PAGE REPLACEMENT ALGORITHMS

- While swapping in a page, if no frame is free, we find one that is not currently being used and free it.
- We can free a frame by writing its contents to swap space and changing the page table (and all other tables) to indicate that the page is no longer in memory.
- We can now use the freed frame to hold the page for which the process faulted.
- If we have multiple processes in memory, we must decide how many frames to allocate to each process; and when page replacement is required, we must select the frames that are to be replaced.
- Designing appropriate algorithms to solve these problems is an important task, because disk I/Ois so expensive. Even slight improvements in demand-paging methods yield large gains in system performance.
- There are many different page-replacement algorithms. Every OS probably has its own replacement scheme.
- How do we select a particular replacement algorithm? In general, we want the one with the lowest page-fault rate.
- We evaluate an algorithm by running it on a particular string of memory references and computing the number of page faults.
- The string of memory references is called a **reference string**. We can generate reference strings artificially (by using a random-number generator, for example), or we can

trace a given system and record the address of each memory reference

- The important page replacement algorithms are
    1. FIFO (First In First Out)
    2. Optimal Page Replacement (OPT)
    3. LRU (Least Recently Used)
    4. LFU (Least Frequently Used)
    5. MFU (Most Frequently Used)

FIFO (First In First Out) Algorithm

- The simplest page-replacement algorithm
- FIFO replacement algorithm associates with each page the time when that page was brought into memory.
- When a page must be replaced, the oldest page is chosen.
- It is not strictly necessary to record the time when a page is brought in. We can create a FIFO queue to hold all pages in memory. We replace the page at the head of the queue. When a page is brought into memory, we insert it at the tail of the queue

Problem:

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

for a memory with three frames.

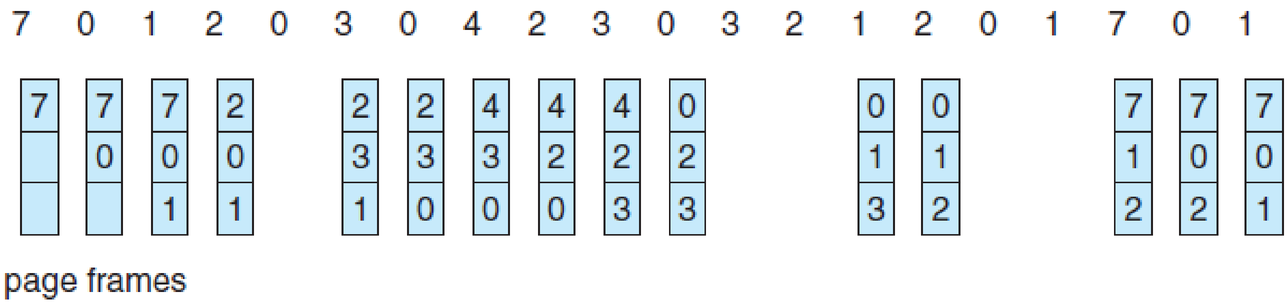


Figure 9.12 FIFO page-replacement algorithm.

- There are fifteen faults altogether.
- The FIFO page-replacement algorithm is easy to understand and program.
- However, its performance is not always good.
- Even if we select for replacement a page that is in active use, everything still works correctly. After we replace an active page with a new one, a fault occurs almost immediately to retrieve the active page.
- Some other page must be replaced to bring the active page back into memory. Thus, a bad replacement choice increases the page-fault rate and slows process execution.
- It does not, however, cause incorrect execution
- Another drawback of FIFO algorithm is **Belady's anomaly**
- Eg: Consider the following reference string:

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

  Run FIFO algorithm with number of frames varies from 1 to 7

  - Let the no of frames be 1: No of page faults = 12
  - Let the no of frames be 2: No of page faults = 12

➢ Let the no of frames be 3:

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

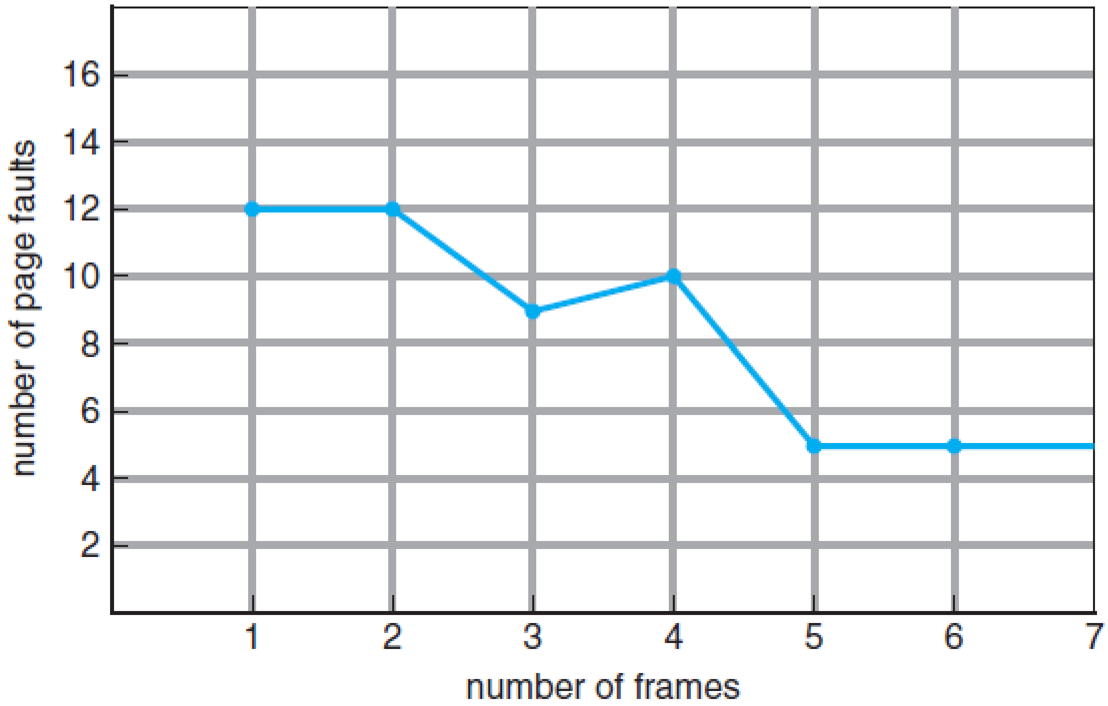
No of page faults = 9

➢ Let the number of frames be 4:

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

No of faults = 10

(No of frames 5, 6 & 7 are left to you as home work)



- Figure shows the curve of page faults for this reference string versus the number of available frames.
- Notice that the number of faults for four frames (ten) is ***greater*** than the number of faults for three frames (nine)!
- This most unexpected result is known as **Belady's anomaly**: for some page-replacement algorithms, the page-fault rate may ***increase*** as the number of allocated frames increases.
- We would expect that giving more memory to a process would improve its performance. But this assumption was not always true.

Optimal Page Replacement (OPT)

- Algorithm that has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly.
- It replaces the page that **will not be used** for the longest period of time.
- Use of this page-replacement algorithm guarantees the lowest possible page fault rate for a fixed number of frames. So this algorithm is also called **MIN**
- It requires future knowledge of the reference string.

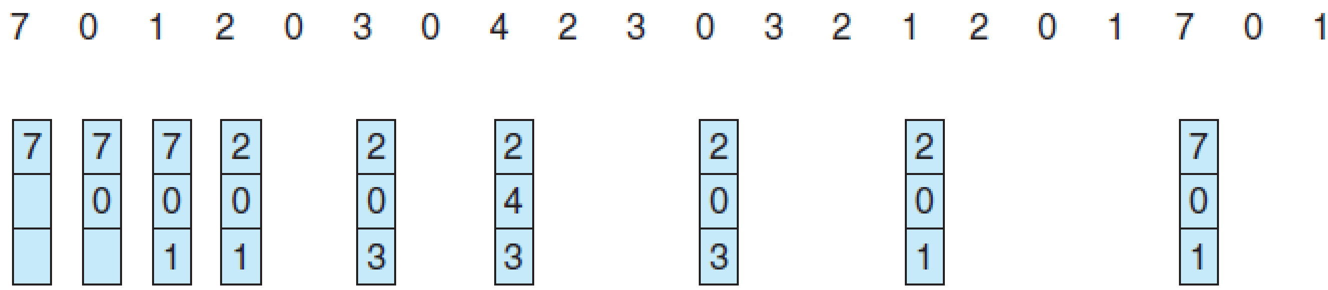
Problem:

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

for a memory with three frames.

Solution:



- With only nine page faults, optimal replacement is much better than FIFO algorithm, which results in fifteen faults.
- If we ignore the first three, which all algorithms must suffer, then optimal replacement is twice as good as FIFO replacement.

- In fact, no replacement algorithm can process this reference string in three frames with fewer than nine faults.
- Unfortunately, the optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.
- We encountered a similar situation with the SJF CPU-scheduling algorithm also
- So, the optimal algorithm is used mainly for comparison studies. For instance, it may be useful to know that, although a new algorithm is not optimal, it is within 12.3 percent of optimal at worst and within 4.7 percent on average.

LRU (Least Recently Used) Algorithm

- If we use the recent past as an approximation of the near future, then we can replace the page that ***has not been used*** for the longest period of time. This approach is the **least recently used (LRU) algorithm**
- It is an approximation of the optimal algorithm
- LRU replacement associates with each page the time of that page's last use.
- When a page must be replaced, LRU chooses the page that has not been used for the longest period of time.
- We can think of this strategy as the optimal page-replacement algorithm looking backward in time, rather than forward.

.

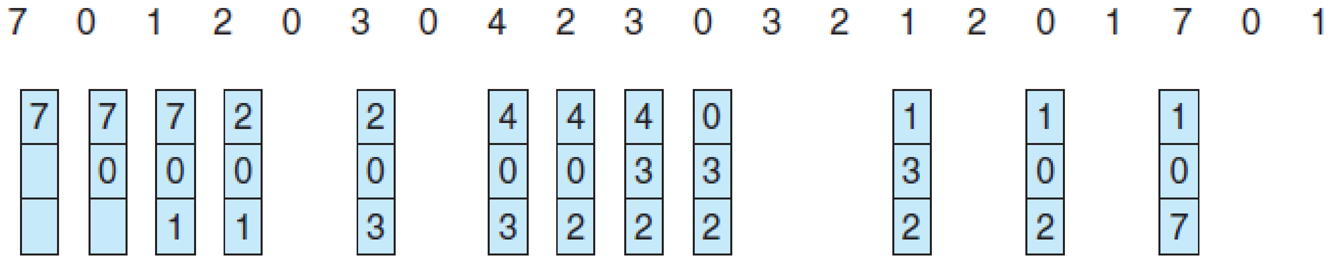
Problem:

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

for a memory with three frames.

Solution:



- The LRU algorithm produces twelve faults. Notice that the first five faults are the same as those for optimal replacement. It is much better than FIFO replacement with fifteen.
- The LRU policy is often used as a page-replacement algorithm and is considered to be good.
- The major problem is ***how*** to implement LRU replacement. An LRU page-replacement algorithm may require substantial hardware assistance.
- The problem is to determine an order for the frames defined by the time of last use.
- Two implementations are feasible:

1. Counters

- ➢ In the simplest case, we associate with each page-table entry a time-of-use field and add to the CPU a logical clock or counter.

- The clock is incremented for every memory reference. Whenever a reference to a page is made, the contents of the clock register are copied to the time-of-use field in the page-table entry for that page.
- We replace the page with the smallest time value.
- This scheme requires a search of the page table to find the LRU page and a write to page table for each memory access.

2. Stack

- Another approach to implementing LRU replacement is to keep a stack of page numbers.
- Whenever a page is referenced, it is removed from the stack and put on the top.
- In this way, the most recently used page is always at the top of the stack and the least recently used page is always at the bottom
- This approach is particularly appropriate for implementations of LRU replacement.

- Like optimal replacement, LRU replacement does not suffer from Belady's anomaly.
- Both belong to a class of page-replacement algorithms, called **stack algorithms**, that can never exhibit Belady's anomaly

Counting-Based Page Replacement Algorithms

1. LFU
2. MFU

- The **least frequently used (LFU)** page-replacement algorithm requires that the page with the smallest count be replaced.
- The reason for this selection is that an actively used page should have a large reference count.
- With LFU, each page table entry has a counter, and for each memory reference, the MMU increments that counter. When a page fault occurs, the OS should choose the page frame whose counter is smallest.
- A problem arises, however, when a page is used heavily during the initial phase of a process but then is never used again. Since it was used heavily, it has a large count and remains in memory even though it is no longer needed.
- So solution is to right-shift the counter every occasionally, so that the counter eventually decays back to 0 if it's not used. This works much better.
- The **most frequently used (MFU)** page-replacement algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used in future.
- The MFU algorithm comes from the same sort of reasoning that brought up the worst-fit memory allocation scheme

- In this case, removing the page frame whose frequency counter is largest actually makes some sense: If a page's frequency counter is large, then that page has had its fair share of the memory, and it's time for somebody else to get its turn.
- Neither MFU nor LFU replacement is common. The implementation of these algorithms is expensive, and they do not approximate OPT replacement well.