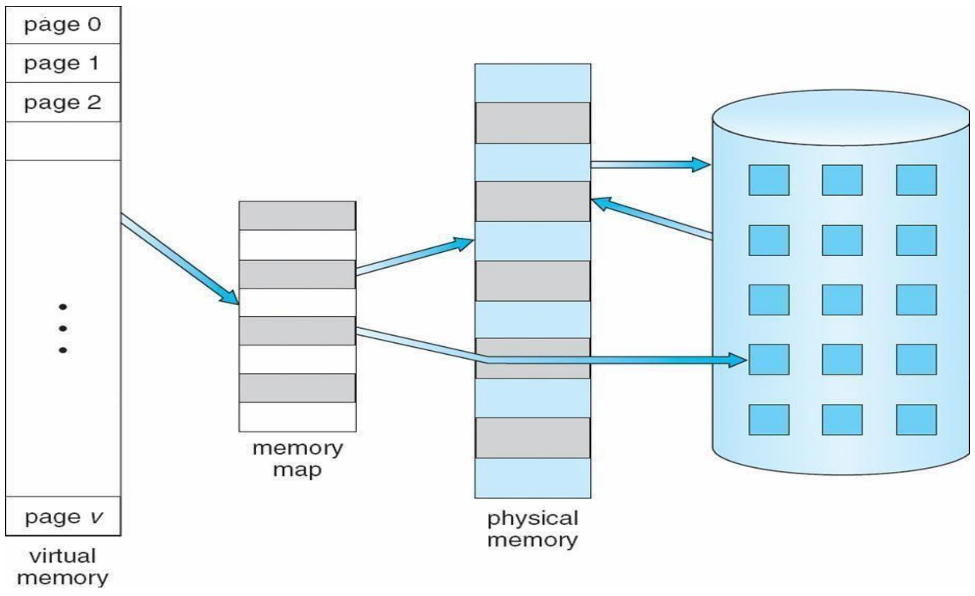
MODULE 4

1. **Virtual Memory Management**
    - Background
    - Demand Paging
    - Page Replacement
2. **Storage Management**
    - Mass Storage Structures
    - Disk Structure
    - Disk Scheduling
    - Swap Space Management
3. **Protection**
    - Goals of Protection
    - Principles of Protection
    - Domain of Protection
    - Domain Structure
    - Access Matrix
    - Implementation of Access Matrix

VIRTUAL MEMORY MANAGEMENT

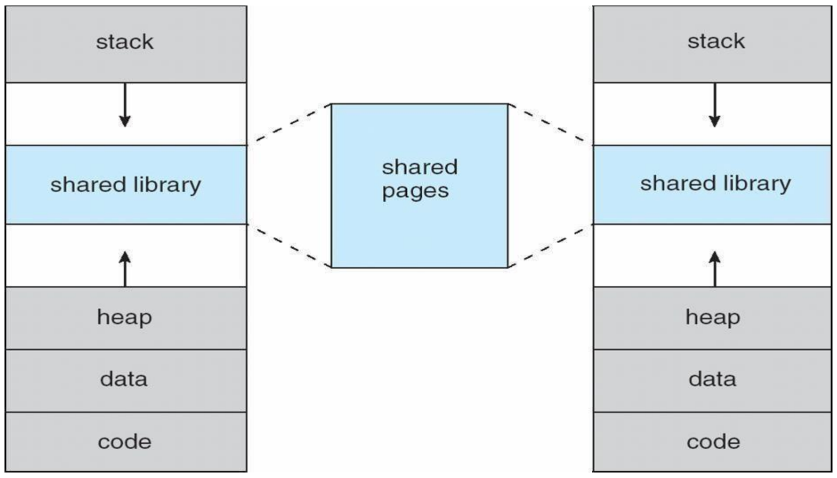
Background

- 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.



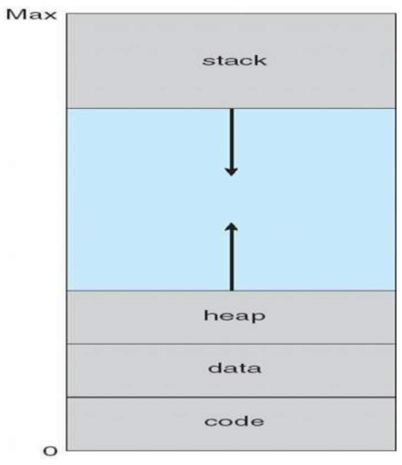
Virtual memory that is larger than physical memory.

- Virtual memory is the separation of user's 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.



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 stackor heap grows in size.

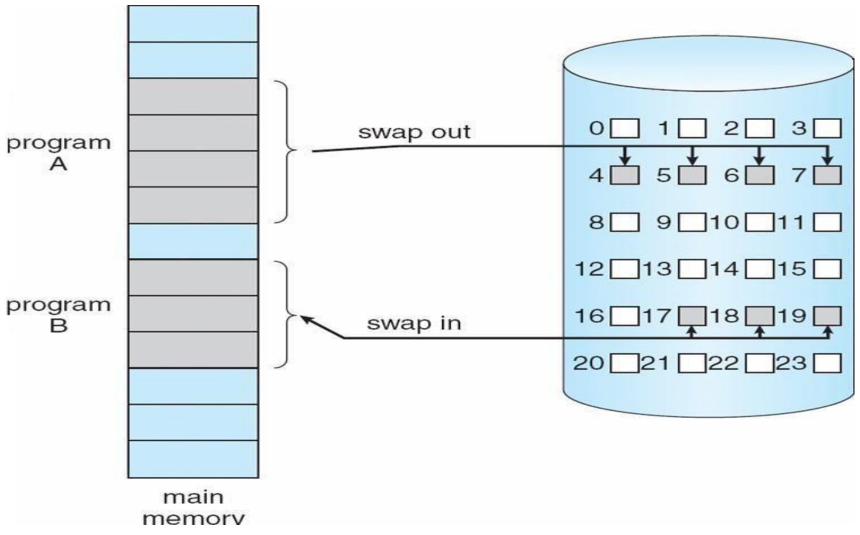


Virtual address space

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 beneeded
- Swapper that deals with pages is a **pager.**

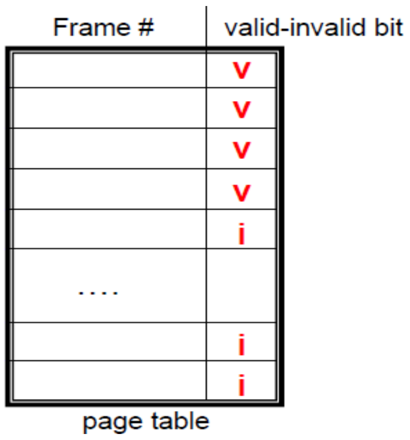


Transfer of a paged memory into continuous disk space

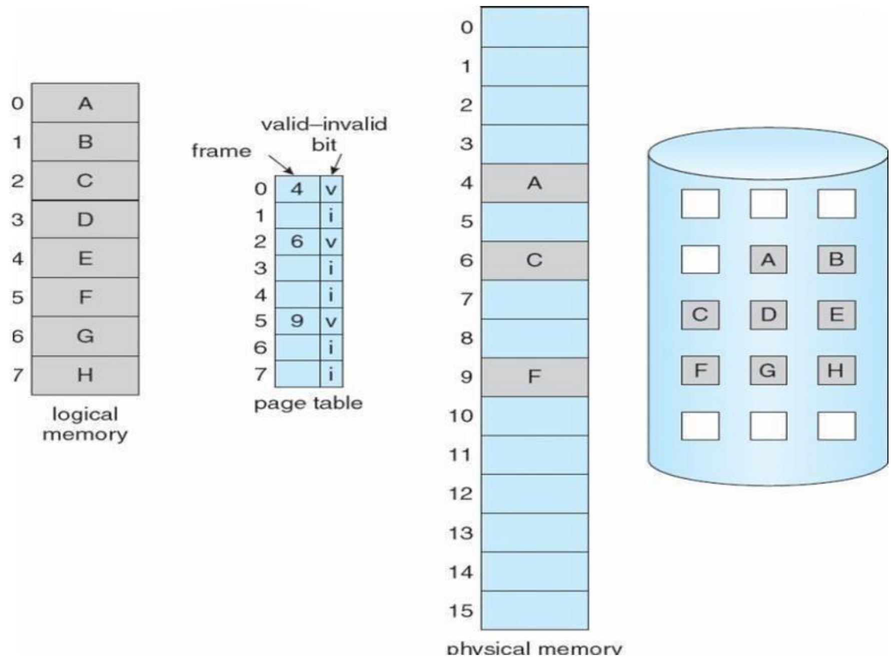
1) Basic concept:

- 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:



- 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



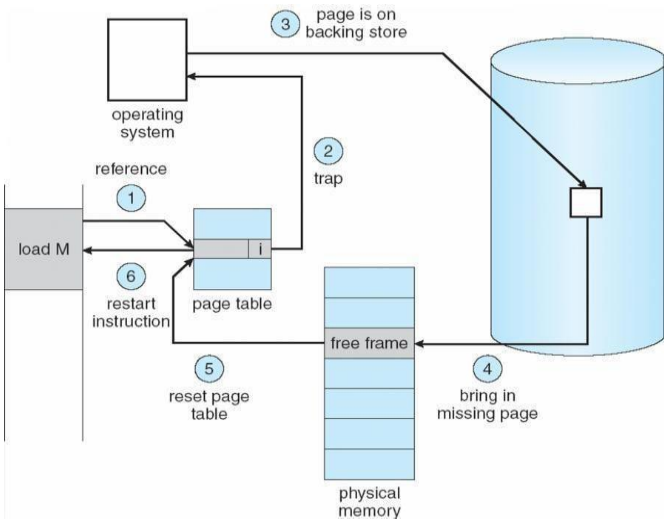
Page Table when some pages are not in main memory

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.



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.

<u>Hardware support</u>: 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.

2) Performance of Demand Paging:

Demand paging can have significant effect on theperformance 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

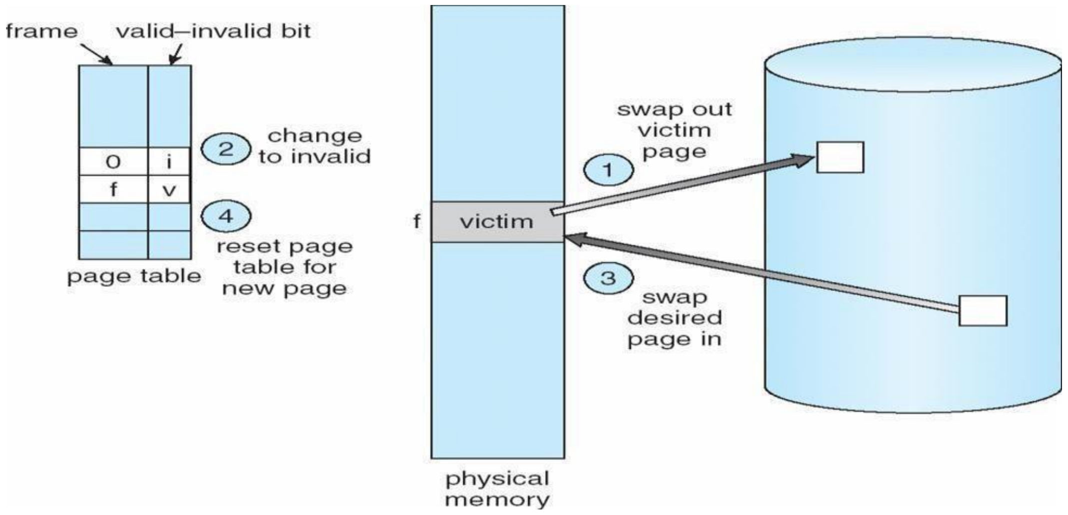
  = **200 + p x 7,999,800**
- If one access out of 1,000 causes a page fault, then EAT = 8.2 microseconds. This isa slowdown by a factor of 40.

Page Replacement

- Page replacement policy deals with the solution of pages in memory to be replaced bya 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 thatthis 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 satisfythe

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.



Page Replacement

Working of Page Replacement Algorithm

1. Find the location of derived page on the disk.
2. Find a free frame
    - If there is a free frame, use it.
    - Otherwise, use a replacementalgorithm 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 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.

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 writingit 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 tothe disk.
- Modify bit is used to reduce overhead of page transfers – only modified pages are written to disk.

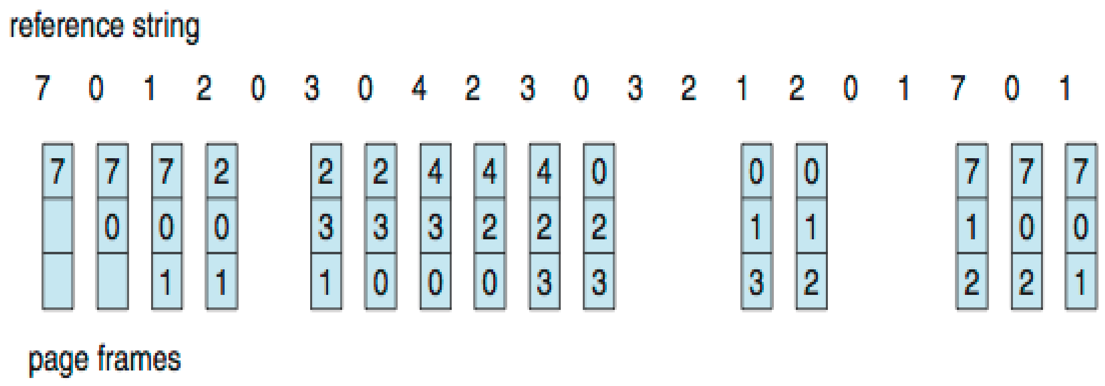
PAGE REPLACEMENT ALGORITHMS

- 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

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

1) 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.



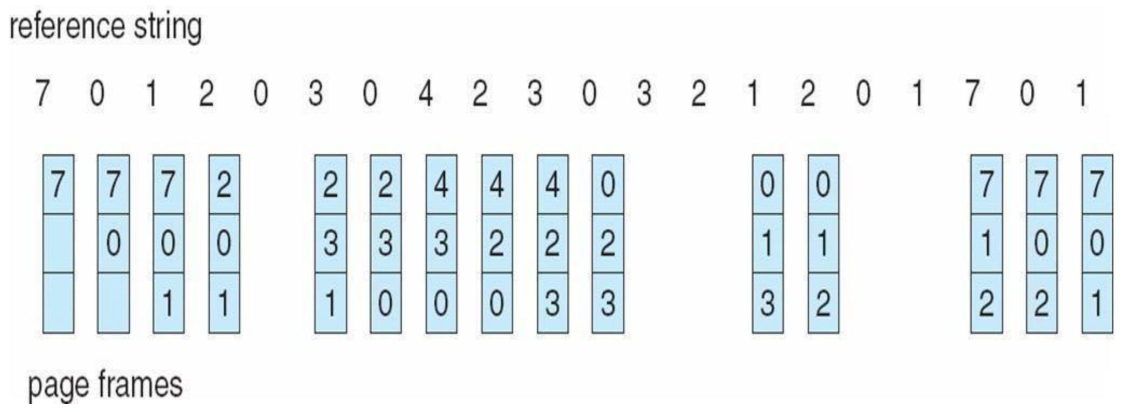
Belady's Anomaly

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

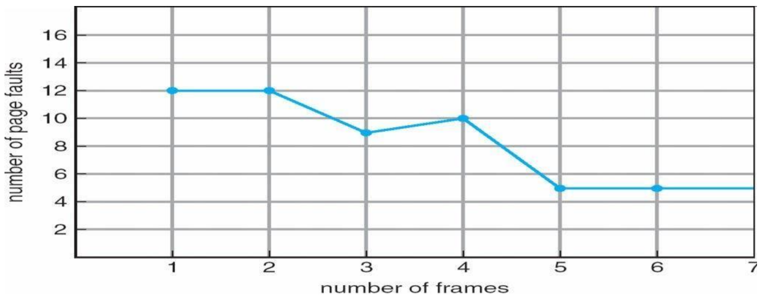
more frames ⇒ more page faults

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. 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



2) 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.

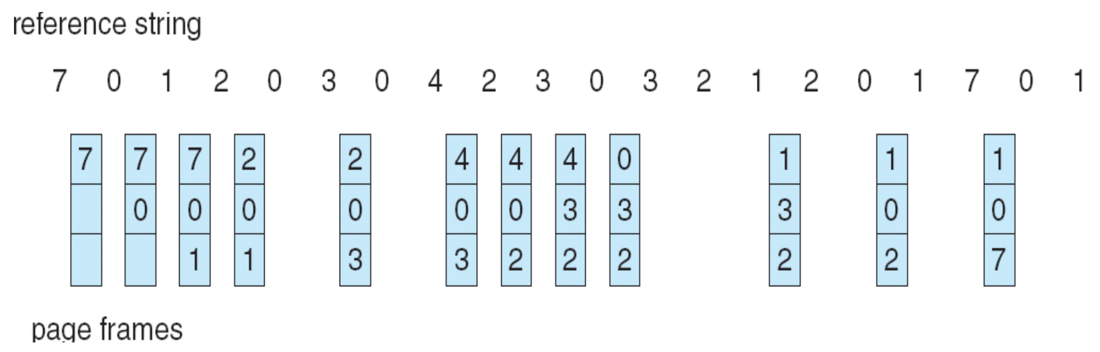
The working is simple **"Replace the page that will not be used for the longest period of time"**

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. 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

3) 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.

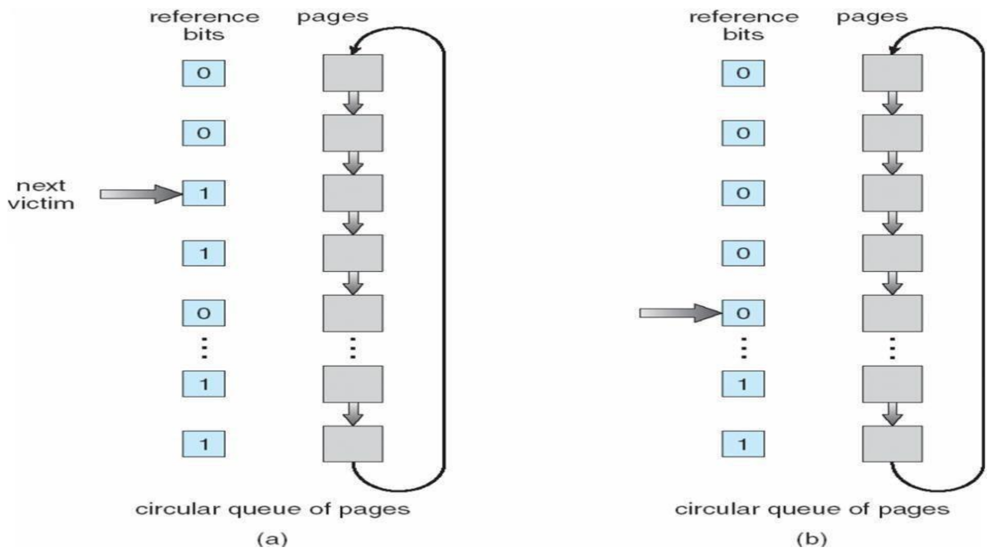
LRU-Approximation Page Replacement

- 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.

i) 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.

ii) 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**.

iii) 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.

iv) Count Based Page Replacement

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

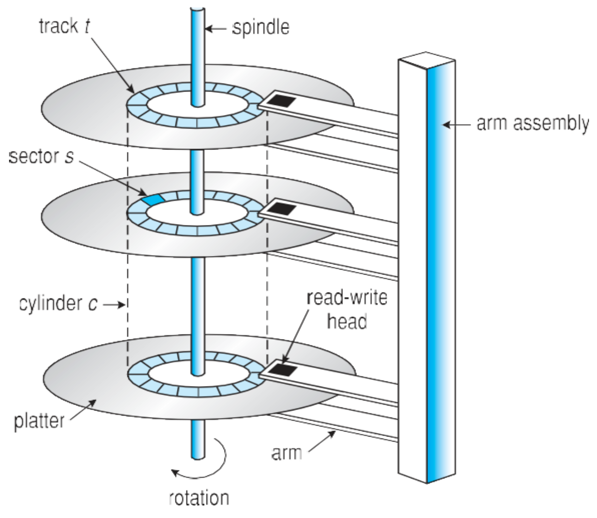
1. **LFU (least frequently used):** This causes the page with the smallest count to be replaced. The reason for this selection is that actively used page should have a large reference count. This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process but never used again. Since it was used heavily, it has a largecount and remains in memory even though it is no longer needed.
2. **MFU Algorithm:** based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

STORAGE MANAGEMENT

Mass Storage Structure

1) Magnetic Disks

- Magnetic disks provide the bulk of secondary storage for modern computer systems.
- Each disk platter has a flat circular shape, like a CD. Common platter diameters range from 1.8 to 5.25 inches.
- The two surfaces of a platter are covered with a magnetic material. The information stored by recording it magnetically on the platters.
- The surface of a platter is logically divided into circular tracks, which are subdivided into sectors. Sector is the basic unit of storage. The set of tracks that are at one arm position makes up a cylinder.
- The number of cylinders in the disk drive equals the number of tracks in each platter.
- There may be thousands of concentric cylinders in a disk drive, and each track may contain hundreds of sectors.



Moving-head disk mechanism

- **Seek Time**:- Seek time is the time required to move the disk arm to the required track.
- **Rotational Latency (Rotational Delay)**:- Rotational latency is the time taken for the disk to rotate so that the required sector comes under the r/w head.
- **Positioning time or random access time** is the summation of seek time and rotational

delay.

- **Disk Bandwidth**:- Disk bandwidth is the total number of bytes transferred divided by total time between the first request for service and the completion of last transfer.
- **Transfer rate** is the rate at which data flow between the drive and the computer.

As the disk head flies on an extremely thin cushion of air, the head will make contact with the disk surface. Although the disk platters are coated with a thin protective layer, sometimes the head will damage the magnetic surface. This accident is called a **head crash**.

2) Magnetic Tapes

- Magnetic tape is a secondary-storage medium. It is a permanent memory and can hold large quantities of data.
- The time taken to access data (access time) is large compared with that of magnetic disk, because here data is accessed sequentially.
- When the nth data has to be read, the tape starts moving from first and reaches the nth position and then data is read from nth position. It is not possible to directly move to the nth position. So tapes are used mainly for backup, for storage of infrequently used information.
- A tape is kept in a spool and is wound or rewound past a read-write head. Moving to the correct spot on a tape can take minutes, but once positioned, tape drives can write data at speeds comparable to disk drives.

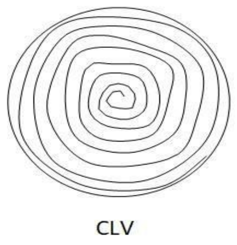
Disk Structure

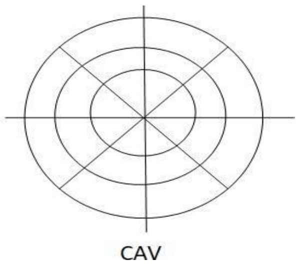
- Modern disk drives are addressed as a large one-dimensional array. The one- dimensional array of logical blocks is mapped onto the sectors of the disk sequentially.
- Sector 0 is the first sector of the first track on the outermost cylinder. The mapping proceeds in order through that track, then through the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

The disk structure (architecture) can be of two types –

1. Constant Linear Velocity (CLV)
2. Constant Angular Velocity (CAV)

1. **CLV –** The density of bits per track is uniform. The farther a track is from the center of the disk, the greater its length, so the more sectors it can hold. As we move from outer zones to inner zones, the number of sectors per track decreases. This architecture is used in CD-ROM and DVD-ROM.
2. **CAV –** There is same number of sectors in each track. The sectors are densely packed in the inner tracks. The density of bits decreases from inner tracks to outer tracks to keep the data rate constant.





Disk Scheduling

Different types of disk scheduling algorithms are as follows:

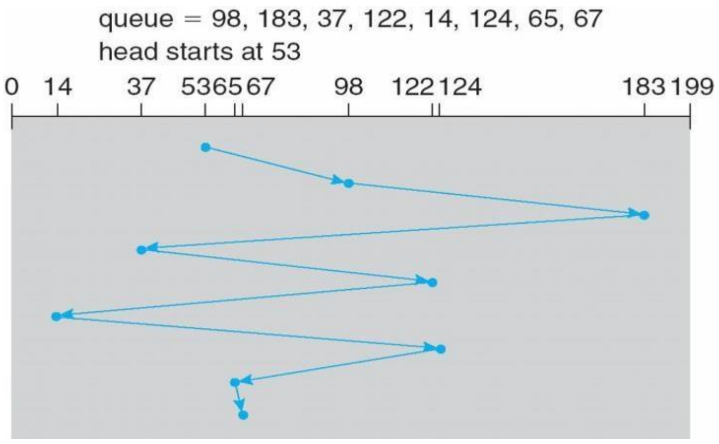
1. FCFS (First Come First Serve)
2. SSTF (Shortest Seek Time First)
3. SCAN (Elevator)
4. C-SCAN
5. LOOK
6. C-LOOK

1) FCFS scheduling algorithm:

This is the simplest form of disk scheduling algorithm. This services the request in the order they are received. This algorithm is fair but do not provide fastest service. It takes no special care to minimize the overall seek time.

Eg:- consider a disk queue with request for i/o to blocks on cylinders. 98, 183, 37, 122, 14, 124, 65, 67

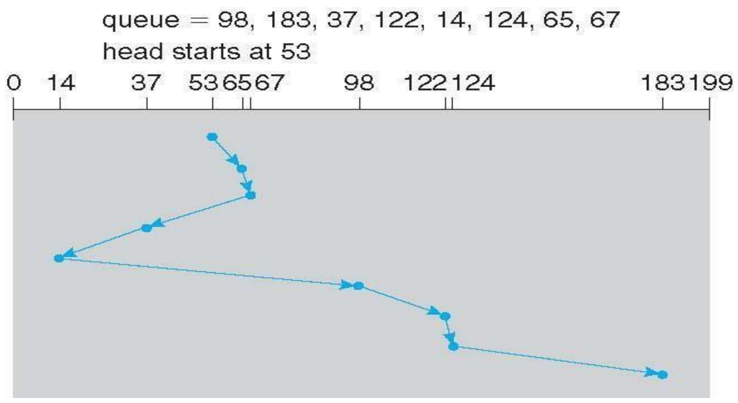
If the disk head is initially at 53, it will first move from 53 to 98 then to 183 and then to 37, 122, 14, 124, 65, 67 for a total head movement of 640 cylinders. The wild swing from 122to 14 and then back to 124 illustrates the problem with this schedule.



2) SSTF (Shortest Seek Time First) algorithm:

This selects the request with minimum seek time from the current head position. SSTF chooses the pending request closest to the current head position.

<u>Eg:</u>- consider a disk queue with request for i/o to blocks on cylinders. 98, 183, 37, 122, 14, 124, 65, 67

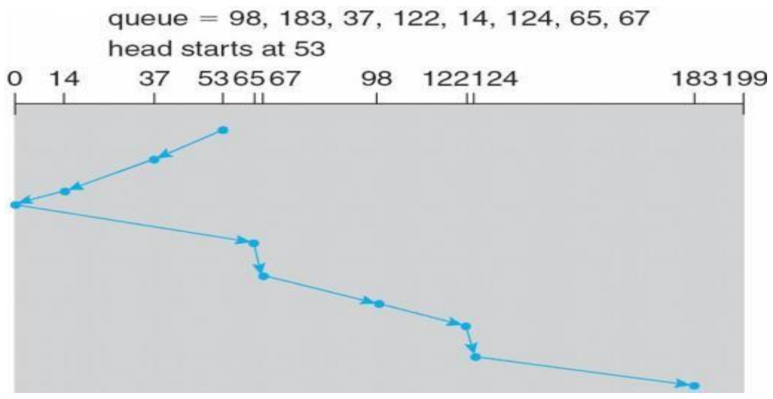


If the disk head is initially at 53, the closest is at cylinder 65, then 67, then 37 is closer than98 to 67. So it services 37, continuing we service 14, 98, 122, 124 and finally 183. The total head movement is only 236 cylinders. SSTF is a substantial improvement over FCFS, it is not optimal.

3) SCAN algorithm:

In this the disk arm starts moving towards one end, servicing the request as it reaches each cylinder until it gets to the other end of the disk. At the other end, the direction of the head movement is reversed and servicing continues. The initial direction is chosen depending uponthe direction of the head.

Eg:- consider a disk queue with request for i/o to blocks on cylinders. 98, 183, 37, 122, 14, 124, 65, 67



If the disk head is initially at 53 and if the head is moving towards the outer track, it services 65, 67, 98, 122, 124 and 183. At cylinder 199 the arm will reverse and will move towards the other end of the disk servicing 37 and then 14. The SCAN is also called as elevator algorithm.

4) C-SCAN (Circular scan) algorithm:

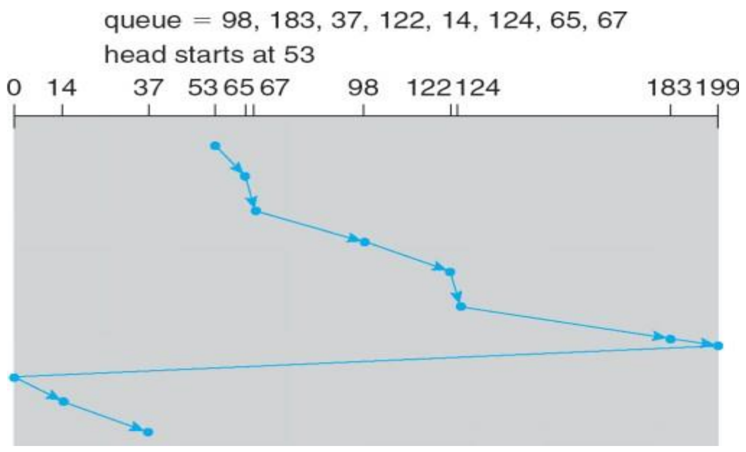
C-SCAN is a variant of SCAN designed to provide a more uniform wait time.

Like SCAN, C-SCAN moves the head from end of the disk to the other servicing the request along the way. When the head reaches the other end, it immediately returns to the beginning of the disk, without servicing any request on the return.

Eg:- consider a disk queue with request for i/o to blocks on cylinders. 98, 183, 37, 122, 14, 124, 65, 67

If the disk head is initially at 53 and if the head is moving towards the outer track, it services 65, 67, 98, 122, 124 and 183. At cylinder 199 the arm will reverse and will move immediately towards the other end of the disk, then changes the direction of head and serves 14 and then 37.

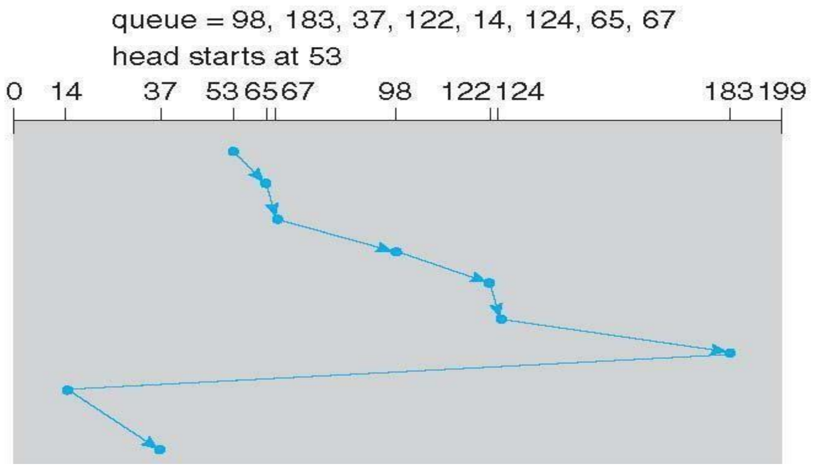
Note: If the disk head is initially at 53 and if the head is moving towards track 0, it services 37 and 14 first. At cylinder 0 the arm will reverse and will move immediately towards the other end of the disk servicing 65, 67, 98, 122, 124 and 183.



5) Look Scheduling algorithm:

Look and C-Look scheduling are different version of SCAN and C-SCAN respectively. Here the arm goes only as far as the final request in each direction. Then it reverses, without going all the way to the end of the disk. The Look and C-Look scheduling look for a request before continuing to move in a given direction.

Eg:- consider a disk queue with request for i/o to blocks on cylinders. 98, 183, 37, 122, 14, 124, 65, 67



C-LOOK disk scheduling.

If the disk head is initially at 53 and if the head is moving towards the outer track, it services 65, 67, 98, 122, 124 and 183. At the final request 183, the arm will reverse and will move towards the first request 14 and then serves 37.

Swap-Space Management

- Swap-space management is another low-level task of the operating system.
- Swapping occurs when the amount of physical memory reaches a critically low pointand processes are moved from memory to swap space to free available memory.

1) Swap-Space Use

- The amount of swap space needed on a system can vary depending on the amount of physical memory, the amount of virtual memory it is backing, and the way in which the virtual memory is used. It can range from a few megabytes of disk space to gigabytes.
- The swap space can overestimate or underestimated. It is safer to overestimate than to underestimate the amount of swap space required. If a system runs out of swap space due to underestimation of space, it may be forced to abort processes or may crash entirely. Overestimation wastes disk space that could otherwise be used for files, butit does no other harm.

2) Swap-Space Location

- A swap space can reside in one of two places: It can be carved out of the normal file system, or it can be in a separate disk partition. If the swap space is simply a large file within the file system, normal file-system routines can be used to create it, name it, and allocate its space.
- External fragmentation can greatly increase swapping times by forcing multiple seeks during reading or writing of a process image. We can improve performance by caching the block location information in physical memory.
- Alternatively, swap space can be created in a separate raw partition. A separate swap-space storage manager is used to allocate and deallocate the blocks from the raw partition.

3) Swap-Space Management: An Example

- Solaris allocates swap space only when a page is forced out of physical memory, rather than when the virtual memory page is first created.
- Linux is similar to Solaris in that swap space is only used for anonymous memory or for

regions of memory shared by several processes. Linux allows one or more swap areas to be established.

- A swap area may be in either a swap file on a regular file system or a raw swap partition. Each swap area consists of a series of 4-KB page slots, which are used to hold swapped pages. Associated with each swap area is a swap map— an array of integer counters, each corresponding to a page slot in the swap area.
- If the value of a counter is 0, the corresponding page slot is available. Values greater than 0 indicate that the page slot is occupied by a swapped page. The value of the counter indicates the number of mappings to the swapped page; for example, a value of 3 indicates that the swapped page is mapped to three different processes.

PROTECTION

Goals of Protection

- Protection is a mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system. Protection ensures that only processes that have gained proper authorization from the operating system can operate on the files, memory segments, CPU, and other resources of a system.
- Protection is required to prevent mischievous, intentional violation of an access restriction by a user.

Principles of Protection

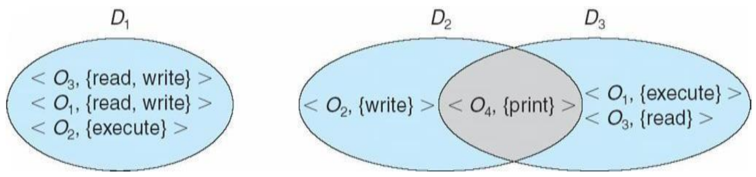
- A key, time-tested guiding principle for protection is the 'principle of least privilege'. It dictates that programs, users, and even systems be given just enough privileges to perform their tasks.
- An operating system provides mechanisms to enable privileges when they are needed and to disable them when they are not needed.

Domain of Protection

- A computer system is a collection of processes and objects. Objects are both hardware objects (such as the CPU, memory segments, printers, disks, and tape drives) and software objects (such as files, programs, and semaphores). Each object (resource) has a unique name that differentiates it from all other objects in the system.
- The operations that are possible may depend on the object. For example, a CPU can only be executed on. Memory segments can be read and written, whereas a CD-ROM or DVD-ROM can only be read. Tape drives can be read, written, and rewound. Data files can be created, opened, read, written, closed, and deleted; program files can be read, written, executed, and deleted.
- A process should be allowed to access only those resources for which it has authorization and currently requires to complete process.

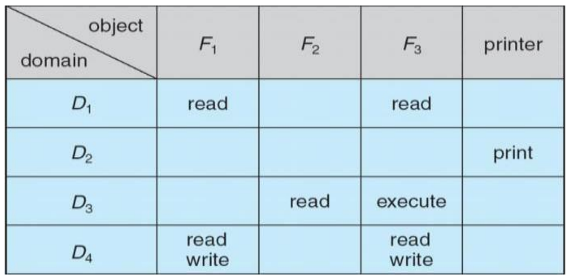
Domain Structure

- A domain is a set of objects and types of access to these objects. Each domain is an ordered pair of <object-name, rights-set>.
- Example, if domain D has the access right <file F,{read,write}>, then all process executing in domain D can both read and write file F, and cannot perform any other operation on that object.
- Domains do not need to be disjoint; they may share access rights. For example, in below figure, we have three domains: D1 D2, and D3. The access right < O4, (print}>is shared by D2 and D3,it implies that a process executing in either of these two domains can print object O4.
- A domain can be realized in different ways, it can be a user, process or a procedure. ie. each user as a domain, each process as a domain or each procedure as a domain.



Access Matrix

- Our model of protection can be viewed as a matrix, called an access matrix. It is a general model of protection that provides a mechanism for protection without imposing a particular protection policy.
- The rows of the access matrix represent domains, and the columns represent objects.
- Each entry in the matrix consists of a set of access rights.
- The entry access (i,j) defines the set of operations that a process executing in domain Di can invoke on object Oj.

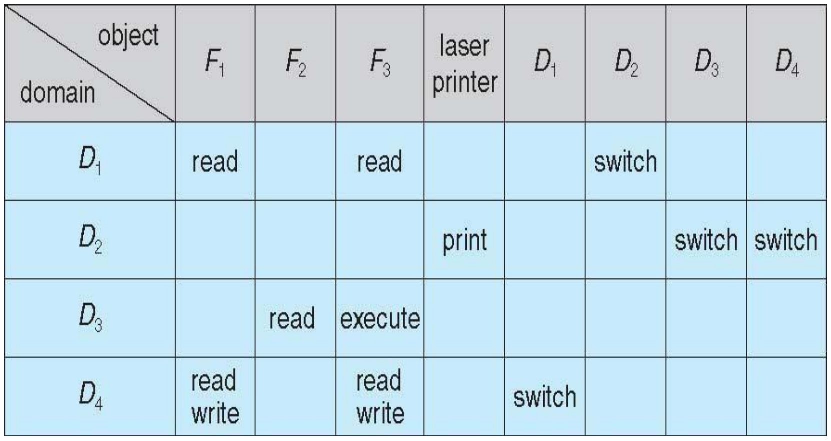


object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

- In the above diagram, there are four domains and four objects—three files (F1, F2,F3) and one printer. A process executing in domain D1 can read files F1 and F3. A process executing in domain D4 has the same privileges as one executing in domain D1; but in addition, it can also write onto files F1 and F3.
- When a user creates a new object Oj, the column Oj is added to the access matrix withthe appropriate initialization entries, as dictated by the creator.

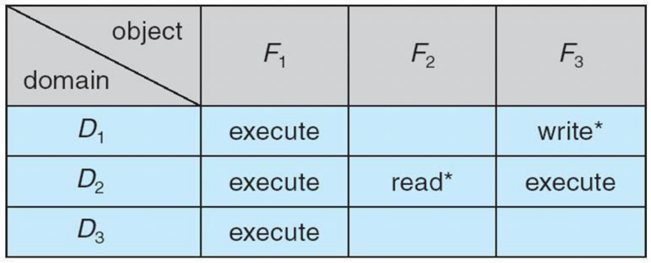
The process executing in one domain and be switched to another domain. When we switch a process from one domain to another, we are executing an operation (switch) on an object (the domain).

Domain switching from domain Di to domain Dj is allowed if and only if the access right switch access(i,j). Thus, in the given figure, a process executing in domain D2 can switch to domain D3 or to domain D4. A process in domain D4 can switch to D1, and one in domain D1 can switch to domain D2.



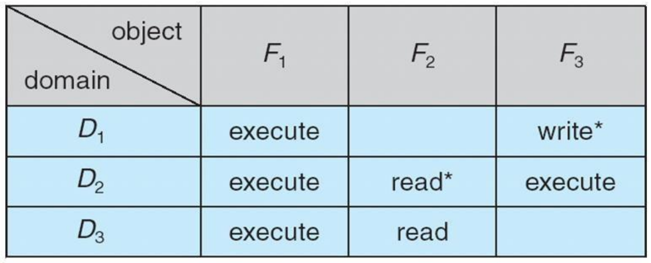
object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Allowing controlled change in the contents of the access-matrix entries requires three additional operations: copy, owner, and control.



object domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)



object domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

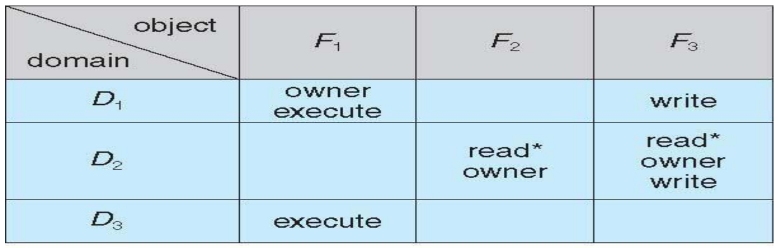
The ability to copy an access right from one domain (or row) of the access matrix to anotheris denoted by an asterisk (*) appended to the access right. The copy right allows the copying of the access right only within the column for which the right is defined. In the below figure,a process executing in domain D2 can copy the read operation into any entry associated with file F2. Hence, the access matrix of figure (a) can be modified to the access matrix shown in figure (b).

<u>This scheme has two variants:</u>

1. A right is copied from access(i,j) to access(k,j); it is then removed from access(i,j). This action is a transfer of a right, rather than a copy.
2. Propagation of the copy right- limited copy. Here, when the right R* is copied from access(i,j) to access(k,j), only the right R (not R*) is created. A process executing in domain Dk cannot further copy the right R.

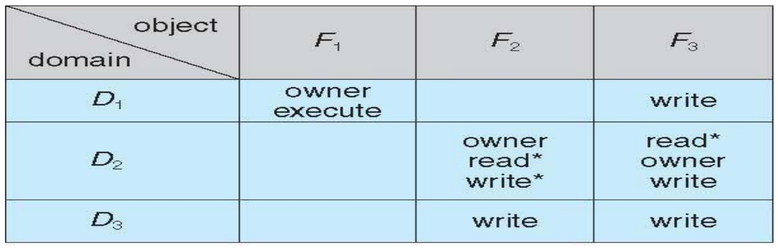
We also need a mechanism to allow addition of new rights and removal of some rights. The owner right controls these operations. If access(i,j) includes the owner right, then a process executing in domain Di, can add and remove any right in any entry in column j.

<u>For example</u>, in below figure (a), domain D1 is the owner of F1, and thus can add and delete any valid right in column F1. Similarly, domain D2 is the owner of F2 and F3 and thus can add and remove any valid right within these two columns. Thus, the access matrix of figure(a) can be modified to the access matrix shown in figure(b) as follows.



object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

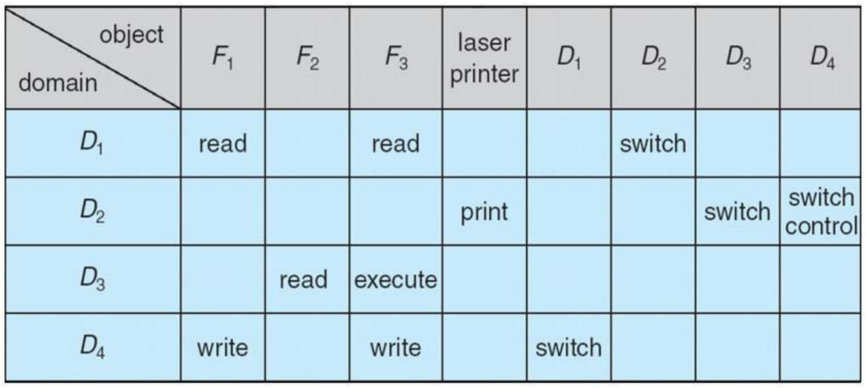
(a)



object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

A mechanism is also needed to change the entries in a row. If access (i,j) includes the control right, then a process executing in domain Di, can remove any access right from row j. For example, in figure, we include the control right in access (D3, D4). Then, a process executing in domain D3 can modify domain D4.



object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Implementation of Access Matrix

Different methods of implementing the access matrix (which is sparse)

- Global Table
- Access Lists for Objects
- Capability Lists for Domains
- Lock-Key Mechanism

1) Global Table

- This is the simplest implementation of access matrix.
- A set of ordered triples <domain, object, rights-set> is maintained in a file. Whenever an operation M is executed on an object Oj, within domain Di, the table is searched for a triple <Di, Oj, Rk>. If this triple is found, the operation is allowed to continue; otherwise, an exception (or error) condition is raised.

Drawbacks -

The table is usually large and thus cannot be kept in main memory. Additional I/O is needed.

2) Access Lists for Objects

- Each column in the access matrix can be implemented as an access list for one object. The empty entries are discarded. The resulting list for each object consists of ordered pairs <domain, rights-set>.
- It defines all domains access right for that object. When an operation M is executed on object Oj in Di, search the access list for object Oj, look for an entry <Di, RK > with M ϵ Rk. If the entry is found, we allow the operation; if it is not, we check the default set. If M is in the default set, we allow the access. Otherwise, access is denied, and an exception condition occurs. For efficiency, we may check the default set first and then search the access list.

3) Capability Lists for Domains

- A capability list for a domain is a list of objects together with the operations allowed on those objects. An object is often represented by its name or address, called a capability.

- To execute operation M on object Oj, the process executes the operation M, specifying the capability for object Oj as a parameter. Simple possession of the capability means that access is allowed.

Capabilities are distinguished from other data in one of two ways:

1. Each object has a tag to denote its type either as a capability or as accessible data.
2. The address space associated with a program can be split into two parts. One part is accessible to the program and contains the program's normal data and instructions. The other part, containing the capability list, is accessible only by the operating system.

4) A Lock-Key Mechanism

- The lock-key scheme is a compromise between access lists and capability lists.
- Each object has a list of unique bit patterns, called locks. Each domain has a list of unique bit patterns, called keys.
- A process executing in a domain can access an object only if that domain has a keythat matches one of the locks of the object.