Page Replacement: Page replacement takes the following approach. If no frame is free, we find one that is not currently being used & free it. We can free a frame by writing its contents to swap space & changing the page table to indicate that the page is no longer in memory. We can use a free frame to hold the page for which the process faulted.

Basic Algorithm:

1) Find the location of the desired page on the disk.
2) Find a free frame:
   a. If frame is free use it.
   b. If there is no free frame, use a page replacement algorithm to select a victim frame
   c. Write the victim frame to the disk, change the page & frame tables accordingly.
3) Read the desired page into the newly freed frame; change the page & frame tables.
4) Restart the user process.

* Notice that, if no frames are free, 2 page transfers (one out & one in) are required. This situation effectively doubles the page-fault service time & increases the effective access time accordingly.

Solution: We can reduce the overhead by using a modify bit (or dirty bit). When this scheme is used, each page or frame has a modify bit associated with it in the hw. The modify bit for a page is set by

the hlw whenever any word or byte in the page is written into, indicating that the page has been modified. When we select a page for replacement, we examine its modify bit. If the bit is set, we know that the page has been modified since it was read in from the disk. In this case, we must write the page to the disk. If the modify bit is not set, however the page has not been modified since it was read into mem. In this case, we need not write the mem. page to the disk, its already there.

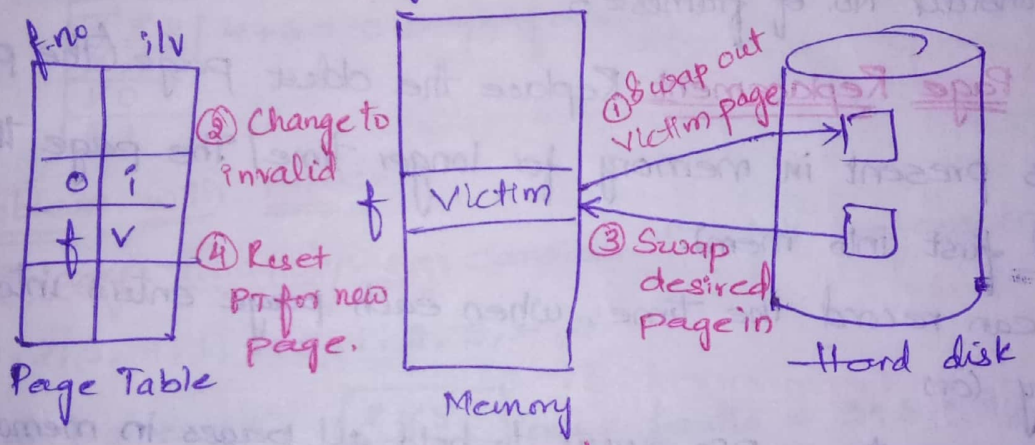


Fig. Page Replacement

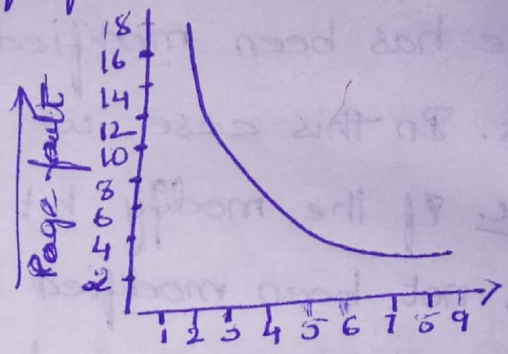
Problems in implementing Demand paging:

1) Frame-Allocation Algs: If we have multiple processes in mem., we must decide how many frames to allocate to each process.

2) Page Replacement Algs: When free frames are not available, we must select the frames that are to be replaced.

* We have to select a page replacement alg. in such a way that it gives lowest page fault rate

Page faults Vs No. of frames: Generally, when the no. of frames allocated to a process is increased, then the page faults will be red. The following is the general graph which represents page faults vs No. of frames.



Page Replacement Algorithms:

We illustrate the page replacement algorithms with the help of following reference string.

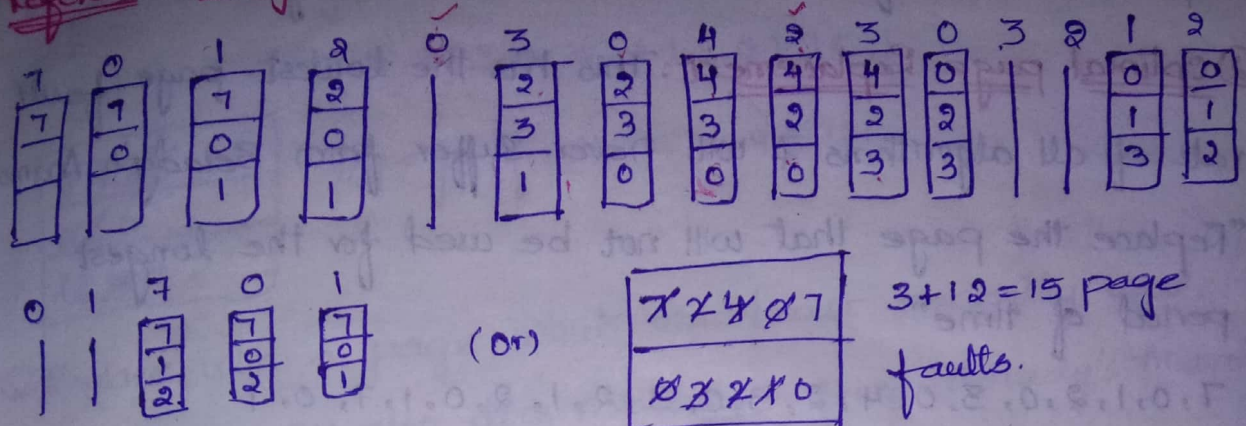
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

We consider No. of frames = 3

1) **FIFO Page Replacement:** Replace the oldest page (The page that is present in memory for longer time / The page that arrived first into mem).

- We can record the time, when each page enters into memory (or)
- We can create a FIFO queue to hold all pages in memory. We replace the page at the head of queue, when the page is brought into the mem, we insert it at the tail of the queue.
- For our example reference string, our 3 frames are initially empty. The first 3 references (7, 0, 1) causes page faults & are brought into mem. (these empty frames). The next reference (2) replaces 7, because 7 was brought in first. Since (0) is the next reference & (0) is already in mem., we have no fault for this reference.

Reference String



(or)

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2
7	0	1	2	0	3	0	4	2	3	0	3	2	1	2
7	0	1	2	0	3	0	4	2	3	0	3	2	1	2
F	F	F	F	H	F	H	F	F	F	F	H	H	F	H

3+12=15 page faults.

✓ The FIFO page replacement algorithm is easy to understand & prg. However its performance is not always good.

* Repeat above with 4 frames.

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2
7	0	1	2	0	3	0	4	2	3	0	3	2	1	2
7	0	1	2	0	3	0	4	2	3	0	3	2	1	2
7	0	1	2	0	3	0	4	2	3	0	3	2	1	2
F	F	F	F	H	F	H	F	F	F	F	H	H	F	H

4+6=10 page faults.

Problem with FIFO:

* Belady's Anamoly: Eg: consider the foll. reference string.

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

i) # frames = 3

1	2	3	4	1	2	5	1	2	3	4	5	
1	2	3	4	1	2	5	1	2	3	4	5	
1	2	3	4	1	2	5	1	2	3	4	5	
F	F	F	F	F	F	F	H	H	F	F	H	

page faults = 3+6 = 9 //

ii) # frames = 4

1	2	3	4	1	2	5	1	2	3	4	5	
1	2	3	4	1	2	5	1	2	3	4	5	
1	2	3	4	1	2	5	1	2	3	4	5	
1	2	3	4	1	2	5	1	2	3	4	5	
F	F	F	F	H	H	F	H	H	F	F	F	

page faults = 4 + 6 = 10 //

Note that no. of page faults for 4 frames (10) is greater than the no. of page faults for 3 frames (9). This most unexpected result is known as Belady's Anamoly.

For some page replacement algs, the page fault rate may increase as the no. of allocated frames increases.

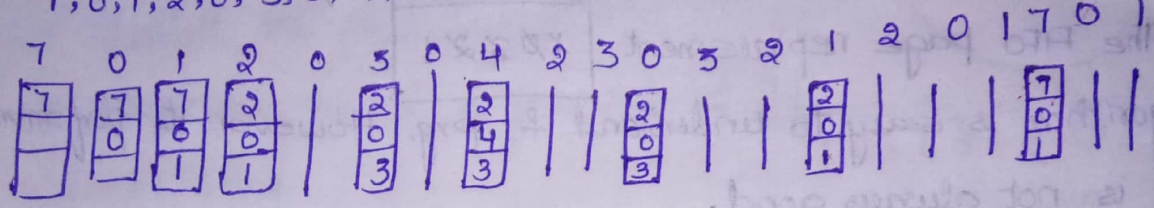
We would expect that giving more mem. to a process would

improve its performance but its not always true.

② Optimal page Replacement: This has the lowest page fault rate of all algorithms & will never suffer from Belady's Anamo

"Replace the page that will not be used for the longest period of time"

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



# of page faults = 9 (or) = 3 + 6 = 9 page faults.

* Disadvantage / Drawback:

The optimal page replacement alg. is difficult to implemen because it requires the future knowledge of the reference string. Hence it is mainly used for comparison studies.

* LRU page Replacement: It's an approximation of optimal page replacement algorithm. (Least Recently Used)

"Replace the page that has not been used for the longest period of time."

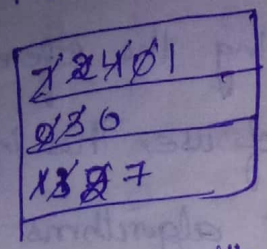
✓ We can think of this strategy as the OPT algorithm lookin backward in time, rather than forward.

✓ If we let S^R be the reverse of a reference string S, then the page fault rate for the OPT alg. on S is the same as the page fault rate for the OPT alg on S^R.

Similary, page fault rate of LRU on S = page fault rate of LRU on S^R.

Reference string:

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



page faults = 3+9 = 12 //

- Replace with a page which was least recently used.
- Like OPT, LRU also does not suffer from Belady's Anamoly.
- OPT & LRU both are called as Stack algorithms.

A stack alg is a alg for which it can be shown that the set of pages in mem for n frames is always a subset of the set of pages that would be in mem. with n+1 frames. For LRU, the set of pages in mem. would be the most recently referenced pages. If the no. of frames is ↑ed, these n pages will still be the most recently referenced & so will still be in mem.

- Implementing LRU → ① counter, ② Stack

4) LRU Approximation Page Replacement: Few computer systems provide sufficient hlw support for true LRU page replacement. Some systems provide no hlw support, & other page replacement algs must be used. Many systems provide some help in the form of reference bits

The reference bit for a page is set by the hlw whenever that page is referenced. (either a read or write to any byte in the page). Reference bits are associated with each entry in the page table.

Initially, all bits are cleared (set to zero) by the OS. As a user process executes, the bit associated with

each page reference is set (to one) by the hlw. After sometime we can determine which pages have been used & which have not been used by examining the reference bits, although we do not known the order of use. This inf. is the basis for many page replacement algorithms that approximate LRU Replacement.

i) **Additional-Reference-bits Algorithm**: We can gain additional ordering inf. by recording the reference bits at regular intervals. We can keep an 8-bit byte for each page in page table in mem. At regular intervals (say every 100 ms), a timer interrupt transfers control to the OS. The OS shifts the reference bit for each page into the high-order bit of its 8-bit byte, shifting the other bits right by 1 bit & discarding the low-order bit.

eg: Lets consider 4 bits (which includes one reference bit & 3 used bit (i.e., U3 U2 U1 U0) Also assume that there are 5 frames.

Reference String: 3,2,3 T 8,0,3 T 3,0,2 T 6,3,4,7

1) Initial State:

P	U3	U2	U1	U0
-	0	0	0	0
-	0	0	0	0
-	0	0	0	0
-	0	0	0	0
-	0	0	0	0

* During first time interval all the 'U' bits are equal to zero & can place pages anywhere.

ii)

P	U3	U2	U1	U0
3	1	0	0	0
2	1	0	0	0
-	0	0	0	0
-	0	0	0	0
-	0	0	0	0

✓Place 3 & 2 pages in the 1st 2 frames & set U3 to 1 for both.

✓Third page reference is 3 & its already present in the frame hence no need to load again.

iii) After first time interval, the bits are shifted to right 1 position.

P	U3	U2	U1	U0
3	0	1	0	0
2	0	1	0	0
-	0	0	0	0
-	0	0	0	0
-	0	0	0	0

During 2nd time interval pages 8, 0, 3 are referenced. pages 8 & 0 are loaded into empty positions & their U3 bits are set to 1. Then the U3 bit for page 3 is also set to 1.

iv)

P	U3	U2	U1	U0
3	1	1	0	0
2	0	1	0	0
8	1	0	0	0
0	1	0	0	0
-	0	0	0	0

At the end of 2nd time interval all 'U' bits are shifted right by 1 position.

v)

P	U3	U2	U1	U0
3	0	1	1	0
2	0	0	1	0
8	0	01	0	0
0	0	01	0	0
-	0	0	0	0

During 3rd time interval pages 3, 0 & 2 are referenced, so U3 is set for pages 3, 0 & 2.

P	U3	U2	U1	U0
3	1	1	1	0
2	1	0	1	0
8	0	1	0	0
0	1	1	0	0
-	0	0	0	0

At the end of 3rd time interval, all 'U' bits are shifted by 1 positi

P	U3	U2	U1	U0
3	0	1	1	1
2	0	1	0	1
8	0	0	1	0
0	0	1	1	0
-	0	0	0	0

During 4th time interval pages 8, 3, 4 & 7 are refered. First page 6 is loaded & its U3 is set to 1. Then the U3 bit for page 3 is set to 1.

P	U3	U2	U1	U0
3	1	1	1	1
2	0	1	0	1
8	0	0	1	0
0	0	1	1	0
6	1	0	0	0

✓ Continuing in the same time interval, when page 4 is required mem. is full, so we choose the page with the lowest 'U' value, which is page 8. page 4 replaces page 8 & U bits are set to 1000. Nxt page 7 is required, replace it by lowest U value ie, 2.

P	U3	U2	U1	U0
3	1	1	1	1
2	0	01	0	01
4	1	0	0	0
0	0	1	1	0
6	1	0	0	0

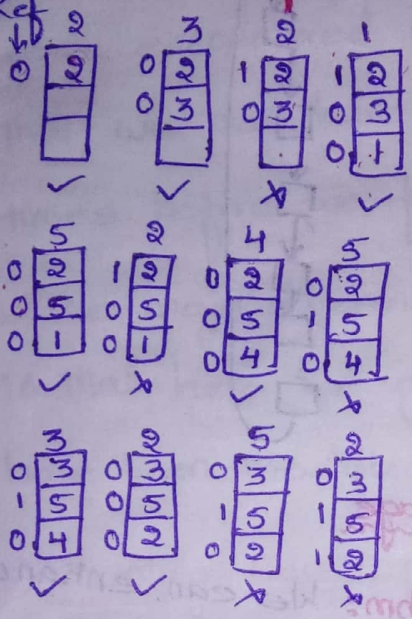
⟹

P	U3	U2	U1	U0
3	1	1	1	1
7	1	0	0	0
4	1	0	0	0
0	0	1	1	0
6	1	0	0	0

* Second chance Algorithm: FIFO Variant of 2nd chance Algorithm (SCA).

Reference String: 2,3,2,1,5,2,4,5,3,2,5,2

Frames = 3, reference bit is taken (No used bits)



✓ Since '2' is again refereneed & its already in mem. we are giving a second chance/reward by setting reference bit to 1.

✓ We are not going replace 2 by 5 because its already got 2nd chance. & it's set to 1 hence look at the pages which are having '0' as reference bit & follow FIFO & as '2' is getting its second chance change reference bit to '0'.

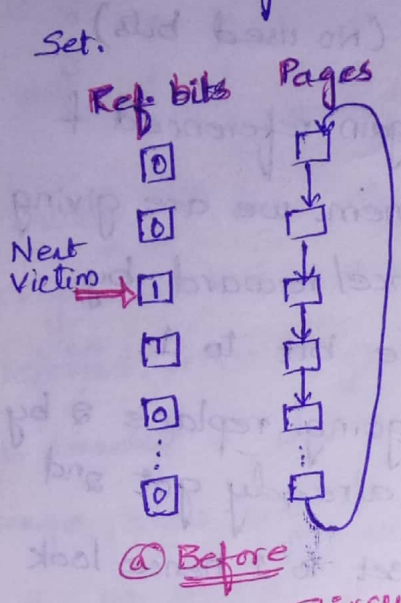
✓ As a reward for saving page fault we will set reference bit to 1.

* Implementing Second chance Algorithm: (clock Algorithm)

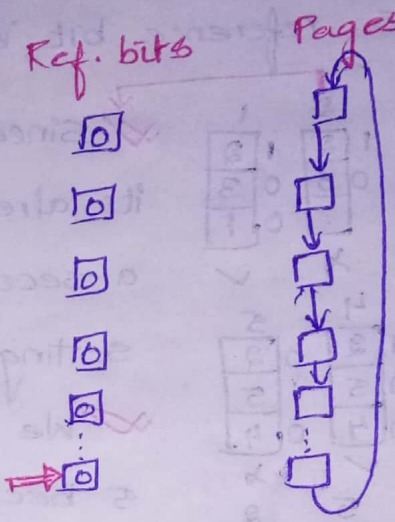
✓ Circular queue can be used. A pointer (i.e., a hand on the clock) indicates which page is to be replaced next. When a frame is needed, the pointer advances until it finds a page with a '0' reference bit set. As it advances, it clears the reference bits (set to 1 to 0) i.e., giving second chance to those pages. Once a victim page is found, the page is replaced, & the new page is inserted in the circular queue in that position.

✓ In the worst case, when all the bits are set, the pointer cycles through the whole queue, giving each

page a second chance. It clears all the reference bi before selecting the next page for replacement. Second chance degenerates to FIFO replacement if all bits are Set.



(a) Before



Circular Queue of Page

iii) Enhanced Second-chance Algorithm: We can enhance the second-chance algorithms by considering the reference bit & the modify bit as an ordered pair. With these 2 bits, we have the following 4 possible cases.

1) (0,0): Neither recently used nor modified - Best page to replace.

2) (0,1): Not recently used but modified - not quite as good, because the page will page need to be written out before replacement.

3) (1,0): Recently used ~~but modified~~ clean - Probably will be used again.

4) (1,1): Recently used & modified - Propably will be used again soon, & the page will be need to be written out to disk before it can replaced.

Each page will be in 1 of these 4 classes. When page replacement is called for, we use the same