3.3 PAGE FAULT

Whenever a processor needs to execute a particular page and that page is not available in main memory, this situation is said to be **"page fault".** When the page fault occurs, the page replacement will be done. The word **"page replacement" means select a victim page in the main memory, replace that page with the required page from the backing store (Disk).** The students may have a doubt, how to select a victim page. **The answer is simple "Page replacement algorithms".** The page replacement algorithms select the victim pages.

Now a days the number of page replacement algorithms are available. Each operating system has its own page replacement policy. Now we consider some of the popular page replacement algorithms.

3.4 PAGE REPLACEMENT ALGORITHMS

3.4.1 FIFO algorithm: (First in First Out algorithm)

FIFO is the simplest page replacement algorithm, the idea behing this is **"Replace a page that page is the oldest page of all the pages of main memory" or "Replace the page that has been in memory longest".** FIFO focuses on the length of time a page has been in memory rather than how much the page is being used. To illustrate the page replacement algorithm, we shall use the reference string.

0 1 2 3 0 1 2 3 0 1 2 3 4 5 6 7

for a memory with 3 frames. This example incurs 16 page faults under FIFO, as shown in the table.

Table FIFO Behaviour

Frame	0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
0	0*	0	0	3*	3	3	2*	2	2	1*	1	1	4*	4	4	7*
1		1*	1	1	0*	0	0	3*	3	3	2*	2	2	5*	5	5
2			2*	2	2	1*	1	1	0*	0	0	3*	3	3	6*	6
	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√

This example incurs 16 pages faults. The symbol "*" indicates the new page in the memory. The symbol "√" indicates page fault. The performance of the page replacement algorithm is measured through the page fault rate.

∴ Page fault rate = Number of page faults/Number of bits in the reference string.

∴ PFR = 16/16 = 100%

An algorithm having the least page fault rate, will be the best one.

If the required page resides in the main memory, the page replacement is not required, in this situation we use the '*x*', symbol. Consider the next example.

The reference string is 0, 1, 2, 3, 2, 5, 6, 3, 4, 6, 3, 7, 3, 1, 5, 3, 6, 3, 4, 2, 4, 3, 4, 5, 1. For a memory with 4 frames. This example incurs 12 page faults shown in the table.

Table FIFO behaviour

Frame	1	2	3	2	5	6	3	4	6	3	7	3	1
0	1*	1	1	1	1	6*	6	6	6*	6	6	6	1*
1	–	2*	2	2*	2	2	2	4*	4	4	4	4	4
2	–	–	3*	3	3	3	3*	3	3	3*	7*	7	7
3	–	–	–	–	5*	5	5	5	5	5	5	3*	3
	√	√	√	×	√	√	×	√	√	×	×	√	√

Frame	5	3	6	3	4	2	4	3	4	5	1
0	1	1	1	1	1	2*	2	2	2	2	2
1	5*	5	5	5	5	5	5	3*	3	3	3
2	7	7	6*	6	6	6	6	6	6	5*	5
3	3	3*	3	3*	4*	4	4*	4	4	4	1*
	√	×	√	×	√	√	×	√	×	√	√

Page fault rate = 16/24 = 75%

3.4.2 Optimal page replacement algorithm

The optimal page replacement algorithm has the lowest page fault rate of all algorithms. The criteria of this algorithm is **"Replace a page that will not be used for the longest period of time"**. To illustrate this algorithm, consider the reference string 1, 2, 3, 2, 5, 6, 3, 4, 6, 3, 4, 6, 3, 7, 3, 1, 5, 3, 6, 3, 4, 2, 4, 3, 4, 5, 1. Assume that the memory size is four frames. Consider the table below to calculate the page fault rate.

Table Optimal Behavior

Frame	1	2	3	2	5	6	3	4	6	3	7	3	1
0	1*	1	1	1	1	1	1	1	1	1	1	1	1*
1		2*	2	2*	2	6*	6	6	6*	6	6	6	6
2			3*	3	3	3	3*	3	3	3*	3	3*	3
3					5*	5	5	4*	4	4	7*	7	7
	√	√	√	×	√	√	×	√	×	×	√	×	×

Frame	5	3	6	3	4	2	4	3	4	5	1
0	1	1	1	1	1	2*	2	2	2	2	1*
1	6	6	6*	6	4*	4	4*	4	4*	4	4
2	3	3*	3	3*	3	3	3	3*	3	3	3
3	5*	5	5	5	5	5	5	5	5	5*	5
	√	×	×	×	√	√	×	×	×	×	√

∴ Page fault = 11/24

The optimal page replacement algorithm is difficult to implement, because it requires future knowledge of reference string, so this algorithm is used mainly for comparison studies.

LRU (Least Recently Used Algorithm)

The criteria of this algorithm is "**Replace a page that has not been used for the longest period of time**". The strategy is "**Page replacement algorithm looking backward in time, rather than forward**". To illustrate the algorithm consider the reference the string.

0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 4, 5, 6, 7 with 3 main frames.

Frame	0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
0	0*	0	0	3*	3	3	2*	2	2	1	1*	1	4*	4	4	7*
1		1*	1	1	0*	0	0	3*	3	3	2	2*	2	5*	5	5
2			2*	2	2	1*	1	1	0*	0	0	3	3	3	6*	6
	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√

∴ Page fault rate = 16/16 = 100%

Software Implementation of LRU

A simple method to implement LRU in software is to use stack of page numbers. Whenever a page is referenced, its number is removed from the stack and put on the top. So, the top of stack is always the most recently used page number and the bottom is the LRU page number. It is time consuming to remove LRU page number from the stack. However, this algorithm does not suffer from Belady's anomaly (to be discussed later).

3.4.6 Least Frequently Used (LFU) algorithm

The Least Frequently used algorithm "**selects a page for replacement, if the page has not been used often in the past**" (or) "**Replace page that page has smallest count**". For this algorithm each page maintains a counter, which counter value shows the least count, replace that page. The reason for this selection is that an actively used page should have a large reference count, so don't replace the activity used page. The frequency counter is reset each time a page is loaded. To illustrate this algorithm consider below reference string 0 1 2 3 0 1 2 3 0 1 2 3 4 5 6 7 with 3 memory frames.

Table LFU Algorithm

Frame	0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
0	0*	0	0	0	0+	0	0	0	0+	01	0	3*	3	3	3	3
1		1*	1	1	1	1+	1	3*	3	1*	1	1	1	1	1	1
2			2	3*	3	3	2*	2	2	2	2+	2	4	5*	6*	7*
	√	√	√	√	×	×	√	√	×	√	×	√	√	√	√	√

∴ Page Fault Rate = 12/16

3.4.7 Most Frequently Used Algorithm (MFU)

The criteria of this algorithm is **replace a page that has the maximum frequency count of all pages.** The implementation of this algorithm is fairly expensive.

3.4.8 Belady's Anomaly

The general principle is if the number of frames is increased, the page fault rate will be decreased. For example, consider the following reference string.

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

if we apply the FIFO algorithm with 3 frames, there are 15 page faults occur, if we increase 4 frames, there are 12 page faults. Consider the reference string.

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

The number of page faults for 4 frames (10) is greater than the number of faults for 3 frames (9). This result is most unexpected and is known as "**Belady's Anomaly**".

Table 4 Frames

Frame	1	2	3	4	1	2	5	1	2	3	4	5
0	1	1	1	1	1+	1	5	5	5	5	4	4
1		2	2	2	2	2+	2	1*	1	1	1	5
2			3	3	3	3	3	3	2*	2	2	2
3				4	4	4	4	4	4	3	3	3
	√	√	√	√	×	×	√	√	√	√	√	√

∴ Page fault for 4 frames = 10/12

Frame	1	2	3	4	1	2	5	1	2	3	4	5
0	1*	1	1	4	4	4	5*	5	5	5	5	5+
1		2*	2	2*	1	1	1	1	1	3	3	3
2			3*	3	3*	2	2	·2	2	2	4	4
	√	√	√	√	√	√	√	√	√	√	√	√

∴ Page fault for 3 frames = 9/12

Belady's Anomaly was not always true it is applicable only to FIFO algorithm.

3.4.9 Working Set Page Replacement Algorithm

The set of pages that a process is currently using is called its working set. Most processes exhibit a **locality of reference** meaning that during any phase of execution, the process references only a relatively small fraction of its pages. This model is based on the concept of current locality of reference of a process. **The working set principle states that:**

1. **A program should not be run, unless its working set is fully loaded in the memory.**
2. **No page of a process should be replaced, if it forms part of its working set.**

Ideally, the working set of a process should comprise its current locality. For this a working set window (say Δ) is the set. The working set of a process comprises the set of pages referenced by it, during the most recent (Δ) references. The working set of a process at a time, t, is defined as:

$$W(t, \Delta) = \{\text{Set of pages } i \mid \text{page } i \text{ appears amongst } r_{t-\Delta+1} \ldots r_t\}$$

where r_t denotes a page referenced at time, t.

Now, Δ is the working set window *i.e.*, a fixed number of page references (say, 10000 instructions). This Δ must be carefully chosen, 3 cases arise—

Case 1 : if Δ is too small then the working-set will not encompass entire locality.

Case 2 : if Δ is too large then the working set may encompass more than the current locality.

Case 3 : *if* $\Delta = \infty$ then it will encompass the entire program.

Please note here that the window size can be fine tuned by observing page fault frequency (PFF) of active processes. If PFF is found to be too high then the window size are increased else if PFF is found to be too low then the window size is reduced.

3.5 THRASHING

If the number of frames allocated to a low-priority process falls below the minimum number then we must suspend that process execution. We should then page out its remaining pages, freeing all its allocated frames. So, now swapping is required. We can find some process in a system that does not have "enough" frames. It is technically possible to reduce the number of allocated frames to the minimum, there is some (larger) number of pages in active use. If the process does not have this number of frames then it will quickly page fault. At this point, it must replace some page that will be needed again. Consequently, it quickly faults again and again. The process continues to fault, replacing pages for which it then faults and brings back in right away. Such a process would spend more time in paging than executing. **This high paging activity is called as thrashing. A process is said to be thrashing if it is spending more time in paging than in execution.** This results in a poor throughput of the system.

Cause of thrashing

Thrashing results in severe performance problems. Let us consider an example to understand this. We know that an OS monitors the CPU utilization. If CPU utilization is too low, we increase the degree of multiprogramming (DOM) by introducing a new process to the system. (A global page-replacement algorithm is used; it replaces pages with no regard to the process to which they belong. Now, say that a process enters a new phase in its execution and needs more frames. It starts faulting and taking frames away from other processes. These processes need those pages, however, and so they also fault, taking frames from other processes. These faulting processes must use the paging device to swap pages in and out. As they queue up for paging device, the ready queue empties. As processes wait, for the paging device, CPU utilization decreases. The CPU schedular sees the decreasing CPU utilization so it increases the degree of multiprogramming. The new process tries to get started by taking frames from running processes, causing more page faults, and a longer queue for the paging device. As a result, the CPU utilization drops even further. The CPU scheduler tries to increase the degree of multiprogramming even more. We say, thrashing has occurred and the system throughput plunges. The page fault rate (PFR) increases tremendously. As a result, the effective memory access time increases. No work is getting done because the processes are spending all their time in paging. This is shown in Figure 3.4.

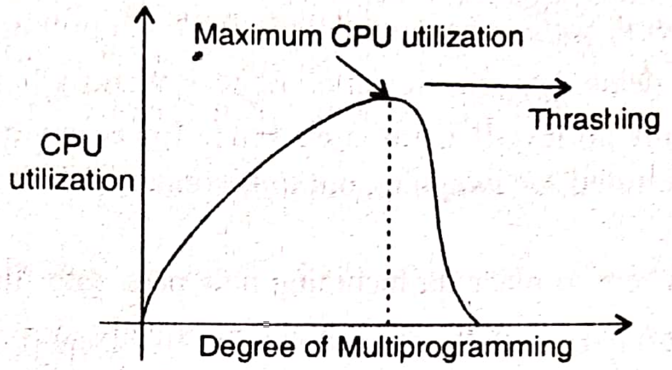

Fig. 3.4 Thrashing

3.6 PAGE REPLACEMENT POLICIES–LOCAL & GLOBAL

As we know that multiple processes compete for frames, so we can classify page replacement algorithms into 2 types—

(*a*) Local page replacement

(*b*) Global page replacement

Local page replacement

It means that when a process requests for a new page to be brought in and there are no free frames in the memory, we choose a frame allocated to only that process for replacement. Thus, this policy requires that each process selects from its own set of allocated frames only.

Global page replacement

It allows a process to select a replacement frame from the set of all frames, even if that frame is currently allocated to some other process. So, one process can take a frame from another. Thus, in this policy any frame from any other process can be replaced. Here the OS has to decide about the set of worst pages and then choose a page amongst them. In this case, process priorities also play a vital role. So, it is quite difficult to implement this policy.