Virtual Memory

Virtual memory is a technique that lows the execution of processes that are not completely in memory. One major advantage of this scheme is that programs can be larger than physical memory.

Virtual memory involves the separation of logical memory as perceived by users from physical memory. This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available (Figure 9.1). Virtual memory makes the task of programming much easier, because the programmer no longer needs to worry about the amount of physical memory available

The **virtual address space** of a process refers to the logical (or virtual) view of how a process is stored in memory. Typically, this view is that a process begins at a certain logical address —say, address 0—and exists in contiguous memory,

physical memory may be organized in page frames and that the physical page frames assigned to a process may not be contiguous. It is up to the memory- management unit (MMU) to map logical pages to physical page frames in memory.

9.2 Demand Paging

With demand-paged virtual memory, pages are loaded only when they are demanded during program execution.

Pages that are never accessed are thus never loaded into physical memory A demand-paging system is similar to a paging.

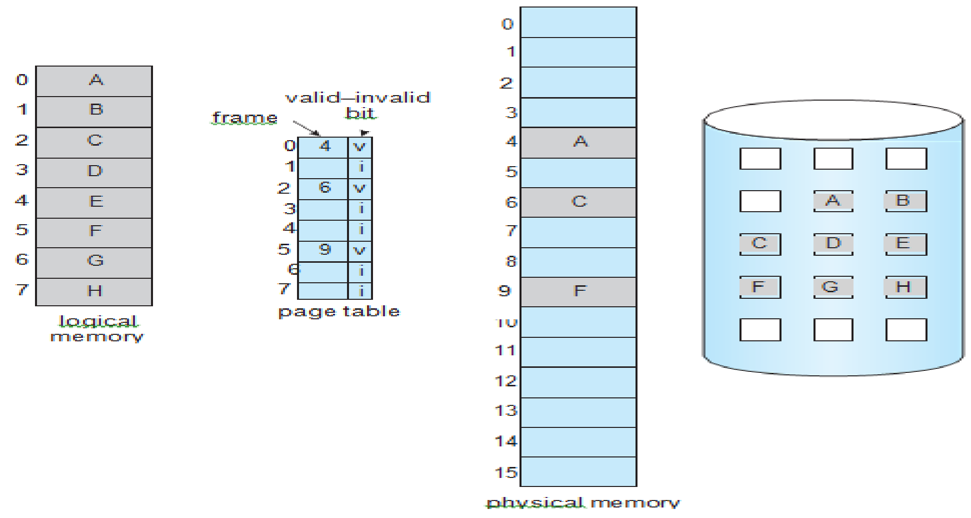

Figure 9.5 Page table when some pages are not in main memory.

system with swapping (Figure 9.4) where processes reside in secondary memory (usually a disk). When we want to execute a process, we swap it into memory.

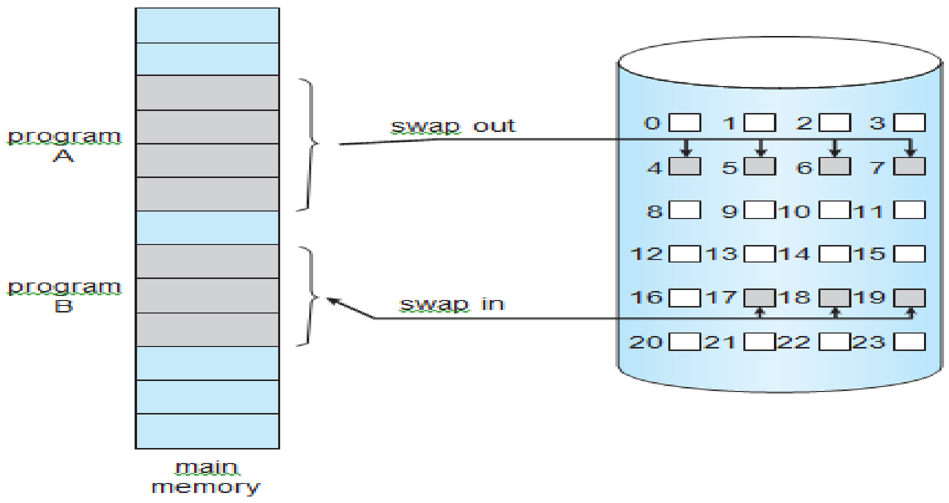

Figure 9.4 Transfer of a paged memory to contiguous disk space.

But what happens if the process tries to access a page that was not brought into memory? Access to a page marked invalid causes a page fault. The paging hardware, in translating the address through the page table, will notice that the invalid bit is set, causing a trap to the operating system. This trap is the result of the operating system's failure to bring the desired page into memoy

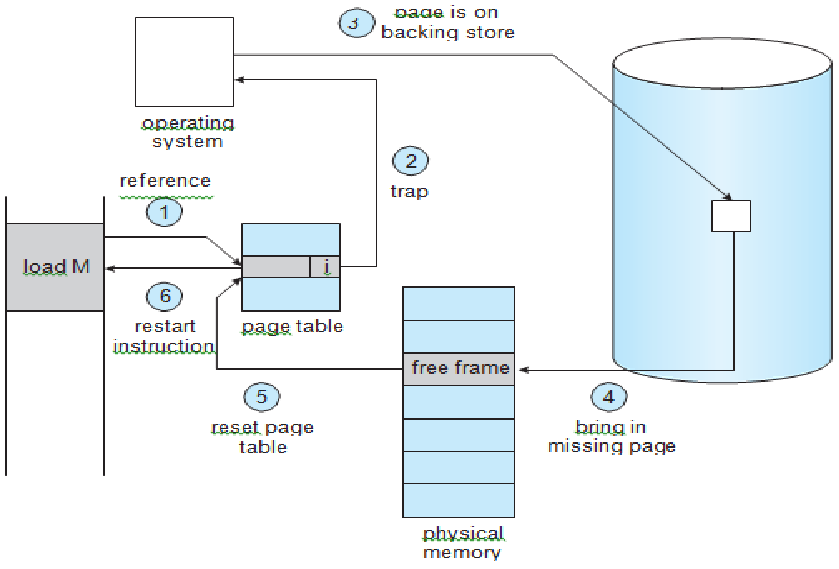

Figure 9.6 Steps in handling a page fault.

9.4 Page Replacement

Page replacement takes the following approach. 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.

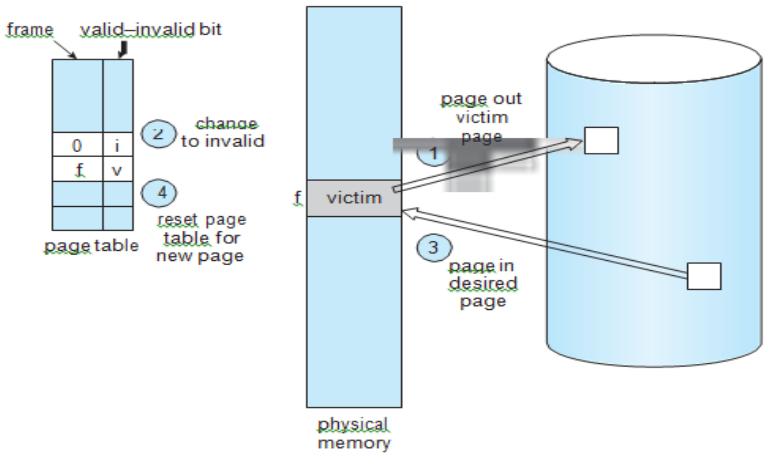

Figure 9.10 Page replacement.

There are many different page-replacement algorithms. Every operating system 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 latter choice produces a large number of data (on the order of 1 million addresses per second). To reduce the number of data, we use two facts. First, for a given page size (and the page size is generally fixed by the hardware or system), we need to consider only the page number, rather than the entire address. Second, if we have a reference to a page *p*, then any references to page *p* that ***immediately*** follow will never cause a page fault. Page *p* will be in memory after the first reference, so the immediately following references will not fault. For example, if we trace a particular process, we might record the following, address sequence

0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103,
0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105

At 100 bytes per page, this sequence is reduced to the following reference string

1, 4, 1, 6, 1, 6, 1, 6, 1, 6,1

We next illustrate several page-replacement algorithms. In doing so, we use the 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

9.4.2 FIFO Page Replacement

The simplest page-replacement algorithm is a first-in, first-out (FIFO) algorithm. A 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. Notice that 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.

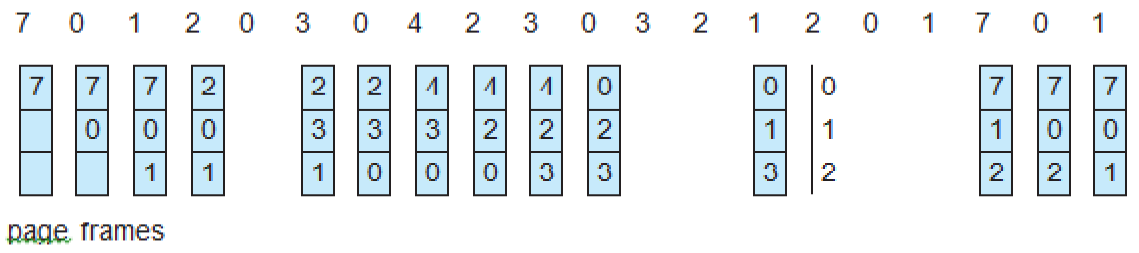

Figure 9.12 FIFO page-replacement algorithm.

Every time a fault occurs, we show which pages are in our three frames. There are fifteen faults altogether To illustrate the problems that are possible with a FIFO page-replacement algorithm, consider the following reference string:

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

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**

9.4.3 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.

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 in Section 6.3.2.) As a result, the optimal algorithm is used mainly for comparison studies.

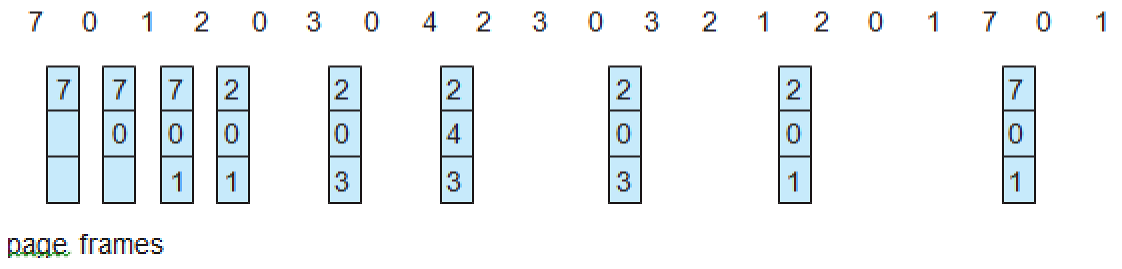

Figure 9.14 Optimal page-replacement algorithm.

With only nine page faults, optimal replacement is much better than a FIFO algorithm, which results in fifteen faults

9.4.4 LRU Page Replacement

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. The result of applying LRU replacement to our example reference string is shown in Figure 9.15.

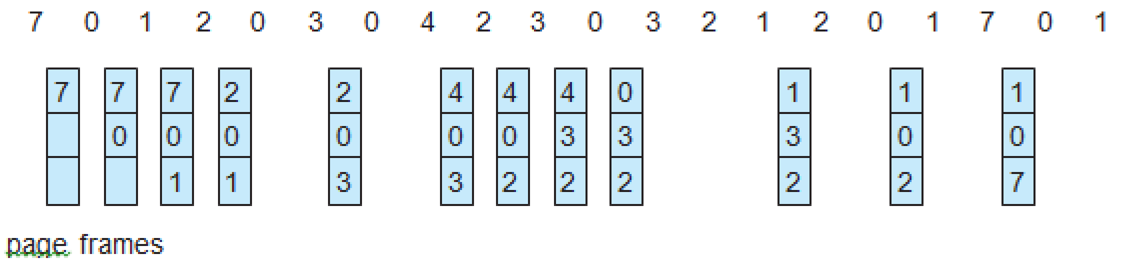

Figure 9.15 LRU page-replacement algorithm.