2.5 VIRTUAL MEMORY

Virtual memory is a memory management technique used by operating systems that provides user an illusion(imaginary) of having a very big main memory. This is done by treating a part of secondary memory as the main memory.

A technique that allows a program to use more memory than in physically available by storing parts of the program on disk and swapping them in and out of RAM as needed.

Main memory has a limit of space and it can only process a limited number of processes at a time. But virtual memory gives extra space for the processes which are larger to be occupied in the main memory.

When space in main memory exceeds the limit then unused processes are moved from RAM to virtual memory. When the CPU needs those inactive processes then they are moved from virtual memory to RAM this movement of processes is known as a swap-in of process.

Logical addresses that are dynamically translated into physical addresses at run time. This means that a process can be swapped in and out of the main memory such that it occupies different places in the main memory at different times during the course of execution.

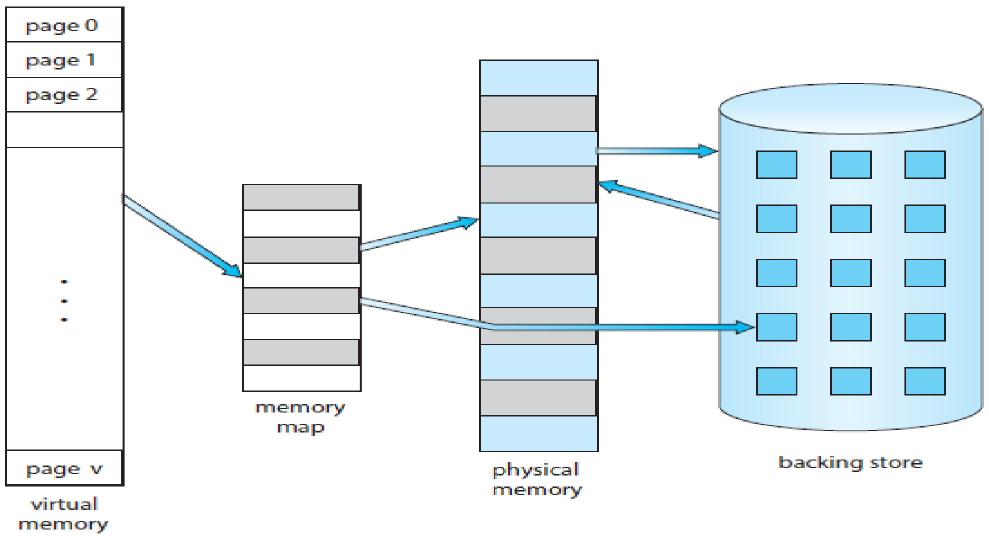


Figure Diagram showing virtual memory that is larger than physical memory.

- A **physical address** is the actual address in the main memory where data is stored. It is a location in physical memory.

- A **logical address**, also known as a virtual address, is an address generated by the CPU during program execution.
- The translation from logical to physical addresses is performed by the operating system's memory management unit (MMU).
- An **address space refers to the range of memory addresses available for storing and retrieving data**.
- ***All physical addresses that match the logical addresses in a logical address space*** are collectively referred to as the "**physical address space**".
- **Virtual address space** – logical view of how process is stored in memory
  - Usually start at address 0, contiguous addresses until end of space.
  - Meanwhile, physical memory organized in page frames
  - MMU must map logical to physical.

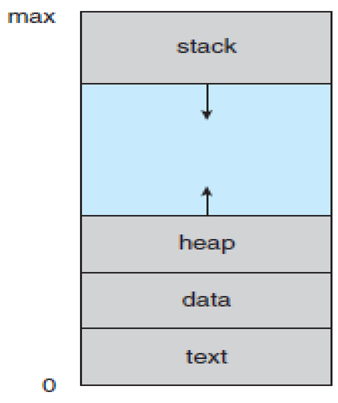


Figure Virtual address space of a process in memory.

- CPU execute process which resides in main memory.
- process/program - collect of pages
- pages refers to collect of instruction.

Benefits of virtual memory are: -

- By using virtual memory many applications or programs can be executed at a time.
- Main memory has limited space but you can increase and decrease the size of virtual memory by yourself.
- Users can run large programs that have a size greater than the main memory
- The data which is common in memory can be shared between RAM and virtual memory

- CPU utilization can be increased because more processes can reside in the main memory
- The cost of buying extra RAM is saved by using virtual memory

There are two methods for implementing virtual memory are as follows

1. Paging **2. Segmentation.**

1. PAGING

- Paging is a memory management technique where memory is partitioned into fixed-sized blocks that are commonly known as **pages**
- Paging eliminates most of the problems
  - Avoids external fragmentation.
  - Avoids problem of varying sized memory chunks.
- Implementation of paging involves
  - logical memory space is divided into blocks of the same size called ***pages***. (logical mem- program resides)
  - Physical memory into a number of equal sized blocks called ***frames*** (physical memory – main memory)
- Any page (from any process) can be placed into any available frame.
- page table is a look up table.
- Page table keep track of which page is stored in which memory location i.e, which page is stored in which frame.

2. Segmentation :-

Segmentation is also a memory management technique where memory is partitioned into variable-sized blocks that are commonly known as **Segments**.

- **Segments**: Programs are divided into logical units, like code, data, stack, and heap, that are stored in segments. Each segment has a unique identifier and is of variable size, matching the needs of the program.

- **Segment Table**: The operating system utilizes a segment table to keep track of information about each segment, including its base address, size (limit), and access rights.

- Logical vs. Physical Address: The CPU generates a logical address containing the segment number and an offset within that segment. This logical address is then translated into a physical address by the Memory Management Unit (MMU) using the segment table

2.1.1 DEMAND PAGING

Demand paging is a memory management technique where pages are loaded in main memory only when they are ***demanded*** during program execution.

- **Lazy swapper** – It is a concept that never swaps a page into memory unless page will be needed
  - Swapper that deals with pages is a **pager**

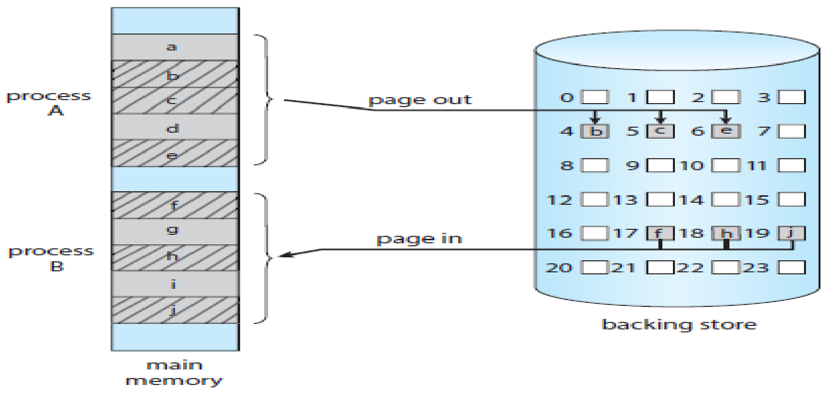


Figure Swapping with paging.

While a process is executing, some pages will be in memory, and some will be in secondary storage. Thus, we need some form of hardware support to distinguish between those pages that are in memory and those pages that are on the disk.

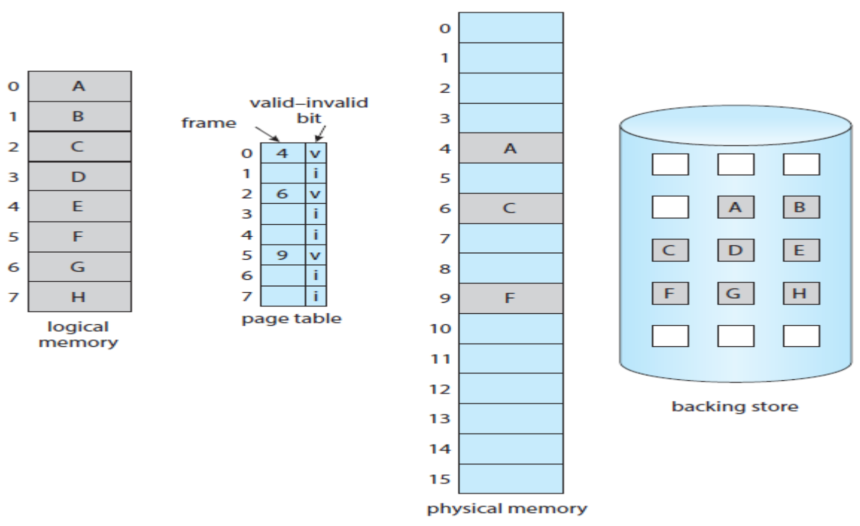


Figure Page table when some pages are not in main memory.

The valid/invalid bit scheme can be used for this purpose. Each entry in the page table has minimum two fields: page frame and valid-invalid bit.

- **If the bit is set to "valid," the page is legal and its present in memory. (vaild-1)**
- **If the bit is set to "invalid," the page is currently not in main memory. (invalid-0)**
    - Access to a page which is invalid causes a **page fault.**

➢ Page fault result causes a trap to the operating system, which indicate the failure and asked to bring the desired page into memory.

➢ When the page fault occur the execution is suspended until missing page is brought into memory.

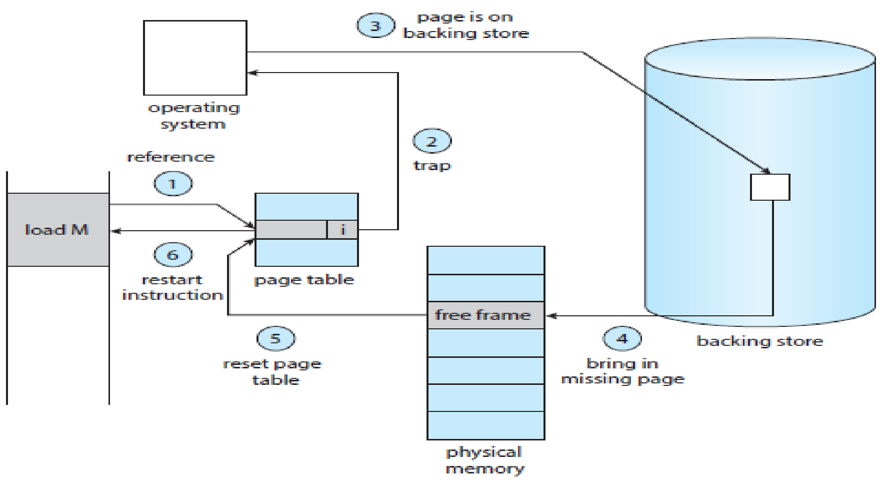


Figure Steps in handling a page fault.

The procedure(steps) for handling this page fault is straightforward

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 not yet brought in that page, now page it in.
3. Find a free frame.

4. We schedule a secondary storage operation to read the desired page into the newly allocated frame.

5. When the storage 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.

2.1.2 Pure Demand Paging:

Pure demand paging will never bring a page into memory until it is needed.

• **Page table**. Page is marked with valid–invalid bit.
• **Secondary memory**. This memory holds those pages that are not present in main memory. The secondary memory which is high-speed disk known as the swap device, used as a storage known as **swap space**

Swap space *is a space on a hard disk that is used as a substitute for physical memory.*

2. Free-Frame List

When a page fault occurs, the operating system must bring the desired page from secondary storage into main memory. To resolve page faults, most operating systems maintain a **free-frame list**, a pool of free frames for satisfying such requests. Operating systems typically allocate free frames using a technique known as **zero-fill-on-demand** . Zero-fill on-demand frames are "zeroed-out" before being allocated, thus erasing their previous contents.

2.1.3 Performance of Demand Paging

Demand paging can significantly affect the performance of a computer system. Computation of the **effective access time** for a demand-paged memory. p- page fault rate; Ma - memory-access time.

Let p be the probability of a page fault ($0 \le p \le 1$).
effective access time is then,

effective access time = (1 − *p*) × *ma* + *p* × page fault time.

2.1.4 COPY ON WRITE

copy-on-write, which works by allowing the parent and child processes initially to share the same pages.

Copy on Write or simply COW is a resource management technique. One of its main use is in the implementation of the fork system call in which it shares the virtual memory(pages) of the OS. In UNIX like OS, fork() system call creates a duplicate process of the parent process which is called as the child process.

The idea behind a copy-on-write is that when a parent process creates a child process then both of these processes initially will share the same pages in memory and these shared pages will be marked as copy-on-write which means that if any of these processes will try to modify the shared pages then only a copy of these pages will be created and the modifications will be done on the copy of pages by that process and thus not affecting the other process.

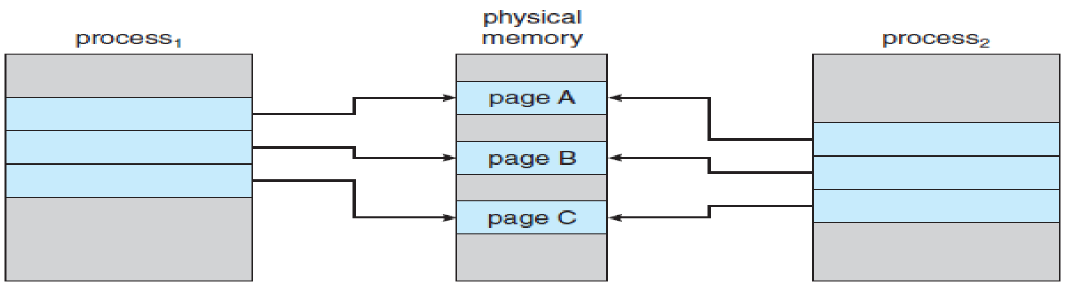


Figure Before process 1 modifies page C.

2.1.4 PAGE REPLACEMENT (PAGE FAULT)

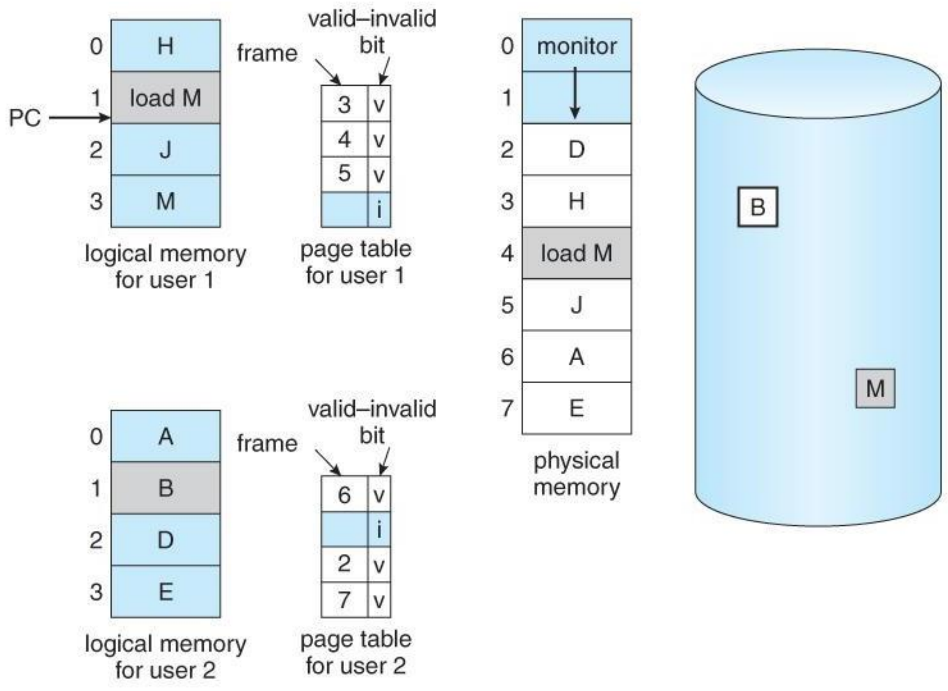
When a **page fault occurs**, page replacement algorithms are used to load the page in memory when there is no free page frames in memory.

A page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.

Find some page in memory that isn't being used right now, and swap that page only out to disk, freeing up a frame that can be allocated to the process requesting it. This is known as ***page replacement.***

- Prevent **over-allocation** of memory by modifying page-fault service routine to include page replacement
- Use **modify** (**dirty**) **bit** to reduce overhead of page transfers – only modified pages are written to disk

Fig: Need for Page Replacement



A page fault causes the following sequence to occur:

1. Trap to the operating system.
2. Save the registers and process state.
3. Determine that the interrupt was a page fault.
4. Check that the page reference was legal, and determine the location of the page in secondary storage.
5. Issue a read from the storage to a free frame:
    a. Wait in a queue until the read request is serviced.

b. Wait for the device seek and/or latency time.

c. Begin the transfer of the page to a free frame.

6. While waiting, allocate the CPU core to some other process.
7. Receive an interrupt from the storage I/O subsystem (I/O completed).
8. Save the registers and process state for the other process (if step 6 is executed).
9. Determine that the interrupt was from the secondary storage device.
10. Correct the page table and other tables to show that the desired page is now in memory.
11. Wait for the CPU core to be allocated to this process again.
12. Restore the registers, process state, and new page table, and then resume
the interrupted instruction.

Advantages

- Large virtual memory.
- More efficient use of memory.
- Unconstrained multiprogramming. There is no limit on degree of multiprogramming.

Disadvantages

- Number of tables and amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.
- Due to lack of an explicit constraint on a job's address space size

Basic Page Replacement

Working of pace 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 an existing frame to be replaced, known as the ***victim frame***.
   c. Write the victim frame to disk. Change all related page tables to indicate that this page is no longer in memory.
3. Bring the desired page and store it in the frame. Update the page and

frame tables to indicate the change.

4. Restart the process that was waiting for this page.

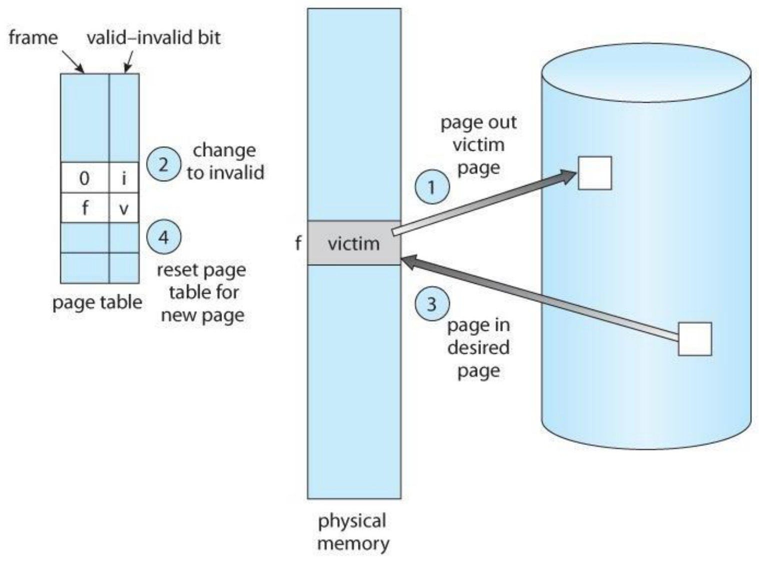


Figure - Page replacement

Assigning a **modify bit, or dirty** bit to each page, indicating whether or not it has been changed since it was last loaded in from disk. If the dirty bit has not been set, then the page is unchanged, and does not need to be written out to disk.

- 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**

Page replacement algorithm are

1. FIFO Page Replacement.
2. Optimal Page Replacement
3. LRU Page Replacement.
4. LRU-Approximation Page Replacement.

1. FIFO Page Replacement

- The simplest page-replacement algorithm is a first-in, first-out (FIFO) algorithm.
- Create a FIFO queue to hold all pages in memory.
- Keep track of all the pages in the queue.
- As new pages are requested and are swapped in, they are added to the tail of a queue and the page which is at the head becomes the victim
- A FIFO replacement algorithm replaces the oldest page that has been present in the main memory for the longest time.
- When a page must be replaced, the oldest page is chosen.

For our example reference string, our three frames are initially empty

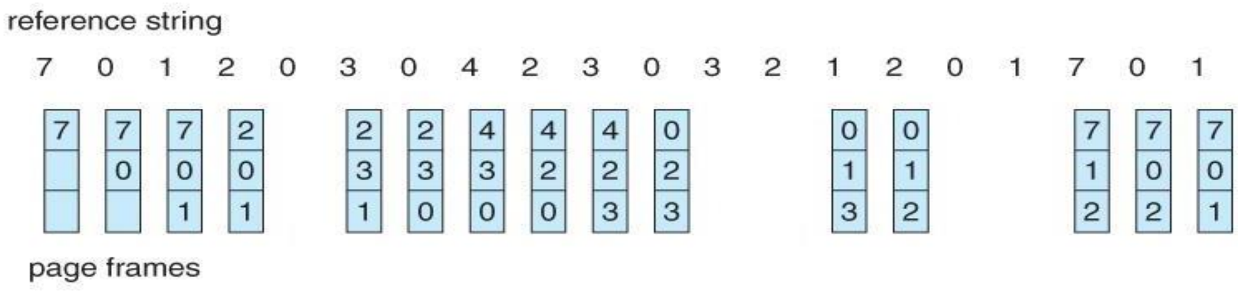


Figure: FIFO page-replacement algorithm.

Total number of Page fault Page fault- 15

Advantages

- This algorithm is simple and easy to use.
- FIFO does not cause more overhead.

Disadvantages

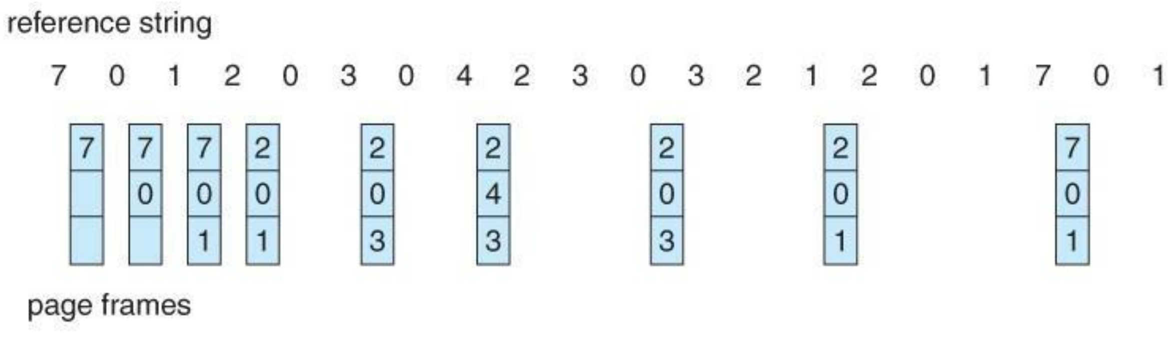
- Performance is not always good.
- ***Belady's anomaly-*** There is an increase in **page faults** as page frames increases.

2. *OPTIMAL PAGE-REPLACEMENT ALGORITHM*

- Belady's anomaly lead to the search for an ***optimal page-replacement algorithm.***
- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms (called OPT or MIN).

- Practical implementation is not possible because we cannot predict in advance those pages that will not be used for the longest time in the future.
- This algorithm leads to less number of page faults and thus is the best-known algorithm
- This algorithm can be used to measure the performance of other algorithms.

Example:-



Total number of Page fault- 9

Advantages of OPR

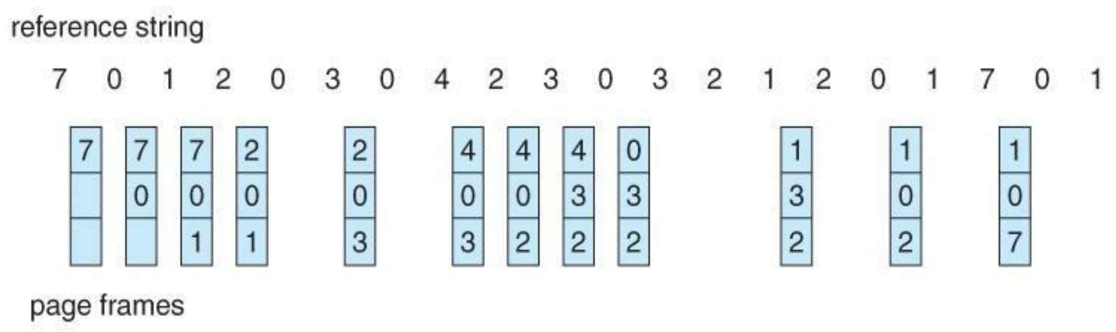
- This algorithm is easy to use.
- This algorithm provides excellent efficiency and is less complex.
- Never suffer from Belady's anomaly (page fault rate decreases)

Disadvantages of OPR

- In this algorithm future awareness of the program is needed.
- Practical Implementation is not possible because the operating system is unable to track the future request

3.LRU Page Replacement Algorithm

- Disadvantage of FIFO and OPT algorithm leads to the search of LRU Page Replacement Algorithm.
- LRU stands for "Least recent used" and this algorithm **replace the page that has not been used for the longest period of time.**
- This algorithm is easy to implement.
- This algorithm makes use of the counter along with the even-page.

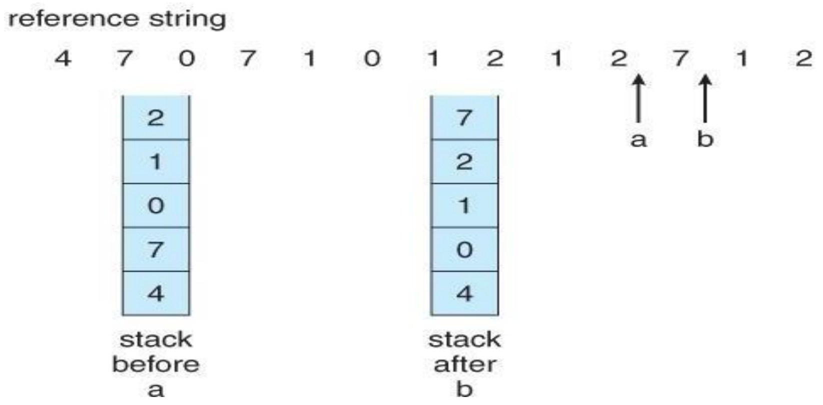


Total number of Page fault- 9

- Total number of Page fault is 12 faults which is much better than FIFO replacement with 15.

  Algorithm is implemented using:

  - **Counters.** Every memory access increments a counter, and the current value of this counter is stored in the page table entry for that page. Then finding the LRU page involves simple searching the table for the page with the smallest counter value. Note that overflowing of the counter must be considered.
  - **Stack.** Another approach is to use a stack, and whenever a page is accessed, pull that page from the middle of the stack and place it on the top. The LRU page will always be at the bottom of the stack. Because this requires removing objects from the middle of the stack, a doubly linked list is the recommended data structure.



Use of a stack to record the most recent page references

Advantages of LRU

- It is an efficient technique.

- With this algorithm, it becomes easy to identify the faulty pages that are not needed for a long time.
- It helps in Full analysis.
- Never suffer from Belady's anomaly

Disadvantages of LRU

- It is expensive and has more complexity.
- There is a need for an additional data structure.

4.LRU-Approximation Page Replacement

- Few computer systems provide sufficient hardware support in the form of a **reference bit** whenever that page is referenced.
- Initially, all bits are cleared (to 0) by the operating system.
- As a user process executes, the **reference bit** associated with each page is set to 1.
- Reference bit gives the details about which pages have been used and which have not been used.

4.1 Additional-Reference-Bits Algorithm

- These 8-bit shift registers contain the history of page use for the last eight time periods.
- At a regular interval (clock timer), 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.
- Example: If the shift register contains 00000000 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.

4.2 Second-Chance Algorithm

- Second chance algorithm is also called **clock algorithm**. It maintains reference bit per page.
- Goal: Remove a page that has not been referenced recently.
- FIFO replacement algorithm with a small modification.
- Implemented using circular queue.

- If the value is 0, replace this page; but if the reference bit is set to 1, the page a gets a second chance and move on to select the next FIFO page.
- When a page gets a second chance, its reference bit is cleared.
- Page with second chance will not be replaced until all other pages have been replaced.

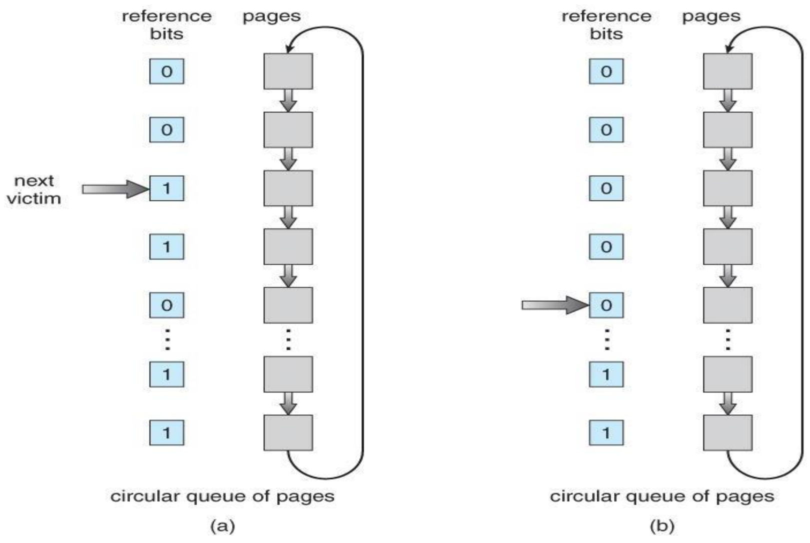


Figure: Second-chance (clock) page-replacement algorithm.

4.3 **Enhanced Second-Chance Algorithm**

- With these two bits, the four possible classes are:
  1. (0, 0) neither recently used nor modified—best page to replace
  2. (0, 1) not recently used but modified—not quite as good, because the page will need to be written out before replacement
  3. (1, 0) recently used but clean—probably will be used again soon
  4. (1, 1) recently used and modified—probably will be used again soon, and the page will be need to be written out to disk before it can be replaced

5. **Counting-Based Page Replacement**

Two schemes:

• The **least frequently used (LFU)** page-replacement algorithm requires that the page with the smallest count be replaced.

• 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.