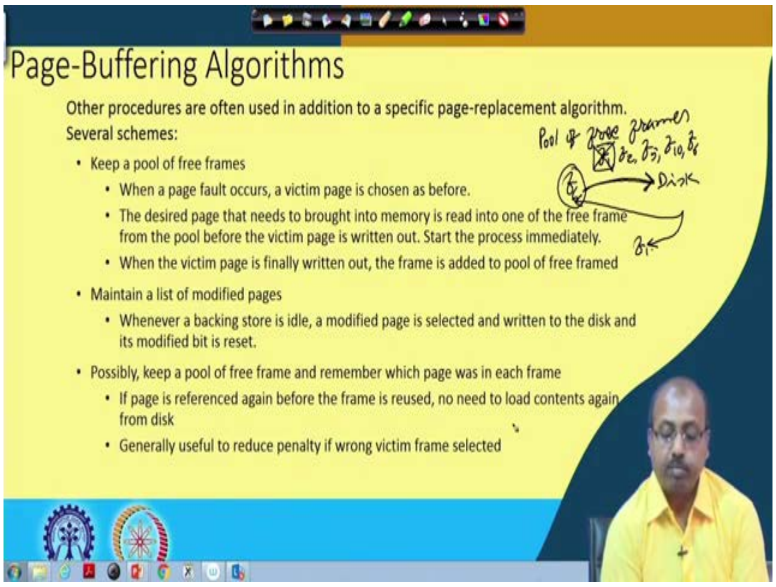
Operating System Fundamentals

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

Lecture - 54
Virtual Memory (Contd.)

(Refer Slide Time: 00:27)



Page-Buffering Algorithms

Other procedures are often used in addition to a specific page-replacement algorithm.
Several schemes:

- Keep a pool of free frames
  - When a page fault occurs, a victim page is chosen as before.
  - The desired page that needs to brought into memory is read into one of the free frame from the pool before the victim page is written out. Start the process immediately.
  - When the victim page is finally written out, the frame is added to pool of free framed
- Maintain a list of modified pages
  - Whenever a backing store is idle, a modified page is selected and written to the disk and its modified bit is reset.
- Possibly, keep a pool of free frame and remember which page was in each frame
  - If page is referenced again before the frame is reused, no need to load contents again from disk
  - Generally useful to reduce penalty if wrong victim frame selected

Next we will be looking into page-buffering algorithm. So these are procedures that are used in addition to a specific page replacement algorithm. So they are actually trying to make this page replacement policy more efficient. There are several such schemes; one of them is to keep a pool of free frames. So, we keep a pool of free frames. So, whenever we whenever a page fault, occur a victim a page will be chosen using some page replacement policy.

Now, the desired page that needs to be brought into memory is read into one of the free frame from the pool before the victim page is written out and it start the process immediately. And when the victim page is finally, written out the frame is added to a pool of free frames. So, the basic idea is like this; so at any point of time so, if the OS will remember a pool of free frames. So, this pool of free frames suppose I have got four frames that kept and kept in the pool frame f 1, f 2, f 5 and f 10.

Now, a page replacement algorithm has run. So, it has selected say frame 6 to be replaced, the page that contained in frame 6 to be replaced. And so, that frame so that frame if the modify bit is on for this corresponding page table entry. So, this frame has to be first written on to the disk. So, this should go to the disk, then only the new page can come to frame 6 and that takes time because the one disk write then another disk read. So, that way two disk transfer so, it will it takes time.

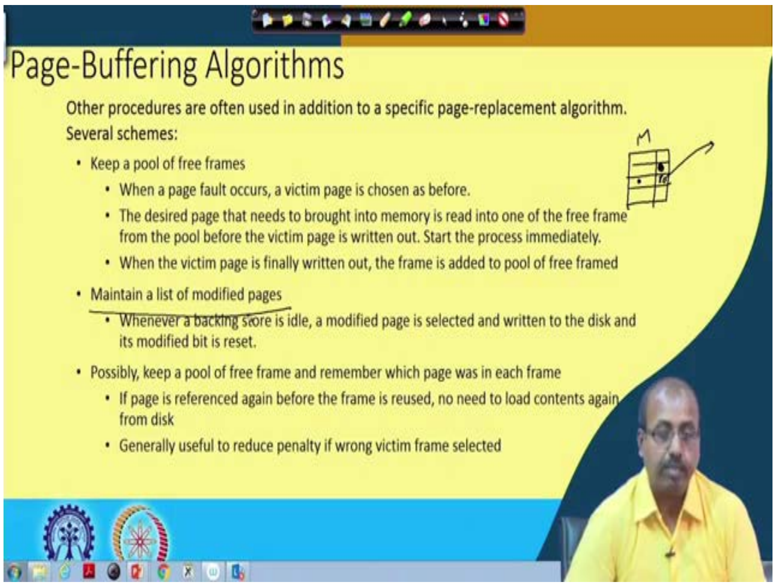
So, instead of that what is done in this particular policy is that we select this frame f 1 where the new frame will be loaded new page is loaded. So, this page instead of being copied to f 6 so, this is copied to f 1. So, after it has been copied to f 1 the process can continue. So, process go goes out of the blocked state, it goes to the ready state and all and this particular write of this frame 6 to disks. So, this is also scheduled and it will take its own time. At some point of time, it will be done and then when this is done then this frame 6 is now free. So, this frame 1 was removed from the pool of free frame and this frame 6 will be added to the pool of free frames.

So, this way by keeping a pool of free frame so, we can save one disk write time. So, that the disk write is actually happening in the background. So, this frame 6 can be written later. So, this is one of the one policy keep a pool of free frames. Then another policy is to maintain a list of modified pages.

So, whenever a backing store is ideal a modified page is selected and written to the disk and its modified bit is reset. So, the problem that we had with this page replacement is that if the bit is not, if the page table entry the modified bit is not set; that means, the there is no write that has been done on to this page. So, while replacing the page so, I do not need to write back to this page into main memory into the disk.

So, many times what happens is that the disk scheduler it may be ideal and you do not have any disk activity. So, operating system it can tell the scheduler to write some of this modified pages to the disk ok.

(Refer Slide Time: 04:04)



Page-Buffering Algorithms

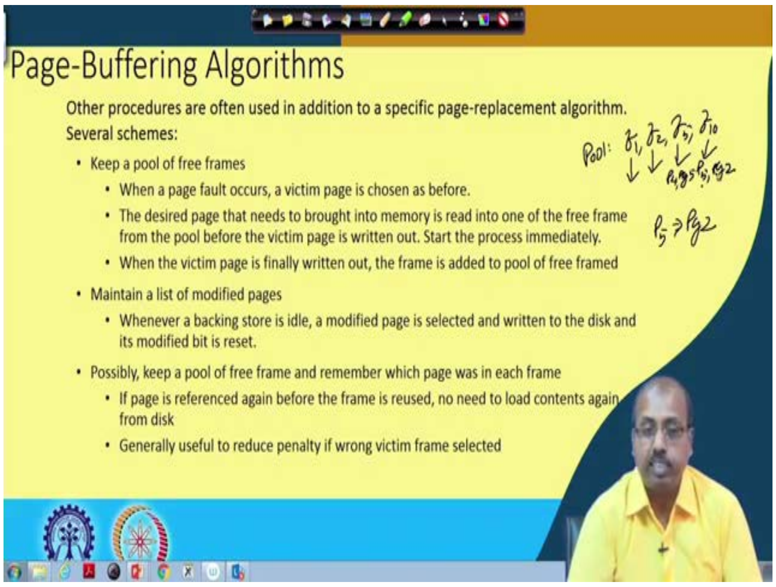
Other procedures are often used in addition to a specific page-replacement algorithm. Several schemes:

- Keep a pool of free frames
  - When a page fault occurs, a victim page is chosen as before.
  - The desired page that needs to brought into memory is read into one of the free frame from the pool before the victim page is written out. Start the process immediately.
  - When the victim page is finally written out, the frame is added to pool of free framed
- Maintain a list of modified pages
  - Whenever a backing store is idle, a modified page is selected and written to the disk and its modified bit is reset.
- Possibly, keep a pool of free frame and remember which page was in each frame
  - If page is referenced again before the frame is reused, no need to load contents again from disk
  - Generally useful to reduce penalty if wrong victim frame selected

So, as it is out what will happen is that if this in the page tables the modified bit so, the this modified bit; so, it may become equal to 0 ok. So, if it was equal to 1 and now so the it was equal to 1, now this page is written back on to the disk. So, now, this bit is said to 0. So, later when a page faults occurs and we select this page as the victim this so, then we find that the modified bit is 0. So, I do not have to write it back onto the disk. So, it can start it can start immediately with the write operation on to that particular frame.

So, this is the we can maintain a list of modified pages and whenever this disk is idle so, we can write those modified pages back to the disk. So, that in subsequent page fault if those pages are selected as victim, we do not have to write back. So, this is the second policy. The third one says the possibly keep a pool of free frame and remember which page was in each frame. So, if page is referenced again before the frame is reduced, no need to load contents again from the disk. So, this third policy is slightly it is doing some sort of housekeeping job we can say. So, it is what it is doing is that it is we have got a pool of free frames ok. So, that pool that we were talking about.

(Refer Slide Time: 05:28)



Page-Buffering Algorithms

Other procedures are often used in addition to a specific page-replacement algorithm.

Several schemes:

- Keep a pool of free frames
  - When a page fault occurs, a victim page is chosen as before.
  - The desired page that needs to brought into memory is read into one of the free frame from the pool before the victim page is written out. Start the process immediately.
  - When the victim page is finally written out, the frame is added to pool of free framed
- Maintain a list of modified pages
  - Whenever a backing store is idle, a modified page is selected and written to the disk and its modified bit is reset.
- Possibly, keep a pool of free frame and remember which page was in each frame
  - If page is referenced again before the frame is reused, no need to load contents again from disk
  - Generally useful to reduce penalty if wrong victim frame selected

Pool: f_1, f_2, f_5, f_{10}

$P_4, pg5$ $\quad P_5, pg2$

$P_5 \rightarrow Pg2$

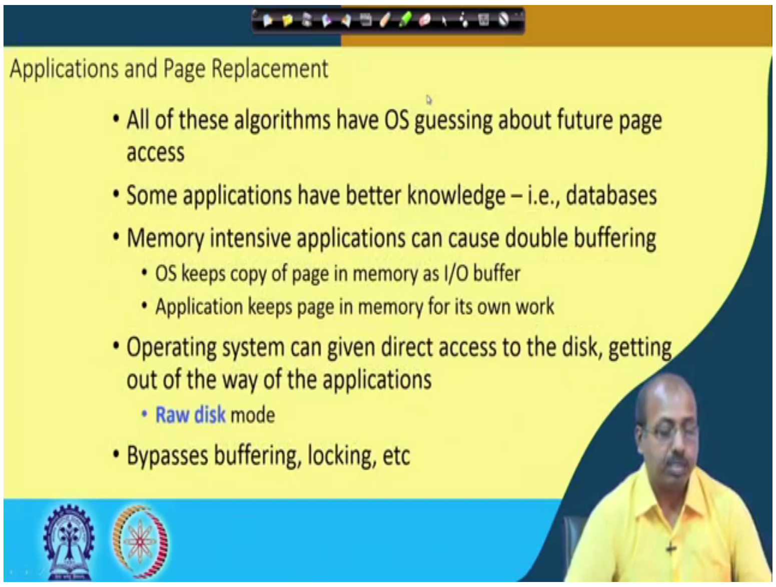
So, we remembered that these are the f 1, f 2, f 5, f 10, so these are the free frames, but not only that it also remembers what was the corresponding page in those frames. May be in frame 10, I have got process 5 page number 2. Similarly frame 5, I may have process 4 page number 5. So, like that we remember what is happening to the was there in the individual frames.

Now, when this page fault occurs suppose after sometime this process 5 creates a page faults and it is looking for page number 2. Now, if we find that in f 10, this process 5 page 2 was there, then I can just instead of loading the page from the secondary storage again. I can just load it on, I can just make f 10 make a part of the process. So, that way we can if the page is referenced again before the frame is used; no need to load contents again from disk.

Generally useful to reduce penalty if wrong victim page wrong victim frame selected, what can happen is that maybe if this page 5 is a very important page for page 2, so for process 2 sorry page 2 is a very important page for process 5. So, by mistake the page replacement algorithm has selected page 2 a process 5 to be replaced. Then what happens is that immediately after some time this process 5 will refer to page number 2. Now, if this page number two this free frame that is remembered that in f 10; it is there, then I do not need to refer to refer to that again.

So, it can very well happen that before this frame 10 was written by some other process so, this process 5 will again asked for page 2 and then it will be copied directly from there. So, we do not need to copy from the disk. So, the page is already there; I just make it part of the page table this frame 10 is made a part of the page table, so that can be done. So, these are some of the page buffering techniques that are useful to augment these replacement policies, so that we can use them for doing operations.

(Refer Slide Time: 07:51)



Applications and Page Replacement

- All of these algorithms have OS guessing about future page access
- Some applications have better knowledge – i.e., databases
- Memory intensive applications can cause double buffering
  - OS keeps copy of page in memory as I/O buffer
  - Application keeps page in memory for its own work
- Operating system can given direct access to the disk, getting out of the way of the applications
  - Raw disk mode
- Bypasses buffering, locking, etc

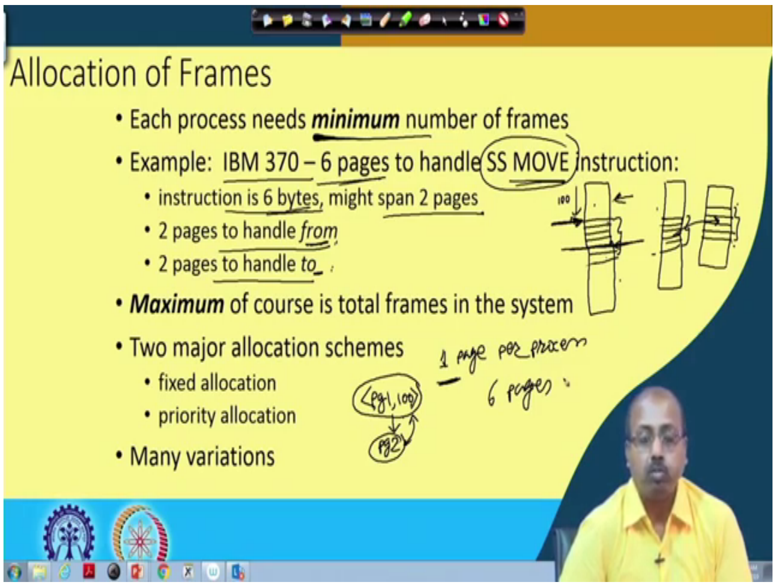
Applications and page replacement, all of these algorithms have OS guessing about future page access. So, what we are trying to do essentially is that we are trying to see like what can happen in future. Some applications are better knowledge for example, data bases. So, databases so, they actually try to go say modify data items for all the records in the file. So, as a result, they will go in a block by block fashion.

So, they have got better knowledge and memory intensive applications can cause double buffering. So, OS keeps copy of page in memory as IO buffer and application keeps page in memory for its own work, so that double buffering may occur. So, for input out output operation so, there is a buffer where this is done and the process may be working with some other another buffer.

So, operating system can give direct access to the disk getting out of the way of the application, so that is raw disk mode. So, some applications so, it may be allow to do disk read write operation. So, it can be the process can be allowed to directly write to some particular sector and all. So that way that is called the raw disk mode and it actually bypasses many of this disk access constant like the buffering, locking etcetera, so though they are all bypass.

So, actually if a process is writing on to the disk, so until and unless it becomes a full block so, it is not written. So, what the system does? Operating system does is that when the process is writing; it is it is written in some memory block. And when the memory block is sufficient for one disk write, then only this entire block is written on the disk. Now, that makes the process slow, so what is done is that is the raw disk access is given to the permission is given to the process, so the process can directly right onto the disk. So, that way this buffering and locking so, this type of OS related delays; they can be bypassed. So, that is another augmentation that we can do.

(Refer Slide Time: 10:01)



Allocation of Frames

- Each process needs ***minimum*** number of frames
- Example: IBM 370 – 6 pages to handle SS MOVE instruction:
  - instruction is 6 bytes, might span 2 pages
  - 2 pages to handle *from*
  - 2 pages to handle *to*
- ***Maximum*** of course is total frames in the system
- Two major allocation schemes
  - fixed allocation
  - priority allocation
- Many variations

Next we will be looking into some important question like how do we allocate frames to the processes. So, how many frames should be allocated to process? So, each process need some minimum number of frames. So, this is a very important thing to understand that each process need some minimum number of frame. For example, IBM 370 if you look into this particular architecture so, there is a move instruction. So, that requires at least 6 pages to be there. So, instruction is 6 bytes and that can span over 2 pages and 2 pages to handle from and 2 pages to handle to.

So, any address; so it may so, happen that this if this move instruction is the code is generated in such a fashion that it is a this is a 6 byte instruction; out of that may be 4 bytes come on to this page and 2 bytes come here. So, this whole thing is the move instruction. Now, when this instruction is say suppose this instruction fetches starts from this point and after getting the first byte, the processor understands that this is a 6 byte instruction, so it has to access all the 6 bytes.

Now, if the if this part if the second page is not there in the memory; the second page is not there in the memory, then it will create a page fault and if it creates a page fault; then after servicing the page fault the second page should be there in the memory. But it may so happened that by this time, this page has already been swapped out the by some other process; this page has already been swapped out.

Or if I say that I will I have got only one page allocated to the process; only one page per process, then also we are got difficulty because so, in initially this particular page has been allocated and then while executing while fetching this instruction; there is a page fault at this point. So, it has to load the second page, but since I am allowing only one page per process. So, this page will be loaded in the frame that contain the previous page. So, when we try to restart the instruction, then again there will be a page fault because the address is this particular address is not there in the memory now, so that way it goes on.

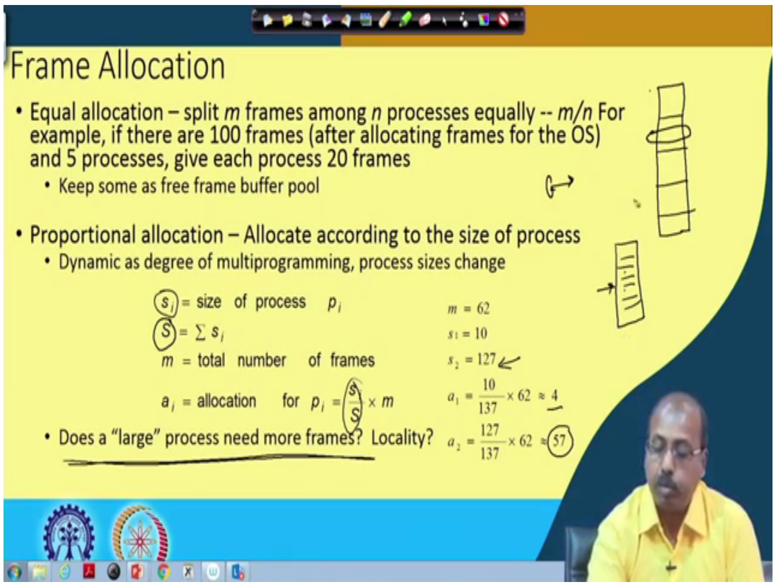
So, basically the first I was trying to access a page 1 this if this offset is say 100. So, page 1, 100, so this address I was trying to access and it created a page fault at this point. Then in this I have so, this page has been overwritten and I have loaded the page 2. Now, to restart the instruction again, I need to access this page 1, 100 because that was the program counter value at which we stopped, but again this is not there. So, again this page 2 will be replaced by page 1 100 and this will go on this will go on doing that thing, so I will not be able to proceed.

Second thing is that so this MOVE so, this is the there is a memory to memory movement. So, there is a from address and there is a to address and both these from and to so, they are they are they can this address can be spanning over two pages. So, this from address maybe spanning over 2 pages maybe I am moving from this particular this location this 4 byte locations out of the 2 bytes say 2 bytes maybe in one page 2 byte maybe in another page and this the 2 addresses. So, that can also span over 2 memory frames; out of that say 2 addresses may be here and 2 bytes maybe there total 4 byte. So, these 4 byte memory I want to copy here.

But this to from address so, that is spanning over two pages and this is also the two addresses also spanning over two pages. So, looking in the instruction set we find that if I looking into this SS MOVE instruction, then at least 6 pages must be there for the instruction to be executed properly. The 6 pages must have been loaded into the memory for this instruction to be executed properly.

So, I can say that for IBM 370, the minimum number of frames that should be allocated to it should be 6 ok. I cannot if I give it less than that there is a chance that the process will not be able to execute and it will just be go going on making page faults without making any progress. So, that is that is the issue and we can of course, the maximum number of pages that or frames that you can allocate to a process. So, there is that is determined by the total number of frames that we have in the system ok; so, that is there. Now so, we have got a minimum which is depicted by the instruction set of the processor and the maximum is depicted by the a number of physical frames that we have.

(Refer Slide Time: 15:41)



Frame Allocation

- Equal allocation – split *m* frames among *n* processes equally -- *m/n* For example, if there are 100 frames (after allocating frames for the OS) and 5 processes, give each process 20 frames
  - Keep some as free frame buffer pool
- Proportional allocation – Allocate according to the size of process
  - Dynamic as degree of multiprogramming, process sizes change

$\boxed{s_i}$ = size of process p_i

$\boxed{S} = \sum s_i$

m = total number of frames

a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$m = 62$

$s_1 = 10$

$s_2 = 127$

$a_1 = \frac{10}{137} \times 62 \approx \underline{4}$

$a_2 = \frac{127}{137} \times 62 \approx \boxed{57}$

  - Does a "large" process need more frames? Locality?

Now between these two so, we can follow any allocation policy and there are two major allocation schemes that are followed; one is called fixed location another is called priority allocation and of course, there can be many variations of these two. So, equal allocation; so we split m frames among n processes equally, so m by n. For example, if there are say 100 frames after allocating the frames to the operating system; if we are left with 100 frames and there are 5 process, then each process I can give 20 frames. Of course, we have to keep some free some for this free frame buffer pool. So, keeping those so, these it may not be 20 may be something less than that. So, that way I can do some do equal allocation or I can have some proportional allocation, so allocate according to the size of processes.

So, this is dynamic as degree of multiprogramming and process sizes change. So if s i is the size of process p i, then and S is the sum of all the sizes and m is the total number of frames that we have, then the allocation for p i is proportional to the size. So, this S is the size of sum of sizes of all processes. So, out of that so if s i is the size of the ith process, then this s i by S. So, this gives the fractional size of a process and based on that I give I multiply it by total number of frames and gives the proportional allocation for this process p i.

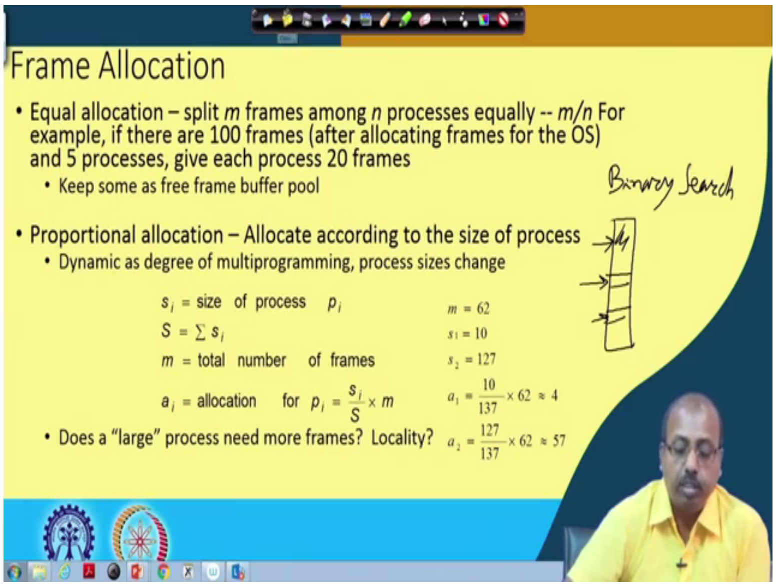
So, this for example, if m equal to 62 total of frame is 62 s 1 is 10, s 2 equal to 127 then a 1 is 10 by 137 into 62 that is around 4 whereas, a 2 is say 127. So, this is almost taking the entire space, so and that is allocated 57 frames. So, out if these 62, 57 are allocate to process a 2 and 4 allocated to process 1.

So, does a large process does a large process need more frames? So, this is of course, a question. So, it may or may not be true like it so happen that the a large process is doing some random reference to all the frames so, all the pages; so, in that case if all those pages are not there in the memory. So, that will create problem; so that will create more number of page faults.

At the same time so, there is something called program locality that we are talking about even if the program; even if a program or process is very large. So, maybe at if you look at some point of time so, and it is near future. So, it will be referring to a portion of the portion of the memory only.

So, that is if it is accessing say this instruction then in near future, it will be around this instruction only. Similarly, if it is access; if it is accessing some array and at some point of time. So, if it is accessing say this entry, then in near future it will be accessing the nearby entries, so that locality is there. So, it says that large process may or may not need this type of large number of frames because if it is using that locality principle then possibly it will hence less number of frames will be sufficient. But if a program is making random reference to ah this locations then of course, this is a problem. A typical example maybe that binary search algorithm ok.

(Refer Slide Time: 19:01)



Frame Allocation

- Equal allocation – split m frames among n processes equally -- m/n For example, if there are 100 frames (after allocating frames for the OS) and 5 processes, give each process 20 frames
  - Keep some as free frame buffer pool
- Proportional allocation – Allocate according to the size of process
  - Dynamic as degree of multiprogramming, process sizes change

s_i = size of process p_i

$S = \sum s_i$

m = total number of frames

a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$m = 62$

$s_1 = 10$

$s_2 = 127$

$a_1 = \frac{10}{137} \times 62 \approx 4$

$a_2 = \frac{127}{137} \times 62 \approx 57$

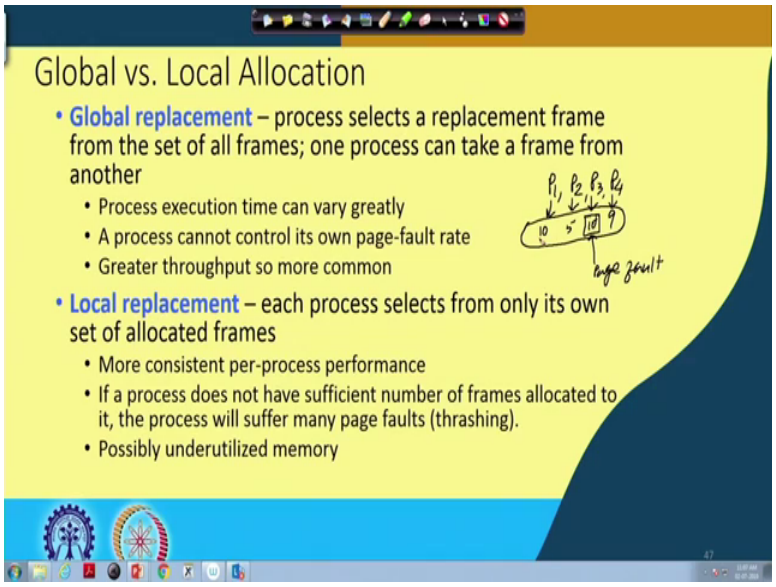
  - Does a "large" process need more frames? Locality?

So, in binary search algorithm what we do is that, we have got an array of numbers and to search for the number you probe at the middle of the array. And if the number is less than the middle the number search is less than this elements, then we search in this portion again probing at the middle; otherwise we probe at the middle of this lower part like that.

Now assuming that this array is very large, it spans over a large number of memory pages. Then this is an example where this access pattern is really random, depending on the data value and data values of this array and the number that we are searching so, this access maybe a very random sort of thing. So, in that case this program does not show any locality behavior. So, that way so, it will be required that all this frames are all these pages are there in memory frames; so, that number of page faults are less.

So, that in that case of course, ah this large process means that they it will need more number of frames. So, this is basically problems specific for some problem the locality will be followed and then less number of frames should be sufficient for some processes, the locality may not be followed and then we will really required large number of frames allocated to their process.

(Refer Slide Time: 20:24)



Global vs. Local Allocation

- **Global replacement** – process selects a replacement frame from the set of all frames; one process can take a frame from another
  - Process execution time can vary greatly
  - A process cannot control its own page-fault rate
  - Greater throughput so more common
- **Local replacement** – each process selects from only its own set of allocated frames
  - More consistent per-process performance
  - If a process does not have sufficient number of frames allocated to it, the process will suffer many page faults (thrashing).
  - Possibly underutilized memory

Then this global verses local allocation. So, we follow we there is something called a global replacement policy like when you have selecting a page to be replaced so, then where do we search like a process selects a replacement frame from the set of all frames and one process can take a frame from another. So, this so, maybe in my system there are 4 processes now p 1, p 2, p 3 and p 4 and each of them has got some frames associated with them maybe p 1 has got 10 frames associated with, p 2 has got 5 frames, p 3 has got say 10 frames, p 4 has got say 9 frames associated with it.

Now, suppose p 3 generates a page fault; so, p 3 generates the page fault, now and we find that there are no free frames available. Then how this p 3 will select a victim page? So, for selecting a victim page, one possibility is that I look into the 10 pages that p 3 has with itself. So, that is one possibility and the other possibility is you look into all this pages and then try to select a page. So, when it is looking into the pages of the particular process only so, this is called local replacement policy. When it is looking into all the pages of all the processes and trying to select a victim so, that is called the global replacement policy.

So, in local in the global replacement policy, the process selects a replacement film from the set of all frames and one process can take a frame from another. So, naturally the process execution time can varies a greatly because it may so, happen that when there are less number of processes in the system, then the free frames are there and a processes pages frames are not grabbed easily by other processes ah. So, that the number of page faults that the that a particular process c is maybe low; so that is there.

But in case of large number of processes in the system, so fighting for this frames will be more and each process may try to grab frames which are allocated to other processes. As a result the processes they may see larger delay because there may be less number of frames available with processes and the process frames may be written by somebody some other process and the corresponding page getting swapped out. And so, naturally when this generates the page fault again the page has to be loaded. So, that makes the total time total execution time of this process significantly high.

So, process execution time can vary greatly depending on the load on the system. A process cannot control its own page fault rate because even if this process is a well behaved process some other processes they may grab the frames held by this process. So, as a result, it process cannot control its own page fault rate.

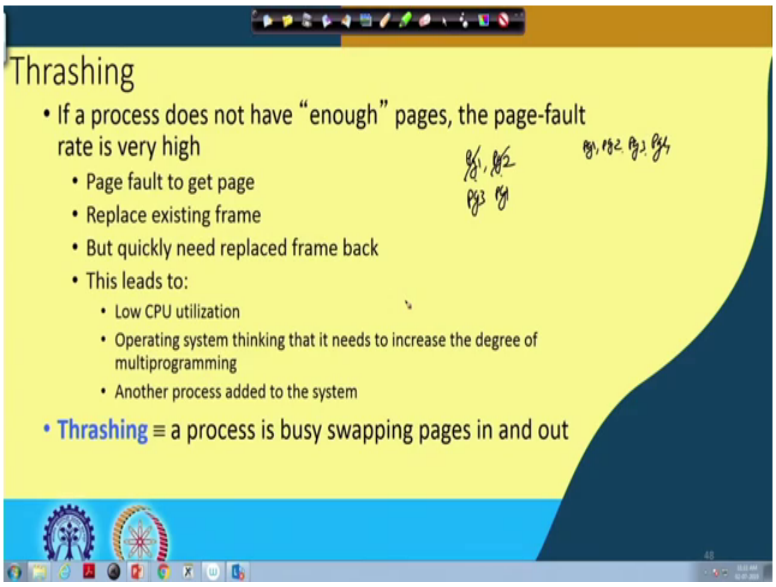
And a greater throughput so, more common because this global replacement since we are trying to give frames to a process which really in need compare to the process which as which as which does not need so many frames. So, this it is trying to improve the throughput. So, throughput maybe higher in case of global replacement, but this execution time for the processes may vary. So, we cannot be sure like within how much time a process will be done and a process cannot control its own page fault rate.

On the other hand this local replacement, each process selects from only its own set of allocated frame. So, whatever frames are allocated so, it will try to take it from there only. More consistent per process performance so, for a particular process since I know that I have been given 10 frames so, my replacement will also be within this 10 frames only. So, that way per process performance is better. If a process does not have sufficient number of frames allocated to it, the process will suffer many page faults; that is called thrashing. So, we will come to this term thrashing slightly later.

So, if a process does not have sufficient number of frames, then definitely it will generate large number of page faults ok. For example, for if we find that for considering the locality and all; for a process maybe it requires at least 4 pages for smooth running of the process. But if it is allocated only say two pages, then there will be large number of page faults generated, so that situation will be called thrashing. And possibly underutilized memory, so maybe so this is the other extreme like a process has been given large amount of large number of frames allocated to it, but it is unable to use them.

But since it is a local replacement policy, no other process can grab this frames also. So, as a result this memory becomes underutilized so, those frames cannot be used by another process. So, that is the other difficulties that we have. So, these local replacement policy so, it has got this problems. So, that way this global replacement is generally followed, but global replacement has got this load variation. So, the load is changing, and then there will be difficulty.

(Refer Slide Time: 25:56)



Thrashing

- If a process does not have "enough" pages, the page-fault rate is very high
  - Page fault to get page
  - Replace existing frame
  - But quickly need replaced frame back
  - This leads to:
    - Low CPU utilization
    - Operating system thinking that it needs to increase the degree of multiprogramming
    - Another process added to the system
- **Thrashing** ≡ a process is busy swapping pages in and out

Next we come to the concept of thrashing. So, if a process does not have enough pages, the page fault rate becomes very high. So, as I was telling that if a process for its smooth running required say 4 pages and it has been given only 2 pages, then so, it has been given two pages say page number 1 and; so, page 1 and page , so they are loaded initially. Now, but ideally for the process to run properly, this page 1, page 2 page 3 and page 4; so, all of them should be there in the memory ok.

Now, see this initially page 1, page 2 is there, but it will page 1 page 2 while they are executing so, they will refer to page 3. Now, if it refers to page 3 so, this page 1 is replaced by page 3 for example, then again this page 1 will again be required. So, that way again this page 2 will be replaced it will become page one then again page two will be referred; so, or page 4 will be referred. So, that way continually the process will generate large number of page fault.

So, if a process does not have enough pages the page fault rate becomes very high. So, page fault to get page and then it will replace existing frame, but quickly need the replaced frame back. So, as I was explaining that page 3 is required so, it replaces page 1, but that replaced page on is again needed very shortly.

So, these needs to low CPU utilization; so, though I have got a large number of processes in the system the CPU utilization will be low and operating system thinks that needs to increase the degree of multiprogramming. Since this CPU utilization has become low, then the OS things that I do not have enough processes in the system. So, that way it tries to push in more number of processes. The long term scheduler is invoked and it brings more number of processes from the new state to the ready state. And as it tries to do that, it has to allocate memory to this new process that have been introduced. So, that way the so, another process added to the system and that increases these thrashing further.

So, that it goes on so, this becomes a positive feedback sort of thing and this fighting between the processes for pages, so that even goes up. So, thrashing is a process it is a busy swapping pages in and out. So, most of the time the system is actually doing this disk io by doing this swapping out of this pages. So, that way the system does not do anything very significant in the computation. So, it just goes on doing this paging in and paging out, so that takes large amount of time.

So, this has to be solved and for the good system performance, we have to understand that the thrashing has got initiated into the system and we should not try to increase the degree of multiprogramming rather try to solve this thrashing problem. So, in the next class, we will see how this thrashing can be solved; how it can be used for it can be solved and then this number of page faults can be reduced and system performance can be improved. We will continue with this in the next class.