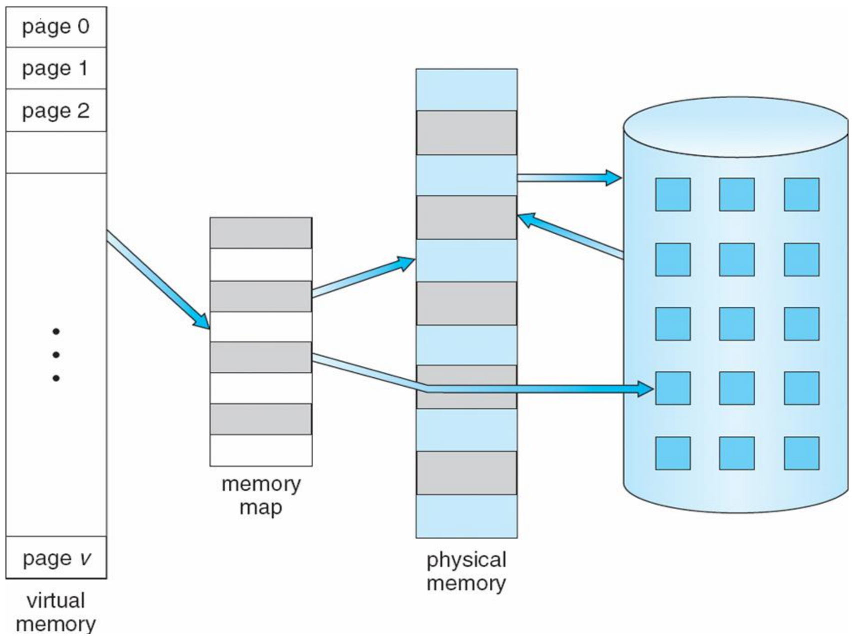
5.7 VIRTUAL MEMORY MANAGEMENT

Virtual memory is a technique that allows for the execution of partially loaded process. There are many advantages of this:

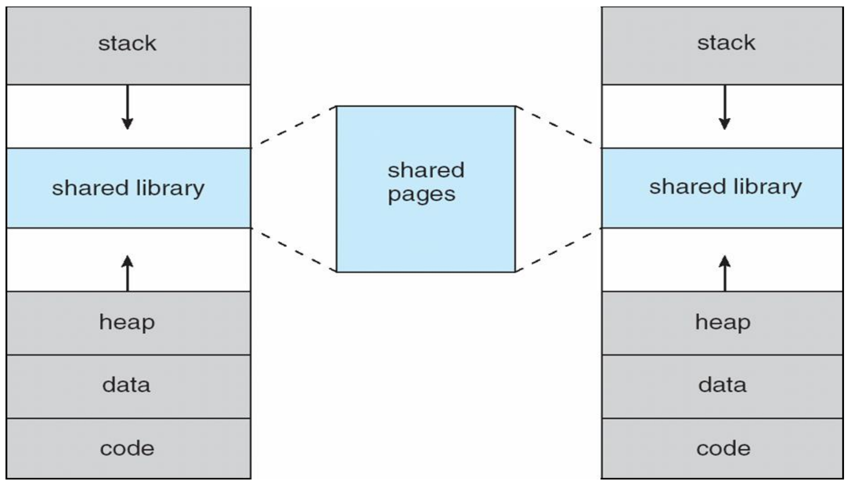
x A program will not be limited by the amount of physical memory that is available user can able to write in to large virtual space. x Since each program takes less amount of physical memory, more than one program could be run at the same time which can increase the throughput and CPU utilization.



*Less i/o operation is needed to swap or load user program in to memory. So each user program could run faster.

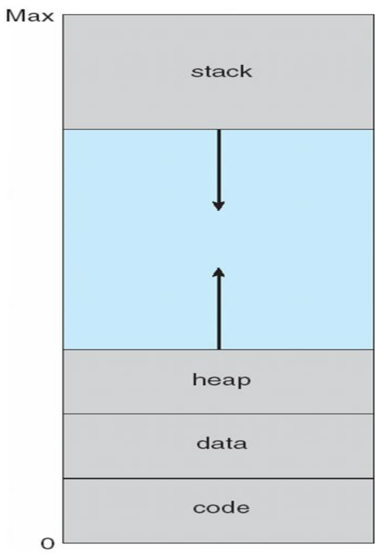
• Virtual memory is the separation of users logical memory from physical memory. This separation allows an extremely large virtual memory to be provided when these is less physical memory.

• Separating logical memory from physical memory also allows files and memory to be shared by several different processes through page sharing.



• Virtual memory is implemented using Demand Paging.

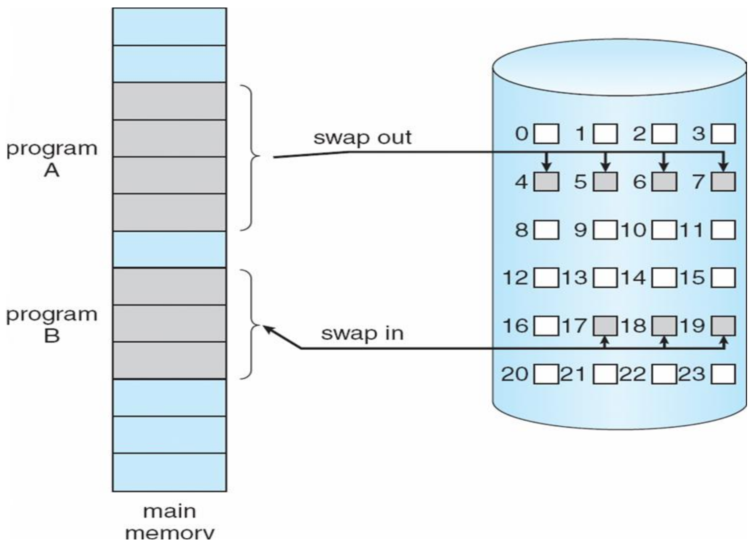
Virtual address space: Every process has a virtual address space i.e used as the stack or heap grows in size.



5.8 DEMAND PAGING

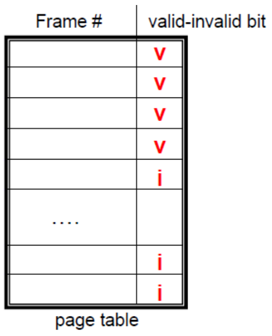
• A demand paging is similar to paging system with swapping when we want to execute a process we swap the process the in to memory otherwise it will not be loaded in to memory.
• A swapper manipulates the entire processes, where as a pager manipulates individual pages of the process.

- 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.**



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:

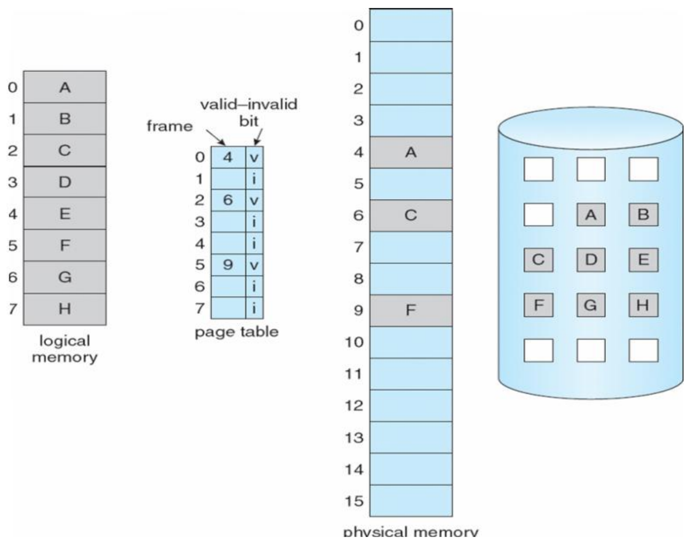


Frame #	valid-invalid bit
	v
	v
	v
	v
	i
….	
	i
	i

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.



Page Fault

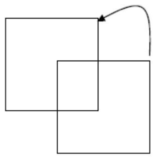
- If there is a reference to a page, first reference to that page will trap to operating system:

page fault

1. Operating system looks at another table to decide:
   - Invalid reference ⇒ abort
   - Just not in memory
2. Get empty frame
3. Swap page into frame

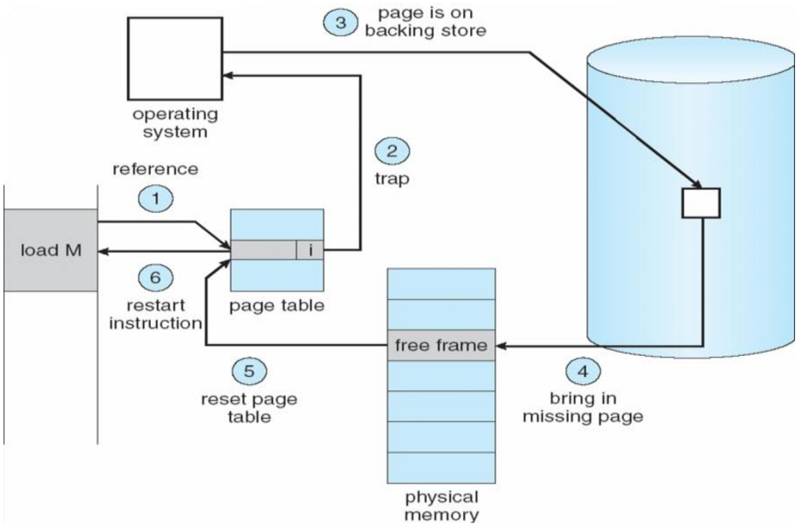
4. Reset tables (page, frame and other tables)
5. Set validation bit = **v**
6. Restart the instruction that caused the page fault

- Restart instruction
- block move



- auto increment/decrement location

Steps in Handling a Page Fault



The step for handling page fault is straight forward and is given below:
1. We check the internal table of the process to determine whether the reference made is valid or invalid.
2. If invalid terminate the process, If valid, then the page is not yet loaded and we now page it in.
3. We find a free frame.
4. We schedule disk operation to read the desired page in to newly allocated frame.
5. When disk reed is complete, we modify the internal table kept with the process to indicate that the page is now in memory.
6. We restart the instruction which was interrupted by illegal address trap. The process can now access the page. In extreme cases, we start the process without pages in memory. When the OS points to the instruction of process it generates a page fault. After this page is brought in to memory the process continues to execute, faulting as necessary until every demand paging i.e., it never brings the page in to memory until it is required.

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. I
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 = 8 milliseconds
- EAT = (1 – p) x 200 + p (8 milliseconds)
  = (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!!

A page fault causes the following sequence to occur:
1. Trap to the OS.
2. Save the user registers and process state.
3. Determine that the interrupt was a page fault.
4. Checks the page references were legal and determine the location of page on disk.
5. Issue a read from disk to a free frame.
6. If waiting, allocate the CPU to some other user.
7. Interrupt from the disk.
8. Save the registers and process states of other users.
9. Determine that the interrupt was from the disk.
10. Correct the page table and other table to show that the desired page is now in memory.
11. Wait for the CPU to be allocated to this process again.
12. Restore the user register process state and new page table then resume the interrupted instruction.

The step for handling page fault is straight forward and is given below:
1. We check the internal table of the process to determine whether the reference made is valid or invalid.
2. If invalid terminate the process,. If valid, then the page is not yet loaded and we now page it in.
3. We find a free frame.
4. We schedule disk operation to read the desired page in to newly allocated frame.
5. When disk reed is complete, we modify the internal table kept with the process to indicate that the page is now in memory.
6. We restart the instruction which was interrupted by illegal address trap. The process can

Demand paging is used when reading a file from disk in to memory. Fork () is used to create a process and it initially bypass the demand paging using a technique called page sharing. Page sharing provides rapid speed for process creation and reduces the number of pages allocated to the newly created process.

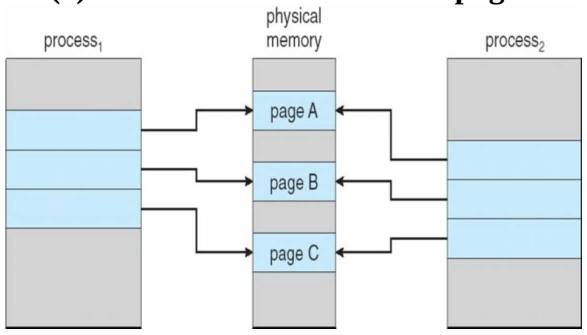
Copy-on-write

Technique initially allows the parent and the child to share the same pages. These pages are marked as copy on- write pages i.e., if either process writes to a shared page, a copy of shared page is created.

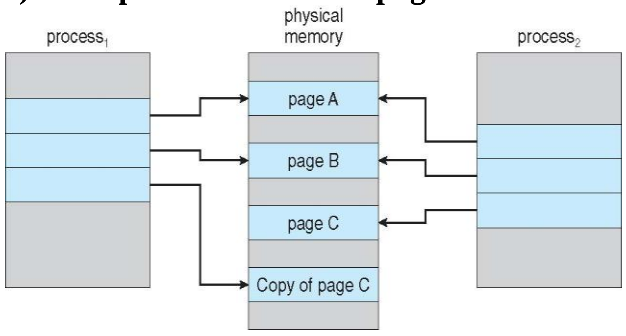
Eg:-If a child process try to modify a page containing portions of the stack; the OS recognizes them as a copy-on-write page and create a copy of this page and maps it on to the address space of the child process. So the child process will modify its copied page and not the page belonging to parent. The new pages are obtained from the pool of free pages.

The previous contents of pages are erased before getting them into main memory. This is called **Zero – on fill demand.**

(a) Before Process 1 modifies page C



b) After process 1 modifies page C



Comparison of demand paging with segmentation:-

Segmentation:

. o Segment may of different size.
. o Segment can be shared.
. o Allows for dynamic growth of segments.
. o Segment map table indicate the address of each segment in memory.
.o Segments are allocated to the program while compilation.

Demand Paging:

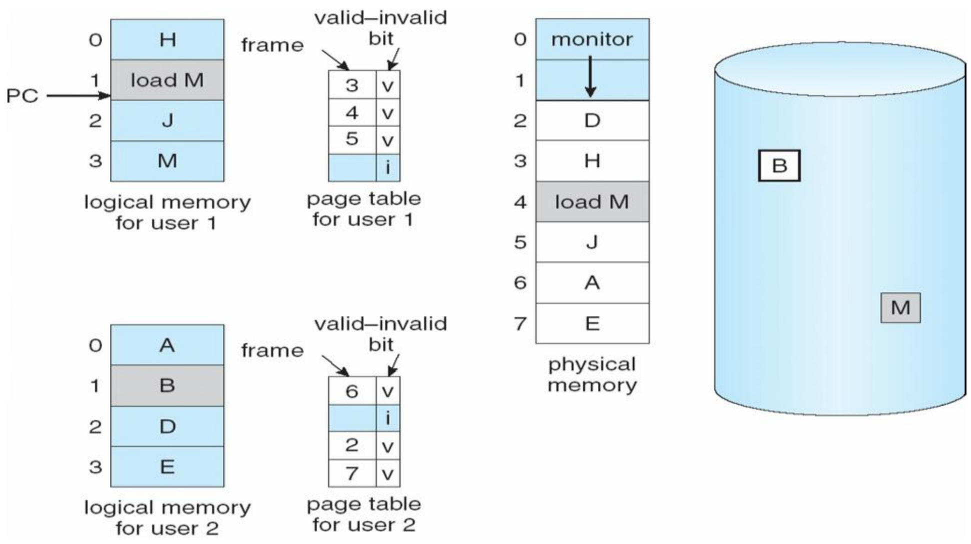
. o Pages are of same size.
. o Pages can’t be shared.
. o Page size is fixed.
. o Page table keeps track of pages in memory.
. o Pages are allocated in memory on demand.

5.10 PAGE REPLACEMENT

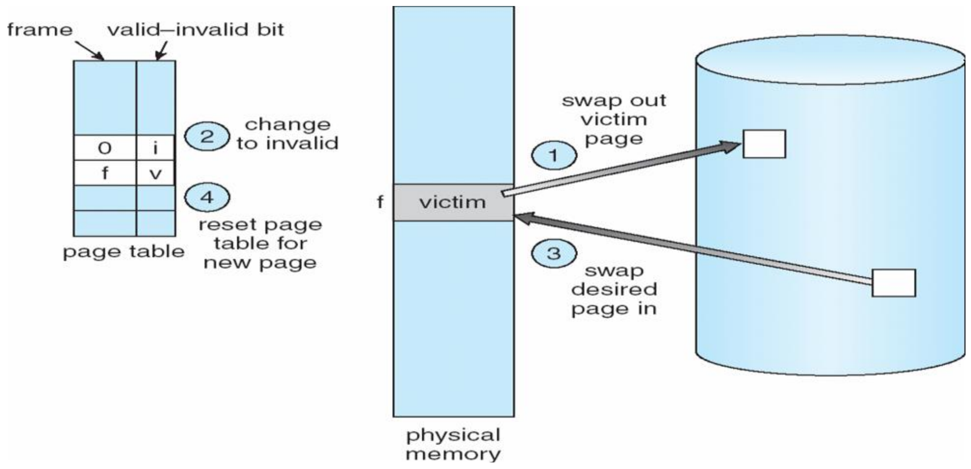
• Demand paging shares the I/O by not loading the pages that are never used.

• Demand paging also improves the degree of multiprogramming by allowing more process to run at the some time.

• 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.



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.
  x 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. x 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 is 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.
We must solve two major problems to implement demand paging: we must develop a frame allocation algorithm and a page replacement algorithm. If we have multiple processors in memory, we must decide how many frames to allocate and page replacement is needed.



Modify bit/ Dirty bit :

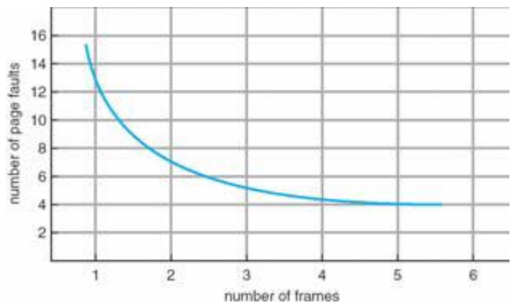
- Each page/frame has a modify bit associated with it.
- If the page is not modified(read-only) then one can discard such page without writing it onto the disk**.** Modify bit of such page is set to 0.
- Modify bit is set to 1, if the page has been modified. Such pages must be written to the disk.
- 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

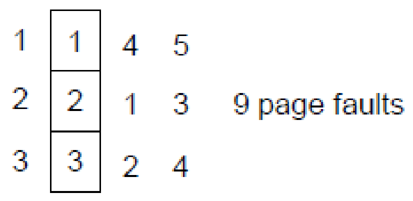
Graph of Page Faults Versus The Number of Frames



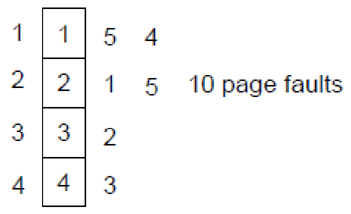
FIFO Algorithm:
x This is the simplest page replacement algorithm. A FIFO replacement algorithm associates each page the time when that page was brought into memory.
x When a Page is to be replaced the oldest one is selected.
x We replace the queue at the head of the queue. When a page is brought into memory, we insert it at the tail of the queue.

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

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



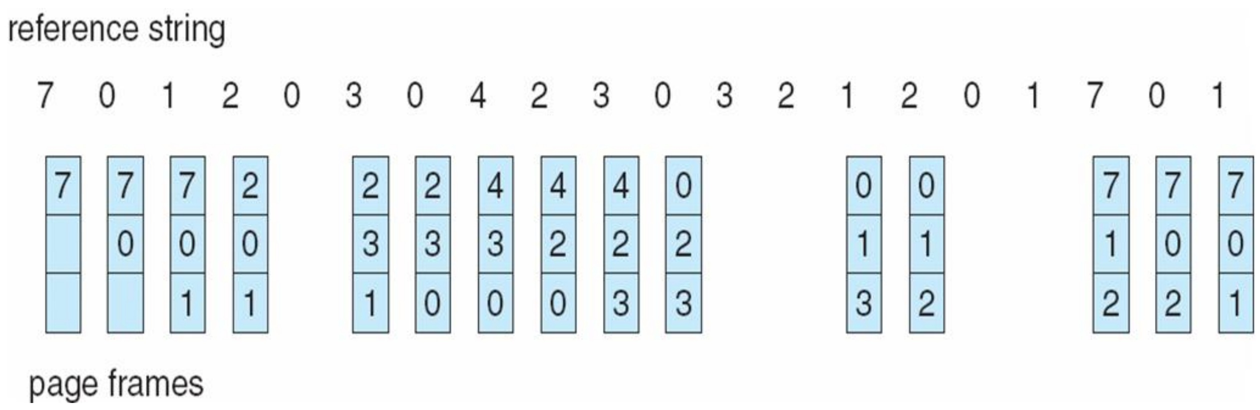
- 4 frames



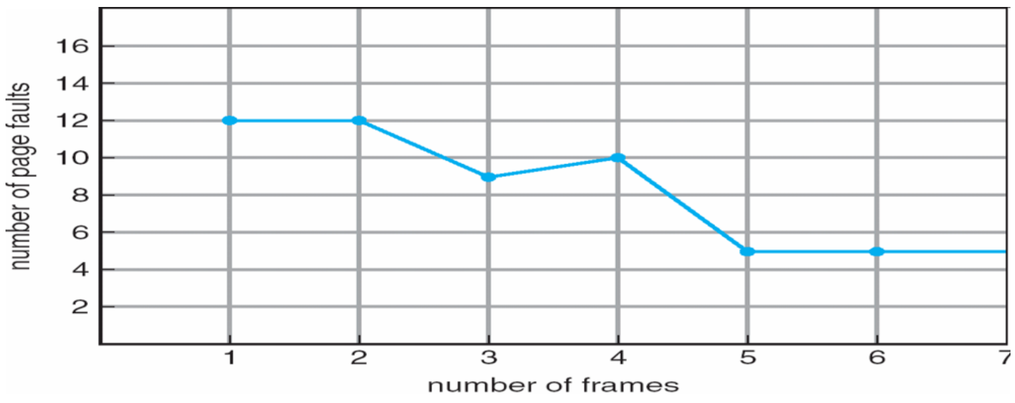
- **Belady's Anamoly**
- For some page replacement algorithm, the page fault may increase as the number of allocated frames increases. FIFO replacement algorithm may face this problem.

  more frames ⇒ more page faults

Example: Consider the following references string with frames initially empty.
x The first three references (7,0,1) cases page faults and are brought into the empty frames.
x 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.
x 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

x Optimal page replacement algorithm is mainly to solve the problem of Belady's Anamoly.
x Optimal page replacement algorithm has the lowest page fault rate of all algorithms.
x 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
x The first three references cause faults that fill the three empty frames.
x 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.
x 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.

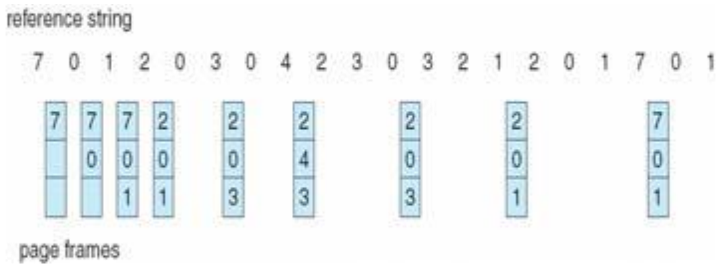
- Replace page that will not be used for longest period of time
- 4 frames example

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

1	4
2	
3	
4	5

6 page faults

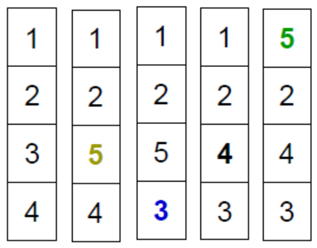
Optimal Page Replacement



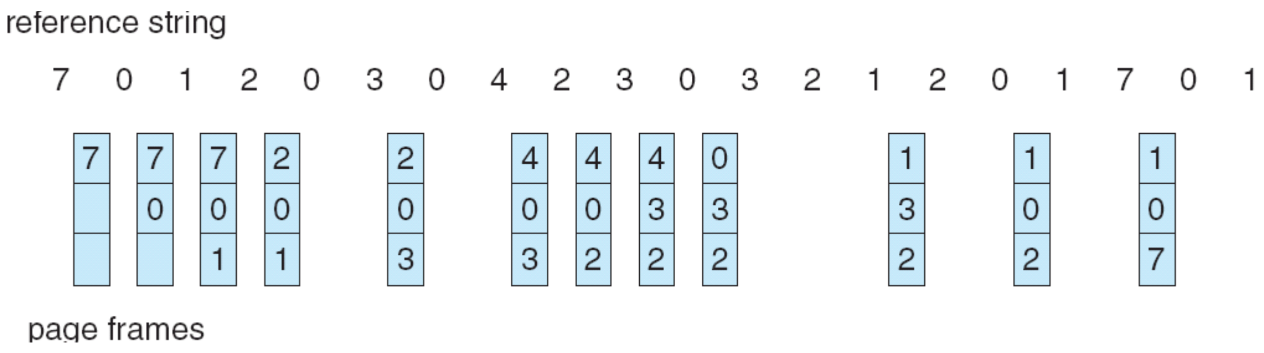
Page faults : 9

Least Recently Used (LRU) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**

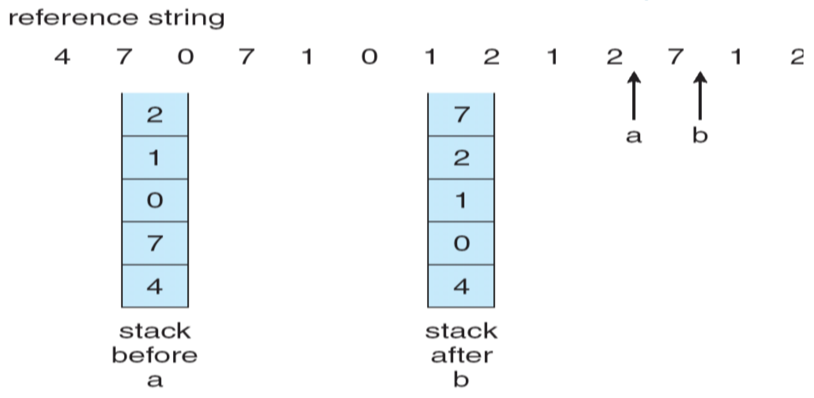


- Counter implementation
- Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
- When a page needs to be changed, look at the counters to determine which are to change



- 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
- No search for replacement

Use Of A Stack to Record The Most Recent Page References



If the optimal algorithm is not feasible, an approximation to the optimal algorithm is possible. The main difference b/w OPTS and FIFO is that;
• FIFO algorithm uses the time when the pages was built in and OPT uses the time when a page is to be used.
• The LRU algorithm replaces the pages that have not been used for longest period of time. x The LRU associated its pages with the time of that pages last use.
x This strategy is the optimal page replacement algorithm looking backward in time rather than forward. Ex: consider the following reference string

x The first 5 faults are similar to optimal replacement.
xWhen reference to page 4 occurs, LRU sees that of the three frames, page 2 as used least recently. The most recently used page is page 0 and just before page 3 was used. The LRU policy is often used as a page replacement algorithm and considered to be good.

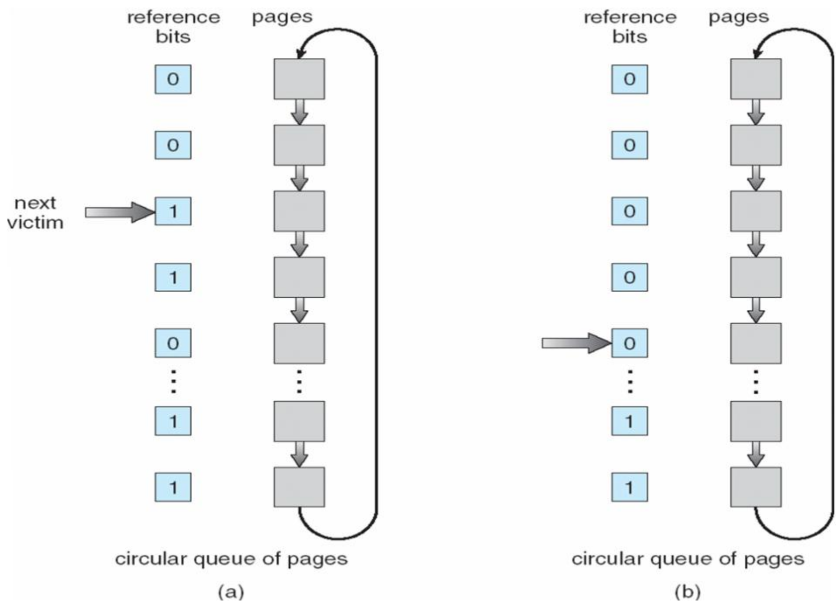
The main problem to how to implement LRU is the LRU requires additional h/w assistance.

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

LRU Approximation
• An LRU page replacement algorithm should update the page removal status information after every page reference updating is done by software, cost increases.
• But hardware LRU mechanism tend to degrade execution performance at the same time, then substantially increases the cost. For this reason, simple and efficient algorithm that approximation the LRU have been developed. With h/w support the reference bit was used. A reference bit associate with each memory block and this bit automatically set to 1 by the h/w whenever the page is referenced. The single reference bit per clock can be used to approximate LRU removal.
• The page removal s/w periodically resets the reference bit to 0, write the execution of the users job causes some reference bit to be set to 1.
• If the reference bit is 0 then the page has not been referenced since the last time the reference bit was set to 0.

- Reference bit
- With each page associate a reference bit, initially = 0
- When page is referenced bit set to 1
- Replace the one which is 0 (if one exists)
    - We do not know the order, however
- Second chance
- Need reference bit
- Clock replacement
- If page to be replaced (in clock order) has reference bit = 1 then:
    - set reference bit 0
    - leave page in memory
    - replace next page (in clock order), subject to same rules

Second- chance(clock) page replacement algorithm



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

5.11 ALLOCATION OF FRAMES

• The allocation policy in a virtual memory controls the operating system decision regarding the amount of real memory to be allocated to each active process.

• In a paging system if more real pages are allocated, it reduces the page fault frequency and improved turnaround throughput.

• If too few pages are allocated to a process its page fault frequency and turnaround times may deteriorate to unacceptable levels.

• The minimum number of frames per process is defined by the architecture, and the maximum number of frames. This scheme is called equal allocation.

• With multiple processes competing for frames, we can classify page replacement into two broad categories

a) Local Replacement: requires that each process selects frames from only its own sets of allocated frame.

b). Global Replacement: allows a process to select frame from the set of all frames. Even if the frame is currently allocated to some other process, one process can take a frame from another.

In local replacement the number of frames allocated to a process do not change but with global replacement number of frames allocated to a process do not change global replacement results in greater system throughput.

Other consideration

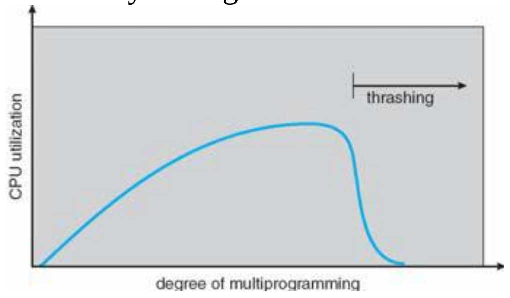
There is much other consideration for the selection of a replacement algorithm and allocation policy.
1) Preparing: This is an attempt to present high level of initial paging. This strategy is to bring into memory all the pages at one time.
2) TLB Reach: The TLB reach refers to the amount of memory accessible from the
TLB and is simply the no of entries multiplied by page size.
3) Page Size: following parameters are considered
a) page size us always power of 2 (from 512 to 16k)
b) Internal fragmentation is reduced by a small page size. c) A large page size reduces the number of pages needed.
4) Invented Page table: This will reduces the amount of primary memory i,e. needed to track virtual to physical address translations.
5) Program Structure: Careful selection of data structure can increases the locality and hence lowers the page fault rate and the number of pages in working state.
6) Real time Processing: Real time system almost never has virtual memory. Virtual memory is the antithesis of real time computing, because it can introduce unexpected long term delay in the execution of a process.

5.12 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

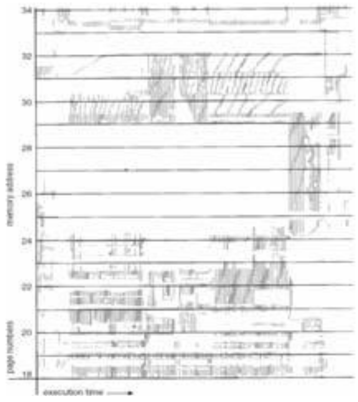
x Thrashing results in severe performance problem.
xThe operating system monitors the cpu utilization is low. We increase the degree of multi programming by introducing new process to the system.
x A global page replacement algorithm replaces pages with no regards to the process to which they belong.



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 multi

programming. 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:
x As the process executes it moves from locality to locality.
x A locality is a set of pages that are actively used.
x A program may consist of several different localities, which may
overlap.
x 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}
x At any given time, all pages referenced by a process in its last 4 seconds of execution are considered to compromise its working set.
x A process will never execute until its working set is resident in main memory.
x Pages outside the working set can be discarded at any movement.
Working sets are not enough and we must also introduce balance set.
a) 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.
b) 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.
c) 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.

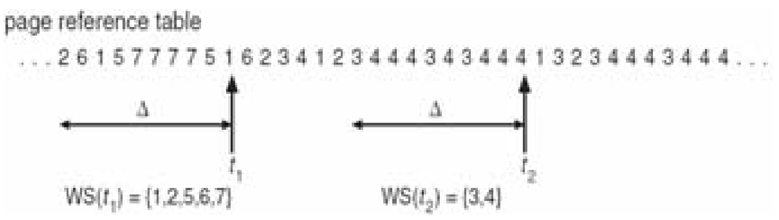


Working-Set Model

- Δ ≡ working-set window ≡ a fixed number of page references

Example: 10,000 instruction

- *WSSi* (working set of Process *Pi*) = total number of pages referenced in the most recent Δ (varies in time)
- if Δ too small will not encompass entire locality
- if Δ too large will encompass several localities
- if Δ = ∞ ⇒ will encompass entire program
- *D* = Σ *WSSi* ≡ total demand frames
- if *D* > *m* ⇒ Thrashing
- Policy if *D* > m, then suspend one of the processes



Keeping Track of the Working Set

- Approximate with interval timer + a reference bit
- Example: Δ = 10,000
- Timer interrupts after every 5000 time units
- Keep in memory 2 bits for each page
- Whenever a timer interrupts copy and sets the values of all reference bits to 0
- If one of the bits in memory = 1 ⇒ page in working set
- Why is this not completely accurate?
- Improvement = 10 bits and interrupt every 1000 time units

Page-Fault Frequency Scheme

- Establish "acceptable" page-fault rate
- If actual rate too low, process loses frame
- If actual rate too high, process gains frame

QUESTIONS:
1. Name two differences between logical and physical addresses.
2. Explain the difference between internal and external fragmentation.
3. Describe the following allocation algorithms:
a. First fit
b. Best fit
c. Worst fit
4. Given memory partitions of 100K, 500K, 200K, 300K, and 600K (in order), how would each of the Firstfit, Best-fit, and Worst-fit algorithms place processes of 212K, 417K, 112K, and 426K (in order)?
Which algorithm makes the most efficient use of memory?
5. Consider a logical address space of eight pages of 1024 words each, mapped onto a physical memory of 32 frames.
a. How many bits are there in the logical address?
b. How many bits are there in the physical address?
6. Why are segmentation and paging sometimes combined into one scheme?

9. Under what circumstances do page faults occur? Describe the actions taken by the operating system when a page fault occurs.
10. Which of the following programming techniques and structures are “good” for a demand paged environment ?
Which are “not good”? Explain your answers.
a. Stack
b. Hashed symbol table
c. Sequential search
d. Binary search
e. Pure code
f. Vector operations
g. Indirection
11. Consider the following page-replacement algorithms. Rank these algorithms on a Five point
scale from “bad” to “perfect” according to their page-fault rate. Separate those algorithms that suffer from Belady’s anomaly from those that do not.
a. LRU replacement
b. FIFO replacement
c. Optimal replacement d. Second-chance replacement
12. Name two differences between logical and physical addresses.
14. Explain the difference between internal and external fragmentation.
15. Why are segmentation and paging sometimes combined into one scheme?
16. Under what circumstances do page faults occur? Describe the actions taken by the operating system when a page fault occurs.