MODULE 4

VIRTUAL MEMORYMANAGEMENT

- Virtual memory is a technique that allows for the execution of partially loaded process.
- Advantages:
  - A program will not be limited by the amount of physical memory that is available user can able to write in to large virtual space.
  - Since each program takes less amount of physical memory, more than one program could be run at the same time which can increase the throughput and CPU utilization.
  - Less i/o operation is needed to swap or load user program in to memory. So each user program could run faster.

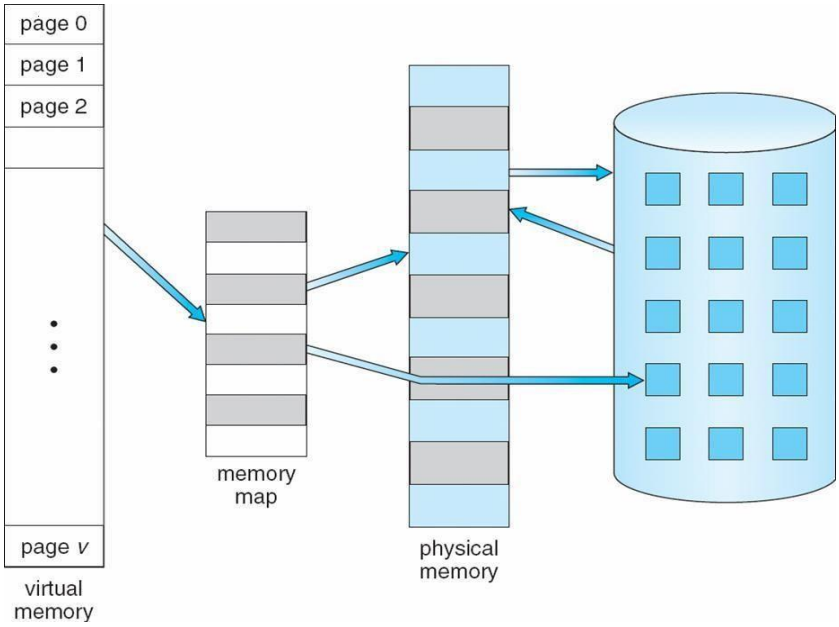


Fig: Virtual memory that is larger than physical memory.

- Virtual memory is the separation of users logical memory from physical memory. This separation allows an extremely large virtual memory to be provided when these is less physical memory.
- Separating logical memory from physical memory also allows files and memory to be shared by several different processes through page sharing.

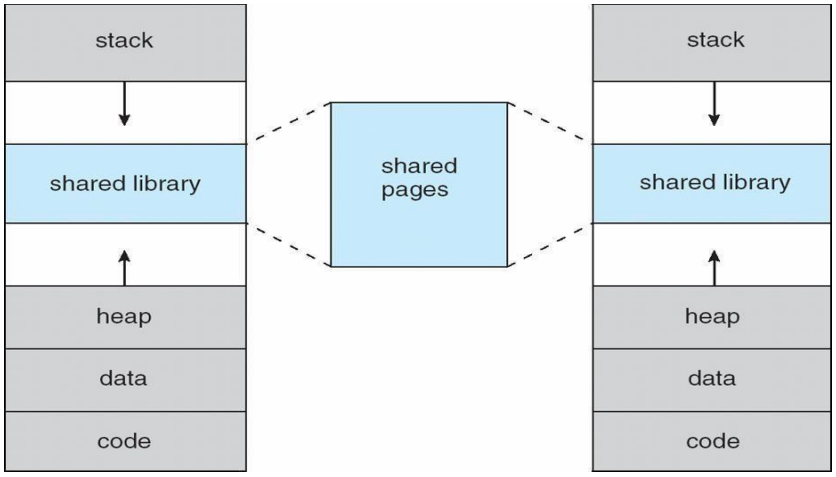


Fig: Shared Library using Virtual Memory

- Virtual memory is implemented using Demand Paging.
- Virtual address space: Every process has a virtual address space i.e used as the stack or heap grows in size.

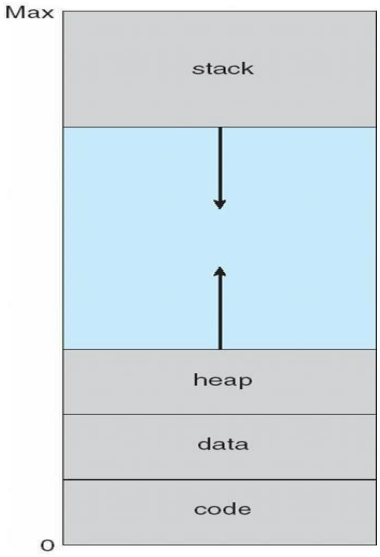


Fig: Virtual address space

DEMAND PAGING

- A demand paging is similar to paging system with swapping when we want to execute a process we swap the process the in to memory otherwise it will not be loaded in to memory.
- A swapper manipulates the entire processes, where as a pager manipulates individual pages of the process.
    - Bring a page into memory only when it is needed
    - Less I/O needed
    - Less memory needed
    - Faster response

- More users
- Page is needed ⇒ reference to it
- invalid reference ⇒abort
- not-in-memory ⇒ bring to memory
- ***Lazy swapper***– never swaps a page into memory unless page will be needed
- Swapper that deals with pages is a **pager.**

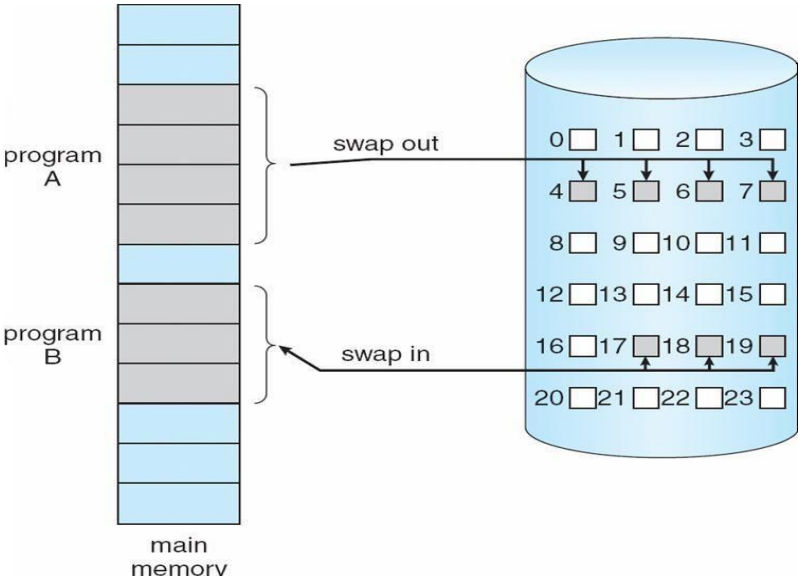


Fig: Transfer of a paged memory into continuous disk space

- **Basic concept:** Instead of swapping the whole process the pager swaps only the necessary pages in to memory. Thus it avoids reading unused pages and decreases the swap time and amount of physical memory needed.
- The valid-invalid bit scheme can be used to distinguish between the pages that are on the disk and that are in memory.
  - With each page table entry a valid–invalid bit is associated
  - (**v** ⇒ in-memory, **i**⇒not-in-memory)
  - Initially valid–invalid bit is set to **i**on all entries
  - Example of a page table snapshot:

Frame #	valid-invalid bit
	v
	v
	v
	v
	i
....	
	i
	i

page table

- During address translation, if valid–invalid bit in page table entry is **I** ⇒ page fault.
- If the bit is valid then the page is both legal and is in memory.
- If the bit is invalid then either page is not valid or is valid but is currently on the disk. Marking

a page as invalid will have no effect if the processes never access to that page. Suppose if it access the page which is marked invalid, causes a page fault trap. This may result in failure of OS to bring the desired page in to memory.

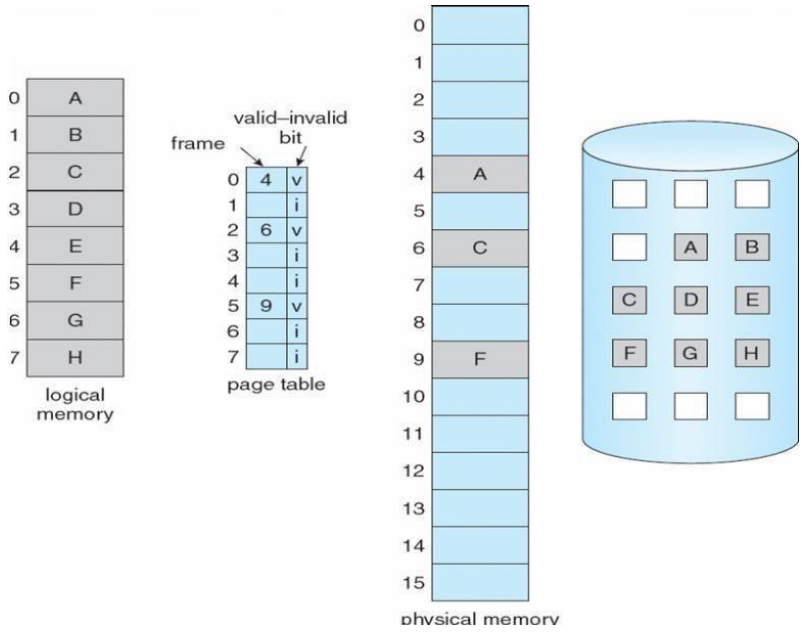


Fig: Page Table when some pages are not in main memory

Page Fault

If a page is needed that was not originally loaded up, then a ***page fault trap*** is generated.

Steps in Handling a Page Fault

1. The memory address requested is first checked, to make sure it was a valid memory request.
2. If the reference is to an invalid page, the process is terminated. Otherwise, if the page is not present in memory, it must be paged in.
3. A free frame is located, possibly from a free-frame list.
4. A disk operation is scheduled to bring in the necessary page from disk.
5. After the page is loaded to memory, the process's page table is updated with the new frame number, and the invalid bit is changed to indicate that this is now a valid page reference.
6. The instruction that caused the page fault must now be restarted from the beginning.

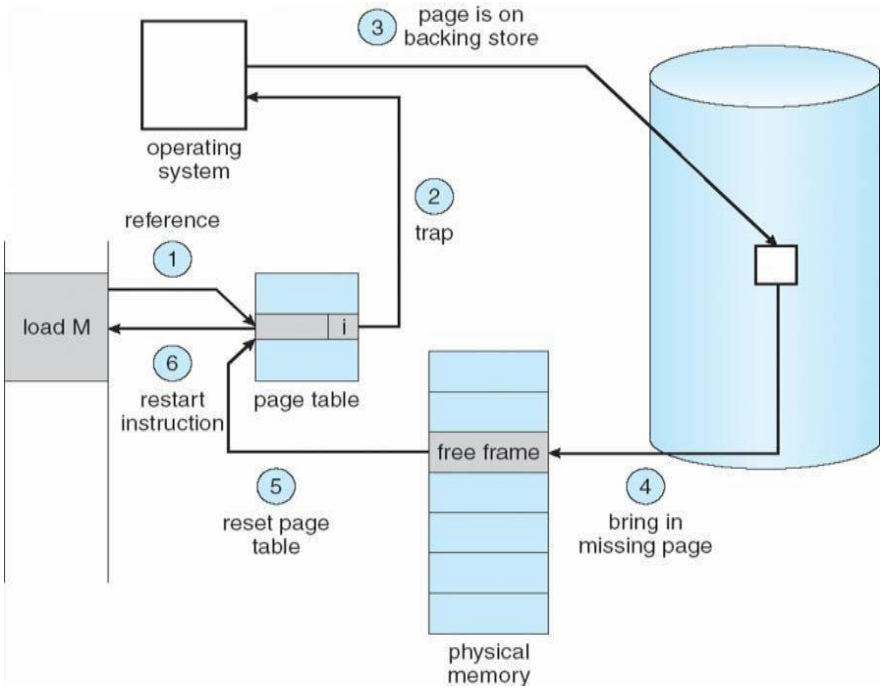


Fig: steps in handling page fault

Pure Demand Paging: Never bring a page into main memory until it is required.

- We can start executing a process without loading any of its pages into main memory.
- Page fault occurs for the non memory resident pages.
- After the page is brought into memory, process continues to execute.
- Again page fault occurs for the next page.

Hardware support: For demand paging the same hardware is required as paging and swapping.

1. Page table:-Has the ability to mark an entry invalid through valid-invalid bit.
2. Secondary memory:-This holds the pages that are not present in main memory.

Performance of Demand Paging: Demand paging can have significant effect on the performance of the computer system.

- Let P be the probability of the page fault (0<=P<=1)
- **Effective access time = (1-P) * ma + P * page fault.**
  - Where P = page fault and ma = memory access time.
- Effective access time is directly proportional to page fault rate. It is important to keep page fault rate low in demand paging.

Demand Paging Example

- Memory access time = 200 nanoseconds
- Average page-fault service time = 8milliseconds
- EAT = (1 – p) x 200 + p (8milliseconds)
  = (1 – p x 200 + p x 8,000,000
  = **200 + p x 7,999,800**
- If one access out of 1,000 causes a page fault, then EAT = 8.2 microseconds. This is a slowdown by a factor of 40.

PAGE REPLACEMENT

- Page replacement policy deals with the solution of pages in memory to be replaced by a new page that must be brought in. When a user process is executing a page fault occurs.
- The hardware traps to the operating system, which checks the internal table to see that this is a page fault and not an illegal memory access.
- The operating system determines where the derived page is residing on the disk, and this finds that there are no free frames on the list of free frames.
- When all the frames are in main memory, it is necessary to bring a new page to satisfy the page fault, replacement policy is concerned with selecting a page currently in memory to be replaced.
- The page i,e to be removed should be the page i,e least likely to be referenced in future.

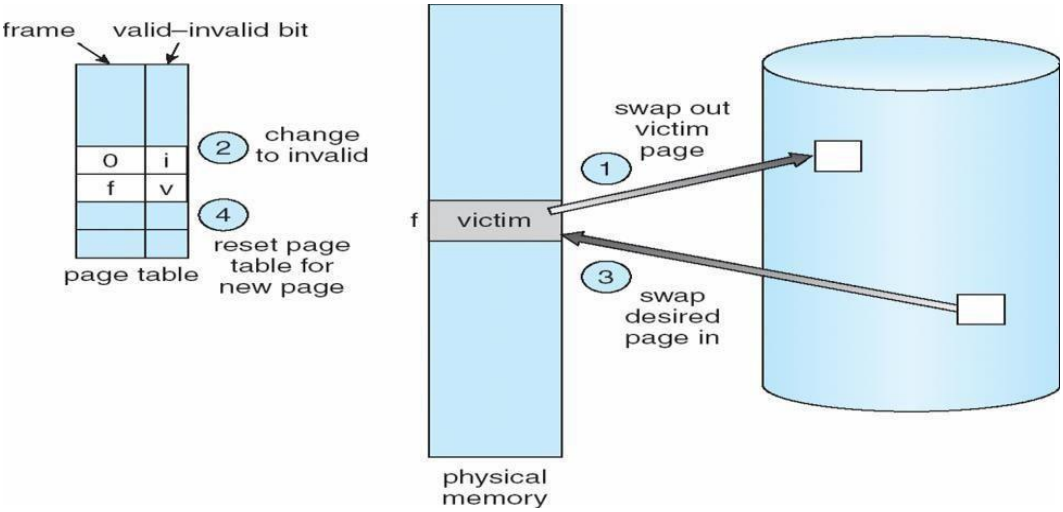


Fig: Page Replacement

Working of Page Replacement Algorithm

1. Find the location of derived page on the disk.
2. Find a free frame x If there is a free frame, use it. x Otherwise, use a replacement algorithm to select a victim.
   - Write the victim page to the disk.
   - Change the page and frame tables accordingly.
3. Read the desired page into the free frame; change the page and frame tables.
4. Restart the user process.

Victim Page

- The page that is supported out of physical memory is called victim page.
- If no frames are free, the two page transforms come (out and one in) are read. This will see the effective access time.
- Each page or frame may have a dirty (modify) bit associated with the hardware. The modify bit for a page is set by the hardware whenever any word or byte in the page i written into, indicating that the page has been modified.
- When we select the page for replacement, we check its modify bit. If the bit is set, then the page is modified since it was read from the disk.
- If the bit was not set, the page has not been modified since it was read into memory. Therefore, if the copy of the page has not been modified we can avoid writing the memory page to the disk, if it is already there. Sum pages cannot be modified.

Modify bit/ Dirty bit :

- Each page/frame has a modify bit associated with it.
- If the page is not modified (read-only) then one can discard such page without writing it onto the disk**.** Modify bit of such page is set to0.
- Modify bit is set to 1, if the page has been modified. Such pages must be written to the disk.
- Modify bit is used to reduce overhead of page transfers – only modified pages are written to disk

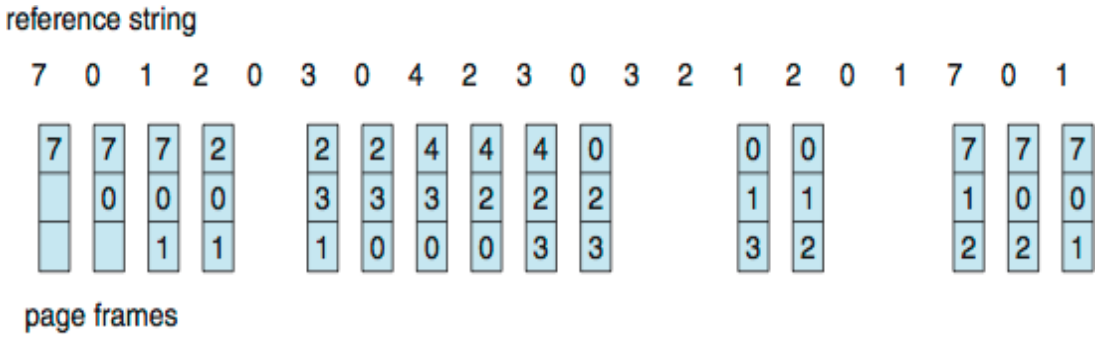
PAGE REPLACEMENT ALGORITHMS

- Want lowest page-fault rate
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
- In all our examples, the reference string is

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

FIFO Algorithm:

- This is the simplest page replacement algorithm. A FIFO replacement algorithm associates each page the time when that page was brought into memory.
- When a Page is to be replaced the oldest one is selected.
- We replace the queue at the head of the queue. When a page is brought into memory, we insert it at the tail of the queue.
- In the following example, a reference string is given and there are 3 free frames. There are 20 page requests, which results in 15 page faults



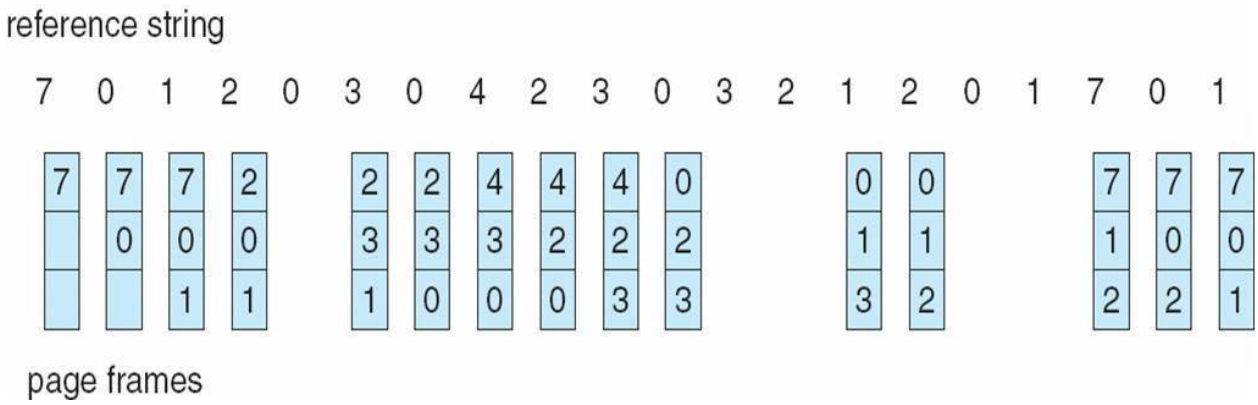
Belady's Anomaly

- For some page replacement algorithm, the page fault may increase as the number of allocated frames increases. FIFO replacement algorithm may face this problem.

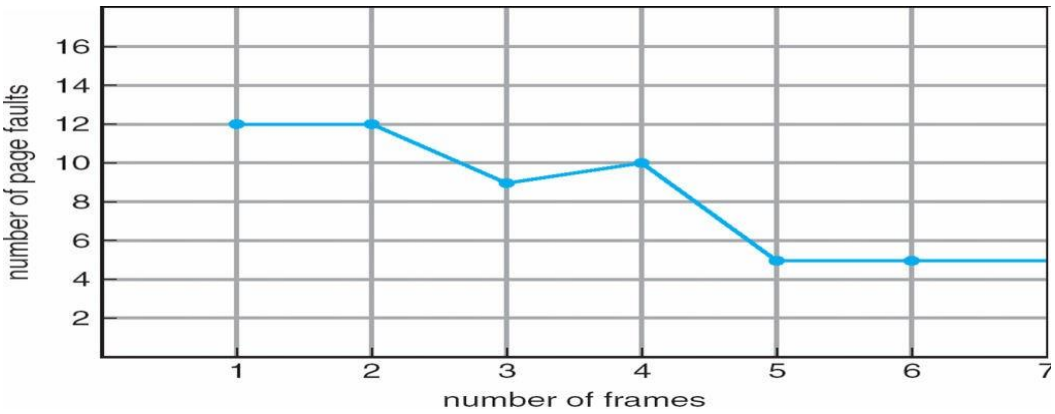
more frames ⇒ more page faults

Example: Consider the following references string with frames initially empty.

- The first three references (7,0,1) cases page faults and are brought into the empty frames.
- The next references 2 replaces page 7 because the page 7 was brought in first. x Since 0 is the next references and 0 is already in memory e has no page faults.
- The next references 3 results in page 0 being replaced so that the next references to 0 causer page fault. This will continue till the end of string. There are 15 faults all together.



FIFO Illustrating Belady's Anomaly



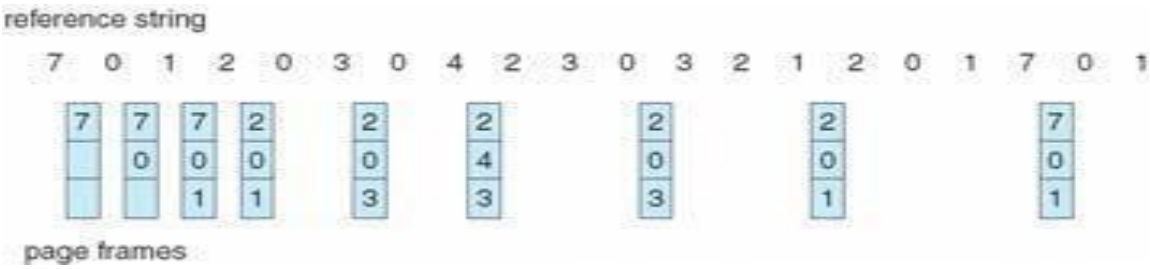
Optimal Algorithm

- Optimal page replacement algorithm is mainly to solve the problem of Belady's Anomaly.
- Optimal page replacement algorithm has the lowest page fault rate of all algorithms.
- An optimal page replacement algorithm exists and has been called OPT.

The working is simple "Replace the page that will not be used for the longest period of time"
Example: consider the following reference string

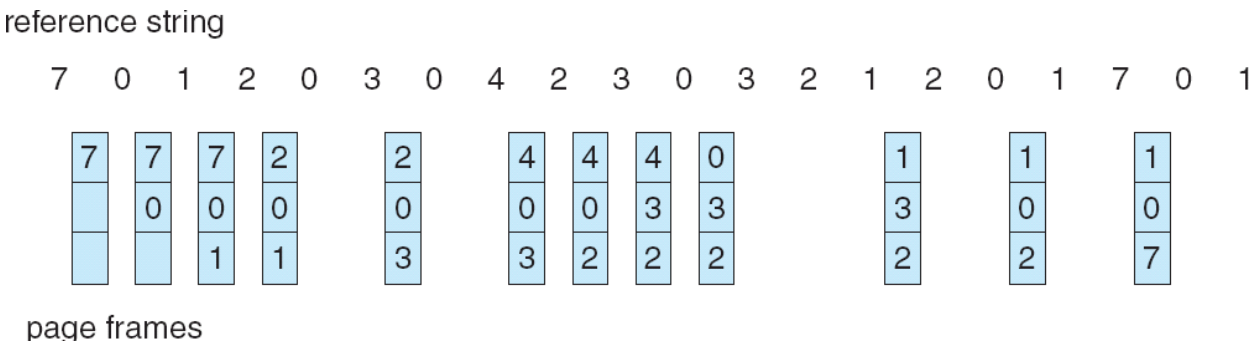
- The first three references cause faults that fill the three empty frames.
- The references to page 2 replaces page 7, because 7 will not be used until reference 18. x The page 0 will be used at 5 and page 1 at 14.
- With only 9 page faults, optimal replacement is much better than a FIFO, which had 15 faults. This algorithm is difficult t implement because it requires future knowledge of reference strings.
- Replace page that will not be used for longest period of time

Optimal Page Replacement



Least Recently Used (LRU) Algorithm

- The ***LRU (Least Recently Used)*** algorithm, predicts that the page that has not been used in the longest time is the one that will not be used again in the near future.
- Some view LRU as analogous to OPT, but here we look backwards in time instead of forwards.



The main problem to how to implement LRU is the LRU requires additional h/w assistance.

Two implementation are possible:

1. **Counters:** In this we associate each page table entry a time -of -use field, and add to the cpu a logical clock or counter. The clock is incremented for each memory reference. When 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 have the time of last reference to each page we replace the page with smallest time value. The time must also be maintained when page tables are changed.
2. **Stack:** Another approach to implement LRU replacement is to keep a stack of page numbers when a page is referenced it is removed from the stack and put on to the top of stack. In this way the top of stack is always the most recently used page and the bottom in least recently used page. Since the entries are removed from the stack it is best implement by a doubly linked list. With a head and tail pointer.

Note: Neither optimal replacement nor LRU replacement suffers from Belady's Anamoly. These are called stack algorithms.

LRU-Approximation Page Replacement

- Many systems offer some degree of hardware support, enough to approximate LRU.
- In particular, many systems provide a ***reference bit*** for every entry in a page table, which is set anytime that page is accessed. Initially all bits are set to zero, and they can also all be cleared at any time. One bit distinguishes pages that have been accessed since the last clear from those that have not been accessed.

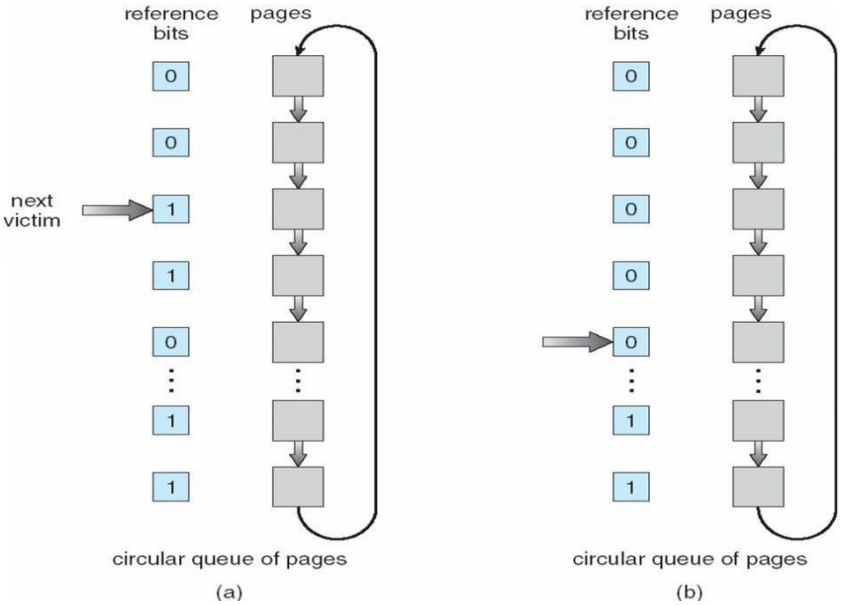
Additional-Reference-Bits Algorithm

- An 8-bit byte (reference bit) is stored 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, then the page has not been used for eight time periods.
- A page with a history register value of 11000100 has been used more recently than one with a value of 01110111.

Second- chance (clock) page replacement algorithm

- The ***second chance algorithm*** is a FIFO replacement algorithm, except the reference bit is used to give pages a second chance at staying in the page table.
- When a page must be replaced, the page table is scanned in a FIFO (circular queue) manner.
- If a page is found with its reference bit as '0', then that page is selected as the next victim.

- If the reference bitvalueis‘1’, then the page is given a second chance and its reference bit value is cleared (assigned as‘0’).
- Thus, a page that is given a second chance will not be replaced until all other pages have been replaced (or given second chances). In addition, if a page is used often, then it sets its reference bit again.
- This algorithm is also known as the ***clock*** algorithm.



Enhanced Second-Chance Algorithm

- The **enhanced second chance algorithm** looks at the reference bit and the modify bit (dirty bit) as an ordered page, and classifies pages into one of four classes:
    1. (0, 0) - Neither recently used nor modified.
    2. (0, 1) - Not recently used, but modified.
    3. (1, 0) - Recently used, but clean.
    4. (1, 1) - Recently used and modified.
- This algorithm searches the page table in a circular fashion, looking for the first page it can find in the lowest numbered category. i.e. it first makes a pass looking for a (0, 0), and then if it can't find one, it makes another pass looking for a(0,1),etc.
- The main difference between this algorithm and the previous one is the preference for replacing clean pages if possible.

Count Based Page Replacement

There is many other algorithms that can be used for page replacement, we can keep a counter of the number of references that has made to a page.

a) **LFU** (least frequently used):

This causes the page with the smallest count to be replaced. The reason for this selection is that actively used page should have a large reference count.

This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process but never used again. Since it was used heavily, it has a large count and remains in memory even though it is no longer needed.

b)**MFU** Algorithm:

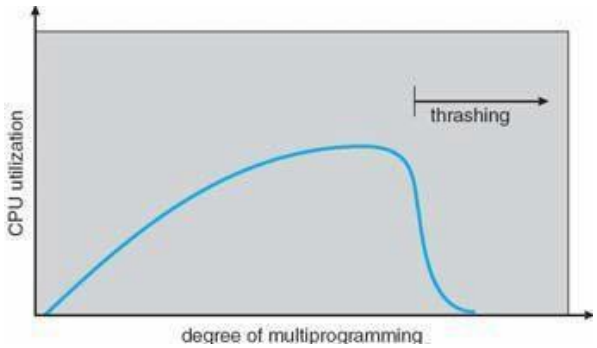
based on the argument that the page with the smallest count was probably just brought in and has yet to be used

THRASHING

- If the number of frames allocated to a low-priority process falls below the minimum number required by the computer architecture then we suspend the process execution.
- A process is thrashing if it is spending more time in paging than executing.
- If the processes do not have enough number of frames, it will quickly page fault. During this it must replace some page that is not currently in use. Consequently it quickly faults again and again.
- The process continues to fault, replacing pages for which it then faults and brings back. This high paging activity is called thrashing. The phenomenon of excessively moving pages back and forth b/w memory and secondary has been called ***thrashing***.

Cause of Thrashing

- Thrashing results in severe performance problem.
- The operating system monitors the cpu utilization is low. We increase the degree of multi programming by introducing new process to the system.
- A global page replacement algorithm replaces pages with no regards to the process to which they belong.



The figure shows the thrashing

- As the degree of multi programming increases, more slowly until a maximum is

reached. If the degree of multi programming is increased further thrashing sets in and the cpu utilization drops sharply.

- At this point, to increases CPU utilization and stop thrashing, we must increase degree of multiprogramming.
- we can limit the effect of thrashing by using a local replacement algorithm. To prevent thrashing, we must provide a process as many frames as it needs.

Locality of Reference:

- As the process executes it moves from locality to locality.
- A locality is a set of pages that are actively used.
- A program may consist of several different localities, which may overlap.
- Locality is caused by loops in code that find to reference arrays and other data structures by indices.
- The ordered list of page number accessed by a program is called reference string.
- Locality is of two types :
  1. spatial locality 2. temporal locality

Working set model

- Working set model algorithm uses the current memory requirements to determine the number of page frames to allocate to the process, an informal definition is "the collection of pages that a process is working with and which must be resident if the process to avoid thrashing". The idea is to use the recent needs of a process to predict its future reader.
- The working set is an approximation of programs locality. Ex: given a sequence of memory reference, if the working set window size to memory references, then working set at time t1 is{1,2,5,6,7} and at t2 is changed to {3,4}
- At any given time, all pages referenced by a process in its last 4 seconds of execution are considered to compromise its working set.
- A process will never execute until its working set is resident in main memory.
- Pages outside the working set can be discarded at any movement.
- Working sets are not enough and we must also introduce balance set.
  - If the sum of the working sets of all the run able process is greater than the size of memory the refuse some process for a while.
  - Divide the run able process into two groups, active and inactive. The collection of active set is called the balance set. When a process is made active its working set is loaded.
  - Some algorithm must be provided for moving process into and out of the balance set. As a working set is changed, corresponding change is made to the balance set.
  - Working set presents thrashing by keeping the degree of multi programming as high as possible. Thus if optimizes the CPU utilization. The main disadvantage of this is keeping track of the working set.

Page-Fault Frequency

- When page- fault rate is too high, the process needs more frames and when it is too low, the process may have too many frames.
- The upper and lower bounds can be established on the page-fault rate. If the actual page- fault rate exceeds the upper limit, allocate the process another frame or suspend the process.
- If the page-fault rate falls below the lower limit, remove a frame from the process. Thus, we can directly measure and control the page-fault rate to prevent thrashing.