Module V
Secondary - Storage Structure

Magnetic disks provide a bulk of secondary storage. Disks come in various sizes and speed. Heretheinformation is stored magnetically. Each disk platter has a flat circular shape like CD. Thetwosurfaces of a platter are covered with a magnetic material. The surface of a platter is logicallydivided into circular **tracks,** which are subdivided into **sectors.** Sector is the basic unit ofstorage. The set of tracks that are at one arm position makes up a **cylinder.**

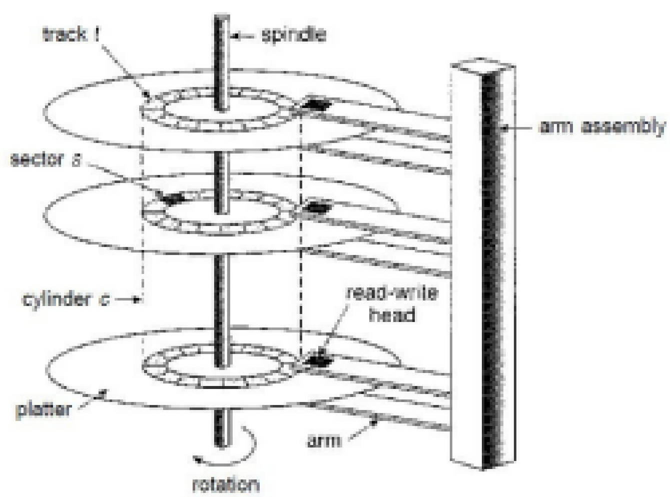


Figure 12.1 Moving-head disk mechanism.

The number of cylinders in the disk dirve equals the number of tracks in each platter. Theremaybe thousands of concentric cylinders in a disk drive, and each track may contain hundreds of sectors. The storage capacityof disk drives is measured in gigabytes.

The head moves fromthe inner trackof the disk to the outer track. Whenthe disk drive is operating the disks is rotating at a constant speed.

To read or write the head must be positioned at the desired trackandat the beginning of the desired sector on that track.

• 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 disktorotate 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 bytotal 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 withthedisk surface. Although the disk platters are coated with a thin protective layer, sometimes thehead will damage the magnetic surface. This accident is called a **head crash.**

Magnetic Tapes

Magnetic tape is a secondary-storage medium. It is a permanent memory and can holdlargequantities of data. The time taken to access data (access time) is large compared withthat ofmagnetic disk, because here data is accessed sequentially. When the nth data has to be read, thetape starts moving from first and reaches the nth position and then data is read fromnth position. It is not possible to directly move to the nth position. So tapes are used mainly for backup, forstorage of infrequently used information.

DISK STRUCTURE

Each disk platter is divided into number of tracks and each track is divided intonumberof sectors. Sectors is the basic unit for read or write operation in the disk. 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 thefirst sector of the first track on the outermost cylinder. The mapping proceeds in order throughthat track, then through the rest of the tracks in that cylinder, and then through the rest of thecylinders from outermost to innermost.

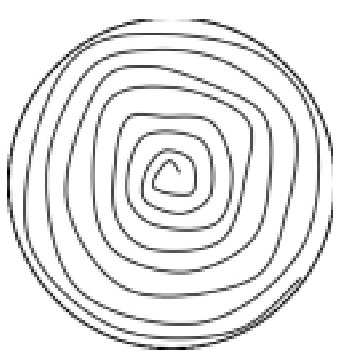
The disk structure (architecture) can be of two types –

i) **Constant Linear Velocity** (CLV)
ii) **Constant Angular Velocity** (CAV)

i) CLV - The density of bits per track is uniform. The farther a track is fromthe center ofthe disk, the greater its length, so the more sectors it can hold. As we move fromouterzones to inner zones, the number of sectors per track decreases. This architectureisused in CD-ROM and DVD-ROM.

ii) CAV – There is same number of sectors in each track. The sectors are densely packedinthe inner tracks. The density of bits decreases from inner tracks to outer tracks tokeepthe data rate constant.

CLVCAV



DISK ATTACHMENT

Computers can access data in two ways.

i) via I/O ports (or host-attached storage)

ii) via a remote host in a distributed file system(or network-attached storage)

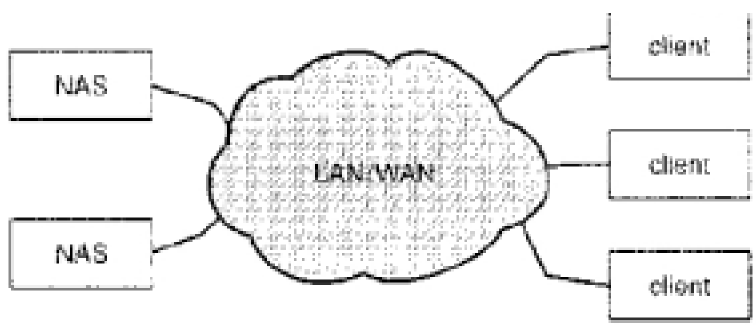
i) Host-Attached Storage

Host-attached storage is storage accessed through local I/O ports. Example : the typical desktop PC uses an I/O bus architecture called IDE or ATA. This architecture supportsamaximum of two drives per I/O bus. The other cabling systems are – **SATA**(Serially AttachedTechnology Attachment), **SCSI**(Small Computer System Interface) and **fiber channel** (FC).

SCSI is a bus architecture. Its physical medium is usually a ribbon cable. FCis ahigh-speed serial architecture that can operate over optical fiber or over a four-conductor coppercable. An improved version of this architecture is the basis of **storage-area networks** (SANs).

ii) Network-Attached Storage

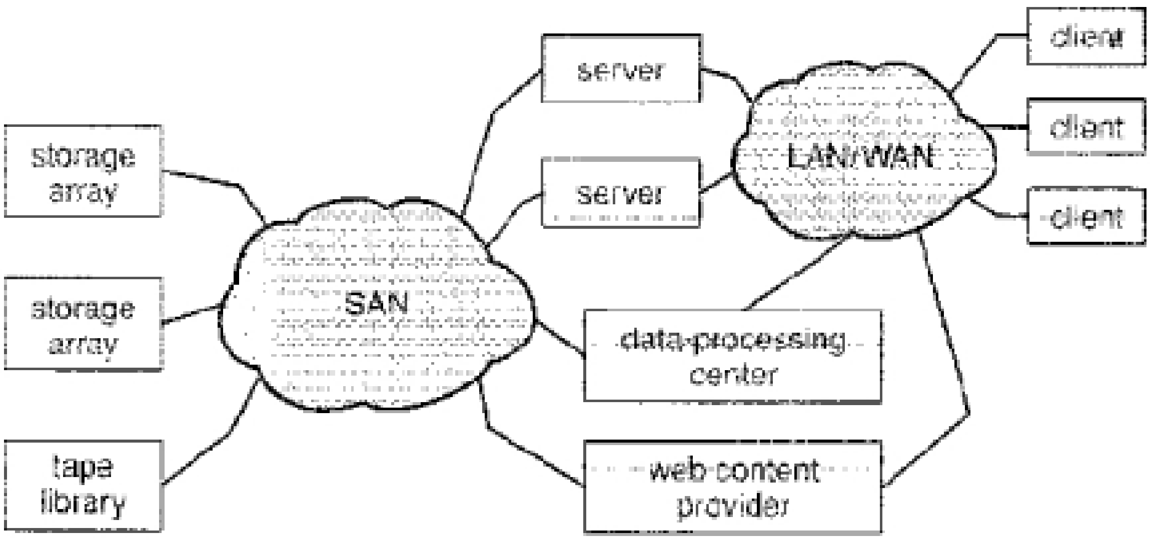
A network-attached storage (NAS) device is a special-purpose storage systemthat isaccessed remotely over a network as shown in the figure. Clients access network-attachedstorage via a remote-procedure-call interface. The remote procedure calls (RPCs) are carriedviaTCP or UDP over an IP network—-usually the same local-area network (LAN) carries all datatraffic to the clients.



Network- attached storage provides a convenient way for all the computers on a LAN to share a pool of storage files.

iii) Storage Area Network(SAN)

A storage-area network (SAN) is a private network connecting servers and storageunits. The power of a SAN lies in its flexibility. Multiple hosts and multiple storage arrays canattachto the same SAN, and storage can be dynamically allocated to hosts. A SANswitch allowsorprohibits access between the hosts and the storage. Fiber Chanel is the most commonSANinterconnect.

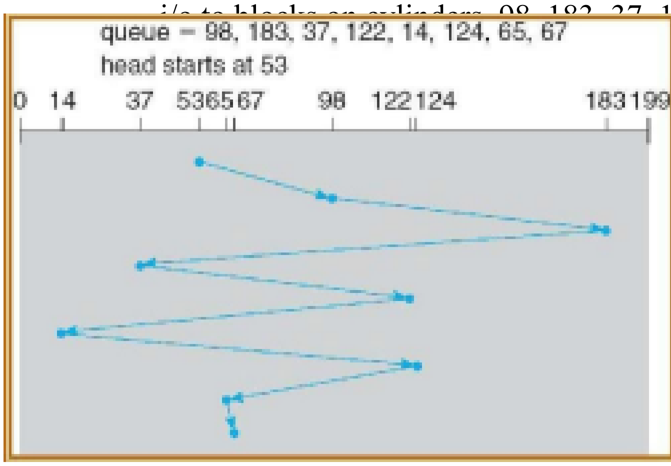


DISK SCHEDULING

Different types of disk scheduling algorithms are as follows:

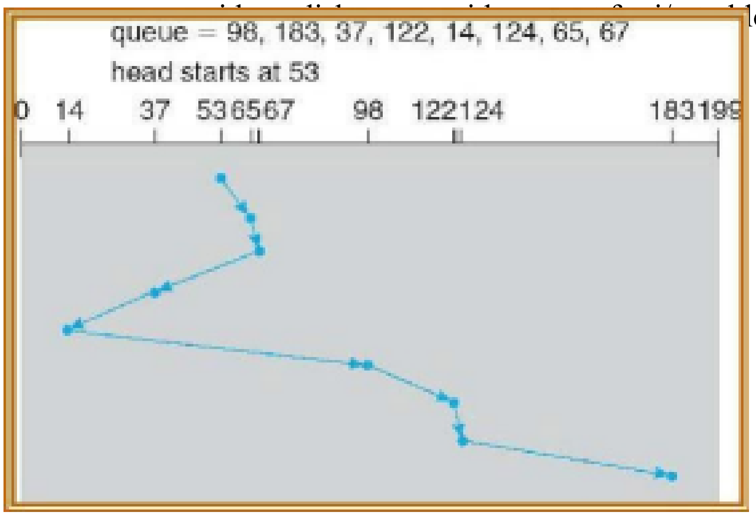
- FCFS (First Come First Serve)
- SSTF(Shortest Seek Time First)
- SCAN (Elecvator)
- C-SCAN
- LOOK
- C-LOOK

i) **FCFS scheduling algorithm**: This is the simplest form of disk scheduling algorithm. This services the request in the order they are received. This algorithmis fair but donot provide fastest service. It takes no special care to minimize the overall seektime. *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 andthento 37, 122, 14, 124, 65, 67 for a total head movement of 640 cylinders. Thewild swing from 122 to 14 and then backto124 illustrates the problem with this schedule.

ii) **SSTF** (Shortest Seek Time First) **algorithm**: This selects the request with minimumseektime from the current head position. SSTF chooses the pending request closest tothecurrent head position. *Eg*:- locksoncylinders. 98, 183, 37, 122, 14, 124, 65, 67

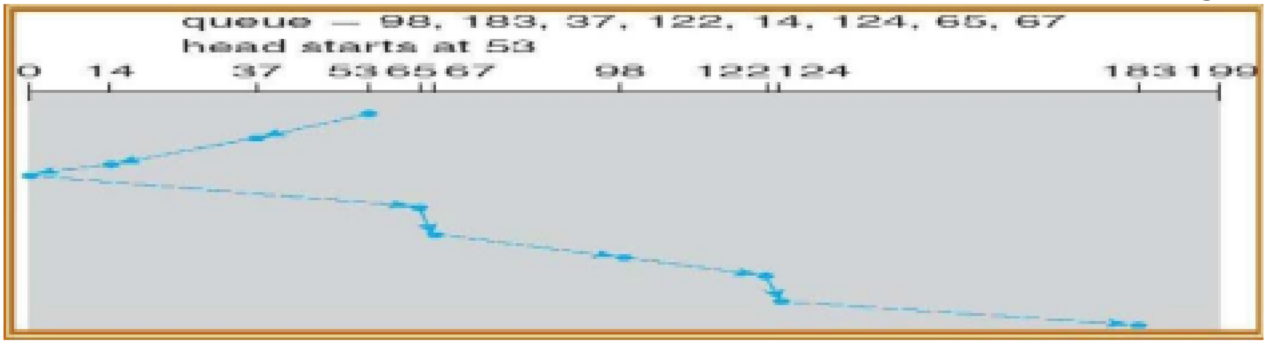


If the disk head is initially at 53, the closest is at cylinder 65, then67, then 37 is closer then 98 to 67. Soit services 37, continuing we service 14, 98, 122, 124 and finally183. The total head movement is only 236 cylinders. SSTF is a substantial improvement over FCFS, it is not optimal.

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

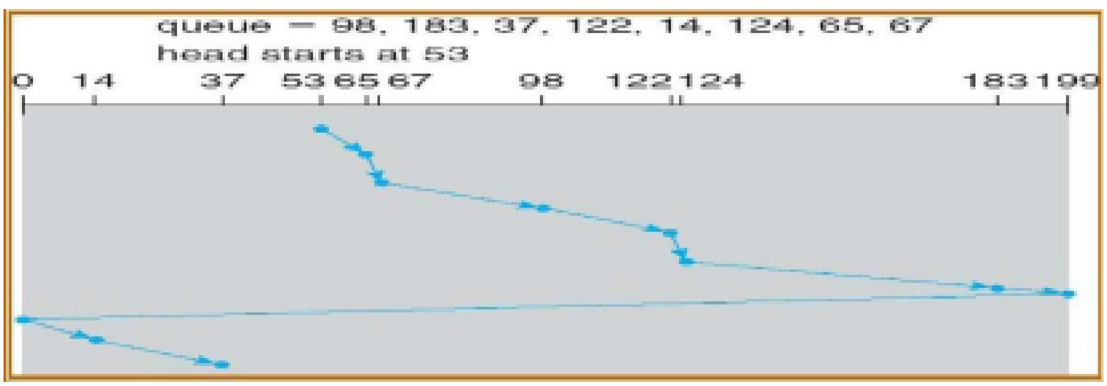
If the disk head is initially at 53 and if the head is moving towards the outer track, it services65, 67, 98, 122, 124 and183. At cylinder 199 the arm will reverse and will move towards theotherend of the disk servicing 37 and then 14. The SCAN is also called as **elevator** algorithm.



If the disk head is initially at 53 and if the head is moving towards **0th track**, it services 37andthen 14. At cylinder 0 the arm will reverse and will move towards the other end of thediskservicing 65, 67, 98, 122, 124 and 183.

iv) **C-SCAN** (Circular scan) **algorithm**:

C-SCAN is a variant of SCAN designed to provide a more uniformwait time. LikeSCAN, C-SCAN moves the head from end of the disk to the other servicing the request alongtheway. When the head reaches the other end, it immediately returns to the beginning of thedisk, 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 services65, 67, 98, 122, 124 and 183. At cylinder 199 the arm will reverse and will move immediatelytowards the other end of the disk, then changes the direction of head and serves 14 and then37.

Note: If the disk head is initially at 53 and if the head is moving towards track 0, it services37and 14 first. At cylinder 0 the arm will reverse and will move immediately towards the other endof the disk servicing 65, 67, 98, 122, 124 and183.

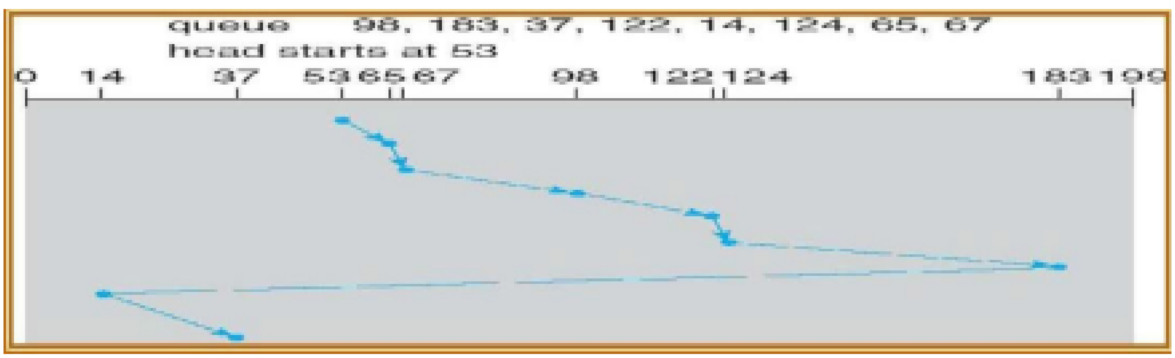
v) **Look Scheduling algorithm**:

Look and C-Look scheduling are different version of SCAN and C-SCANrespectively. 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

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 and183. At the final request 183, the arm will reverse and will move towardsthefirst request 14 and then serves 37.

vi) C-Look Scheduling algorithm:



If the disk head is initially at 53 and if the head is moving towards the outer track, it services65, 67, 98, 122, 124 and 183. At the last request, the arm will reverse and will move immediatelytowards the first request 14 and then serves 37.

Selection of a Disk-Scheduling Algorithm

SSTF is commonly used and it increases performance over FCFS.
SCAN and C-SCAN algorithm is better for a heavy load on disk.

SCAN and C-SCAN have less starvation problem.
Disk scheduling algorithm should be written as a separate module of the operating system. SSTF or Look is a reasonable choice for a default algorithm.

SSTF is commonly used algorithms has it has a less seek time when compared withotheralgorithms. SCAN and C-SCAN perform better for systems with a heavy load on the disk, (ie. more read and write operations from disk).

Selection of disk scheduling algorithm is influenced by the file allocation method, if contiguousfile allocation is choosen, then FCFS is best suitable, because the files are stored in contiguousblicks and there will be limited head movements required. A linked or indexed file, in contrast, may include blocks that are widely scattered on the disk, resulting in greater head movement.

The location of directories and index blocks is also important. Since every file must be openedtobe used, and opening a file requires searching the directory structure, the directories will beaccessed frequently. Suppose that a directory entry is on the first cylinder and a file's data areonthe final cylinder. The disk head has to move the entire width of the disk. If the directoryentrywere on the middle cylinder, the head would have to move, at most, one-half the width. Cachingthe directories and index blocks in main memory can also help to reduce the disk-armmovement, particularly for read requests.

Because of these complexities, the disk-scheduling algorithm is very important and is writtenasa separate module of the operating system.

Disk Management

Disk Formatting

The process of dividing the disk into sectors and filling the disk with a special data structureiscalled **low-level formatting**. Sector is the smallest unit of area that is read/written bythediskcontroller. The data structure for a sector typically consists of a header, a data area (usually512bytes in size) and a trailer. The header and trailer contain information used by the disk controller, such as a sector number and an error-correcting code (ECC).

When the controller writes a sector of data during normal I/O, the ECC is updated with a valuecalculated from all the bytes in the data area. When a sector is read, the ECC is recalculatedandis compared with the stored value. If the stored and calculated numbers are different, thismismatch indicates that the data area of the sector has become corrupted and that the disksector may be bad.

Most hard disks are low-level-forniatted at the factory as a part of the manufacturingprocess. This formatting enables the manufacturer to test the disk and to initialize the mappingfromlogical block numbers to defect-free sectors on the disk.

When the disk controller is instructed for low-level-formatting of the disk, the size of datablockof all sector sit can also be told how many bytes of data space to leave between the header andtrailer of all sectors. It is of sizes, such as 256, 512, and 1,024 bytes. Formatting a diskwithalarger sector size means that fewer sectors can fit on each track; but it also means that fewerheaders and trailers are written on each track and more space is available for user data.

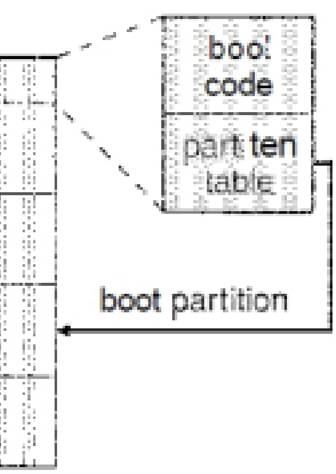
The operating system needs to record its own data structures on the disk. It does so in twosteps. **partition** and **logical formatting.**
Partition – is to partition the disk into one or more groups of cylinders. The operatingsystemcan treat each partition as though it were a separate disk. For instance, one partition canholdacopy of the operating system's executable code, while another holds user files. **logical formatting** (or creation of a file system) - Now, the operating systemstores the initial file-system data structures onto the disk. These data structures may include maps of freeandallocated space (a FAT or modes) and an initial empty directory.

To increase efficiency, most file systems group blocks together into larger chunks, frequentlycalled **clusters.**

Boot Block
When a computer is switched on or rebooted—it must have an initial programto run. Thisiscalled the *bootstrap* program. The bootstrap program – —initializes the CPU registers, device controllers, main memory, and then starts theoperating system.
—Locates and loads the operatingsystem from the disk —jumps to beginning the operating-system execution.

The bootstrap is stored in read-only memory (ROM). Since ROM is read only, it cannot beinfected by a computer virus. The problem is that changing this bootstrap code requires changingthe ROM, hardware chips. So most systems store a tiny bootstrap loader programin theboot ROM whose only job is to bring in a full bootstrap program from disk. The full bootstrapprogram can be changed easily: A new version is simply written onto the disk. The full bootstrapprogram is stored in "the boot blocks" at a fixed location on the disk. A disk that has aboot partition is called a **boot disk** or **system disk.**

The Windows 2000 system places its boot code in the first sector on the hard disk (**masterbootrecord,** or MBR). The code directs the system to read the boot code from, the MBR. Inadditionto containing boot code, the MBR contains a table listing the partitions for the hard diskandaflag indicating which partition the system is to be booted from.



Partition Table

Bad Blocks

Disk are prone to failure of sectors due to the fast movement of r/w head. Sometimes thewhole disk will be changed. Such group of sectors that are defective are called as bad blocks. Different ways to overcome bad blocks are -

—Some bad blocks are handled manually, eg. In MS-DOS.

—Some controllers replace each bad sector logically with one of the spare sectors(extrasectors). The schemes used are **sector sparing** or **forwarding** and **sector slipping**.

In MS-DOS format command, scans the disk to find bad blocks. If format finds a bad block, it writes a special value into the corresponding FAT entry to tell the allocation routines not tousethat block.
In SCSI disks , bad blocks are found during the low-level formatting at the factory and is updatedover the life of the disk. Low-level formatting also sets aside spare sectors not visible totheoperating system. The controller can be told to replace each bad sector logically with one of thespare sectors. This scheme is known as **sector sparing** or **forwarding.**

A typical bad-sector transaction might be as follows:

- The operating system tries to read logical block 87.
- The controller finds that the sector is bad. It reports this finding to the operating system. • The next time the system is rebooted, a special, command is run to tell the SCSI controller to replace the bad sector with a spare.
- After that, whenever the system requests logical block 87, the request is translatedintothe replacement sector's (spare) address by the controller.

Some controllers replace bad blocks by **sector slipping**. Here is an example: Supposethat logical block 17 becomes defective and the first available spare follows sector 202. Then, sectorslipping remaps all the sectors from 17 to 202, moving them all down one spot. That is, sector202 is copied into the spare, then sector 201 into 202, and then 200 into 201, and so on, until

sector 18 is copied into sector 19. Slipping the sectors in this way frees up the space of sector 18, so sector 17 can be mapped to it.

SWAP SPACE MANAGEMENT

The amount of swap space needed on a system can vary depending on the amount ofphysical 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 overestimated or underestimated. It is safer to overestimate thantounderestimate the amount of swap space required. If a system runs out of swap space duetounderestimation of space, it may be forced to abort processes or may crash entirely. Overestimation wastes disk space that could otherwise be used for files, but it does nootherharm.

Swap-Space Location

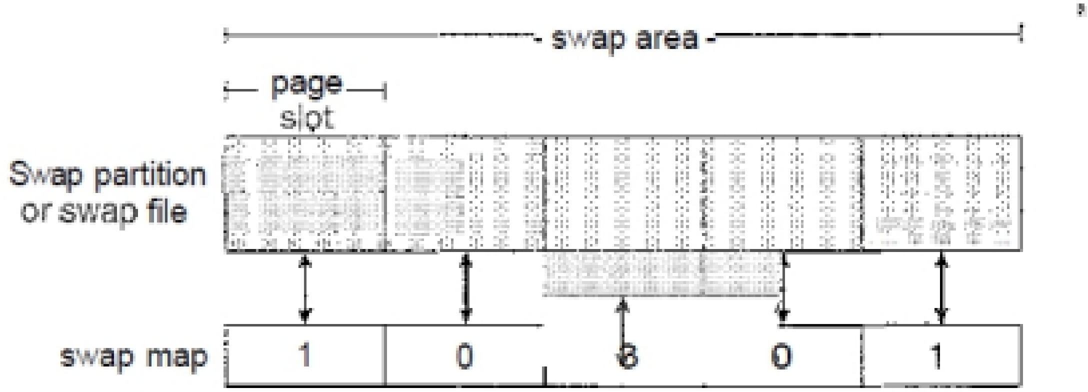
A swap space can reside in one of two places: It can be carved out of the normal **file system**, or it can be in a separate **disk partition**. If the swap space is simply a largefilewithin the file system, normal file-system routines can be used to create it, name it, andallocateits space. External fragmentation can greatly increase swapping times by forcing multipleseeksduring reading or writing of a process image. We can improve performance by caching theblocklocation information in physical memory.

Alternatively, swap space can be created in a separate raw partition. Aseparate swap-space storage manager is used to allocate and deallocate the blocks from the rawpartition.

Swap-Space Management: An Example

Solaris allocates swap space only when a page is forced out of physical memory, ratherthan when the virtual memory page is first created.

Linux is similar to Solaris in that swap space is only used for anonymous memoryor forregions of memory shared by several processes. Linux allows one or more swap areas tobe



established. A swap area may be in either a swap file on a regular file systemor a rawswappartition. Each swap area consists of a series of 4-KB page slots, which are used to hold swappedpages. Associated with each swap area is a swap map—an array of integer counters, eachcorresponding to a page slot in the swap area. If the value of a counter is 0, the correspondingpage slot is available. Values greater than 0 indicate that the page slot is occupied by a swappedpage. The value of the counter indicates the number of mappings to the swapped page; forexample, a value of 3 indicates that the swapped page is mapped to three different processes. Thedata structures for swapping on Linux systems are shown in below figure.

PROTECTION:

GOALS OF PROTECTION

Protection is a mechanism for **controlling the access** of programs, processes, or userstothe resources defined by a computer system. Protection ensures that only processes that havegained proper authorization from the operating system can operate on the files, memorysegments, CPU, and other resources of a system.

Protection is required to prevent mischievous, intentional violation of anaccessrestriction 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 performtheir 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. O*bjects* are both **hardwareobjects** (such as the CPU, memory segments, printers, disks, and tape drives) and **softwareobjects** (such as files, programs, and semaphores). Each object (resource) has a unique namethat differentiates it from all other objects in the system.

The operations that are possible may depend on the object. For example, a CPUcanonlybe executed on. Memory segments can be read and written, whereas a CD-ROMor DVD-ROMcan 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, anddeleted.

A process should be allowed to access only those resources
a) for which it has authorization
b) currently requires to complete process

Domain Structure

A domain is a set of objects and types of access to these objects. Each domainisanordered pair of <object-name,rights-set>. Example, if domain D has the access right <*fileF,* {read,write}>, then all process executing in domain D can both read and write file *F*, andcannot perform any other operation on that object.

Domains do not need to be disjoint; they may share access rights. For example, inbelowfigure, we have three domains: D1 D2*,* and D3. The access right < *O4,* (print}> is sharedbyD2and *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 isageneral model of protection that provides a mechanism for protection without imposingaparticular protection policy. The rows of the access matrix represent domains, and the columnsrepresent 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*.

domain \ object	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. Aprocess executingin domain D4 has the same privileges as one executing in domain *D1;* but in addition, it canalsowrite onto files F1 and F3. When a user creates a new object Oj, the column Oj is added to the access matrix withtheappropriate initialization entries, as dictated by the creator.

The process executing in one domain and be **switched** to another domain. When we switchaprocess from one domain to another, we are executing an operation (**switch**) on an object (thedomain). Domain switching from domain Di to domain *Dj* is allowed if and only if the accessright switch € access(i,j). Thus, in the given figure, a process executing in domain D2 canswitchto 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 threeadditional 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 another is denoted by an asterisk (*) appendedtothe access right. The *copy* right allows the copying of the access right only within the column for whichthe right is defined. In the belowfigure, a process executing in domain*D2* can copy the read operationinto any entry associated with fileF2. Hence, the access matrix of figure (a) can be modified to the access matrix shown in figure (b).

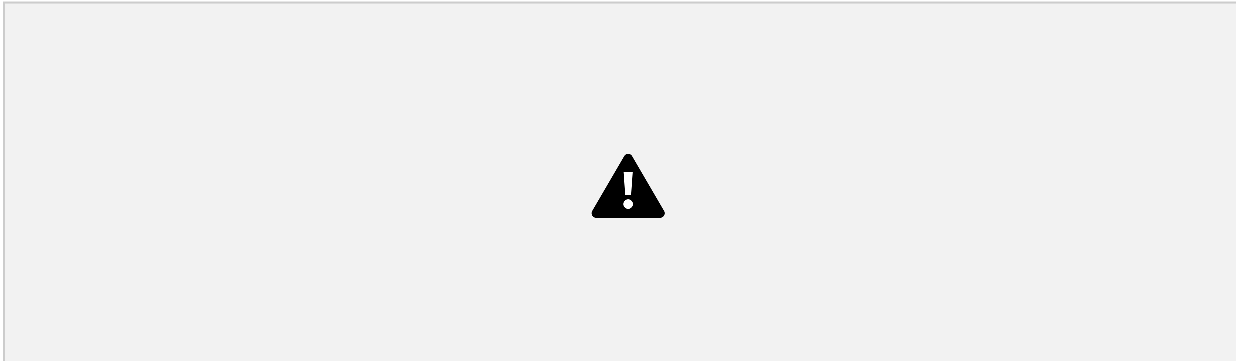
This scheme has two variants:

1) A right is copied from access(i,j) to access(k,j); it is then removed fromaccess(i,j). Thisaction is a *transfer* of a right, rather than a copy.
2) Propagation of the *copy* right- limited copy. Here, when the right *R** is copiedfromaccess(i,j) to access(k,j), only the right *R* (not R**)* is created. A process executingindomain *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 processexecuting in domain Di, can add and remove any right in any entry in column j. For example, inbelow figure (a), domain D1 is the owner of F1, and thus can add and delete any validright incolumn F1. Similarly, domain *D2* is the owner of *F2* and F3 and thus can add and removeanyvalid right within these two columns. Thus, the access matrix of figure(a) can be modifiedtotheaccess matrix shown in figure(b) as follows.

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 fromrowj. Forexample, in figure, we include the *control* right in access(D3, D4). Then, a process executingindomain D3 can modify domain D4.

Control



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

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, theoperation 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

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 orderedpairs<*domain, rights-set*>. It defines all domains access right for that object. When an operationMisexecuted on object Oj in Di, search the access list for object Oj, look for an entry <*Di,* Rj >withM e Kj. 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.

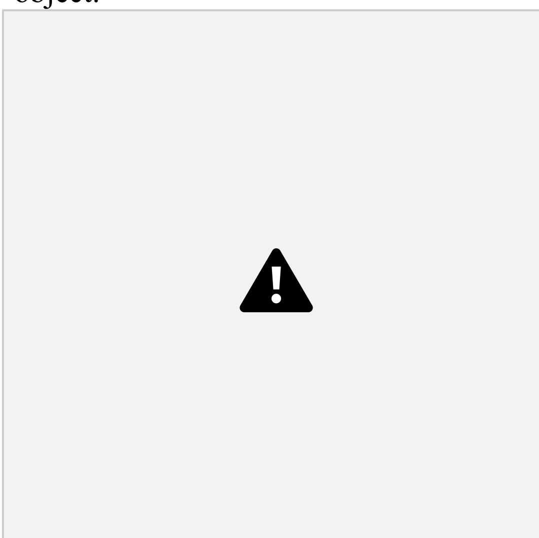
Capability Lists for Domains

A capability list for a domain is a list of objects together with the operations allowedonthose objects. An object is often represented by its name or address, called a **capability**. Toexecute operation M on object Oj, the process executes the operation *M,* specifying the capabilityfor object Oj as a parameter. Simple possession of the capability means that access is allowed.

Capabilities are usually distinguished from other data in one of two ways: Each object has a tag to denote its type either as a capability or as accessible data. Alternatively, the address space associated with a program can be split into two parts. One part isaccessible to the program and contains the program's normal data and instructions. Theotherpart, containing the capability list, is accessible only by the operating system.

A Lock-Key Mechanism

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



Access Control

Each file and directory are assignedan owner, a group, or possibly a list of users, andfor each of those entities, access-control informationis assigned.

Solaris 10 advances the protection available in the Sun Microsystems operating systemby explicitly adding the principle of least privilegevia **role-based access control (RBAC).** This facility revolves around privileges. A privilege is theright to execute a system call or to use an optionwithin that system call (such as opening a file withwrite access). Privileges can be assigned to processes, limiting them to exactly the access they needto

perform their work. Privileges and programs can also be assigned to **roles.** Users are assignedroles or can take roles based on passwords to the roles. In this way, a user can take a rolethat enables a privilege, allowing the user to run a program to accomplish a specific task, as depictedin Figure 14.8. This implementationof privileges decreases the security risk associatedwithsuperusers and setuid programs.

Revocation of Access Rights

Since the capabilities are distributed throughout the system, we must find thembeforewecan revoke them. Schemes that implement revocation for capabilities include the following: • **Reacquisition.** Periodically, all capabilities are deleted from each domain. If a process wants touse a capability, it may find that that capability has been deleted. The process may then trytoreacquire the capability. If access has been revoked, the process will not be able to reacquire thecapability.

• **Back-pointers.** A list of pointers is maintained with each object, pointing to all capabilities associated with that object. When revocation is required, we can follow these pointers, changingthe capabilities as necessary.

• **Indirection**. The capabilities point indirectly to the objects. Each capability points to a uniqueentry in a global table, which in turn points to the object. We implement revocation by searchingthe global table for the desired entry and deleting it. Then, when an access is attempted, the capability is found to point to an illegal table entry.

• **Keys**. A key is a unique bit pattern that can be associated with a capability. This key is definedwhen the capability is created, and it can be neither modified nor inspected by the process owning the capability. A master key is associated with each object; it can be defined or replacedwith the **set-key** operation. When a capability is created, the current value of the master keyis associated with the capability. When the capability is exercised, its key is compared with the master key. If the keys match, the operation is allowed to continue; otherwise, an exceptioncondition is raised.

In key-based schemes, the operations of defining keys, inserting theminto lists, anddeleting them from lists should not be available to all users.

CAPABILITY-BASED SYSTEM

Here, survey of two capability-based protection systems is done.

1) An Example: Hydra

Hydra is a capability-based protection system that provides considerableflexibility. A fixed set of possible access rights is known to and interpreted by the system. Theserights include such basic forms of access as the right to read, write, or execute a memorysegment. In addition, a user (of the protection system) can declare other rights.

Operations on objects are defined procedurally. The procedures that implement suchoperations are themselves a form of object, and they are accessed indirectly by capabilities. Thenames of user-defined procedures must be identified to the protection systemif it is to deal with

objects of the user defined type. When the definition of an object is made known to Hydra, thenames of operations on the type become auxiliary rights.

Hydra also provides rights amplification. This scheme allows a procedure to be certifiedas *trustworthy* to act on a formal parameter of a specified type on behalf of any process that holds a right to execute the procedure. The rights held by a trustworthy procedureareindependent of, and may exceed, the rights held by the calling process.

When a user passes an object as an argument to a procedure, we may need to ensurethat the procedure cannot modify the abject. We can implement this restriction readily by passinganaccess right that does not have the modification (write) right.

The procedure-call mechanism of Hydra was designed as a direct solution to the *problemof mutually suspicious subsystems.*

A Hydra subsystem is built on top of its protection kernel and may require protectionofits own components. A subsystem interacts with the kernel through calls on a set of kernel-defined primitives that define access rights to resources defined by the subsystem.

2) An Example: Cambridge CAP System A different approach to capability-based protection has been taken in the designof theCambridge CAP system. CAP's capability system is simpler and superficially less powerful thanthat of Hydra. It can be used to provide secure protection of user-defined objects. CAPhas twokinds of capabilities.

The ordinary kind is called a **data capability. It** can be used to provide access toobjects, but the only rights provided are the standard read, write, and execute of the individual storagesegments associated with the object.

The second kind of capability is the **software capability,** which is protected, but not interpreted, by the CAP microcode. It is interpreted by a *protected* (that is, a privileged)procedure, which may be written by an application programmer as part of a subsystem. Aparticular kind of rights amplification is associated with a protected procedure.