Virtual Memory

Contents of Lecture:

- ❖ Background
- ❖ Demand Paging
- ❖ Page Replacement
- ❖ Page and Frame Replacement Algorithms

References for This Lecture:

- ✓ *Abraham Silberschatz, Peter Bear Galvin and Greg Gagne, Operating System Concepts, 9th Edition,* ***Chapter 9***

Background

- ❖ The requirement that instructions must be in physical memory to be executed, the **entire program** is not needed:
    - ✓ Programs often have code to handle unusual error conditions (Error code).
    - ✓ large data structures (Arrays, lists, tables)
    - ✓ Certain options and features of a program may be used rarely (unusual routines).
- ❖ Even in those cases where the entire program is needed, it may not all be needed at the same time.

- ❖ **Virtual memory:** technique that allows the execution of processes that are not completely in memory
    - ✓ Separation of user logical memory from physical memory
    - ✓ Only part of the program needs to be in memory for execution
    - ✓ Logical address space can therefore be much larger than physical address space
    - ✓ Allows address spaces to be shared by several processes
    - ✓ Allows for more efficient process creation
    - ✓ More programs running concurrently
    - ✓ Less I/O needed to load or swap processes
    - ✓ Virtual memory can be implemented via **Demand paging**

Demand Paging

- ❖ Demand Paging is a technique bring a page into memory only when it is needed during program execution
- ❖ Similar to paging system with swapping next figure (Figure 9.4)
- ❖ If page needed and not memory resident
    - ✓ Need to load the page into memory from storage

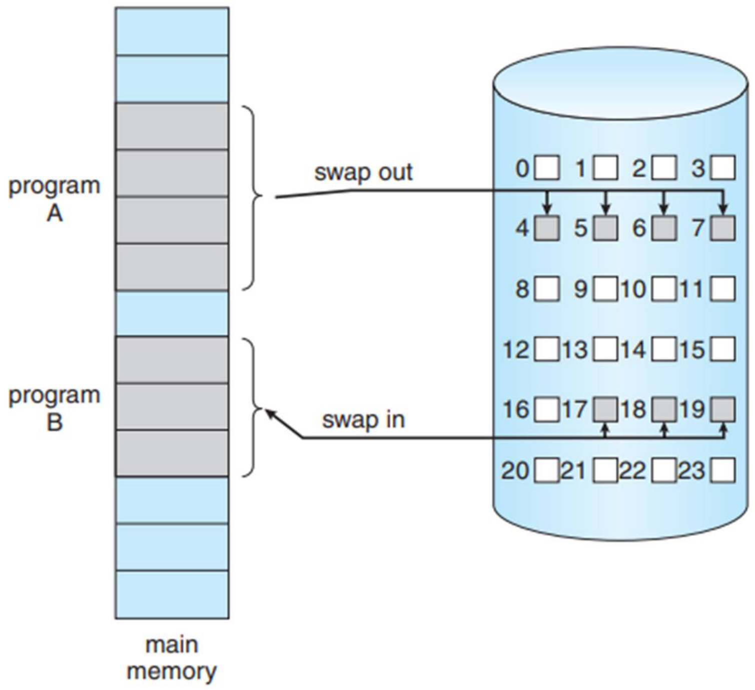

Figure 9.4 Transfer of a paged memory to contiguous disk space.

Basic Concepts

- ❖ With swapping, pager (swapper) guesses which pages will be used before swapping out again
- ❖ Instead, pager brings in only those pages (pages that will be used) into memory
- ❖ If page needed and not memory resident
  - ✓ Need to detect and load the page into memory from storage
    - ▪ Without changing program behavior
    - ▪ Without programmer needing to change code

Valid-Invalid Bit

- ❖ hardware support needed to distinguish between the pages that are in memory and the pages that are on the disk.
- ❖ With each page table entry a valid–invalid bit is associated
  - ✓ **v** → in-memory (memory resident)
  - ✓ **i** → not-in-memory
- ❖ Initially valid–invalid bit is set to **i** on all entries
- ❖ Example of a page table when some pages are not in main memory next figure (Figure 9.5):

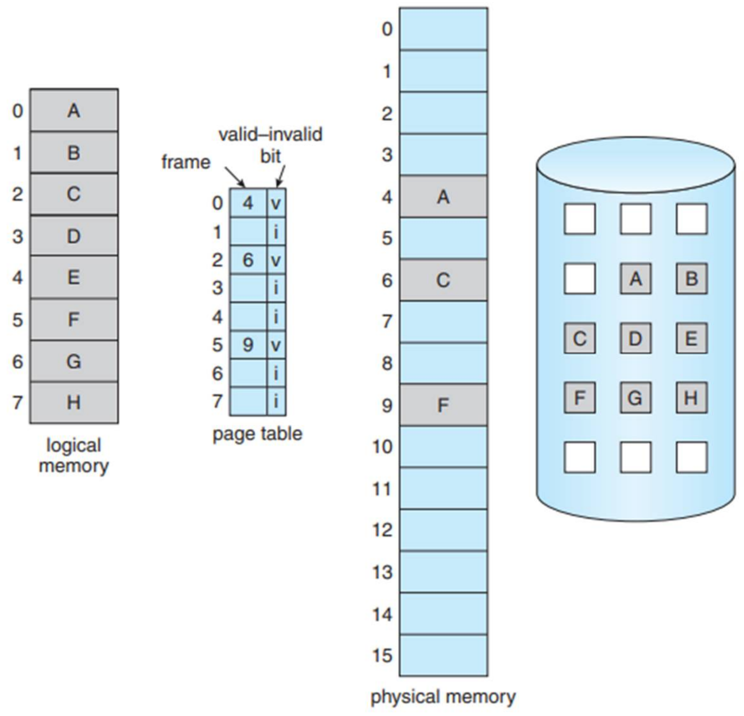

Figure 9.5 Page table when some pages are not in main memory.

❖ During MMU address translation, if valid–invalid bit in page table and entry is **i** then access to it causes **page fault**

Page Fault handling steps

❖ The procedure for handling this page fault is straightforward next figure (Figure 9.6):

1. Check an internal table (usually kept with the process control block) for this process to determine whether the reference was a valid or an invalid memory access.

2. If the reference was invalid, terminate the process. If it was valid but have not yet brought in that page, now page it in.

3. Find a free frame (by taking one from the free-frame list, for example).
4. Schedule a disk operation to read the desired page into the newly allocated frame.

5. When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.

6. restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory

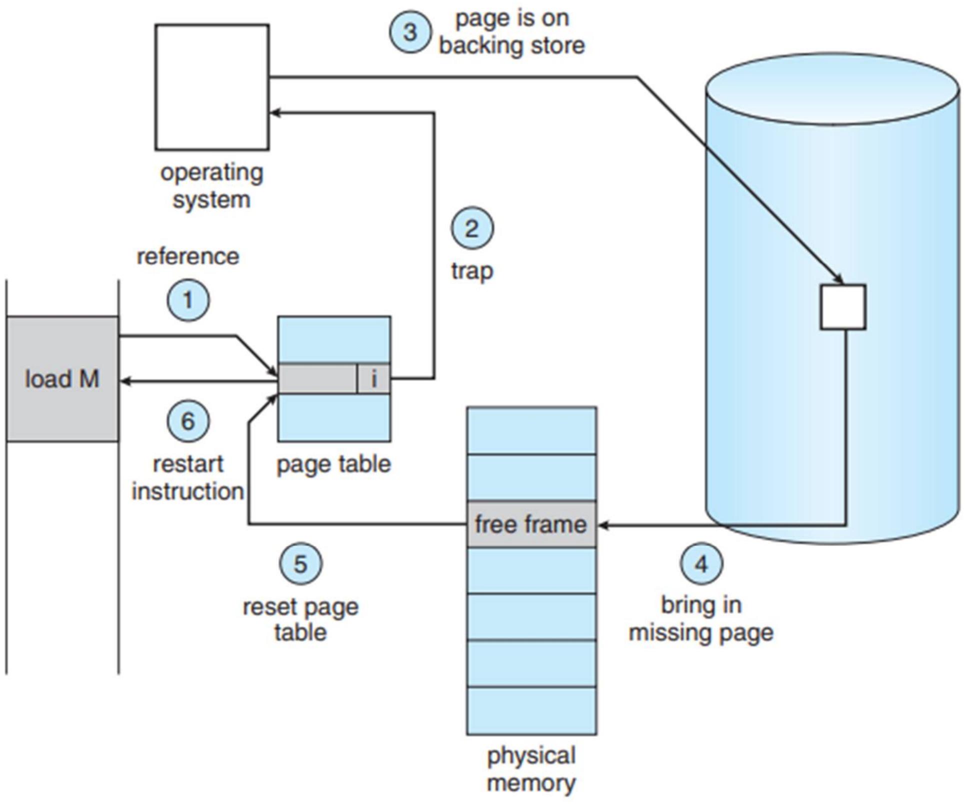

Figure 9.6 Steps in handling a page fault.

Aspects of Demand Paging

- ❖ **Extreme case** – start process with no pages in memory
  - ✓ OS sets instruction pointer to first instruction of process, if it non-memory-resident -> page fault
  - ✓ And for every other process pages on first access
- ❖ **Pure demand paging**:
  - ✓ Never bring a page into memory until it is required.

- ❖ Hardware support needed for demand paging
  - ✓ Page table with valid / invalid bit
  - ✓ Secondary memory (swap device with swap space)
  - ✓ Instruction restart

What Happens if There is no Free Frame?

- ❖ Page replacement – find some page in memory, but not really in use, page it out
  - ✓ **Algorithm:** terminate? swap out? replace the page?
  - ✓ **Performance:** want an algorithm which will result in minimum number of page faults
- ❖ Same page may be brought into memory several times

Page Replacement

- ❖ If no frame is free find one that is not currently being used and free it.
  - ✓ Write its contents to swap space and changing the page table (and all other tables) to indicate that the page is no longer in memory
  - ✓ Use the freed frame to hold the page for which the process faulted.

- ❖ Use **modify (dirty) bit** to reduce overhead of page transfers – only modified pages are written to disk
- ❖ Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory

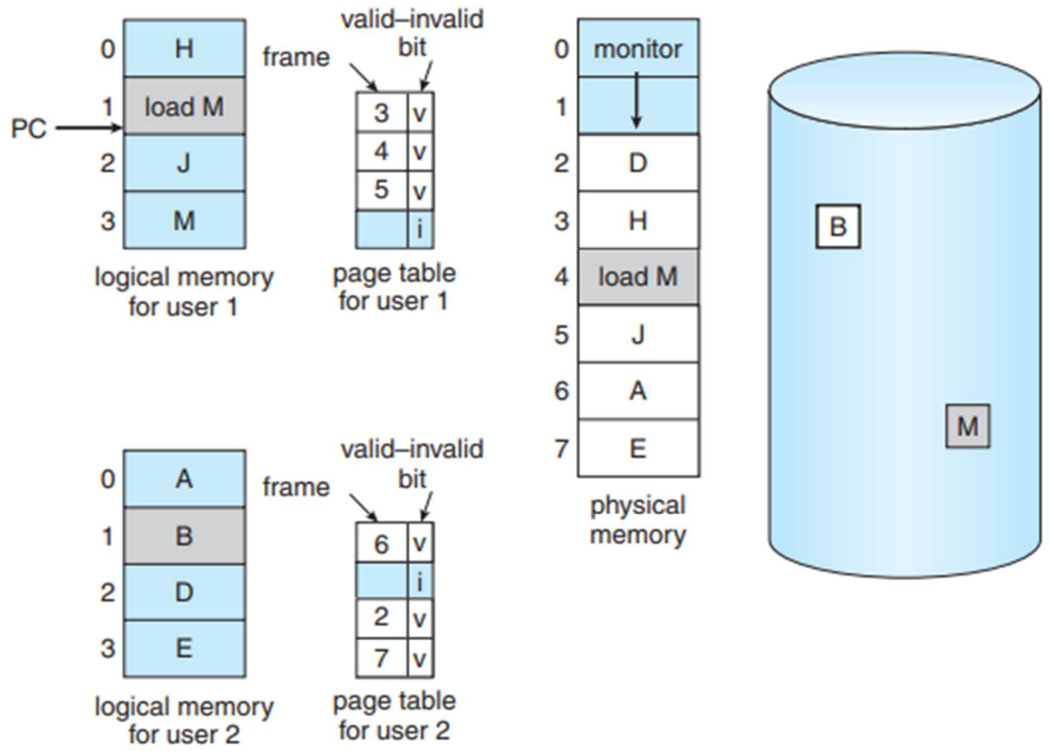

Figure 9.9 Need for page replacement.

Basic Page Replacement

1. Find the location of the desired page on the disk.
2. Find a free frame:
   a) If there is a free frame, 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 and frame tables accordingly.
3. Read the desired page into the newly freed frame; change the page and frame tables.
4. Continue the user process from where the page fault occurred.

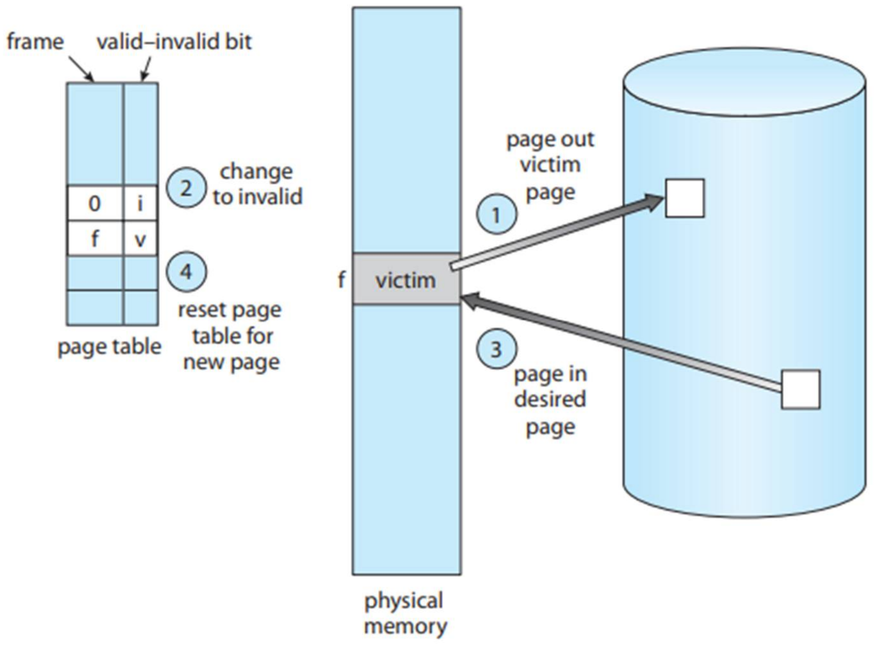

Figure 9.10 Page replacement.

Page and Frame Replacement Algorithms

❖ **Frame-allocation algorithm:** determines
- ✓ How many frames to give each process
- ✓ Which frames to replace

❖ **Page-replacement algorithm:** want lowest page-fault rate on both first access and re-access

❖ Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
- ✓ String is just page numbers, not full addresses
- ✓ Repeated access to the same page does not cause a page fault
- ✓ Results depend on number of frames available

Note :In all our examples, the reference string of referenced page numbers is
7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

1. First-In-First-Out (FIFO) Algorithm

❖ 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.

❖ Replace the page at the head of the queue. When a page is brought into memory, insert it at the tail of the queue.

❖ Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

❖ 3 frames (3 pages can be in memory at a time per process)

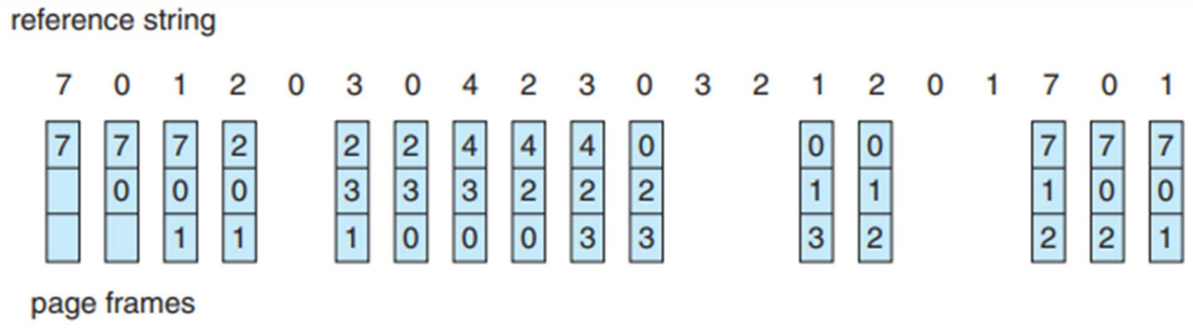

Figure 9.12 FIFO page-replacement algorithm.

2. Optimal Algorithm

- ❖ Replace the page that will not be used for the longest period of time.
- ❖ This algorithm that has the lowest page-fault rate of all algorithms.

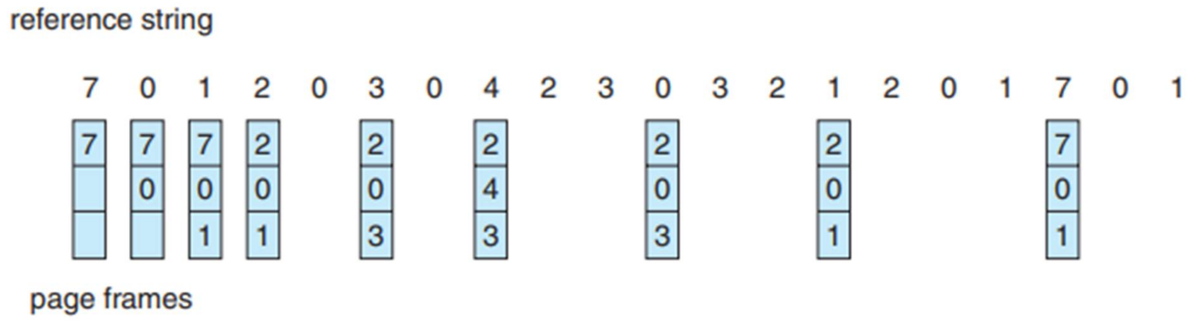

Figure 9.14 Optimal page-replacement algorithm.

3. Least Recently Used (LRU) Algorithm

- ❖ Use past knowledge rather than future
- ❖ 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

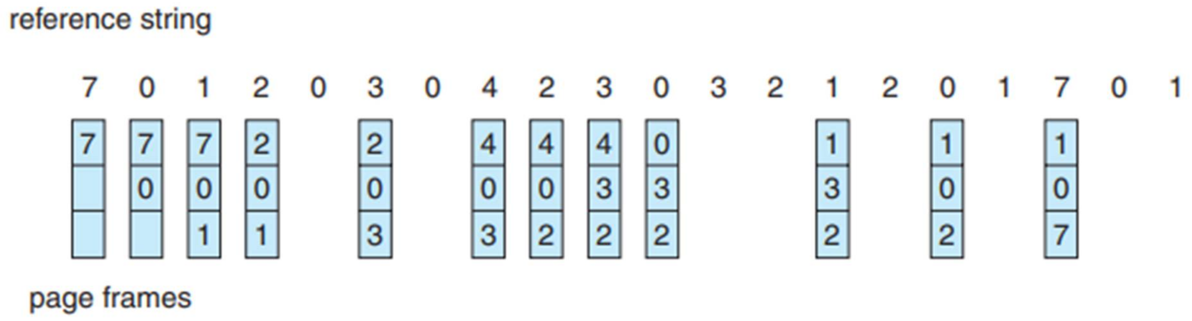

Figure 9.15 LRU page-replacement algorithm.

Two implementations are feasible

a) Counter implementation

- ✓ Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
- ✓ When a page needs to be changed, look at the counters to find smallest value
- ✓ Search through table needed

b) Stack implementation

- ✓ Keep a stack of page numbers in a double link form:
- ✓ Page referenced:
  - ▪ move it to the top
  - ▪ requires 6 pointers to be changed
- ✓ But each update more expensive
- ✓ No search for replacement

❖ LRU and OPT are cases of stack algorithms that don't have Belady's Anomaly

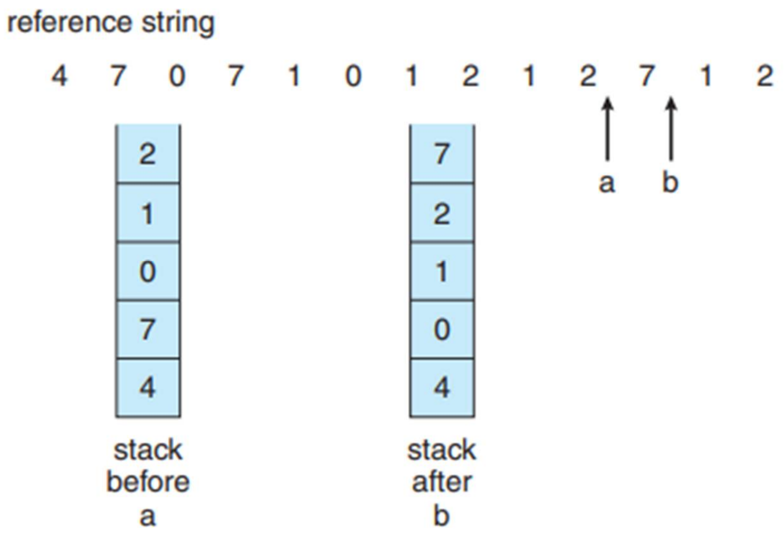

Figure 9.16 Use of a stack to record the most recent page references.