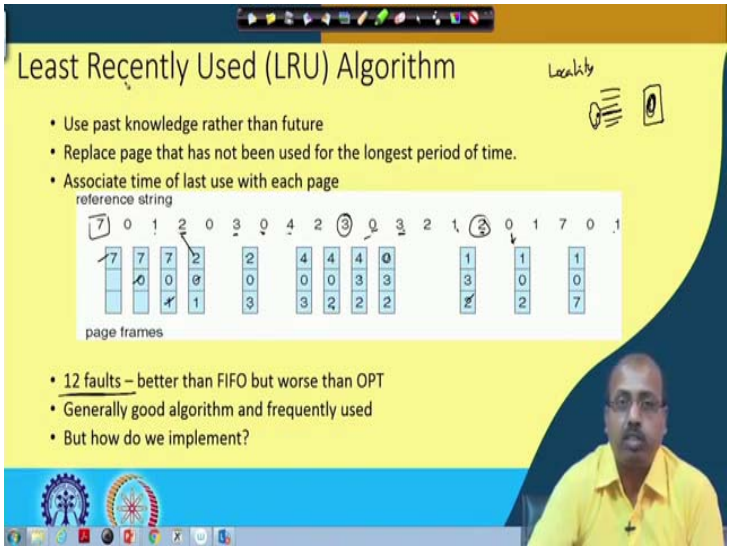
Operating System Fundamentals
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 53
Virtual Memory (Contd.)

The next algorithm that we will look into for this page replacement is the Least Recently Used or LRU algorithm.

(Refer Slide Time: 00:31)



Least Recently Used (LRU) Algorithm

- Use past knowledge rather than future
- Replace page that has not been used for the longest period of time.
- Associate time of last use with each page

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7	7	7	2	2	4	4	4	0	1	1	1
	0	0	0	0	0	0	3	3	3	0	0
		1	1	3	3	2	2	2	2	2	7

page frames

- 12 faults – better than FIFO but worse than OPT
- Generally good algorithm and frequently used
- But how do we implement?

So, it uses past knowledge rather than future. So, previously we have seen that optimal replacement policy. So, that is fine, but we cannot look into the future, but we can always look into the past ok. So, looking into the past so, we try to see whether we can try to predict like behaviour of a program. So, it does not change all on a sudden.

So, based on that principal so, we try to predict like what can be the possible future references and based on that we try to select the victim page. So, it uses past knowledge rather than future and replace the page that has not been used for the longest period of time. So, if a page has not been referred to for the longest period of time, then it is very much likely that compared to other pages. So, this page is not going to be referred to in the near future as well. So, replace page that has not been used for the longest period of time and associate time of last use with each page.

So, this information has to be kept with each page like when it was last accessed. So, suppose this is my reference string 7 0 1 2 etcetera. So, if you are following this LRU or least recently used policy, for the first few pages so, there is nothing very interesting because this page was not there and frames are also free. So, 7 has been loaded here, then this 0 has been loaded here, 1 has been loaded here.

Now comes this page number 2; now page number 2 when we are looking into. So, which page is least recently used. So, in the recent past you see that this page number 7 it has been used long back. So, it is not being used in the recent past. So, we can replace this page number 7 by this page number 2. So, that way we try to select a victim. So, basically we look back and try to see which page has not been referred to for the farthest amount of time or largest amount of time then comes. So, this 7 is replaced by 2 here, then this 0. So, 0 is already there; now comes 3.

Now if you look back this pages 2 and 0; they have been referred to in the near past, but this page number 1 has not been referred to in the near past. So, this 1 is replaced by 3 then comes this 0. So, this 0 is already there, then comes this 4. Now this 4 again the page 2 has not been referred to in the near past. So, that 2 is this 2 is replaced by 4, then again there is a reference to page 2.

Again looking back so, out of this 4 0 and 3. So, 3 has been referred long back. So, this 3 is replaced by 2, then again there is a reference to 3 and again looking back out of 4 0 2. So, 0 has been referred long back. So, this 0 is replaced by 3, then again there is a reference to 0 and looking back 4 has been referred long back. So, this 4 is replaced by 0, then this 3 2; so, those pages were already there are no page fault.

Then at 1 so, gain if you look back out of this 0 3 2. So, 0 has been referred long back. So, 0 is replaced by 1. So, it becomes 1 3 2 then there is a reference to 2 and so, that is fine 2 is already there. Now there is a reference to 0 now this when this reference to 0 is there. So, this is it again looking back, so, 3 is here. So, 3 is replaced by 0 and then after that 1 is already there. Now comes 7. So, out of this 1 0 and 2, so, this 7 is 2 is the farthest page that has been referred 2.

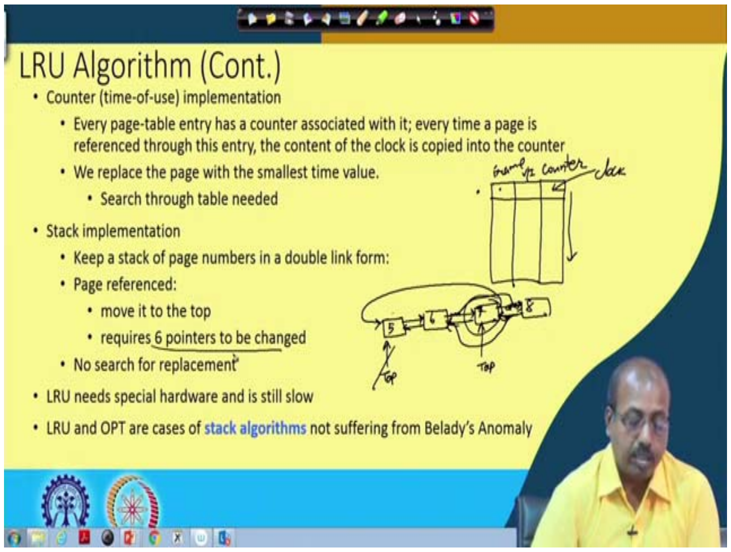
So, this 2 is replaced by 7. Now this 0 and 1 so, they are there. Now how many page fault? So, this is gives rise to 1 2 3 4 5 6 7 8 9 10 11 12 page faults. So, in the FIFO, I have replacement. So, we had more number of page faults compared to that here we have got less number of page faults. So, better than FIFO, but worst than the OPT. So, generally it is a good algorithm and frequently used because it is the relying on this process behaviour. So, we will see after sometimes that there is a concept of process locality.

So, if you look into a programs behaviour; so, if it is referring to some code so, it is referring to some locations here then it is very much likely that in near future. So, it will also refer in the vicinity only. So, if this page if this particular instruction belongs to say this particular page then in the near future so, this will be referring to the process will be referring to this page only. So, it is better that if in near future in the near past some page has been referred to. So, it is better that we do not replace that page because there is a there it is likely that the page will again be referred to in the near future.

So, whichever page has not been referred to for a long time so, that page we can possibly replace by the new page; so,. So, that is the problem locality behaviour; so, if since because this locality is there so, it is expected that we should be we can be able to do better if this is if this algorithm can be implemented. But the question is how do we implement this algorithm?

So, how do you remember like which page was referred when so, we have to have some as timing associated with the pages and that makes the algorithm implementation a bit costly compared to say FIFO algorithm where we can just keep a simple first in first out queue of the page numbers and from there we could very easily select the victim page. But here that is not so, we have to remember the time part of timing of this reference.

(Refer Slide Time: 06:32)


LRU Algorithm (Cont.)

- Counter (time-of-use) implementation
  - Every page-table entry has a counter associated with it; every time a page is referenced through this entry, the content of the clock is copied into the counter
  - We replace the page with the smallest time value.
    - Search through table needed
- Stack implementation
  - Keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed
  - No search for replacement
- LRU needs special hardware and is still slow
- LRU and OPT are cases of stack algorithms not suffering from Belady's Anomaly

So, one possibility is that we use a counter based counter for the pages. So, every page table entry has a counter associated with it and every time a page is referenced through this entry, the content of the clock is copied into the counter. So, basically what it says is that we have the page table and in the page table. So, we store we are storing the frame number; we are storing the frame number and the valid invalid bit and also we keep another entry which is the time value or the counter value.

So, whenever a particular page is referred to so, this page has been referred; then this particular counter so, we copy the clock value the current clock value. So, in that way at any point of time so, if you have to find which page to replace. So, you find out the page which has got the minimum clock time. So, this content of the clock is copied to the counter field and you replace the page with the smallest time value and naturally we need to search through the table.

So, this is another problem. So, whenever you are going to do page replacement we have to search through this entire table to find out which page has got the minimum timing. So, that is the counter based implementation another possible implementation is to have a stack based implementation keep a stack of page numbers in a doubly link form. If the page is referenced, they move it to the top and then so, that way we have got a situation like this.

So, we suppose at some point of time we have got the pages referred to like say 5 6 7 8. Suppose I have got 4 memory frames and it is kept in a doubly link list ok. So, this is the;

this is the top of the stack. Now suppose my next reference is for page number 7. So, what I have to do is that I have to bring this page number 7 to the top of the stack.

So, this top has to go here. So, that can be done, but I need to modify some of the pointers. So, this since this was at the top. So, this was the; this was the next referred most referred page. So, what we do is that we take out this pointer and make it point to this 5 and then; from this 5 was pointing to 6 so, 6 will be made to.

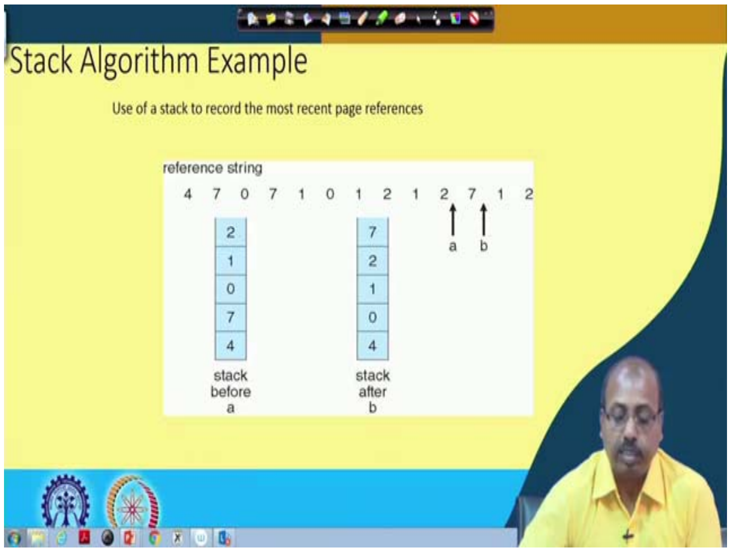
So, this pointer is this. So, this pointer will be made to point to 8 this pointer will be made to point to 8 and so, this pointer. So, this becomes 7 5 6 8 so, this side is done and on the reverse side. So, this is a 8 is the last one. So, this pointer has to be modified now. So, this 8 was a so, this 8 should point to 6 ok. So, 8 should point to 6. So, this pointer has to be modified and it will point to point like this. So, that way you can modify this page this set of pointers so, that this 7 comes to the top of the stack.

So, you can it can be shown that by doing 6 pointer manipulation so, this can be done. So, you can bring this page that is referred next to the top of the stack. So, the advantage that you get is the top is always pointing to the top of the stack. So, as a result so I do not need to search unlike the previous example of where I need to search through the table to find out the entry with minimum clock time. So, that will not be required. So, that is a stack based implementation.

So, naturally whatever we do it is not very simple this LRU implementation is not very simple. So, it requires some special hardware and it is a still, but it is still slow why because we need to have some either some counter and this counter has to be there and particularly as we know that some part of the page table is kept in the associative memory. So, there this TLBs so, this TLBs should accommodate for this timing and this clock timing and all or if it is a pointer this pointer manipulation has to be done. So, that way it takes time.

So, this that way we have to have some special hardware. So, possibly the memory management unit it will have the corresponding hardware built in so, that this LRU can be supported. Now LRU and OPT; so, these are cases of stack algorithms they do not suffer from Belady’s anomaly. So, FIFO is also there, but FIFO suffers from Belady’s anomaly, but LRU and OPT. So, these are stack algorithms and it can be shown that whenever you have got a stack algorithm so, they do not suffer from this Belady's anomaly.

(Refer Slide Time: 11:45)



Stack Algorithm Example

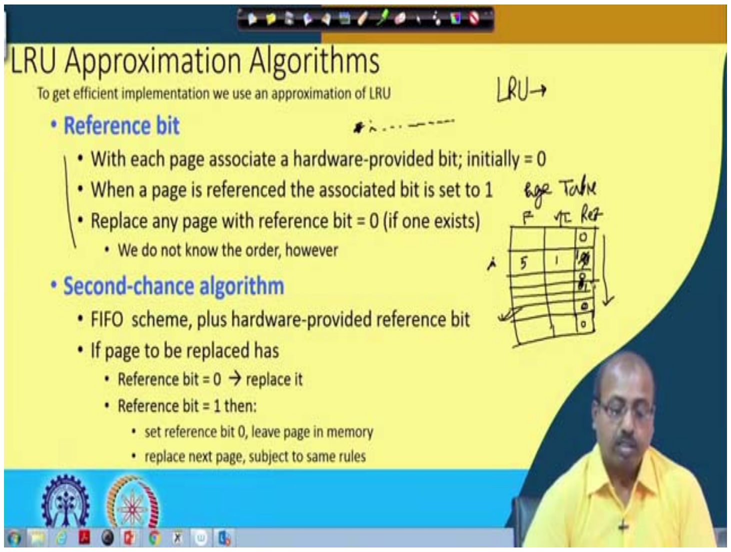
Use of a stack to record the most recent page references

So, this is an example of stack algorithm. So, use of a stack to record the most recent page references. So, suppose this is the situation so, this so, up to this point a. So, if you are looking at this point a suppose this was the situation.

So, this 2 was the most recently accessed page then it was 1, then 0, then 7 and then 4. So, this is the stack, now after this; after this page 7 has been accessed that is reference after this point access 7 so, the we have got 7 as the topmost ones. So, it is 7 2 1 0 4 so, that entire thing gets rearranged. So, it becomes 7 2 1 0 4, so, after this stack after b. So, this we can have a stack to record the most recent page number page references and we keep them in this stack.

So, how this stack will be implemented that is one issue? But if we can do that then definitely this stack algorithms can be realised and that can be helpful for running this page replacement algorithms.

(Refer Slide Time: 12:57)



LRU Approximation Algorithms

To get efficient implementation we use an approximation of LRU

- **Reference bit**
  - With each page associate a hardware-provided bit; initially = 0
  - When a page is referenced the associated bit is set to 1
  - Replace any page with reference bit = 0 (if one exists)
    - We do not know the order, however
- **Second-chance algorithm**
  - FIFO scheme, plus hardware-provided reference bit
  - If page to be replaced has
    - Reference bit = 0 → replace it
    - Reference bit = 1 then:
      - set reference bit 0, leave page in memory
      - replace next page, subject to same rules

Now, since LRU implementation is costly so, there are some approximation algorithms for the LRU. So, one of them is known as the reference bit algorithm. So, reference bit algorithm it says that with each page associate a hardware provided bit which is initially 0. When the page is reference associated bit is set to 1 and replace any page with reference bit equal to 0 if it exist, but we do not know the order however.

So, it is like this. So, basically we have got this we have to what the LRU algorithm was trying to do. So, it was trying to select a page which is which has been referred long back ok. Now so, if a page so, whenever this page is; so, at any point of time; so, if we look into the page table. So, if we have so, if we look into the page table then we have got apart from this frame number and this valid invalid bit. So, we have got another bit. So, which we call the reference bit ok.

So, this reference bit is initially set to 0. So, whenever a particular page is load into this frame, suppose in this particular page i is loaded into frame 5. So, this is valid and this reference bit is set to 0 ok. So, this is not yet referred to sometime later when this page will be referred to so, this bit will be set to 1. Now after sometime so, if we have to search out a page which is going to be selected as victim what we can do we can search through this page search through this table looking for a page whose reference bit is set to 0.

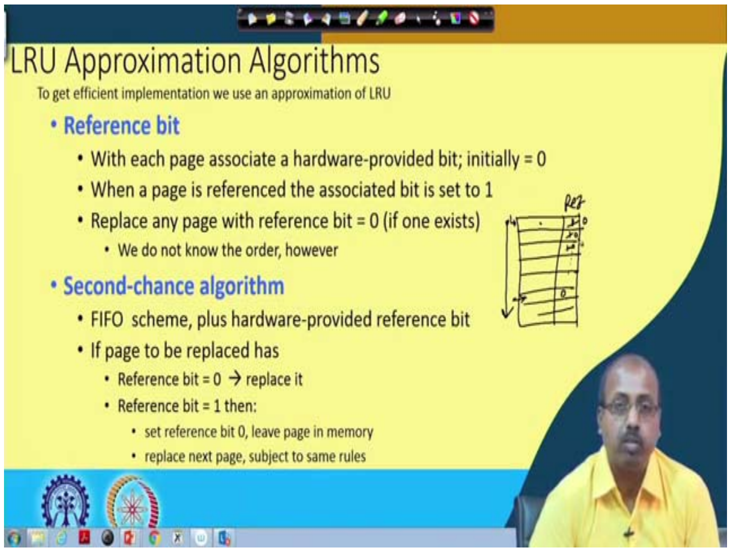
And if the reference bit is 0; that means, that the page has not been referred to so, naturally it is by means of by the principle of locality of a program reference. So, it is not going to be referred to in the near future. So, we can select this page as the victim page. Now one thing that we should try to understand that suppose this particular page was referred long back. So, this page number 5. So, it was referenced long back; page number i, it was referenced long back. So, this bit was set to 1 and after that lot of time has passed. But whatever we have written here so, it does not give us any provision to note that the page i has not been referred to in the near past.

So, what we can possibly do is periodically we can set all these reference bits to 0 or we can reset all this reference bits to 0. So, periodically we reset all these bits to 0 ok. Then as the program is running so, the it will generate page references and if this page i is really an useful page, then very soon the bit will be turned to 1. This reference bit will turn to 1. So, some other page so, if this is another very important page then within a small amount of time. So, this bit will also be becoming equal to 1. So, that way the pages which are really important for the process they will possibly become 1 within a very short while after they had been reset.

So, when a page is referenced the associated bit is set to 1 and replace any page with reference bit equal to 0; if 1 exists. Of course, we do not know the order in which so, the it is not the full LRU because it is not keeping track of which page was not referred to for a long time. So, it is treating all of them equally ok. All the pages whose reference bit is 0 so, they have got equal chance of getting replaced getting selected as victim. So, that is of course, a problem. So, it is not an LRU algorithm so, this an this an approximation to the LRU.

So, another appropriation that you have is known as the second chance algorithm. So, in second chance algorithm what happens is that we have got this first in first out policy plus one hardware provided reference bit. If page is to be page to be replaced has reference bit equal to 0, we replace it; otherwise reference if the reference bit is 1 in that case, you set the reference bit to 0 and leave page in memory and replace next page subject to same to same rules.

(Refer Slide Time: 17:41)



LRU Approximation Algorithms

To get efficient implementation we use an approximation of LRU

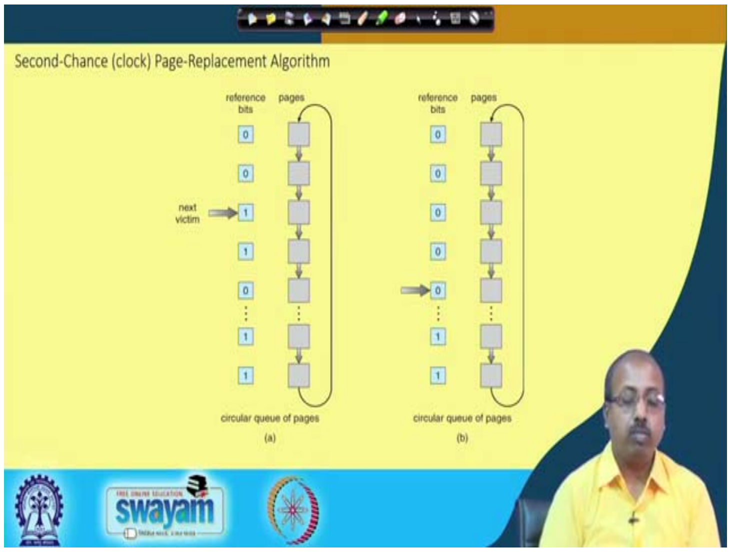
- Reference bit
  - With each page associate a hardware-provided bit; initially = 0
  - When a page is referenced the associated bit is set to 1
  - Replace any page with reference bit = 0 (if one exists)
    - We do not know the order, however
- Second-chance algorithm
  - FIFO scheme, plus hardware-provided reference bit
  - If page to be replaced has
    - Reference bit = 0 → replace it
    - Reference bit = 1 then:
      - set reference bit 0, leave page in memory
      - replace next page, subject to same rules

So, here what happens is that we have got this page table and there is a reference bit. So, there is a reference bit. So, this is otherwise following the FIFO policy. So, supposed I have got these pages stored in this order. So, this is the oldest page. So, it is increasing in this is the oldest page and then it goes like this. And initially so, whenever a page is referred to so, that bit is set to 1 suppose all this bits are 1 means these pages has been referred. So, if this page has been referred to so, even if this is the topmost entry in the this is the oldest entry in the memory than it does not get replaced because this reference bit is 0.

So, rather what we do is set this reference bit to 0. So, if we search out a page for which the reference bit is 0 ok, then that page will be in replaced. So, this particular page will be replaced and in the, but while looking for this. So, all this bits they will be so, all this reference bits o they will be turned to 0. So, we look into the reference bit; so, if we find that the reference bit is equal to 1 so, we do not replace that page.

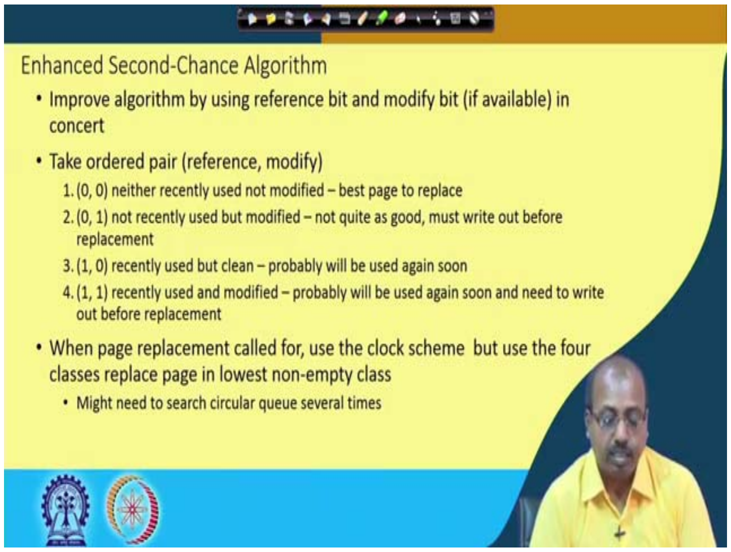
We go to the next page in the order, but before going to that so, we leave we make the reference bit of this particular page equal to 0. So, this is known as a second chance algorithm. So, the name comes from the fact that the page which has got referred to. So, it gets another chance before being replaced ok. So, in between the page may be referred to again ok. So, if it t is referred to then it will again become equal the bit will become equal to 1. So, that way it gets another chance to survive.

(Refer Slide Time: 19:28)



So, this is the situation. So, suppose the next victim is this one ok. So, this pages are there in the queue like this the circular queue. So, the next victim is say this particular page, but since this page reference bit is equal to 1 so, it is not replaced. So, rather so, the reference bit is reset to 0 and we come to the page whose reference bit is 0 at present. So, this is the thing. So, it comes to this one and this page gets replaced. So, this is how the second chance or clock algorithm works. So, this has got another name called clock algorithm. So, this is the page replacement algorithm works like this.

(Refer Slide Time: 20:07)



Enhanced Second-Chance Algorithm

- Improve algorithm by using reference bit and modify bit (if available) in concert
- Take ordered pair (reference, modify)
  1. (0, 0) neither recently used not modified – best page to replace
  2. (0, 1) not recently used but modified – not quite as good, must write out before replacement
  3. (1, 0) recently used but clean – probably will be used again soon
  4. (1, 1) recently used and modified – probably will be used again soon and need to write out before replacement
- When page replacement called for, use the clock scheme but use the four classes replace page in lowest non-empty class
  - Might need to search circular queue several times

There is another enhanced second chance algorithm that improves the algorithm by using reference bit and modified bit in concerts. So, that uses two bits; reference bits and modified bit. So, actually if you look into the reference bit what it says is that for a page if the reference bit is set then that page has been referred to. On the other and the modified bit so, it was telling that whether the page has been modified or not. And this modified thing; so, if a page has not been modified, then while doing the replacement I do not need to write back the page on to the secondary storage because there is no writing done on that page.

On the other hand, if the modified bit is equal to 1, then when the page is if the page is chosen as a victim to be replaced by another page; it has to be first written back on to the disc. So, if you look into this reference bit and modified bit; so for a particular page; so, if both the bits at 0; that means, it is neither recently used nor modified. So, it was loaded into the memory long back and after that it has not been referred to in the recent past.

So, this is the best page to be replaced because I do not have it is not referred to in the near past and it there is no modification also, then I do not need to write back. Then this 0, 1 so, this is not recently used; but modified ok. So, it is not quite good quite as good as this previous category that is 0, 0 category because we have to write out before replacement.

And 1, 0 so, this is recently used, but it is a clean page. So, I do not have to write back and but since it is one it has been referred to so, it is likely that it will be referred again in the near future. And 1, 1 so, recently used and modified; so, this is possibly the worst page to be modified to be replaced because it has been referred to and it has also been modified. So, probably will be used again soon and also, if I select it for as a victim, then I will need to write it out before replacement. So, which page can we select for replacement? Naturally 0 0 is the best category because that is clean and that is not referred to in the near past.

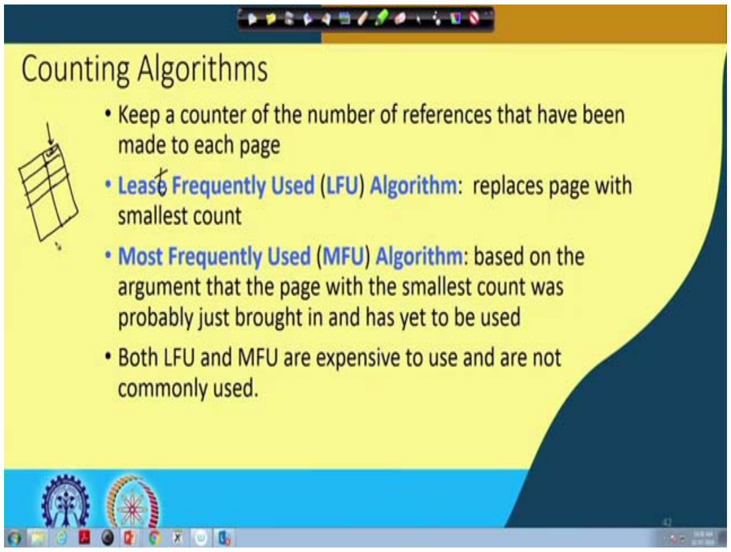
So, that is the 0 0 page if we can find so, that is that can be replaced. So, if we do not find 0 0, then 0 1 is the next good option because it has not been referred to in near past. So, naturally it is likely that it is not going to be used again in the near future. But it has to be modified though we have to take care of that writing part.

Then we have got this 1, 0 categories. So, this is recently used, but clean. So, the; so, it may be used again soon. So, if we so basically I can go in this order. First I try to select a page of this category 0, 0; if I do not find anybody then I look into the category 0 1.

Again if I do not find anybody, then look into the category 1 0 and again 1 1. So, go in this order. So, when page replacement is called for use the clock scheme, but use the 4 classes replace 4 classes replace page in the lowest non empty class. So, we have got this circular queue and that previously that clock algorithm that we looked; so, this clock policy second chance policy is followed ok.

But we are looking into instead of only the referred bit. So, we also look into this modify bit and the category. And we may need to search the circular queue several times for finding the a page of suitable type.

(Refer Slide Time: 23:59)



Counting Algorithms

- Keep a counter of the number of references that have been made to each page
- Least Frequently Used (LFU) Algorithm: replaces page with smallest count
- Most Frequently Used (MFU) Algorithm: based on the argument that the page with the smallest count was probably just brought in and has yet to be used
- Both LFU and MFU are expensive to use and are not commonly used.

Next we look in to the counting algorithm. So, it keep a counter of the number of references that have been made to each page. So, that is the another very interesting algorithm. So, any all these counting algorithm; so, they keep a counter of the number of times a number of references that have been made to each page.

Then so, we have got; we have got two important algorithms in this category; one of them is known as the Least Frequently Used or LFR algorithm. So, replace page with the smallest count. So, every page table entry it has got a counter holding the number of references of course, there is an issue of this counter size and all; so, the overflow of counter and all. So, assuming that if there are; so, assuming that we are checking the counter quite frequently and then maybe we are remember; we can give sufficient number of bits to the counter or maybe after sometime we will reduce all the counter values to um modulo something.

So, that way that the modulo operation is done together for all the counter bits all the counter entries of all the pages then of course, we can do that. So, can we can have this counter overflow problem can be resolved. What I am mean trying to say is like this that if I have got say every entry in the page table, it has got a counter; it has got a counter.

Now whenever the corresponding page is referred to so, this count value is incremented. Now this count has got a finite size. So, after sometime this counter may overflow. So, to avoid that so, if I periodically look into all these counter values and we can just normalise them or we can do all of them round to some 0 modulo operation, then I can get rid off that overflow problem. So, that way I can have this least frequently used algorithm. So, that will replace the page with smallest count. We can also have most frequently used algorithm. So, based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

So, least frequently used policy so, it says that the since the count value is low. So, it has been referred to for the least amount of time; so, least number of time. So, naturally this page if I replace, then it is expected that in it is not going to hamper system performance significantly because this page will not be referred to that much. However, the counter logic is like this may be a page has got a low count value because the page was loaded very recently. So, it has not been referred to in the near past because it is not old enough.

So, in that from that angle so, most frequently used algorithm is another possibility. So, if a page has got the maximum frequency of reference. So,; that means, this is possibly has been used to a large extent and it is not going to be referred that much in the near future. So, that way the most frequently used maybe the other category. So, both LFU and MFU are expensive to use and are not commonly used because of this counter overflow problem that we discussed. So, both this LFU and MFU has got this thing and as we have seen that both of them have got their counter logic like which one is going to be better and all.

So, this counter based algorithms are not that much useful. So, in most of the systems we will find this LRU policy, second chance policy and because of this ease of implementation the FIFO policy is also supported. So, there are more complex algorithm where it will try to take a combination of many of these policies and that way it will come up with even more complex page replacement policy like which are actually supported by the hardware.

What we see that all these page replacement algorithm so, they need some amount of hardware support. So, until and unless your memory management unit provides those supports so, it is not possible for an OS designer to choose upon a particular page replacement scheme. So, we will continue with this in the next class.