COURSE NAME: OPERATING SYSTEMS

COURSE CODE: BCS303

SEMESTER: 3

MODULE: 5

NUMBER OF HOURS: 10

CONTENTS:

- **Secondary Storage Structures**
  - Mass storage structures
  - Disk structure
  - Disk attachments
  - Disk scheduling
  - Disk management
  - swap space management
- **Protection**
  - Goals of protection
  - Principles of protection
  - Domain of protection
  - Access matrix
  - Access control
  - Revocation of access rights
  - Capability- Based systems

Design principles

Question Bank:

Process management

WEB RESOURCES:

Memory Management

https://www.geeksforgeeks.org/operating-systems/

Input/Output

https://www.tutorialspoint.com/operating_system/index.htm

MODULE 5

SECONDARY STORAGE STRUCTURES

OVERVIEW OF MASS-STORAGE STRUCTURE

Web link- https://youtu.be/ZjMwUhapSEM

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.

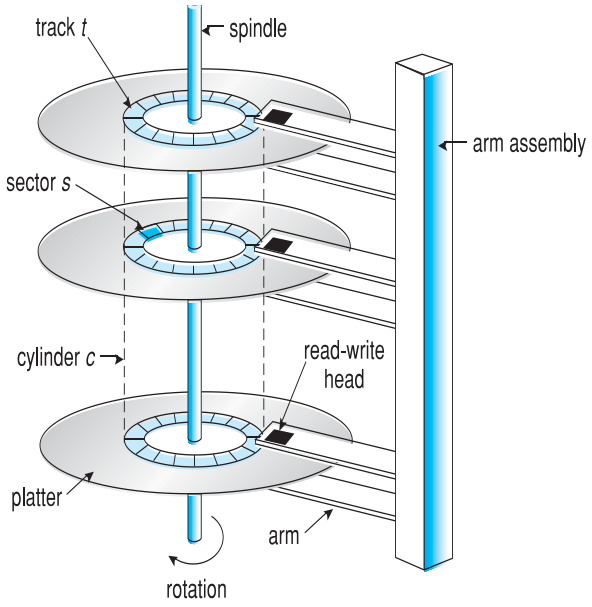


Figure: Moving-head disk mechanism

- 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.

- **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**.

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 n^{th} data has to be read, the tape starts moving from first and reaches the n^{th} position and then data is read from n^{th} position. It is not possible to directly move to the n^{th} 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

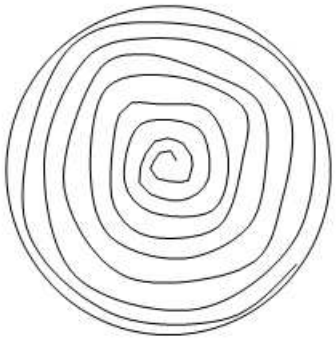
- Modern disk drives are addressed as a large one-dimensional array. The one-dimensional array of logical blocks is mapped onto the sectors of the disk sequentially.
- Sector 0 is the first sector of the first track on the outermost cylinder. The mapping proceeds in order through that track, then through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

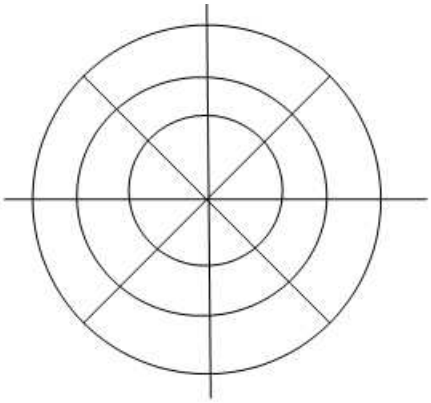
1. Constant Linear Velocity (CLV)
2. Constant Angular Velocity (CAV)

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.



CLV



CAV

DISK ATTACHMENT

Computers can access data in two ways.

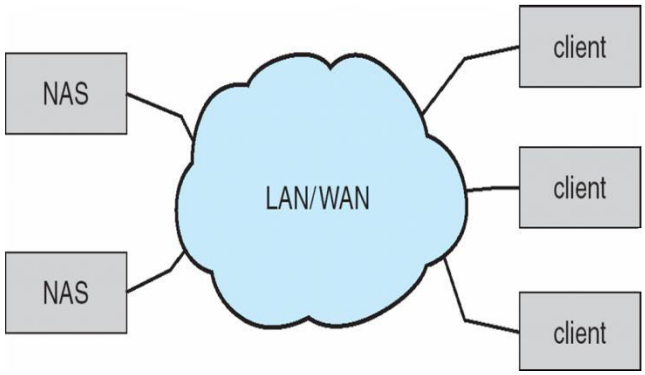
1. via I/O ports (or host-attached storage)
2. via a remote host in a distributed file system (or network-attached storage)

1. 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 supports a maximum of two drives per I/O bus.
- The other cabling systems are – SATA (Serially Attached Technology Attachment), SCSI (Small Computer System Interface) and fiber channel (FC).
- SCSI is a bus architecture. Its physical medium is usually a ribbon cable. FC is a high- speed serial architecture that can operate over optical fiber or over a four-conductor copper cable. An improved version of this architecture is the basis of storage-area networks (SANs).

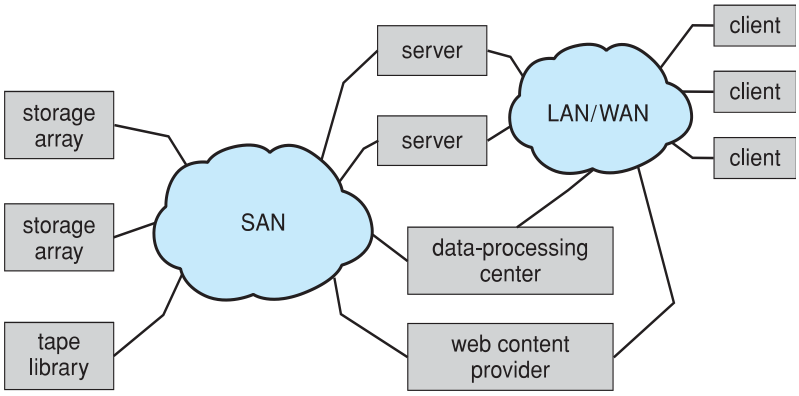
2. Network-Attached Storage

- A network-attached storage (NAS) device is a special-purpose storage system that is accessed remotely over a network as shown in the figure.
- Clients access network-attached storage via a remote-procedure-call interface. The remote procedure calls (RPCs) are carried via TCP or UDP over an IP network usually the same local-area network (LAN) carries all data traffic to the clients.
- Network- attached storage provides a convenient way for all the computers on a LAN to share a pool of storage files.



3. Storage Area Network (SAN)

- A storage-area network (SAN) is a private network connecting servers and storage units.
- The power of a SAN lies in its flexibility. Multiple hosts and multiple storage arrays can attach to the same SAN, and storage can be dynamically allocated to hosts.
- A SAN switch allows or prohibits access between the hosts and the storage. Fiber Chanel is the most common SAN interconnect.



DISK SCHEDULING

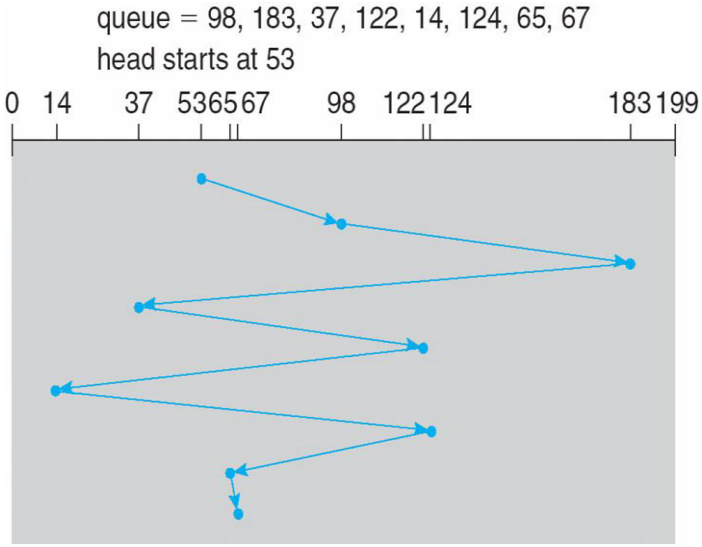
Different types of disk scheduling algorithms are as follows:

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



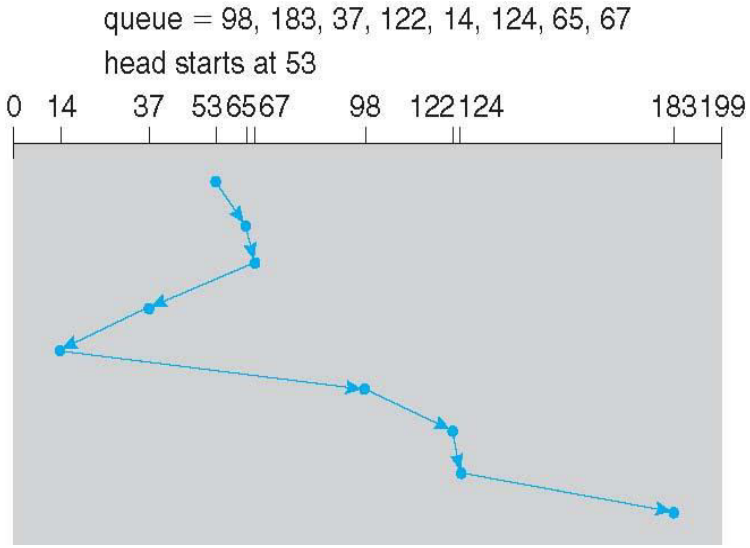
   If the disk head is initially at 53, it will first move from 53 to 98 then to 183 and then to 37, 122, 14, 124, 65, 67 for a total head movement of 640 cylinders. The wild swing from 122 to 14 and then back to 124 illustrates the problem with this schedule.

Web link: https://youtu.be/hSaPhBtU_BA

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.

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



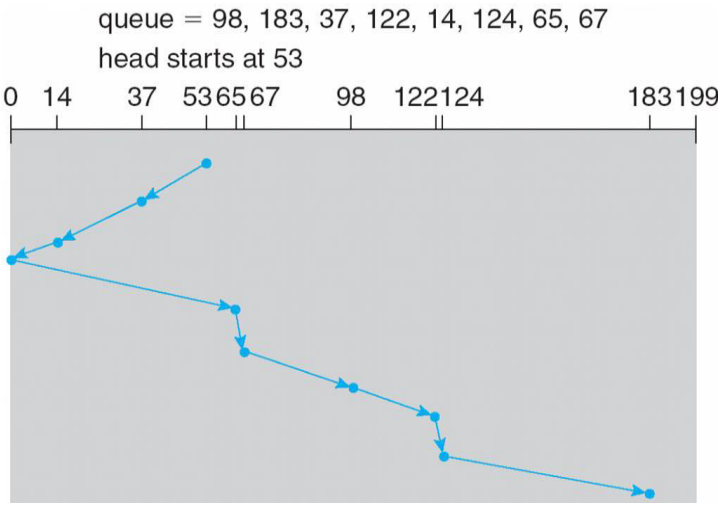
If the disk head is initially at 53, the closest is at cylinder 65, then 67, then 37 is closer than 98 to 67. So it services 37, continuing we service 14, 98, 122, 124 and finally 183. The total head movement is only 236 cylinders. SSTF is a substantial improvement over FCFS, it is not optimal.

Web link: https://youtu.be/DOnrmOGzooA

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 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 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

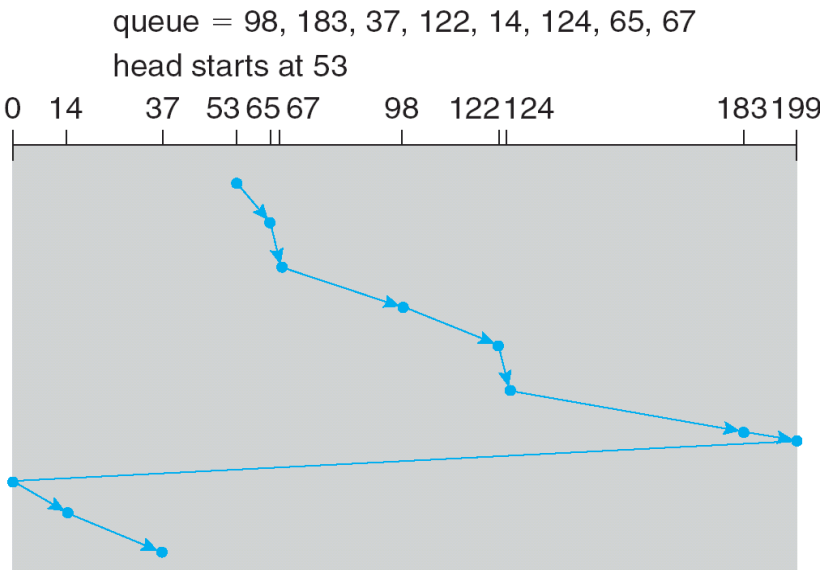
Web link: https://youtu.be/lQAYseBJPuA

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.

Web link: https://youtu.be/EgUctpbHUsQ

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

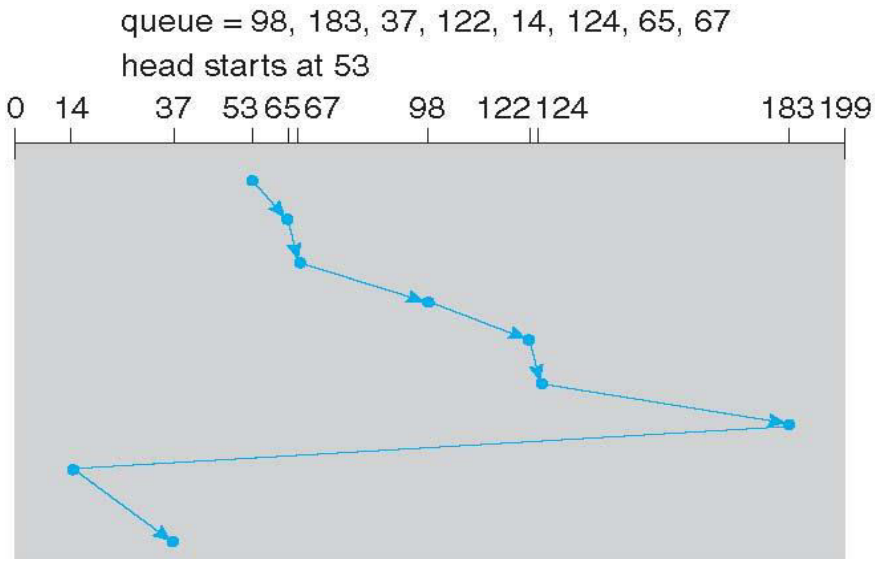


Figure: C-LOOK disk scheduling.

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.

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.
- SSTF or Look is a reasonable choice for a default algorithm.
- Selection of disk scheduling algorithm is influenced by the file allocation method, if contiguous file allocation is chosen, then FCFS is best suitable, because the files are stored in contiguous blocks and there will be limited head movements required.
- A linked or indexed file 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 opened to be used, and opening a file requires searching the directory structure, the directories will be accessed frequently.
- Suppose that a directory entry is on the first cylinder and a file's data are on the final cylinder. The disk head has to move the entire width of the disk. If the directory entry were on the middle cylinder, the head would have to move, at most, one-half the width. Caching the directories and index blocks in main memory can also help to reduce the disk-arm movement, particularly for read requests.

DISK MANAGEMENT

Disk Formatting

- The process of dividing the disk into sectors and filling the disk with a special data structure is called low-level formatting. Sector is the smallest unit of area that is read / written by the disk controller. The data structure for a sector typically consists of a header, a data area (usually 512 bytes 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 value calculated from all the bytes in the data area. When a sector is read, the ECC is recalculated and is compared with the stored value. If the stored and calculated numbers are different, this mismatch indicates that the data area of the sector has become corrupted and that the disk sector may be bad.
- Most hard disks are low-level- formatted at the factory as a part of the manufacturing process. This formatting enables the manufacturer to test the disk and to initialize the mapping from logical 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 data block of all sector sit can also be told how many bytes of data space to leave between the header and trailer of all sectors. It is of sizes, such as 256, 512, and 1,024 bytes. Formatting a disk with a larger sector size means that fewer sectors can fit on each track; but it also means that fewer headers 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 two steps i.e., Partition and logical formatting.

1. **Partition** – is to partition the disk into one or more groups of cylinders. The operating system can treat each partition as though it were a separate disk. For instance, one partition can hold a copy of the operating system's executable code, while another holds user files.
2. **Logical formatting (or creation of a file system)** - Now, the operating system stores the initial file-system data structures onto the disk. These data structures may include maps of free and allocated space (a FAT or modes) and an initial empty directory.

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

Boot Block

When a computer is switched on or rebooted, it must have an initial program to run. This is called the bootstrap program.
The bootstrap program –

- Initializes the CPU registers, device controllers, main memory, and then starts the operating system.
- Locates and loads the operating system 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 be infected by a computer virus. The problem is that changing this bootstrap code requires changing the ROM, hardware chips. So most systems store a tiny bootstrap loader program in the boot ROM whose only job is to bring in a full bootstrap program from disk. The full bootstrap program can be changed easily: A new version is simply written onto the disk. The full bootstrap program is stored in "the boot blocks" at a fixed location on the disk. A disk that has a boot partition is called a boot disk or system disk.

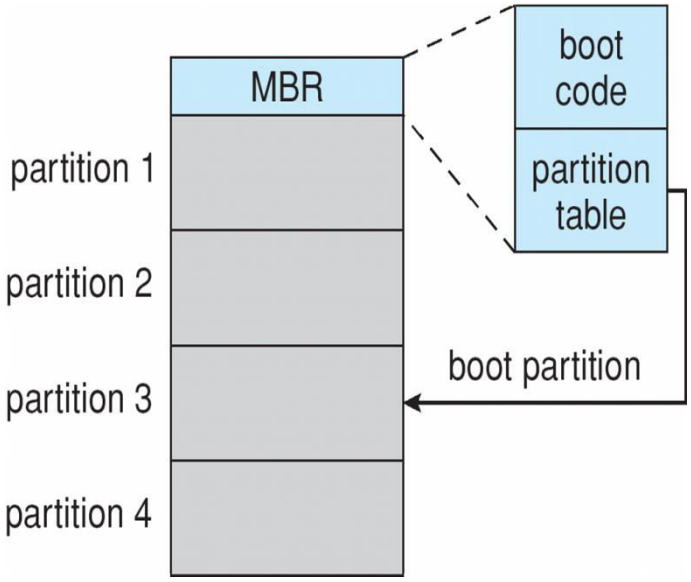


Figure: Booting from disk in Windows 2000.

The Windows 2000 system places its boot code in the first sector on the hard disk (master boot record, or MBR). The code directs the system to read the boot code from, the MBR. In addition to containing boot code, the MBR contains a table listing the partitions for the hard disk and a flag indicating which partition the system is to be booted from.

Bad Blocks

Disks are prone to failure of sectors due to the fast movement of r/w head. Sometimes the whole 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 (extra sectors). 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 to use that block.
In SCSI disks, bad blocks are found during the low-level formatting at the factory and is updated over the life of the disk. Low-level formatting also sets aside spare sectors not visible to the operating system. The controller can be told to replace each bad sector logically with one of the spare 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 translated into the replacement sector's (spare) address by the controller.

Some controllers replace bad blocks by sector slipping.

Example: Suppose that logical block 17 becomes defective and the first available spare follows sector 202. Then, sector slipping remaps all the sectors from 17 to 202, moving them all down one spot. That is, sector 202 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

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

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, but it does no other harm.

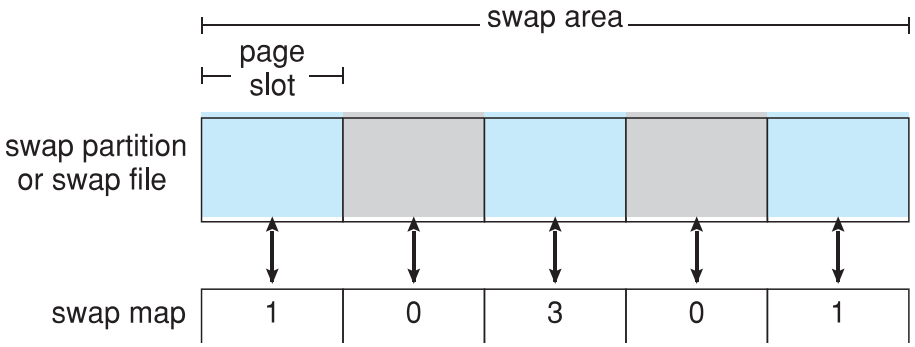
Swap-Space Location

- 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.

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.
- The data structures for swapping on Linux systems are shown in below figure.



PROTECTION

Web Link: https://youtu.be/O_WbprDZMDw

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.

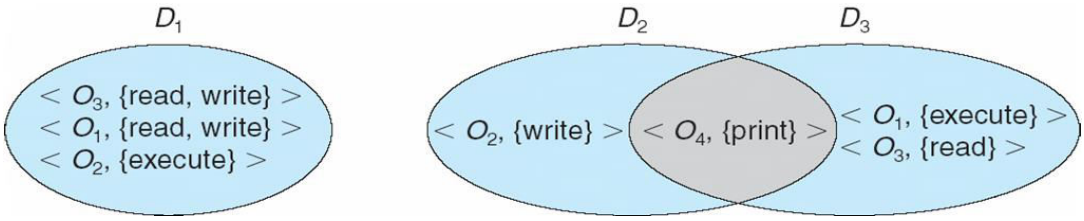
DOMAIN OF PROTECTION

- 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: D_1 D_2, and D_3. The access right < O_4, (print}> is shared by D_2 and D_3,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 with the 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 D_i to domain D_j is allowed if and only if the access right switch access(i,j). Thus, in the given figure, a process executing in domain D_2 can switch to domain D_3 or to domain D_4. A process in domain D_4 can switch to D_1, and one in domain D_1 can switch to domain D_2.

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 another is 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 D_2 can copy the read operation into any entry associated with file F_2. 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 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 D_k 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.

For example, 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 D_i, can remove any access right from row j. For example, in figure, we include the control right in access(D_3, D_4). Then, a process executing in domain D_3 can modify domain D_4.

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 O_j, within domain D_i, the table is searched for a triple $<D_i, O_j, R_k>$. 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 O_j in D_i, search the access list for object O_j, look for an entry $<D_i, R_K>$ with $M \in R_k$. 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 O_j, the process executes the operation M, specifying the capability for object O_j as a parameter. Simple possession of the capability means that access is allowed.

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

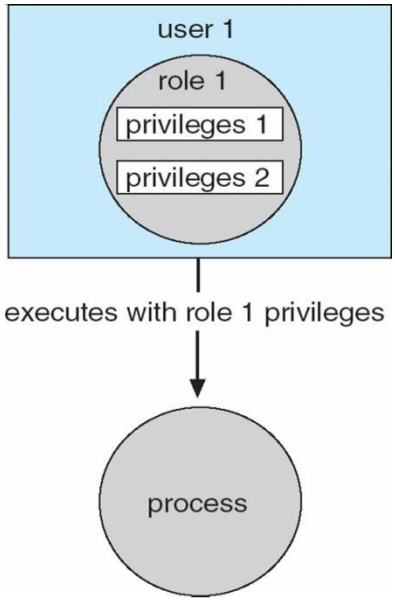
1. Each object has a tag to denote its type either as a capability or as accessible data.
2. The address space associated with a program can be split into two parts. One part is accessible to the program and contains the program's normal data and instructions. The other part, containing the capability 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 key that matches one of the locks of the object.

ACCESS CONTROL

- Each file and directory are assigned an owner, a group, or possibly a list of users, and for each of those entities, access-control information is assigned.
- Solaris 10 advances the protection available in the Sun Microsystems operating system by explicitly adding the principle of least privilege via role-based access control (RBAC). This facility revolves around privileges.
- A privilege is the right to execute a system call or to use an option within that system call.
- Privileges can be assigned to processes, limiting them to exactly the access they need to perform their work. Privileges and programs can also be assigned to roles.
- Users are assigned roles or can take roles based on passwords to the roles. In this way, a user can take a role that enables a privilege, allowing the user to run a program to accomplish a specific task, as depicted in below figure.
- This implementation of privileges decreases the security risk associated with super users and setuid programs.



REVOCATION OF ACCESS RIGHTS

The capabilities are distributed throughout the system, we must find them before we can revoke them. Schemes that implement revocation for capabilities include the following:

1. **Reacquisition** - Periodically, all capabilities are deleted from each domain. If a process wants to use a capability, it may find that that capability has been deleted. The process may then try to reacquire the capability. If access has been revoked, the process will not be able to reacquire the capability.
2. **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, changing the capabilities as necessary.
3. **Indirection** - The capabilities point indirectly to the objects. Each capability points to a unique entry in a global table, which in turn points to the object. We implement revocation by searching the 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.
4. **Keys** - A key is a unique bit pattern that can be associated with a capability. This key is defined when 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 replaced with the set-key operation.
   When a capability is created, the current value of the master key is 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 exception condition is raised.
   In key-based schemes, the operations of defining keys, inserting them into lists, and deleting 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 considerable flexibility. A fixed set of possible access rights is known to and interpreted by the system. These rights include such basic forms of access as the right to read, write, or execute a memory segment. In addition, a user (of the protection system) can declare other rights.
- Operations on objects are defined procedurally. The procedures that implement such operations are themselves a form of object, and they are accessed indirectly by capabilities. The names of user-defined procedures must be identified to the protection system if it is to deal with objects of the user defined type. When the definition of an object is made known to Hydra, the names of operations on the type become auxiliary rights.
- Hydra also provides rights amplification. This scheme allows a procedure to be certified as 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 procedure are independent 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 ensure that the procedure cannot modify the abject. We can implement this restriction readily by passing an access right that does not have the modification (write) right.
- The procedure-call mechanism of Hydra was designed as a direct solution to the problem of mutually suspicious subsystems.
- A Hydra subsystem is built on top of its protection kernel and may require protection of its 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 design of the Cambridge CAP system. CAP's capability system is simpler and superficially less powerful than that of Hydra. It can be used to provide secure protection of user-defined objects.

CAP has two kinds of capabilities.

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

2. 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. A particular kind of rights amplification is associated with a protected procedure.