University Institute of Engineering

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Unit- 2

Subject Name : Operating System

Chapter : Segmentation and Virtual Memory

Prepared By: Er. Inderjeet Singh

Outline

- Virtual Memory
- Demand Paging
- Page Fault
- Page Replacement Algorithms
- Thrashing

VIRTUAL MEMORY

- A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard disk that's set up to emulate the computer's RAM.
- The main visible advantage of this scheme is that programs can be larger than physical memory.
- Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk.
- Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

How Virtual Memory Works?

- In modern word, virtual memory has become quite common these days. In this scheme, whenever some pages needs to be loaded in the main memory for the execution and the memory is not available for those many pages, then in that case, instead of stopping the pages from entering in the main memory, the OS search for the RAM area that are least used in the recent times or that are not referenced and copy that into the secondary memory to make the space for the new pages in the main memory.
- Since all this procedure happens automatically, therefore it makes the computer feel like it is having the unlimited RAM.

VIRTUAL MEMORY BENEFITS

- • Less I/O would be needed to load or swap user programs into memory, so each user program would run faster.
- Thus, running a program that is not entirely in memory would benefit both the system and the user.
- Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory

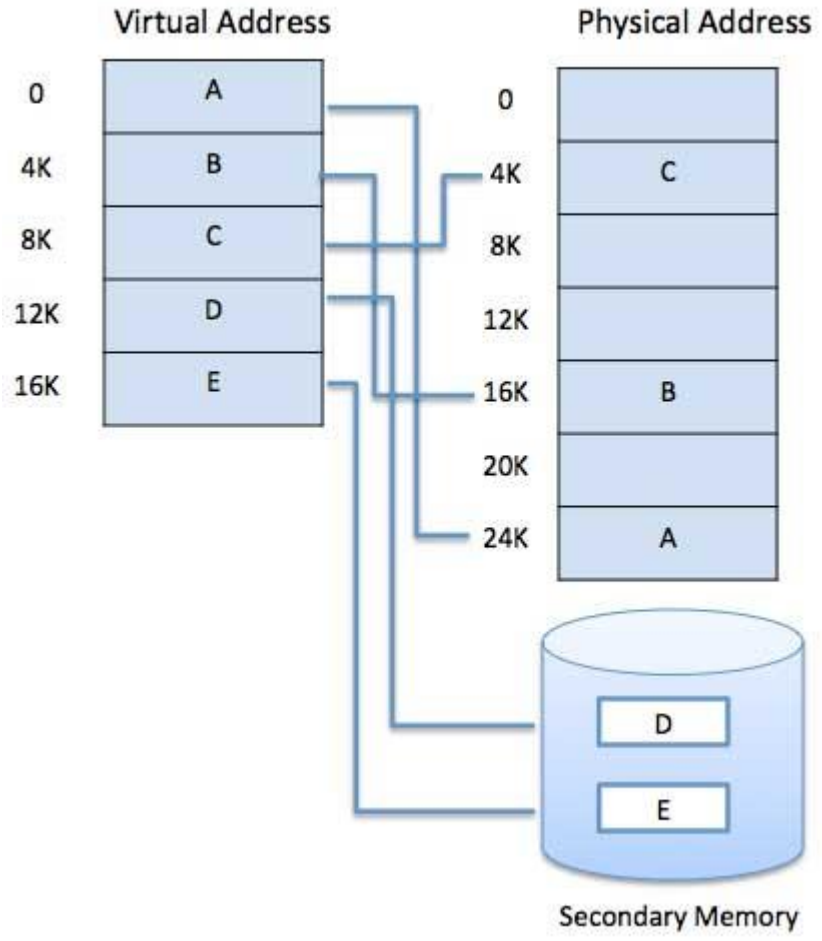


DIAGRAM FOR VIRTUAL MEMORY

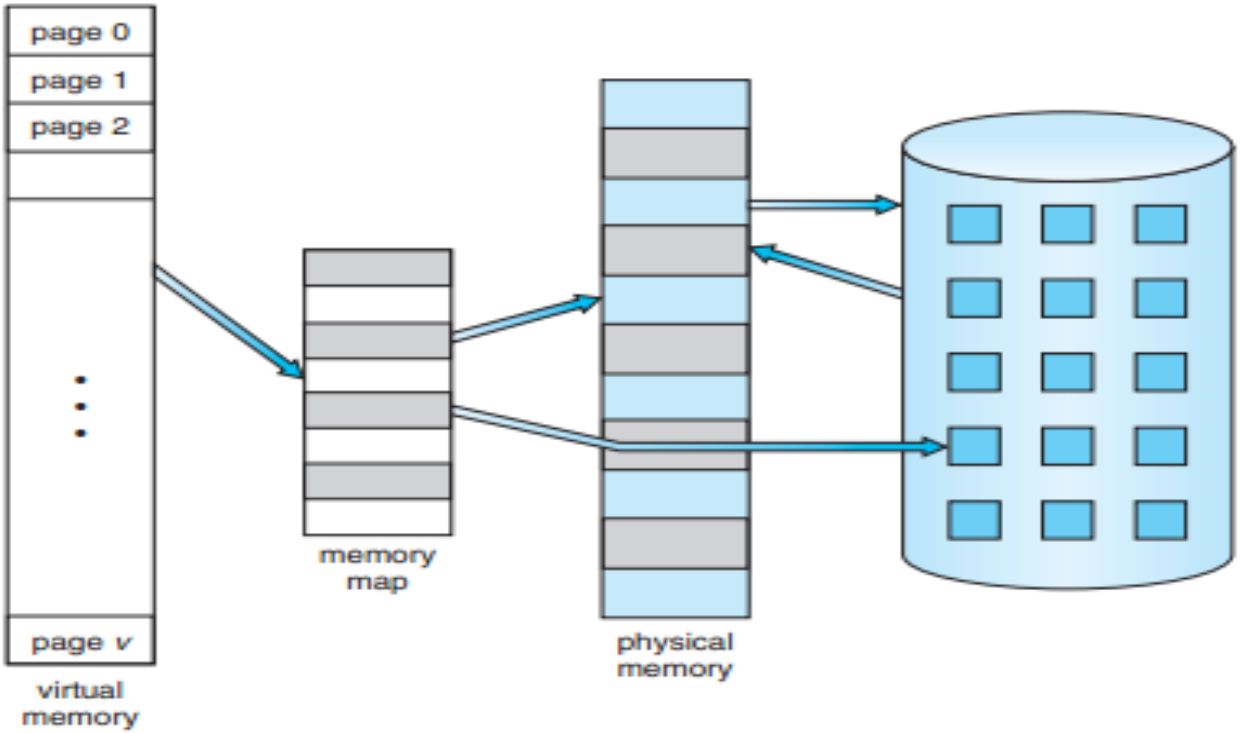


Figure 9.1 Diagram showing virtual memory that is larger than physical memory.

DEMAND PAGING

- Virtual memory is commonly implemented by demand paging.
- A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance.

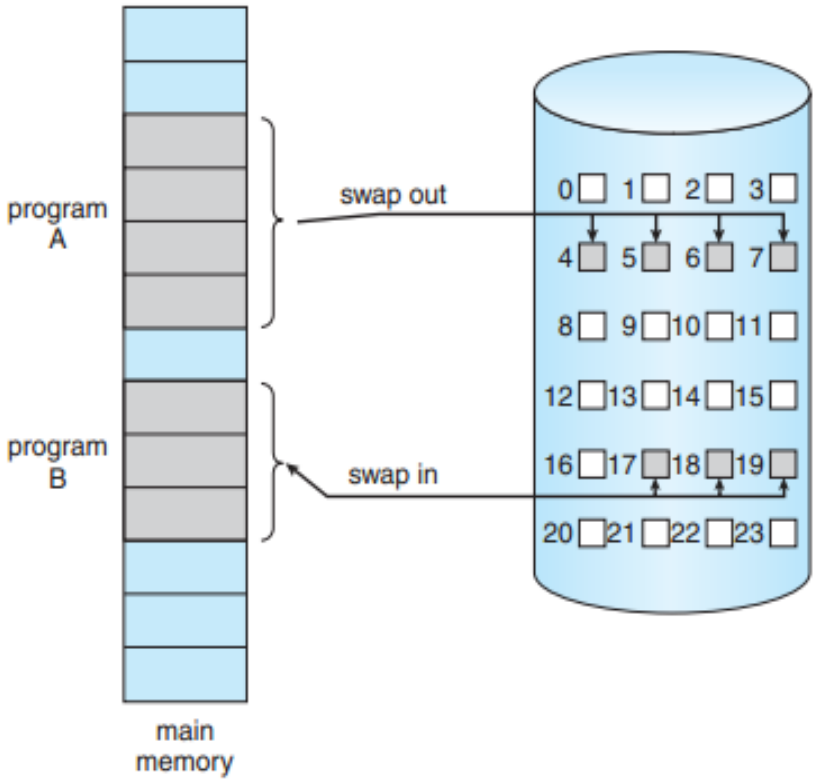


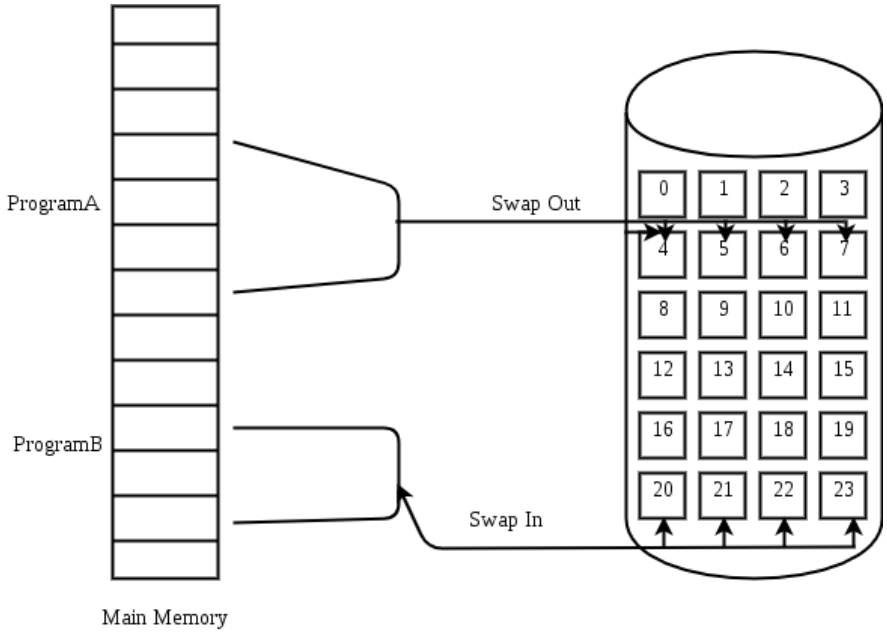
Figure 9.4 Transfer of a paged memory to contiguous disk space.

ADVANTAGES OF DEMAND PAGING

- More processes may be maintained in the main memory: Because we are going to load only some of the pages of any particular process, there is room for more processes. This leads to more efficient utilization of the processor because it is more likely that at least one of the more numerous processes will be in the ready state at any particular time.
- A process may be larger than all of main memory: One of the most fundamental restrictions in programming is lifted. A process larger than the main memory can be executed because of demand paging. The OS itself loads pages of a process in main memory as required.

Swapping

Swapping a process out means removing all of its pages from memory, or marking them so that they will be removed by the normal page replacement process. Suspending a process ensures that it is not runnable while it is swapped out. At some later time, the system swaps back the process from the secondary storage to main memory. When a process is busy swapping pages in and out then this situation is called thrashing.



DEMAND PAGING: Valid and Invalid Bit

- The valid – invalid bit scheme is used to identify weather the required page is available in main memory or not.
- If the bit is set to (v)valid , it means page is available in memory.
- If the bit is set to i(invalid), it means page is not available in main memory but available on disk.

DEMAND PAGING

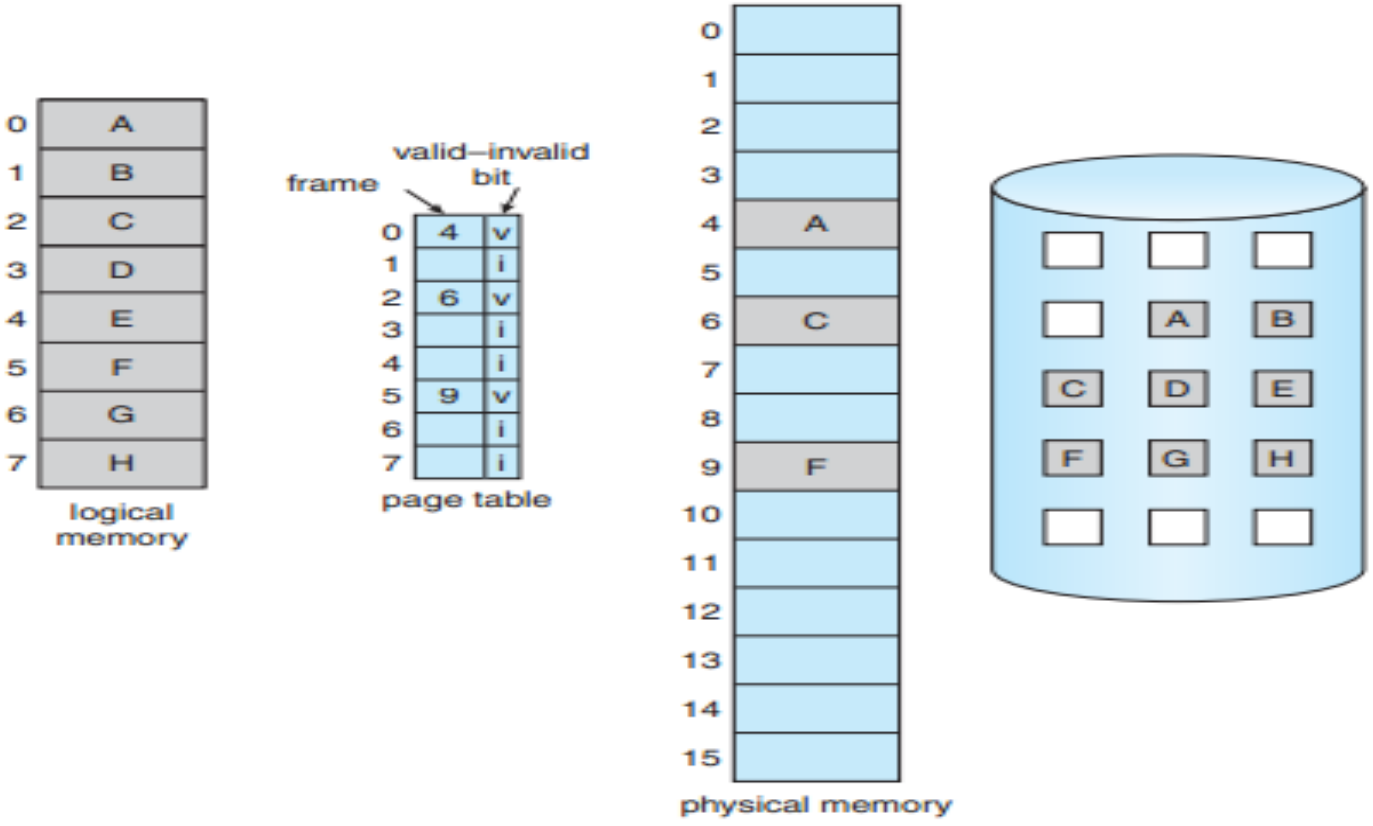


Figure 9.5 Page table when some pages are not in main memory.

PAGE FAULT

- 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.

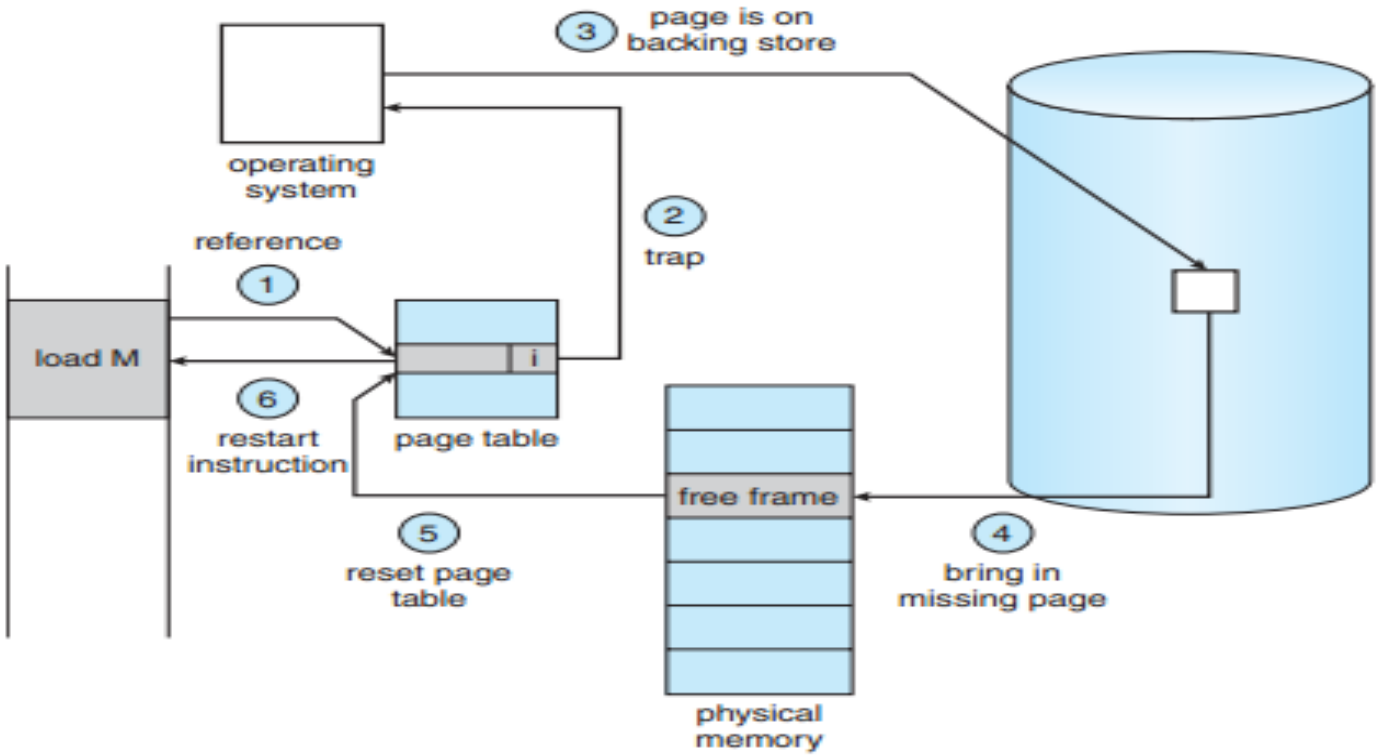


Figure 9.6 Steps in handling a page fault.

PAGE FAULT

1. We 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, we terminate the process. If it was valid but we have not yet brought in that page, we now page it in.

3. We find a free frame (by taking one from the free-frame list, for example).

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

PAGE FAULT SERVICE TIME

The time taken to service the page fault is called as page fault service time. The page fault service time includes the time taken to perform all the above six steps.

Let Main memory access time is: m

Page fault service time is: s

Page fault rate is : p

Then, Effective memory access time = (p*s) + (1-p)*m

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 (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. We modify the page-fault service routine to include page replacement:

PAGE REPLACEMENT

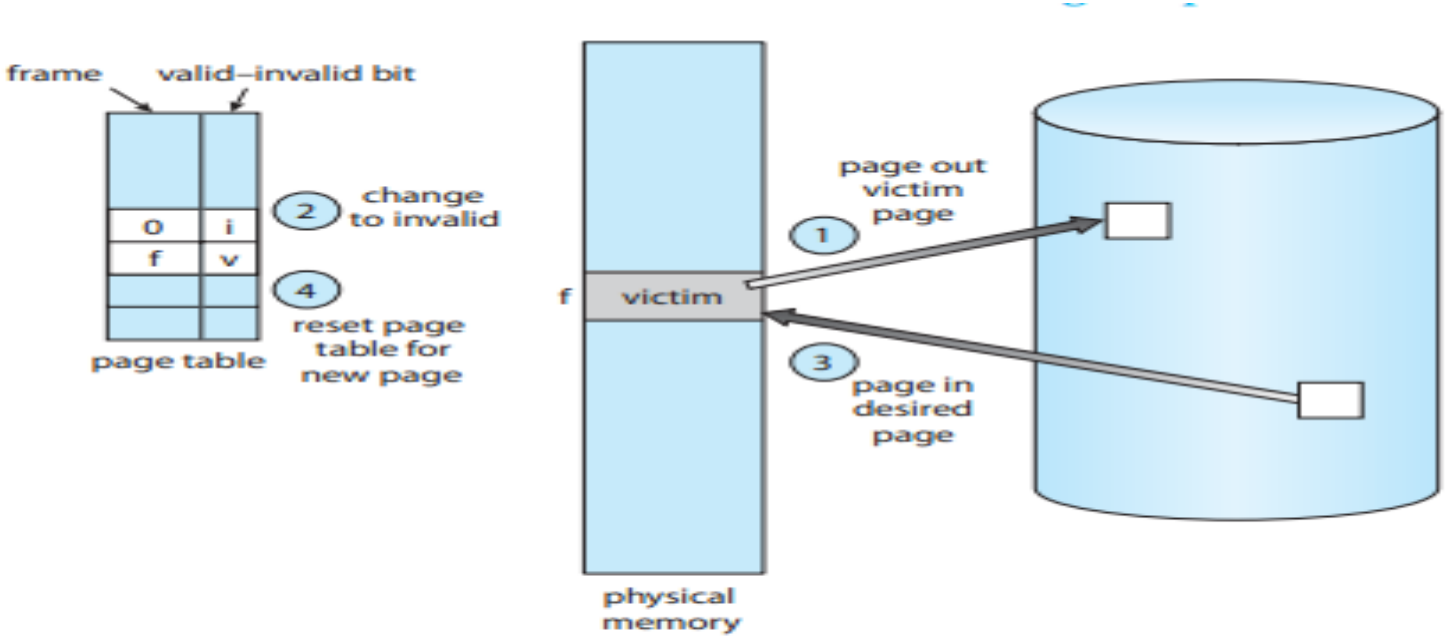


Figure 9.10 Page replacement.

PAGE REPLACEMENT ALGORITHM

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.

FIFO PAGE REPLACEMENT 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.

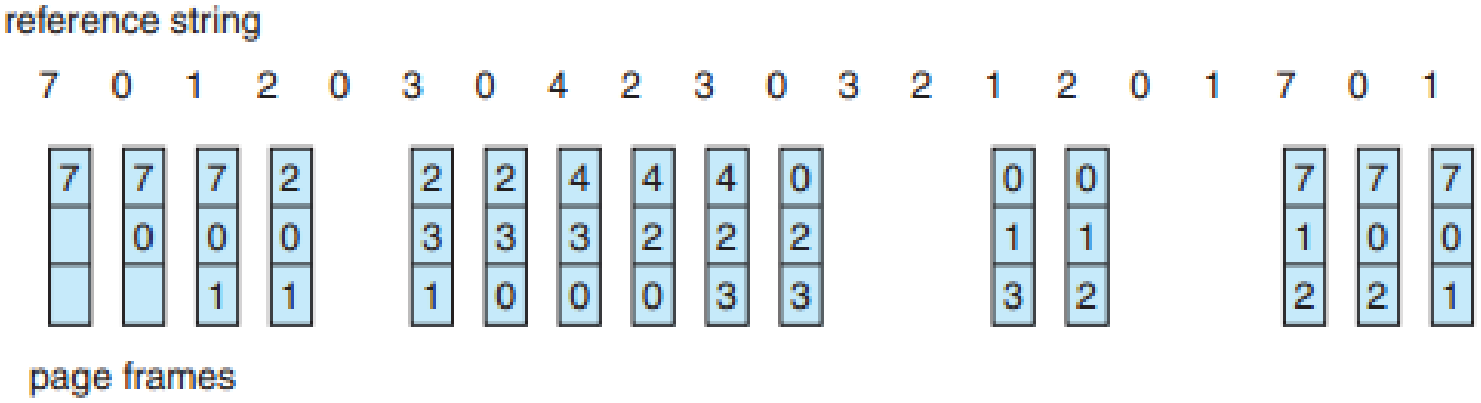


Figure 9.12 FIFO page-replacement algorithm.

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.
- It is simply this: Replace the page that will not be used for the longest period of time.

OPTIMAL PAGE REPLACEMENT

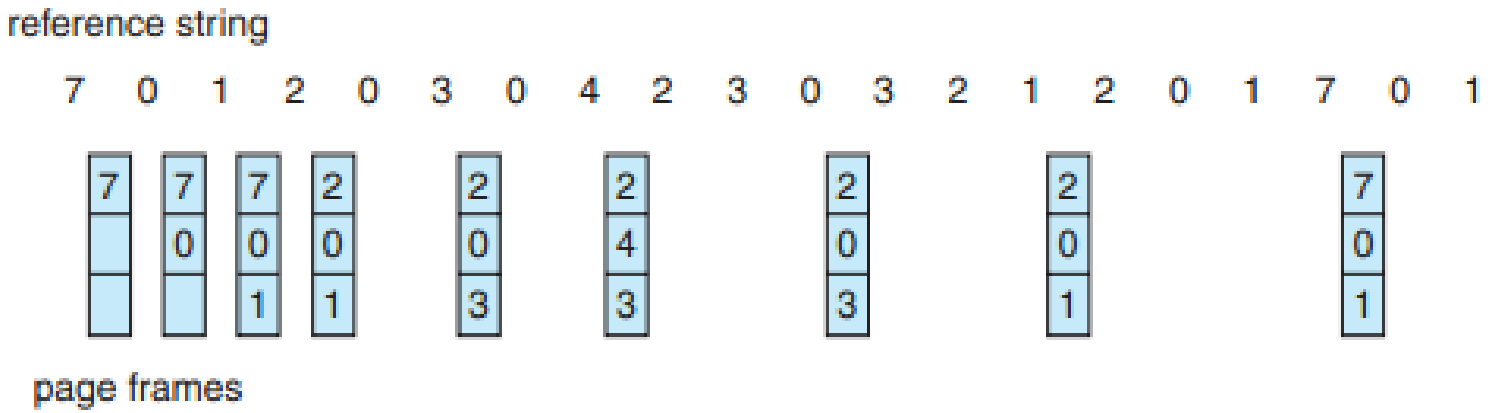


Figure 9.14 Optimal page-replacement algorithm.

Unfortunately, the optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.

LRU PAGE REPLACEMENT

LRU chooses the page that has not been used for the longest period of time in the past.

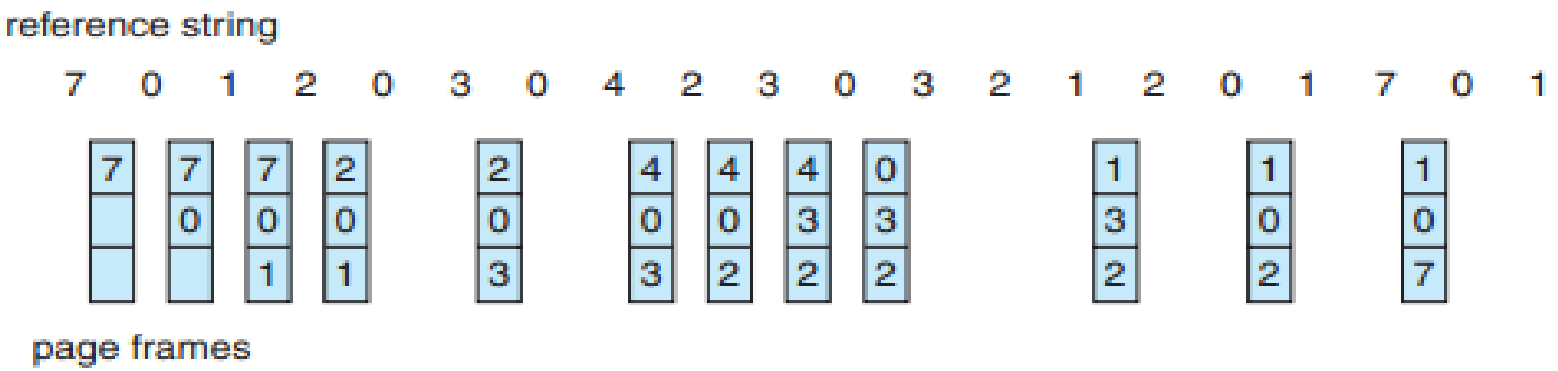


Figure 9.15 LRU page-replacement algorithm.

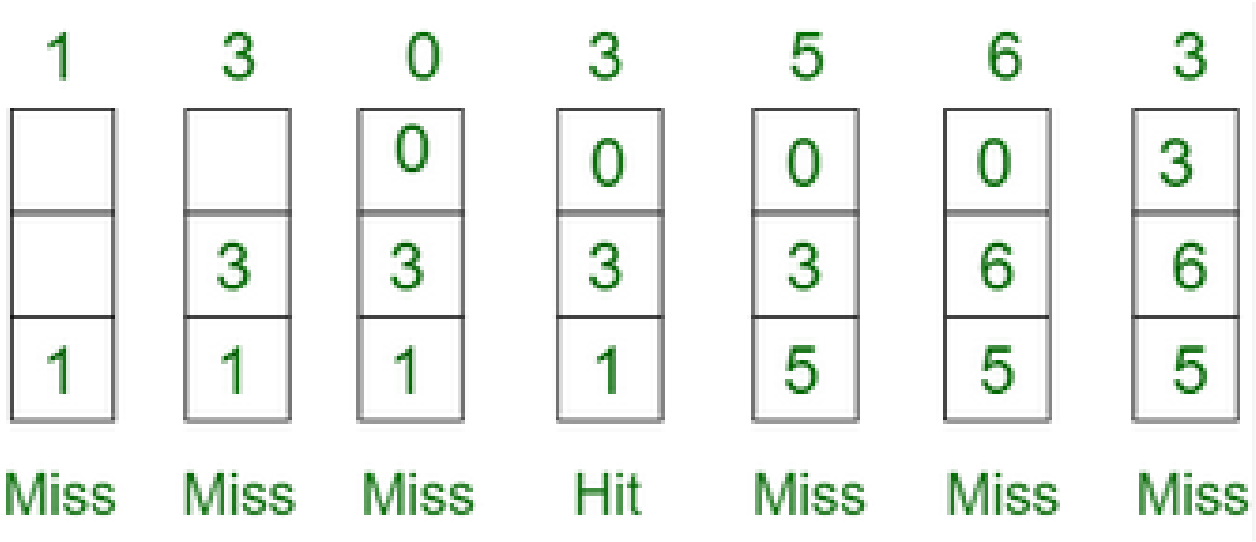
Like optimal replacement, LRU replacement does not suffer from Belady's anomaly.

Exercise: FIFO PAGE REPLACEMENT

Example-1 Consider page reference string 1, 3, 0, 3, 5, 6,3 with 3 page frames. Find number of page faults.

Exercise: FIFO PAGE REPLACEMENT

Example-1 Consider page reference string 1, 3, 0, 3, 5, 6,3 with 3 page frames. Find number of page faults.



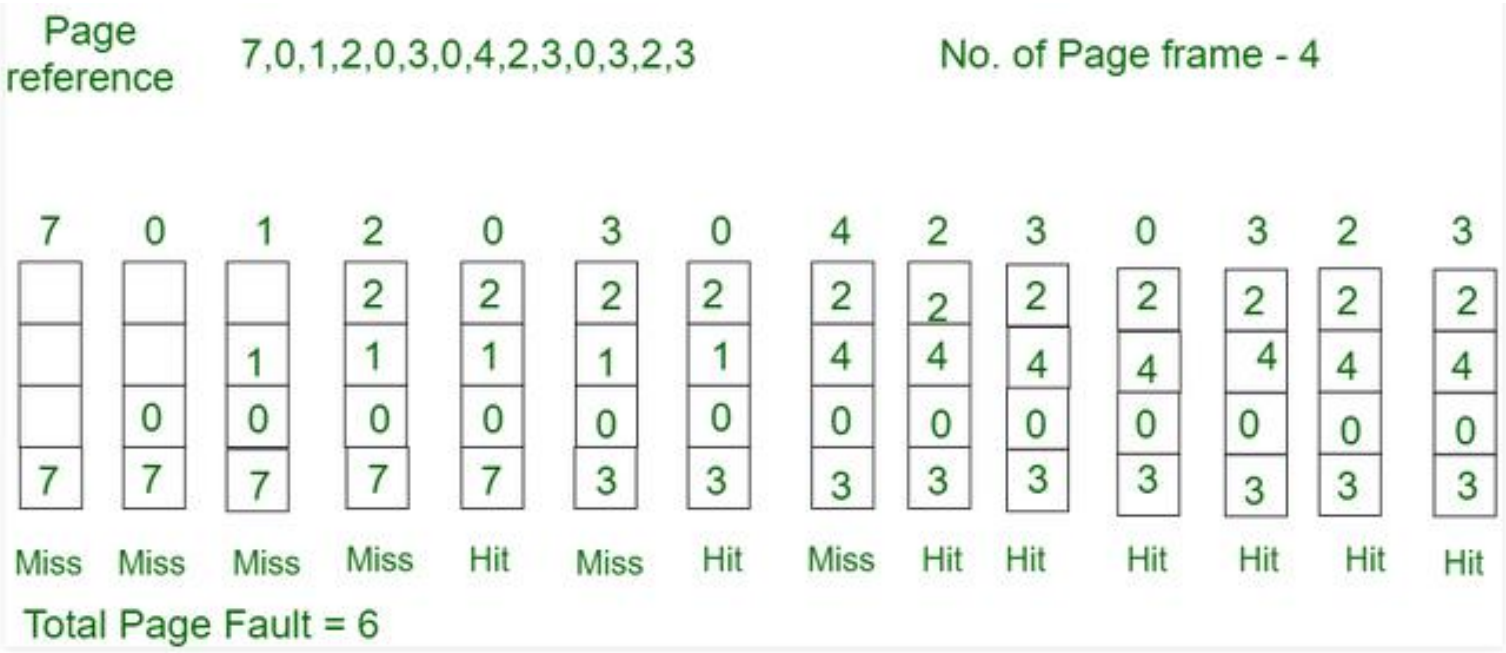
Total Page Fault = 6

Exercise: Optimal PAGE REPLACEMENT

Example-2: Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frame. Find number of page fault.

Exercise: Optimal PAGE REPLACEMENT

Example-2: Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frame. Find number of page fault.

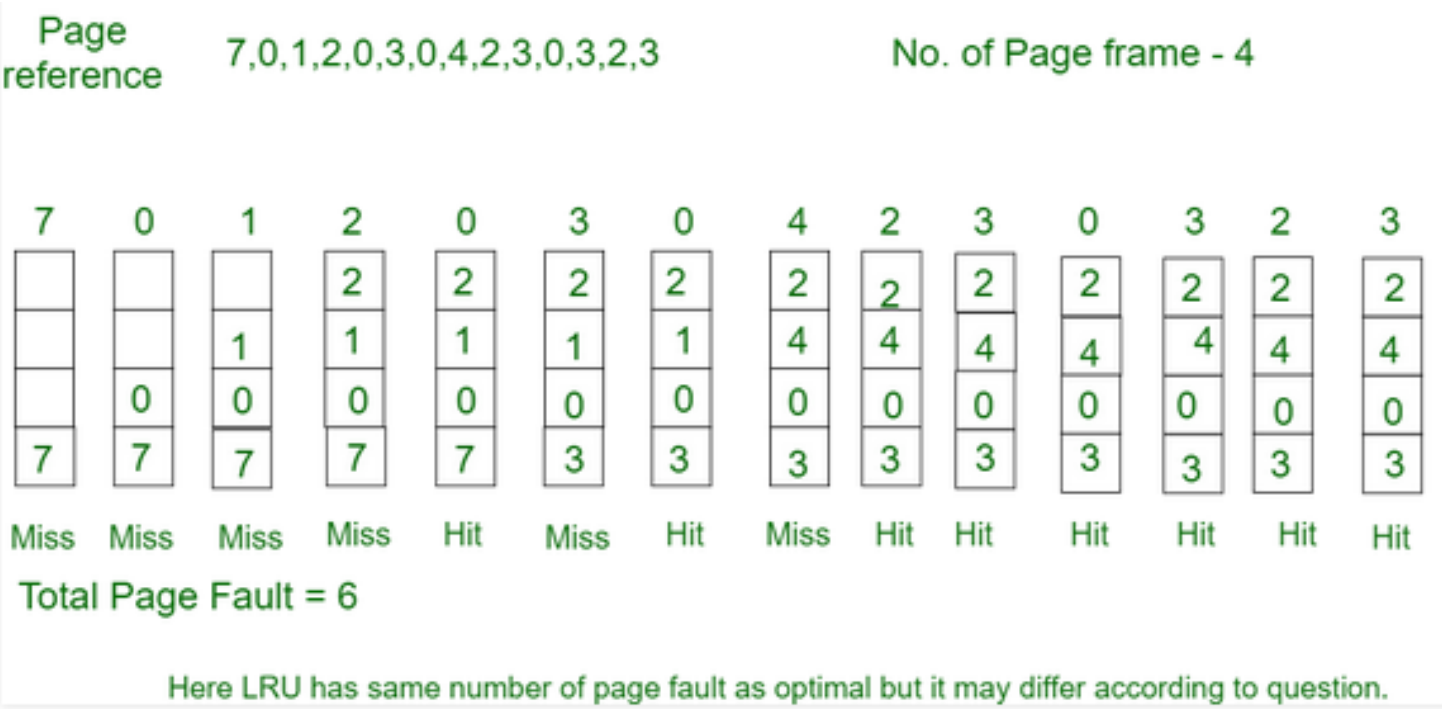


Exercise: LRU PAGE REPLACEMENT

Example-3 Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frames .Find number of page faults.

Exercise: LRU PAGE REPLACEMENT

Example-3 Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frames .Find number of page faults.



Exercise: LRU PAGE REPLACEMENT

Q. Consider a reference string: 4, 7, 6, 1, 7, 6, 1, 2, 7, 2. the number of frames in the memory is 3. Find out the number of page faults respective to:

1. Optimal Page Replacement Algorithm
2. FIFO Page Replacement Algorithm
3. LRU Page Replacement Algorithm

Exercise: LRU PAGE REPLACEMENT

Q. Consider a reference string: 4, 7, 6, 1, 7, 6, 1, 2, 7, 2. the number of frames in the memory is 3. Find out the number of page faults respective to:

1. Optimal Page Replacement Algorithm

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	2	2	2
Frame 2		7	7	7	7	7	7	7	7	7
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Hit	Hit

Total Page Faults: 5

Exercise: LRU PAGE REPLACEMENT

Q. Consider a reference string: 4, 7, 6, 1, 7, 6, 1, 2, 7, 2. the number of frames in the memory is 3. Find out the number of page faults respective to:

1. LRU Page Replacement Algorithm

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	6	7	7
Frame 2		7	7	7	7	7	7	2	2	2
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Miss	Hit

Total Page Faults: 6

Exercise: LRU PAGE REPLACEMENT

Q. Consider a reference string: 4, 7, 6, 1, 7, 6, 1, 2, 7, 2. the number of frames in the memory is 3. Find out the number of page faults respective to:

1. FIFO Page Replacement Algorithm

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	6	7	7
Frame 2		7	7	7	7	7	7	2	2	2
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Miss	Hit

Total Page Faults: 6

Belady's Anomaly

- In the case of LRU and optimal page replacement algorithms, it is seen that the number of page faults will be reduced if we increase the number of frames. However, Balady found that, In FIFO page replacement algorithm, the number of page faults will get increased with the increment in number of frames.
- This is the strange behaviour shown by FIFO algorithm in some of the cases. This is an Anomaly called as Belady's Anomaly.

Belady's Anomaly

Let's examine such example :

- The reference String is given as 0 1 5 3 0 1 4 0 1 5 3 4. Let's analyze the behavior of FIFO algorithm in two cases.

Case 1: Number of frames = 3

Request	0	1	5	3	0	1	4	0	1	5	3	4
Frame 3			5	5	5	1	1	1	1	1	3	3
Frame 2		1	1	1	0	0	0	0	0	5	5	5
Frame 1	0	0	0	3	3	3	4	4	4	4	4	4
Miss/Hit	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Hit	Hit	Miss	Miss	Hit

Number of Page Faults = 9

Belady's Anomaly

Case 2: Number of frames = 4

Request	0	1	5	3	0	1	4	0	1	5	3	4
Frame 4				3	3	3	3	3	3	5	5	5
Frame 3			5	5	5	5	5	5	1	1	1	1
Frame 2		1	1	1	1	1	1	0	0	0	0	4
Frame 1	0	0	0	0	0	0	4	4	4	4	3	3
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Miss	Miss	Miss	Miss	Miss	Miss

Number of Page Faults = 10

Therefore, in this example, the number of page faults is increasing by increasing the number of frames hence this suffers from Belady'sAnomaly.

THRASHING

- If the number of frames allocated to a low-priority process falls below the minimum number required by the computer architecture, we must suspend that process's execution. We should then page out its remaining pages, freeing all its allocated frames. This provision introduces a swap-in, swap-out level of intermediate CPU scheduling.
- In fact, look at any process that does not have "enough" frames. If the process does not have the number of frames it needs to support pages in active use, it will quickly page-fault. At this point, it must replace some page. However, since all its pages are in active use, it must replace a page that will be needed again right away. Consequently, it quickly faults again, and again, and again, replacing pages that it must bring back in immediately.
- This high paging activity is called thrashing. A process is thrashing if it is spending more time paging than executing.

CAUSES OF THRASHING

- Consider the following scenario, which is based on the actual behaviour of early paging systems.
- The operating system monitors CPU utilization. If CPU utilization is too low, we increase the degree of multiprogramming by introducing a new process to the system. A global page-replacement algorithm is used; it replaces pages without regard to the process to which they belong. Now suppose 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 the paging device, the ready queue empties. As processes wait for the paging device, CPU utilization decreases.

CAUSES OF THRASHING

- The CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming as a result. 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, CPU utilization drops even further, and the CPU scheduler tries to increase the degree of multiprogramming even more. Thrashing has occurred, and system throughput plunges. The page-fault rate increases tremendously. As a result, the effective memory-access time increases. No work is getting done, because the processes are spending all their time paging.

THRASHING

Figure 9.18, in which CPU utilization is plotted against the degree of multiprogramming. As the degree of multi-programming increases, CPU utilization also increases, although more slowly, until a maximum is reached. If the degree of multiprogramming is increased even further, thrashing sets in, and CPU utilization drops sharply. At this point, to increase CPU utilization and stop thrashing, we must decrease the degree of multiprogramming.

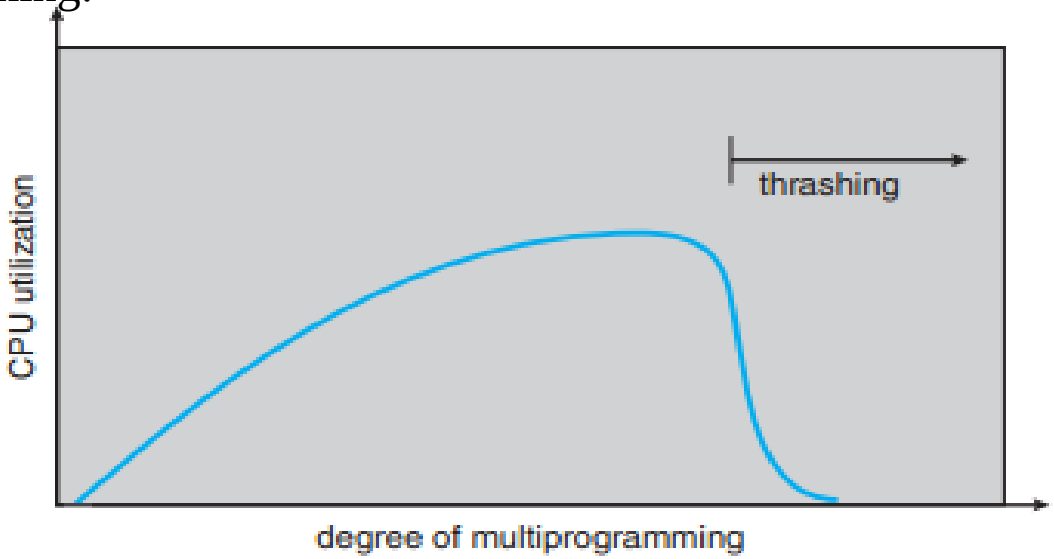


Figure 9.18 Thrashing.

TECHNIQUES TO HANDLE THRASHING

Working set Model-

- As mentioned, the working-set model is based on the assumption of locality. This model uses a parameter, to define the working-set window. The idea is to examine the most recent page references. The set of pages in the most recent page references is the working set.
- If a page is in active use, it will be in the working set. If it is no longer being used, it will drop from the working set time units after its last reference.

For example, given the sequence of memory references shown in Figure 9.20, if $\Delta = 10$ memory references, then the working set at time t_1 is {1, 2, 5, 6, 7}. By time t_2, the working set has changed to {3, 4}.

The accuracy of the working set depends on the selection of Δ. If Δ is too small, it will not encompass the entire locality; if Δ is too large, it may overlap several localities. In the extreme, if Δ is infinite, the working set is the set of pages touched during the process execution.

TECHNIQUES TO HANDLE THRASHING

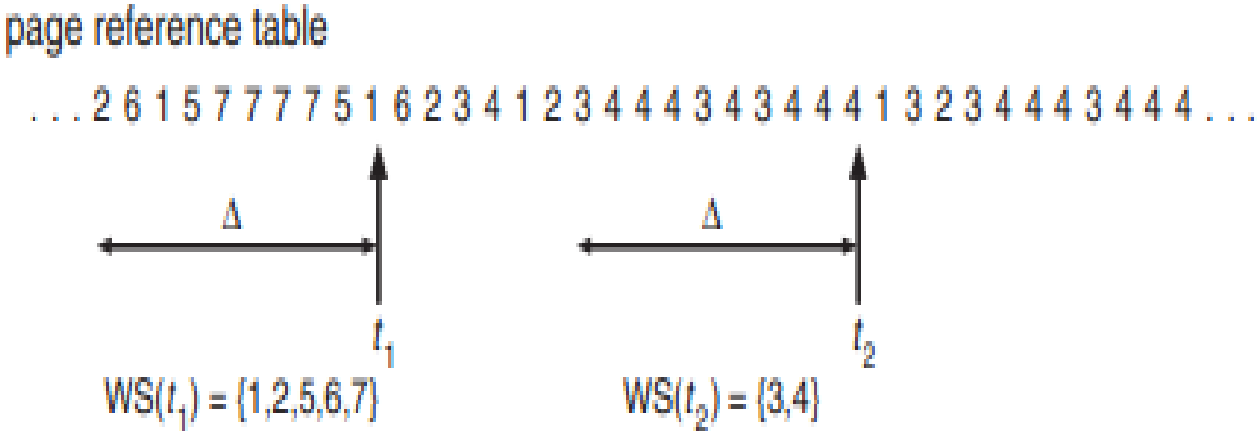


Figure 9.20 Working-set model.

TECHNIQUES TO HANDLE THRASHING

The most important property of the working set, then, is its size. If we compute the working-set size, WSS_i, for each process in the system, we can then consider that

$$D = \sum WSS_i,$$

where D is the total demand for frames. Each process is actively using the pages in its working set. Thus, process i needs WSS_i frames. If the total demand is greater than the total number of available frames ($D > m$), thrashing will occur, because some processes will not have enough frames.

This working-set strategy prevents thrashing while keeping the degree of multiprogramming as high as possible. Thus, it optimizes CPU utilization. The difficulty with the working-set model is keeping track of the working set. The working-set window is a moving window. At each memory reference, a new reference appears at one end, and the oldest reference drops off the other end.

PAGE FAULT FREQUENCY

- Thrashing has a high page-fault rate. Thus, we want to control the page-fault rate.
- When it is too high, we know that the process needs more frames. Conversely, if the page-fault rate is too low, then the process may have too many frames. We can establish upper and lower bounds on the desired page-fault rate.
- If the actual page-fault rate exceeds the upper limit, we allocate the process another frame. If the page-fault rate falls below the lower limit, we remove a frame from the process.
- Thus, we can directly measure and control the page-fault rate to prevent thrashing.

PAGE FAULT FREQUENCY

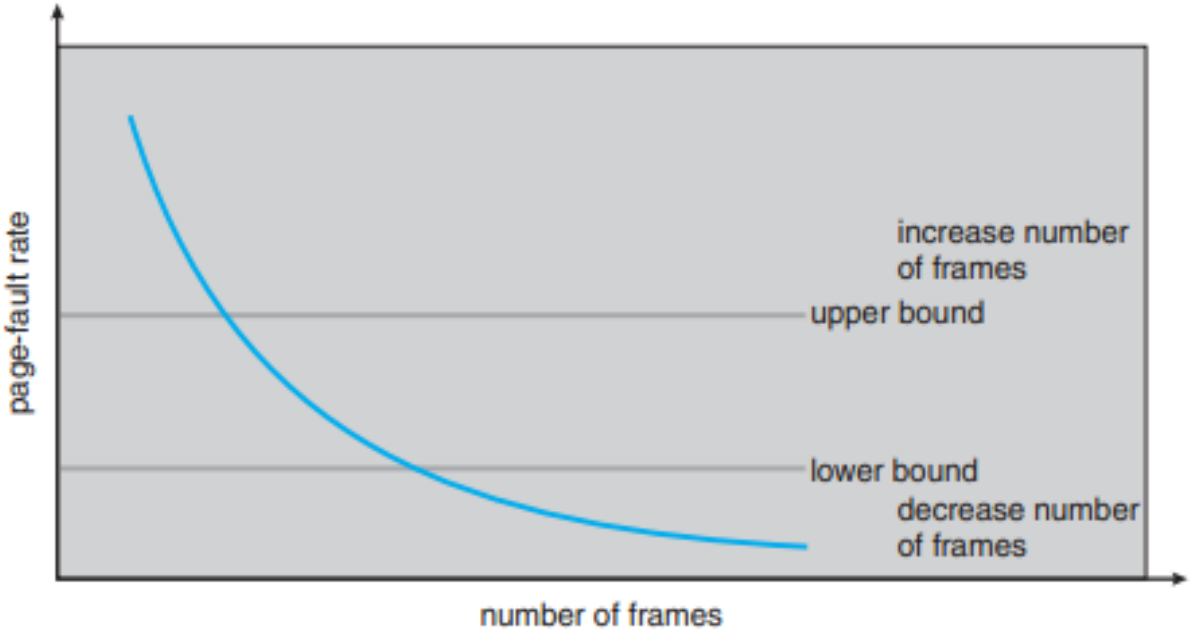
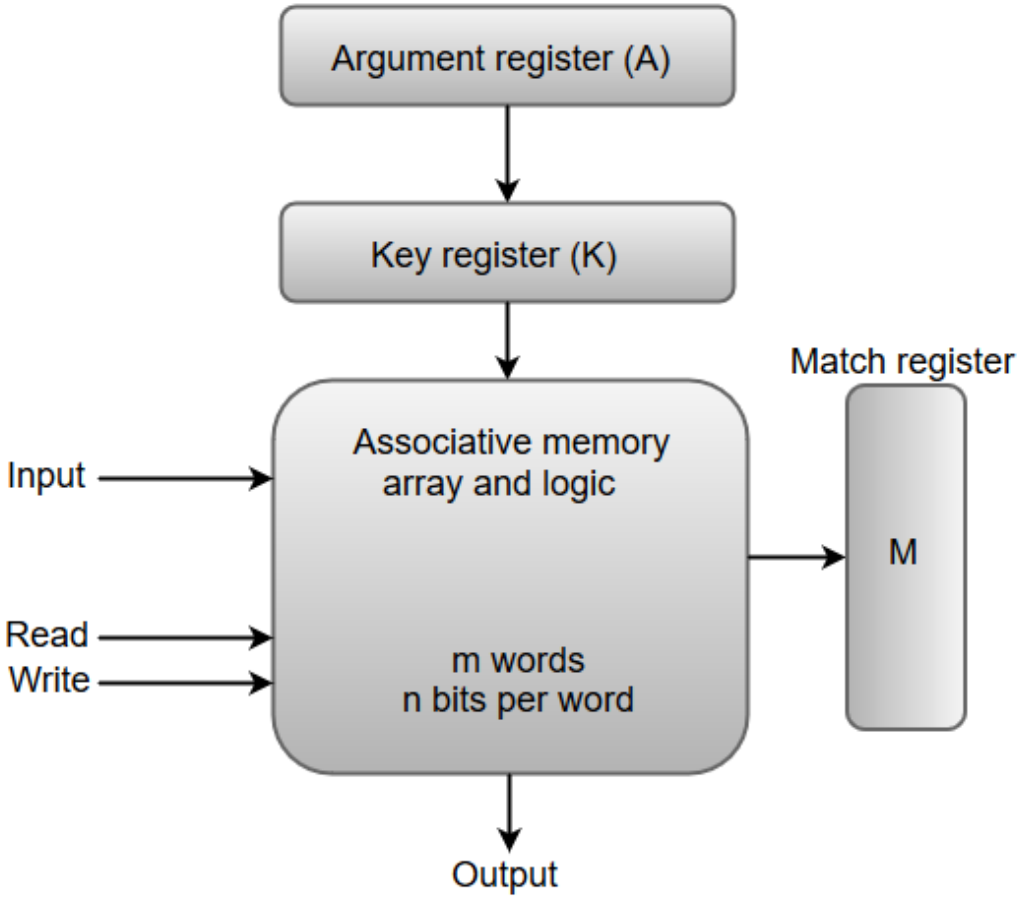


Figure 9.21 Page-fault frequency.

ASSOCIATIVE MEMORY

- Memory is accessed simultaneously and in parallel on basis of data rather than address or location. Associative memory is often referred to as **Content Addressable Memory (CAM)**.
- It is more expensive than RAM – Each cell must have storage capability & logic circuits for matching content with external argument
- When a write operation is performed on associative memory, no address or memory location is given to the word. The memory itself is capable of finding an empty unused location to store the word.

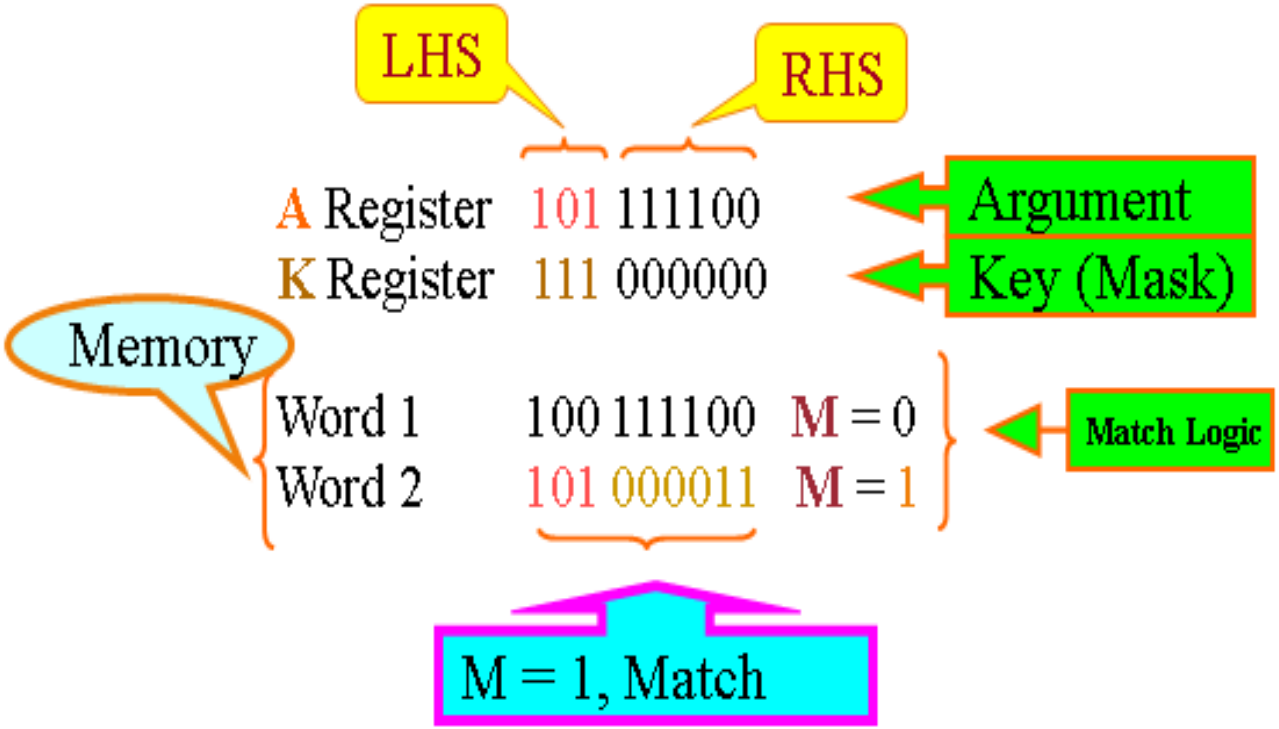
ASSOCIATIVE MEMORY



ASSOCIATIVE MEMORY

- Argument Register: It contains the word to be searched. It has n bits(one for each bit of the word).
- Key Register: This specifies which part of the argument word needs to be compared with words in memory. If all bits in the register are 1, the entire word will be matched.
- Associative Memory Array: It contains the words which are to be compared with the argument word.
- Match Register(M): It has m bits, one bit corresponding to each word in the memory array. After the matching process, the bits corresponding to matching words in match register are set to 1.

ASSOCIATIVE MEMORY



References

- "Operating System Concepts" by Avi Silberschatz and Peter Galvin
- "Operating Systems: Internals and Design Principles" by William Stallings
- www.tutorialpoint.com
- www.javapoint.com

Thank You

For any Query, you can contact

Er. Inderjeet Singh(e8822)

Ph. No: 86-99-100-160

Email id: inderjeet.e8822@cumail.in