

2. Optimal Page Replacement

One result of the discovery of Belady's anomaly was the search for an **optimal page-replacement algorithm**—the algorithm that has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly. Such an algorithm does exist and has been called OPT or MIN. It is simply this:

Replace 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.

For example, on our sample reference string, the optimal page-replacement algorithm would yield nine page faults, as shown in Figure 9.14. The first three references cause faults that fill the three empty frames. The reference to page 2 replaces page 7, because page 7 will not be used until reference 18, whereas page 0 will be used at 5, and page 1 at 14. The reference to page 3 replaces page 1, as page 1 will be the last of the three pages in memory to be referenced again. With only nine page faults, optimal replacement is much better than a 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.)

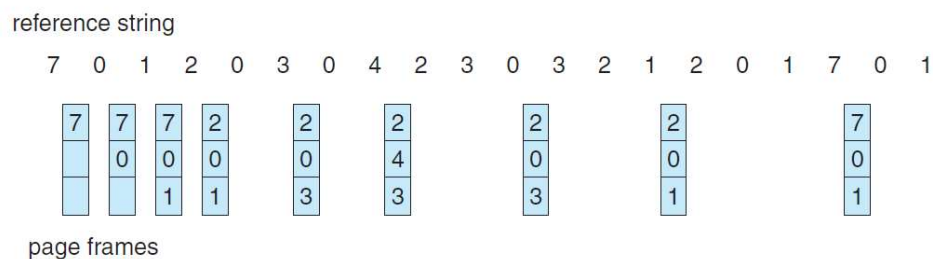


Figure 9.14 Optimal page-replacement algorithm.

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. As a result, 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.

3. LRU Page Replacement

Least Recently Used. The least recently used (LRU) method will replace the frame that has not been used for the longest time. In this method, when a page is referenced that is already in $M1$, it is placed in the top row and the other pages are shifted down. In other words, the most used pages are kept at the top. See Figure 2.56a. An improvement of PF is made using LRU by adding a page frame to $M1$. See Figure 2.56b.

In general, LRU is more efficient than FIFO, but it requires more hardware (usually a counter or a stack) to keep track of the least and most recently used pages.

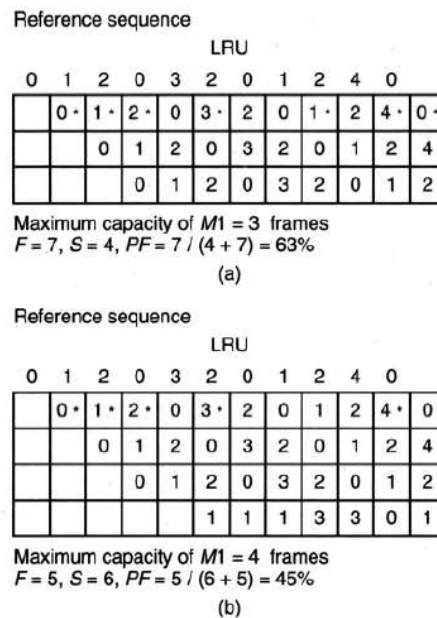


Figure 2.56 Performance of LRU replacement technique on two different memory configurations.

If the optimal algorithm is not feasible, perhaps an approximation of the optimal algorithm is possible. The key distinction between the FIFO and OPT algorithms (other than looking backward versus forward in time) is that the FIFO algorithm uses the time when a page was brought into memory, whereas the OPT algorithm uses the time when a page is to be used. 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.

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. The result of applying

LRU replacement to our example reference string is shown in Figure 9.15. The LRU algorithm produces twelve faults. Notice that the first five faults are the same as those for optimal replacement. When the reference to page 4 occurs, however, LRU replacement sees that, of the three frames in memory, page 2 was used least recently. Thus, the LRU algorithm replaces page 2, not knowing that page 2 is about to be used. When it then faults for page 2, the LRU algorithm replaces page 3, since it is now the least recently used of the three pages in memory. Despite these problems, LRU replacement with twelve faults 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:

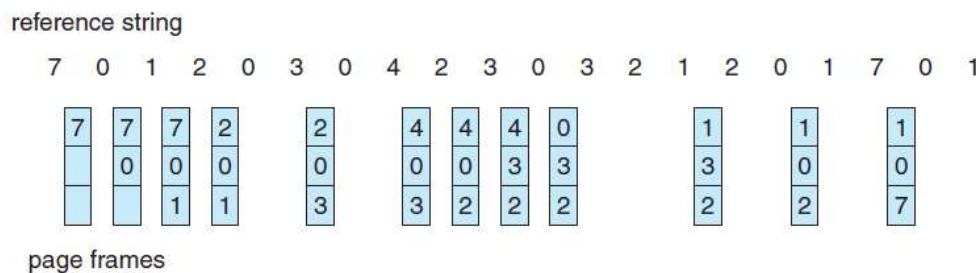


Figure 9.15 LRU page-replacement algorithm.

- **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. In this way, we always have the “time” of the last reference to each 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 memory (to the time-of-use field in the page table) for each memory access. The times must also be maintained when page tables are changed (due to CPU scheduling). Overflow of the clock must be considered.

- **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 (Figure 9.16). Because entries must be removed from the middle of the stack, it is best to implement this approach by using a doubly linked list with a head pointer and a tail pointer. Removing a page and putting it on the top of the stack then requires changing six pointers at worst. Each update is a little more expensive, but there is no search for a replacement; the tail pointer points to the bottom of the stack, which is the LRU page. This approach is particularly appropriate for software or microcode 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. A stack algorithm is an algorithm for which it can be shown that the set of pages in memory for n frames is always a **subset** of the set of pages that would be in memory with $n + 1$ frames. For LRU replacement, the set of pages in memory would be the n most recently referenced pages. If the number of frames is increased, these n pages will still be the most recently referenced and so will still be in memory.

Note that neither implementation of LRU would be conceivable without hardware assistance beyond the standard TLB registers. The updating of the clock fields or stack must be done for **every** memory reference. If we were to use an interrupt for every reference to allow software to update such data structures, it would slow every memory reference by a factor of at least ten,

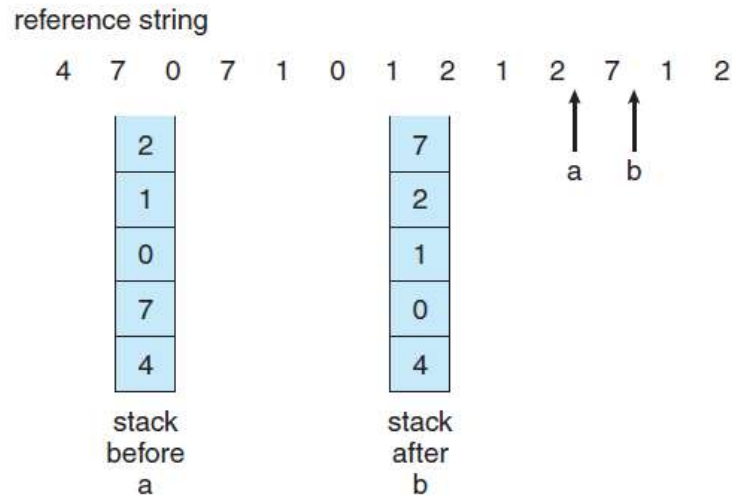


Figure 9.16 Use of a stack to record the most recent page references.

4. LRU-Approximation Page Replacement

Few computer systems provide sufficient hardware support for true LRU page replacement. In fact, some systems provide no hardware support, and other page-replacement algorithms (such as a FIFO algorithm) must be used. Many systems provide some help, however, in the form of a reference bit. The reference bit for a page is set by the hardware whenever that page is referenced (either a read or a write to any byte in the page). Reference bits are associated with each entry in the page table.

Initially, all bits are cleared (to 0) by the operating system. As a user process executes, the bit associated with each page referenced is set (to 1) by the hardware. After some time, we can determine which pages have been used and which have not been used by examining the reference bits, although we do not know the *order* of use. This information is the basis for many page-replacement algorithms that approximate LRU replacement.

- Additional-Reference-Bits Algorithm

We can gain additional ordering information by recording the reference bits at regular intervals. We can keep an 8-bit byte for each page in a table in memory. At regular intervals (say, every 100 milliseconds), a timer interrupt transfers control to the operating system. The operating system 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 and discarding the low-order bit. These 8-bit shift registers contain the history of page use for the

last eight time periods. If the shift register contains 00000000, for example, then the page has not been used for eight time periods.

A page that is used at least once in each period has a shift register value of 11111111. A page with a history register value of 11000100 has been used more recently than one with a value of 01110111. If we interpret these 8-bit bytes as unsigned integers, the page with the lowest number is the LRU page, and it can be replaced. Notice that the numbers are not guaranteed to be unique, however. We can either replace (swap out) all pages with the smallest value or use the FIFO method to choose among them.

- Second-Chance Algorithm
- Enhanced Second-Chance Algorithm

5. Counting-Based Page Replacement

6. Page-Buffering Algorithms

7. Applications and Page Replacement