optimal algorithm this is a theoretical one. So, it says that I would like to replace a page which is not going to be referred to in the longest future time; so, longest period of time in future. So, that is the optimal policy so, replace page that will not be used for the longest period of time.

So, with 3 frame; so, suppose I have got 3 frame here so, this 7 so, 7 0 1. So, first 3 memory reference first 3 memory references. So, they are fine so, 7 0 1. So, that is done. Now this 2 comes so, for these 2 I look into the string after this ok. So, I have to look this side and find out which page out of this 7 0 1 is not referred to for the longest amount of time and then we find that this page number 7 is going to be referred after a long time fine. So, this page 7 will be replaced and 2 will come here.

Now, this 0 is already there 0 frame is already there; now this 3 comes. So, between 2 0 and 1 you see that. So, this between this 2 0 and 1, you see that 1 is referred the long after longest time. So, 1 will be replaced by this 3 ok, then again 0 comes; so, 0 is already there. Now 4 comes, and for the 4 for getting the page which is not referred in the near future so, you see that the pages 2 and 3 so, they are going to be referred again, but 0 is the longest one; so, the farthest 1.

So, this 0 will be replaced by this 4. So, it becomes 2 4 3 now 2 3. So, they are fine, they are already there. Now this 0 comes ok. Now, 0 again if we look through, then you see 4 is not referred to in the near future. So, this 4 is replaced by this 0 then these 3 2. So, they are already there now this 1 comes and 1 you see that 2 and 0. So, they are referred in near future. So, this 3 will be replaced by 1. Now this 2 0 1 so, they are fine again when the 7 comes in near future 0 and 1 are going to be referred to. So, this 2 will be replaced by this 7 and that is the thing. So, what is the number of page fault? It is 1 2 3 4 5 6 7 8 9 page fault.

So, this is an optimal policy because we are looking forward to the history and then we are looking forward into the reference string and then we are replacing the frame which is replacing the page which is not going to be referred to in near future. So, how do you know which page will not be used for longest period of time so, that is the big question ok. And as you know that until and unless the process executes so, we cannot tell which page it is going to refer to.

So, we cannot read the future. So, that way this is going to be a theoretical algorithm only and it is used mainly for measuring how well a given algorithm performs ok. So, if you are proposing a new page replacement policy, then to judge the quality of the page replacement policy so, you can compare with this optimal replacement and that can tell you like what is the how far are you from the optimal. So, this is mainly for the benchmarking purpose. So, we will continue discussing on other page replacement algorithms in the next class.