MODULE 5

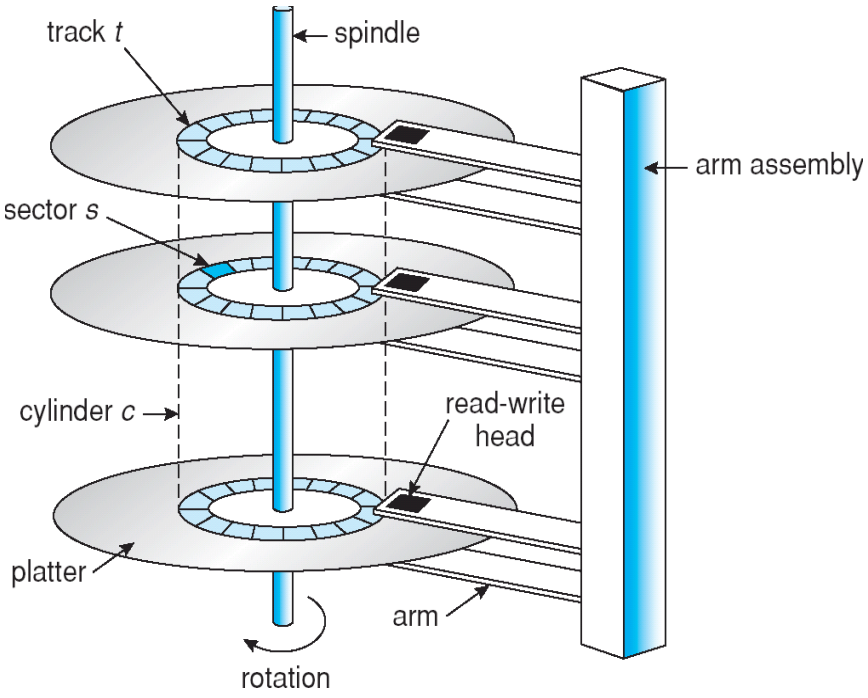
- ***Secondary Storage Structures, Protection:***
  - *Mass storage structures*
  - *Disk structure*
  - *Disk attachment*
  - *Disk scheduling*
  - *Disk management*
  - *Swap space management*
- ***Protection:***
  - *Goals of protection*
  - *Principles of protection*
  - *Domain of protection*
  - *Access matrix*
  - *Implementation of access matrix*
  - *Access control*
  - *Revocation of access rights*
  - *Capability- Based systems*
- ***Case Study: The Linux Operating System:***
  - *Linux history*
  - *Design principles*
  - *Kernel modules*
  - *Process management Scheduling*
  - *Memory Management*
  - *File systems*
  - *Input and output*
  - *Inter-process communication***.**

Secondary Storage Structure

➢ Overview of Mass Storage Structure

- **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 as shown in below **figure**. The two surfaces of a platter are covered with a magnetic material.
  - ✓ We store information by recording it magnetically on the platters. A Read-write head flies just above each surface of every platter. The heads are attached to a **disk arm** that moves all the heads as a unit.
  - ✓ Surface of a platter is logically divided into circular **tracks** which are subdivided into **sectors.**
  - ✓ When the disk is in use, a drive motor spins it at high speed. Most drivers rotate 60 to 200 times per second.
  - ✓ Disk **speed has two parts.** The **Transfer rate** is the rate at which data flow between the drive and the computer. The **Positioning time** (**random-access time**) consists of the time necessary to move the disk arm to the desired cylinder, called the **seek time** and the time necessary for the desired sector to rotate to the disk head, called the **rotational latency.**



  - ✓ A disk can be **removable,** allowing different disks to be mounted as needed. **Ex:** floppy disks.
  - ✓ A disk drive is attached to a computer by a set of wires called an **I/O bus.** Several kinds of buses are available, including Enhanced Integrated Drive Electronics (**EIDE),**

Advanced Technology Attachment (ATA), Serial ATA (SATA), Universal serial Bus(USB), Fiber Channel (FC) and SCSI buses.

- ✓ The data transfers on a bus are carried out by special electronic processors called **controllers**. The **host controller** is the controller at the computer end of the bus. A **disk controller** is built into each disk drive.
- ✓ To perform a disk I/0 operations, the computer places a command into the host controller using **memory-mapped I/0 ports**.

- **Magnetic Tapes**
  - ✓ Magnetic tape was used as an early secondary-storage medium. Its access time is slow when compared to main memory and magnetic disk.
  - ✓ Random access to magnetic tape is slower than disk so it is not very useful for secondary storage.
  - ✓ Tapes are used for backup and to store infrequently accessed data.

➢ Disk Structure

- ✓ Disk drives are addressed as large one-dimensional arrays of **logical blocks** where logical block is the smallest unit of transfer.
- ✓ Sector 0 is the first sector of the first track on the outermost cylinder.
- ✓ Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.
- ✓ By using this mapping, we can convert a logical block number into an old-style disk address that consists of a cylinder number, a track number within that cylinder, and a sector number within that track.
- ✓ It is difficult to perform this translation, **for two reasons**. First, most disks have some **defective sectors**; **second, the number of sectors** per track is **not constant** on some drives.
- ✓ On media that use **constant linear velocity (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.
- ✓ The disk rotation speed can stay constant that is the density of bits decreases from inner tracks to outer tracks to keep the data rate constant. This method is used in hard disks and is known as **constant angular velocity (CAV).**

➢ Disk Attachment

- ✓ Computers access disk storage in **two ways**
  - o Via I/O ports (host attached storage).
  - o via a remote host in a distributed file system (network attached storage)

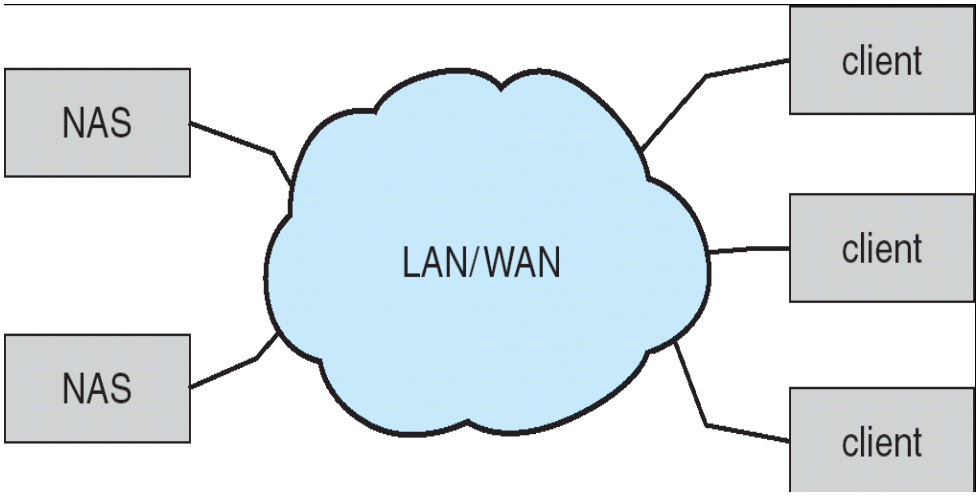
- **Host Attached Storage**

  - ✓ Host-attached storage is storage accessed through local I/O ports.
  - ✓ High-end workstations and servers use more sophisticated I/O architectures like SCSI and Fiber channel (FC).

- ✓ SCSI supports 16 devices on the bus.
- ✓ Fiber-channel is a high-speed serial architecture that can operate over optical fiber.
- ✓ A wide variety of storage devices are suitable for use as host-attached storage.
- ✓ Ex: RAID arrays, CD, DVD,etc

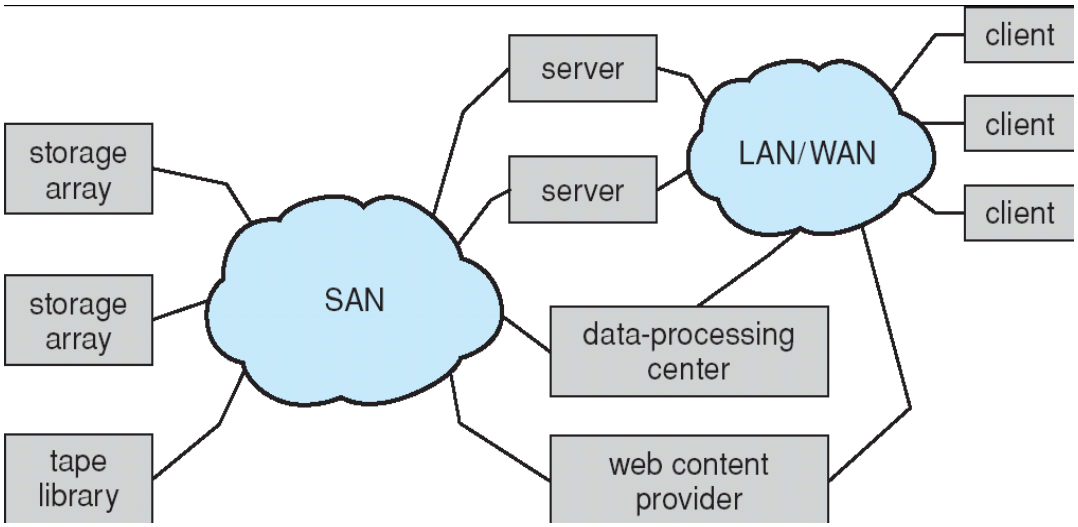
- **Network-Attached Storage**

  - ✓ A network-attached storage (NAS) device is a special-purpose storage system that is accessed remotely over a data network as shown in below **figure**.
  - ✓ Clients access network–attached storage via a remote-procedure-call interface such as NFS for UNIX and CIFS for Windows.
  - ✓ The remote procedure calls are carried via TCP or UDP over all IP network.
  - ✓ ISCSI is the latest network-attached storage protocol.
  - ✓ Drawback: Storage I/O operations consume bandwidth on the data network, thereby increasing the latency of network communication.



- **Storage-Area Network**

  - ✓ A storage-area network (SAN) is a private network connecting servers and storage units as shown in below **figure**.
  - ✓ 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.
  - ✓ SAN's have more ports and less expensive ports than storage arrays.
  - ✓ Fiber channel is used to interconnect multiple storage area networks.

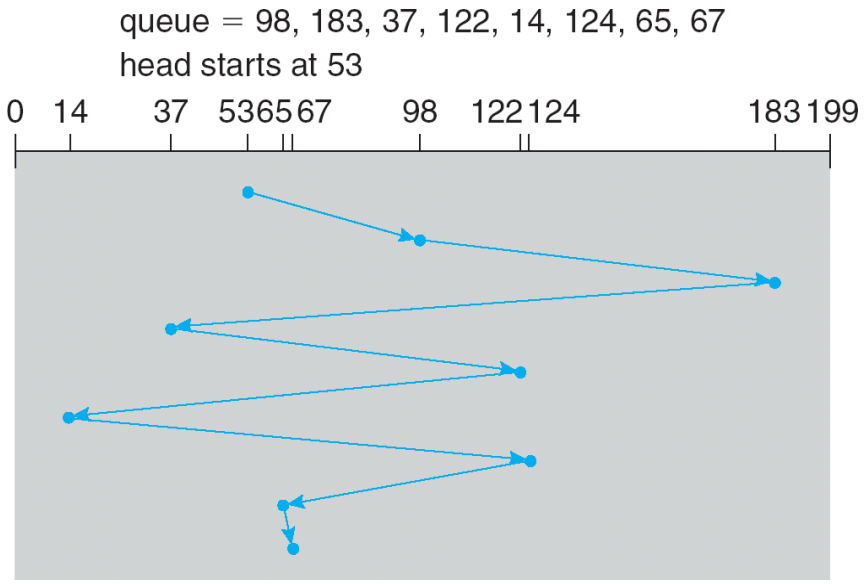


➢ Disk Scheduling

- ✓ Disk **access time has two major components.**
  - **Seek Time:** It is the time for the disk arm to move the heads to the cylinder containing the desired sector.
  - **Rotational Latency:** It is the additional time for the disk to rotate the desired sector to the disk head.
- ✓ **Disk Bandwidth:** It is the total number of bytes transferred, divided by the total time between the first request for service and the completion of last transfer.
- ✓ Whenever a process needs I/O from the disk, it issues a system call to the operating system.
- ✓ The request specifies several pieces of information,
  - Whether this operation is input or output
  - What the disk address for the transfer is
  - What the memory address for the transfer is
  - What the number of sectors to be transferred is
- ✓ If the desired disk drives available, the request can be serviced immediately. If the driver is busy, request will be placed in the queue. When one request is completed, OS chooses another pending request to service next. Several **disk scheduling** are used for this purpose.

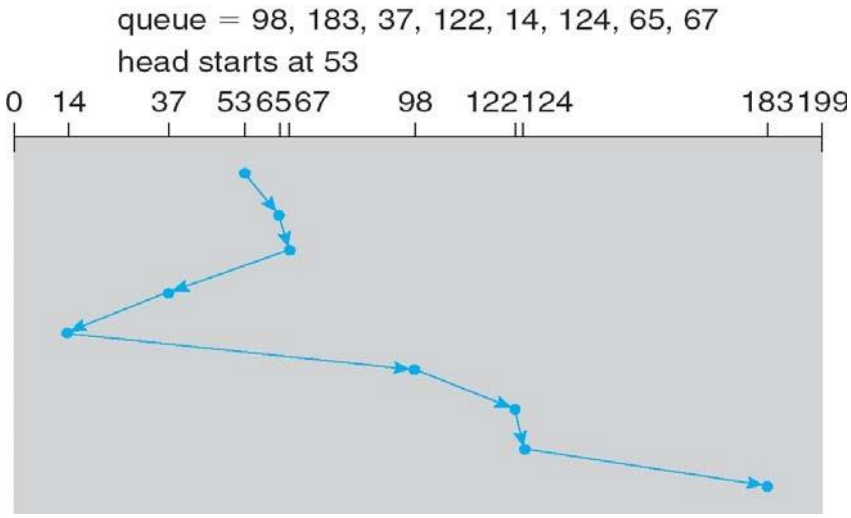
- **FCFS Scheduling**

  - ✓ Simplest form of disk scheduling.
  - ✓ Generally doesn't provide fastest service.
  - ✓ **For example:** A disk queue with requests for I/O to blocks on cylinders 98, 183, 37,122,14,124,65,67 and disk head is initially at cylinder 53.
  - ✓ It will first move from 53 to 98, then to 183,37,122,14,124,65 and 67 as shown in below **figure** for a total head movement of **640 cylinders**.



- **SSTF Scheduling (Short-seek-Time-First)**

  - ✓ SSTF assumes that it is better to service all the requests close to the current head position before moving the head for away to service other requests.
  - ✓ SSTF choose the pending request closest to the current head position.
  - ✓ For the queue 9, 7, 183, 37, 122, 14, 124, 65, 67 with head position = 53. Closest request to the initial head position is 65. Once we are at cylinder 65 next request served is 67 next is 37 This scheduling results in a total head movement of only 236 cylinders
  - ✓ SSTF may cause starvation of some process.

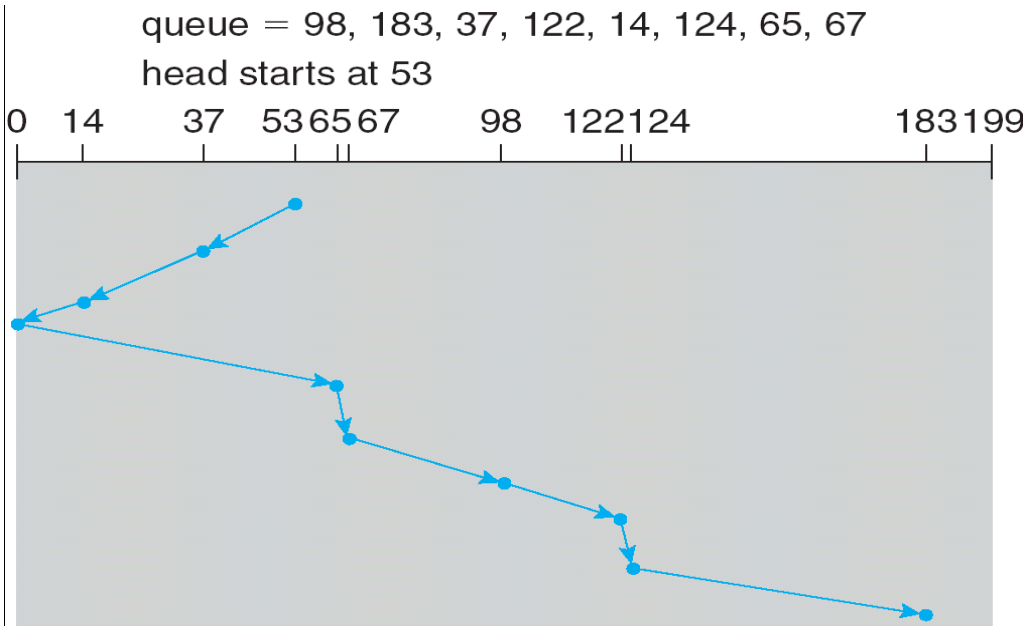


- **SCAN Scheduling (Elevator algorithm)**

  - ✓ In SCAN algorithm, the disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder, until it get to other end of the

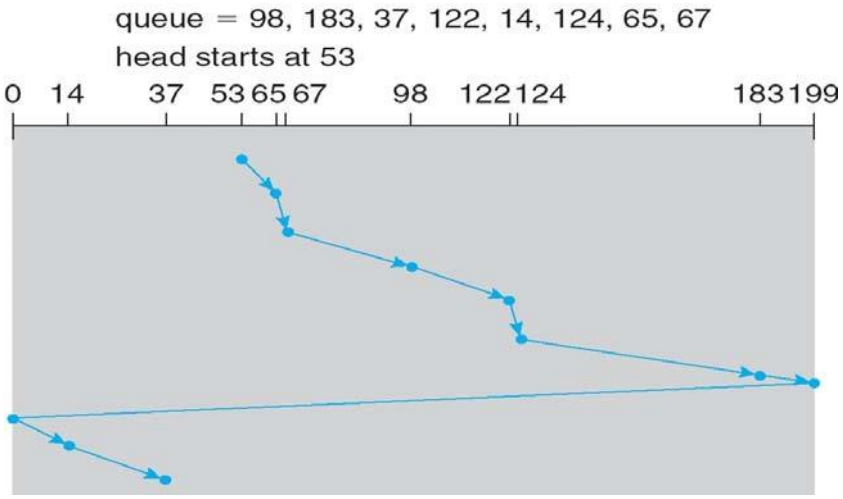
disk. At the other end, the direction of head movement is reversed and servicing continues.

- ✓ Head continuously scans back and forth across the disk.
- ✓ Consider requests 98, 183, 37, 122, 14, 124, 65 and 67, head position = 53
- ✓ For this algorithm we need to know the direction of head movement
- ✓ If the disk arm is moving towards 0, the head will service 37 and then 14.
- ✓ At cylinder 0, the arm will reverse and move towards the other end servicing the requests at 65, 67, 98, 122, 124 and 183 as shown in below **figure.**
- ✓ This scheduling method results in a total head movement of only **236 cylinders.**



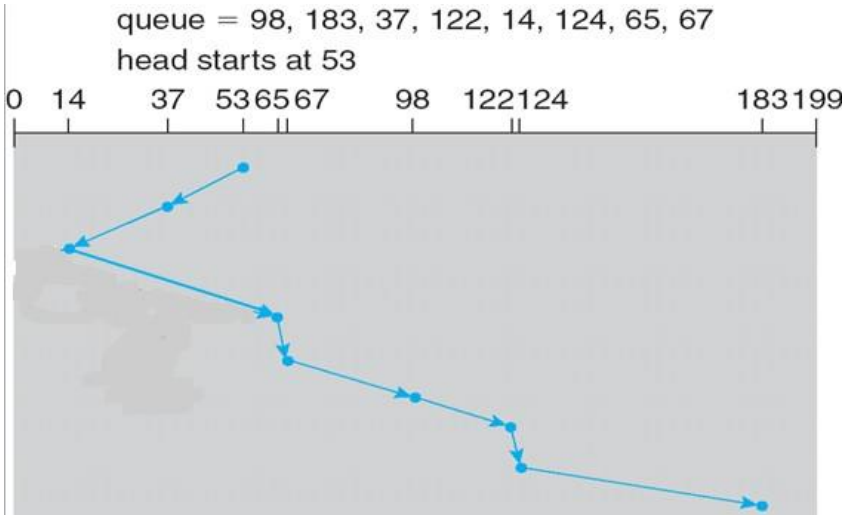
C-Scan Scheduling

- ✓ Circular SCAN (C-SCAN) Scheduling is a variant of SCAN designed to provide a more uniform wait time.
- ✓ C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, it immediately returns to the beginning of the disk, without servicing any requests on the return trip as shown in below **figure.**
- ✓ The C-SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.



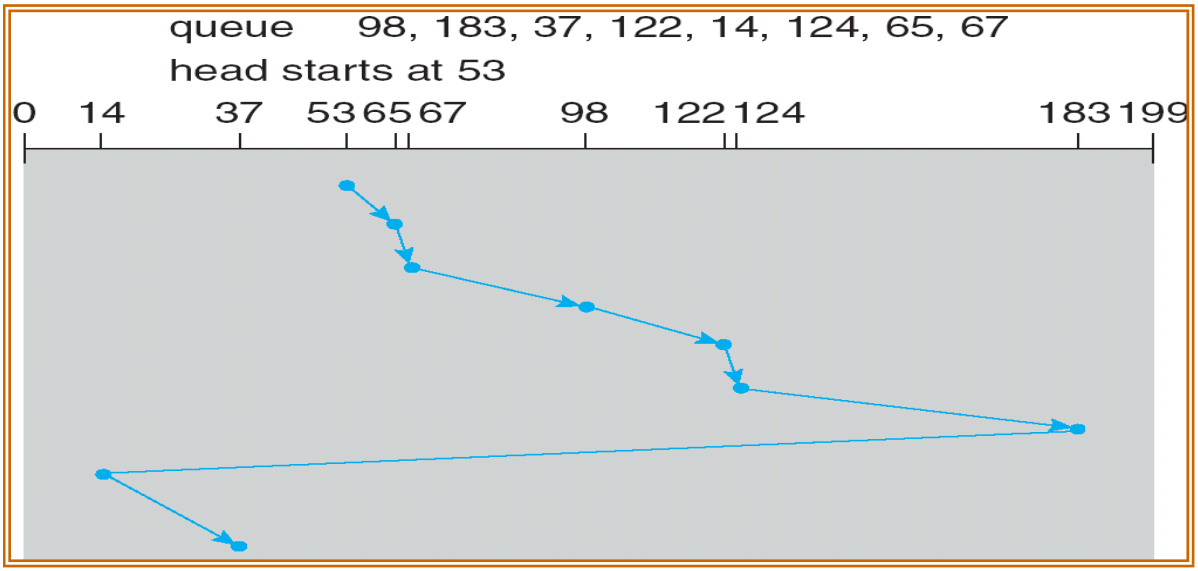
- **Look Scheduling**

  - ✓ In this disk arm does not move across the full width of the disk.
  - ✓ The arm goes only as far as the final request in each direction. Then it reverses direction immediately, without going all the way to the end of the disk.
  - ✓ Version of SCAN and C-SCAN that follows this pattern are called LOOK and C-LOOK Scheduling because they look for a request before continuing to move in a given direction as shown in below **figure.**



- **C-Look Scheduling**

  - ✓ Version of C-SCAN
  - ✓ Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.



- **Selection of Disk Scheduling Algorithm**

  - ✓ SSTF is a common scheduling algorithm because it increases performance over FCFS.
  - ✓ SCAN and C-SCAN used in the heavily loaded disk because these avoids starvation problem.
  - ✓ File-allocation method must be considered while selecting scheduling algorithm.
  - ✓ The location of directories and index blocks is also important.
  - ✓ Caching the directories and index blocks in main memory can also help to reduce disk-arm movement particularly for read requests.
  - ✓ Because of these complexities, the disk-scheduling algorithm should be written as a separate module of the operating system, so that it can be replaced with a different algorithm if necessary.

➢ Disk Management

OS is responsible for several disk management activities like,

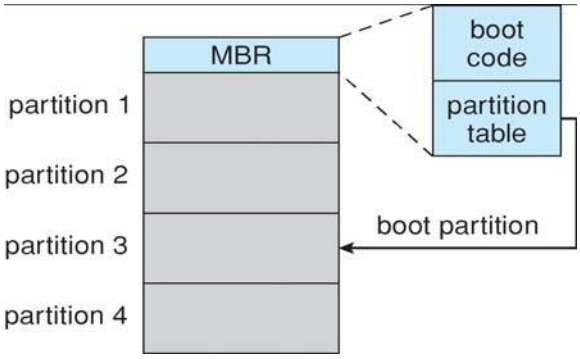
- **Disk Formatting**

  - ✓ A new magnetic disc is a blank slate, before a disk can store data, it must be divided into sectors that the disk controllers can read and write. This process is called **Low level formatting or Physical formatting.**
  - ✓ Low–level formatting fills the disk with a special data structure for each sector. It consists of a header, data area 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 the sector is read, the ECC

is recalculated and compared with the stored value. If it does not match then it is corrupted. Soft errors can be corrected by using ECC information

- ✓ To use a disk to hold files, the OS needs to record its own data structures on the disk. Two steps involved in this process are,
  - **Partition:** The disks are partitioned into one or more groups of cylinders.
  - **Logical formatting:** In this step, OS stores the initial file-system data structure onto the disk.
- ✓ To **increase efficiency,** most file systems group the blocks together into larger chunks called as **clusters.**

- **Boot Block**
  - ✓ Initial **bootstrap** program is required for a computer to start running. It initializes the system and then starts the OS.
  - ✓ Boot strap program finds the OS kernel on disk, loads that kernel into memory, and jumps to an initial address to begin the OS execution.
  - ✓ It is stored in **Read Only Memory (ROM)** but 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 job is to bring in a full bootstrap program from disk.
  - ✓ Full bootstrap program is stored in the **boot blocks** at a fixed location on the disk. A disk that has boot partition is called a **Boot disk** or **System disk.**
  - ✓ **For example**: The boot process in Windows 2000 system places its boot code in the **first sector** on the hard disk and it is termed as **master boot record (MBR).**
  - ✓ Windows 2000 allows a hard disk to be **divided into one or more partitions**; one partition is the **boot partition** and it contains the **operating system and device drivers.** The other partition contains a **table** listing the partitions for the hard disk and a **flag** indicating which partition the system is to be booted from, as shown in **figure.**
  - ✓ Once the system identifies the boot partition, it reads the first sector from that partition which is called the boot sector and continues with the remainder of the boot process, which includes loading the various subsystems and system services.



- **Bad Blocks**
  - ✓ Because disks have moving parts and small tolerances they are prone to failures. Failure may affect complete disk or it may affect one or two sectors. Most disks even come from factory with **bad blocks**.

- ✓ On simple disks **bad blocks are handled manually.** If blocks go bad during normal operation, a special program like **chkdsk** must be run manually to search for the bad blocks and to lock them. Data that resided on the bad blocks usually are lost.
- ✓ In sophisticated disks like **SCSI**, controller maintains a list of bad blocks on the disk. 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.**
- ✓ As an **alternative** to sector sparing some controllers can be instructed to replace a bad block by **sector slipping**. **For example,** suppose that logical block 17 becomes defective and the first available spare follows sector 202. Then, sector slipping remaps all sectors front 17 to 202, moving them all down one spot. That is, sector 202 is copied into spare, then sector 201 into 202, then 200 into 201 and so on.

➢ Swap - Space Management

- ✓ **Swap - Space Management** is another low-level task of the operating system. Virtual memory uses disk space as an extension of main memory.
- ✓ Since disk access is much slower than memory access, using **swap space** significantly decreases system performance. The main goal for the design and implementation of swap space is to provide the best throughput for the virtual memory system.

- **Swap - Space Use**

  - ✓ Amount of Swap Space needed on a system can vary depending on the amount of physical memory, amount of virtual memory, way in which virtual memory is used etc.
  - ✓ It is safer to over estimate than to underestimate the amount of Swap Space required, because if a system runs out of space it may abort processes. Overestimation wastes disk space but it will not cause any harm.

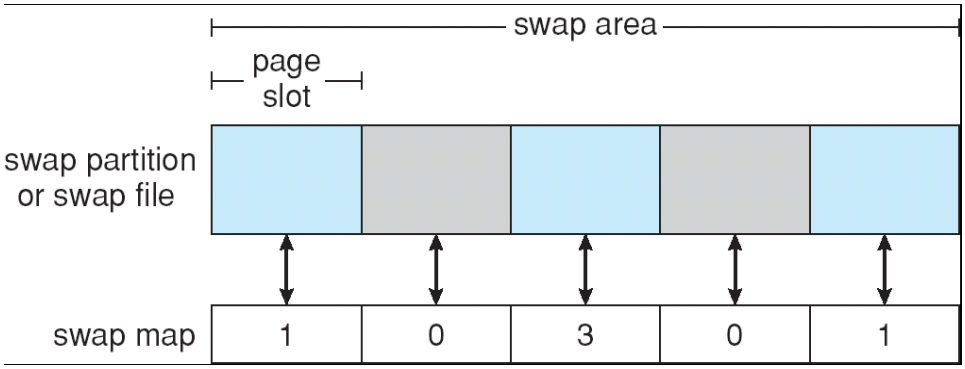
- **Swap - Space Location**

  - ✓ Swap Space can reside in one of **two places**.
    - o It can be simply a large file within the **file system** where normal file-system routines can be used to create it, name it and allocate its space. This approach is **easy to implement** but **inefficient**. Navigating the directory structure and the disk allocation data structures takes **time and extra disk accesses**.
    - o Swap space can be created in a separate **disk partition** (**raw partition).** A separate **swap space storage manager** is used to allocate and deallocate the blocks from the raw partition.
  - ✓ Some operating systems are flexible and can swap both in raw partitions and in file-system space.

- **Swap - Space Management: An Example**

  - ✓ In Solaris 1, when a process executer text - segment pages containing code are brought in form the file system, accessed in main memory, and thrown away if select for page out.

- ✓ It is more efficient to reread a page from the file system than to write it to swap space and then reread it from there.
- ✓ Swap Space is only used as a backing store for pages of **anonymous memory,** which includes memory allocated for the stack, heap and uninitialized data of a process.
- ✓ The biggest change is that Solaris now allocates swap space only when a page is forced out of physical memory, rather than when the virtual memory page is first created. This scheme gives better performance
- ✓ 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-space partition. Each swap area consists of a series of 4-KB **page slots** which are used to hold swapped pages.
- ✓ An array of integer counters, each corresponding to a page slot called **swap map** is associated with a 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 **figure 12.10.**



SYSTEM PROTECTION

- ✓ The processes in an operating system must be protected from one another's activities. To provide such protection, we can use various mechanisms to ensure 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 refers to a mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system.

➢ Goals of Protection

- ✓ Prevention of **mischievous, intentional violation** of an access restriction by a user.
- ✓ Ensures that each program component in a system uses system resources according to stated policies.
- ✓ Protection can improve **reliability** by detecting latent errors at the interfaces between component subsystems.
- ✓ A protection - oriented system provides means to distinguish between authorized and unauthorized usage.
- ✓ Role of **protection** in a computer system is to provide a **mechanism** for the enforcement of the **policies** governing resource use.

➢ Principles of Protection

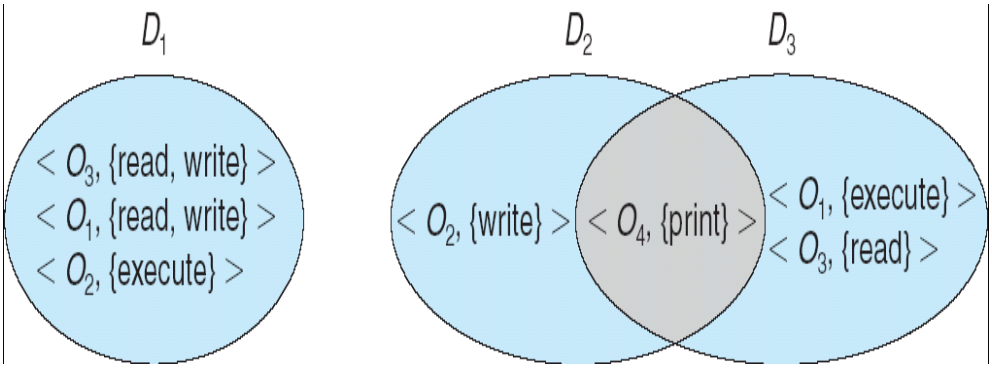
- ✓ Guiding principle for protection is the **Principle of Least Privilege**. It dictates that programs, users and even system be given just enough privileges to perform their tasks.
- ✓ Principle of least privilege implements programs, system calls in such a way that failure of a component does the minimum damage.
- ✓ It provides mechanisms to enable privileges when they are needed and to disable them when they are not needed.
- ✓ Managing users with the principle of least privilege entails creating a separate account for each user, with just the privileges that the user needs.
- ✓ Eg: Analogy of a Security Guard

➢ Domain of Protection

- ✓ A computer system is a collection of processes and objects such as **hardware objects** (the CPU, memory segments, printers, disks, and tape drives) and **software objects** (files, programs, and semaphores).
- ✓ The operations that are possible may depend on the object. A process should be allowed to access only those resources for which it has authorization.
- ✓ At anytime, a process should be able to access only those resources that it currently requires to complete its task. This is referred as **Need-to-Know** principle. It limits the amount of damage caused by faulty process.

- **Domain Structure**

- ✓ A process operates within a Protection Domain which specifies the resources that the process may access. Each domain defines a set of objects and the types of operations that may be invoked on each object.
- ✓ Ability to execute an operation on an object is an Access Right.
- ✓ A **domain** is a collection of access rights. It is denoted by **ordered pair - object- name, right-set** . **For example,** if domain D has the access right <file F, {read, write}>, then a process executing in domain D can both read and write file F; it cannot perform any other operation on that object.
- ✓ Domains do not need to be disjoint; they may **share access rights**. **For example**, in **figure,** we have **three domains: D_1, D_2, and D_3**. The access right < 0_4, {print}> is shared by D_2 and D_3, implying that a process executing in either of these two domains can print object 0_4.



- ✓ **Association** between a process and a domain may be **static or dynamic**. Dynamic association supports **domain switching** i.e., it enables the process to switch from one domain to another. A domain can be realized in a **variety of ways:**
  - Each **user** may be a domain. In this case the set of objects that can be accessed depends on the identity of the user. Domain switching occurs when one user logs out and another user logs in.
  - Each **process** may be a domain. Domain switching occurs when one process and then waits for a response.
  - Each **procedure** may be a domain. Domain switching occurs when a procedure call is made.

- **An example - UNIX**

  - ✓ In UNIX Operating system, domain is related with the user. S**witching the domain** corresponds to changing the user identification temporarily.
  - ✓ Owner **identification and a domain bit (known as the setuid bit)** are associated with each file. When the **setuid bit is on**, and a user executes that file, the user ID is set to that of the owner of the file; when the bit is **off** the user ID does not change.
  - ✓ **Other methods** are used to change domains in operating systems in which user IDs are used for domain definition, because almost all systems need to provide such a mechanism.

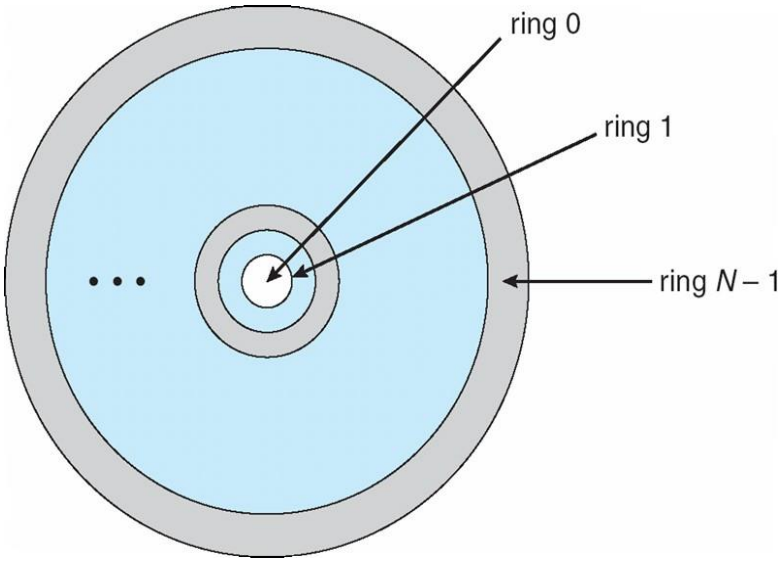
- ✓ An alternative to this method used in other operating systems is to place **privileged programs** in a special directory. The operating system would be designed to **change the user lD** of any program run from this directory, either to the equivalent of root or to the user lD of the owner of the directory.
- ✓ Even more restrictive, and thus more protective, are systems that simply do not allow a change of user ID. In these instances, special techniques must be used to allow users access to privileged facilities. For instance, a **daemon process** may be started at boot time and run as a special user ID.
- ✓ In any of these systems, great care must be taken in writing privileged programs.

- **An example – MULTICS**

  - ✓ Protections of domains are organized hierarchically into a **ring structure**. Rings are numbered from 0 to ring N-1. Each ring is a single domain as shown in **figure.**
  - ✓ Consider any two domain rings, i.e, D_i& D_j. If value of j is less than i (j<i), then domain D_i is subset of domain D_j. The process executing in domain D_j has more privileges than the process executing in domain D_i. **Ring 0 has full privileges.**
  - ✓ MULTICS has a **segmented address space**; each segment is a file, and each segment is associated with one of the rings. A segment description includes an entry that identifies the ring number and **three access bits** to control reading, writing, and execution.
  - ✓ A current-ring-number counter is associated with each process, identifying the ring in which the process is executing currently. When a process is executing in ring i, it cannot access a segment associated with ring j (j <i). It can access a segment associated with ring k (k>= i). The type of access is restricted according to the access bits associated with that segment.
  - ✓ **Domain switching** in MULTICS occurs when a process crosses from one ring to another by calling a procedure in a different ring. This switch must be done in a controlled manner; otherwise, a process could start executing in ring 0, and no protection would be provided.
  - ✓ To allow **controlled domain switching,** we modify the ring field of the segment descriptor to include the following:
    - o **Access bracket**. A pair of integers, bl and b2, such that bl <= b2.
    - o **Limit.** An integer b3 such that b3 >b2.
    - o **List of gates.** Identifies the **entry points** (or **gates)** at which the segments may be called.
  - ✓ If a process executing in ring i calls a procedure (or segment) with access bracket (bl,b2), then the call is allowed if **bl <= i <= b2,** and the current ring number of the process remains i. Otherwise, a **trap** to the operating system occurs, and the situation is **handled as follows**:
    - o If i <bl, then the call is allowed to occur, because we have a transfer to a ring (or domain) with fewer privileges. If parameters are passed that refer to segments in a lower ring then these segments must be copied into an area that can be accessed by the called procedure.
    - o If i >b2, then the call is allowed to occur only if b3 is greater than or equal to i and the call has been directed to one of the designated entry points in the list of gates.

This scheme allows processes with limited access rights to call procedures in lower rings that have more access rights, but only in a carefully controlled manner.

- ✓ The main **disadvantage** of the ring structure is that it does not allow us to enforce the **need-to-know principle.** The MULTICS protection system is generally more **complex and less efficient**.



➢ Access Matrix

- ✓ Our model of protection can be viewed abstractly as a matrix, called an **access matrix.**
- ✓ 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 O_i can invoke on object O_j.
- ✓ **For Example,** consider the access matrix shown in below **figure.**

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	

- ✓ The access matrix consists of four domains, four objects, three files and one printer. The **summary of access matrix** is as follows:
  - Process in domain D_1 can read file F_1 and file F_3.
  - Process in domain D_2 can only use printer.
  - Process in domain D_3 can read file F_2 and execute file F_3.
  - Process in domain D_4 can read and write file F_1 and file F_3.
- ✓ Access matrix scheme provides us with the mechanism for specifying a variety of policies. Mechanism consists of implementing the access matrix. Policy decisions involve which rights should be included in the $(i, j)^{th}$ entry. We must also decide the domain in which each process executes.
- ✓ When a user creates a new object O_j, the column O_j , is added to the access matrix. Blank entries indicate no access rights. A process is switched from one domain to another domain by executing **switch operation** on the object.
- ✓ Each entry in the access matrix may be modified individually. Domain switch is only possible if and only if the access right **switch € access (i, j)**. The below **figure** shows the access matrix with domains as objects. Process can change domain as follows,
  - Process in domain D_2 can switch to domain D_3 and domain D_4.
  - Process in domain D_4 can switch to domain D_1.
  - Process in domain D_1 can switch to domain D_2.
- ✓ Access matrix is inefficient for storage of access rights in computer system because they tend to be large and sparse.

Figure 17.4

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
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.**
- ✓ 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 access right to be copied only within the column for which the right is defined.
- ✓ **For example,** as shown in below **figure 17.5(a)**, 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 17.5(a) can be modified to the access matrix shown in **figure 17.5(b).** This scheme has **two variants:**

- 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.
- Propagation of the copy right may be limited. That is, 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.

✓ A system may select only one of these three copy rights, or it may provide all three by identifying them as separate rights: copy, transfer, and limited copy.

✓ 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,** as shown in below **figure 17.6(a)**, domain D_1 is the owner of F_1 and thus can add and delete any valid right in column F_1. Similarly, domain D_2 is the owner of $F_2$and $F_3$and thus can add and remove any valid right within these two columns. Thus, the access matrix of figure 17.6(a) can be modified to the access matrix as shown in **figure 17.6(b).**

Figure 17.5

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 copy and owner rights allow a process to **change the entries** in a column.

Figure 17.6

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)

- ✓ **control** is a mechanism to change the entries in a row. The **control** right is applicable only to domain objects. 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 17.4, we include the control right in access(D_2, D_4). Then, a process executing in domain D_2 could modify domain D_4, as shown in **figure 17.7.**

Figure 17.7

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			

➢ Implementing Access matrix

- ✓ Methods for implementing access matrix are,
  - o Global table.
  - o Access lists for objects.
  - o Capability.

- A lock key mechanism.

- **Global Table**

  - ✓ It is the simplest method for implementation of access matrix. Global table consists of domain, object and right set. The order of syntax is
    < domain, object, right-set >
  - ✓ If **operation P** is executed on an object O_j within domain D_j the global table is searched for a triple- < D_j, O_j, R_k> with M € R_k. If the above triple is found, then operation is allowed to continue. If suppose triple is not found then an exception error condition occurs.
  - ✓ **Limitations of Global table:** Global table is large and it cannot be kept in memory and additional Input/ Output required to bring the required data to memory.

- **Access list for objects**

  - ✓ Each column in the access matrix can be implemented as an access list for one object. The empty entries can be discarded.
  - ✓ The resulting list for each object consists of **ordered pairs <domain, rights-set>,** which define all domains with a nonempty set of access rights for that object.
  - ✓ This approach can be extended easily to define a list plus a default set of access rights. When an operation M on an object O_i is attempted in domain D_i, we search the access list for object 0_i, looking for an entry < D_i, R_k> with M € 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.

- **Capability list for domains**

  - ✓ Each row is associated with its domain.
  - ✓ 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 **physical name or address, called a Capability.**
  - ✓ Process executes operation M by specifying the capability (or pointer) for object O_j as a parameter.

  - ✓ Capabilities are **distinguished** from other data in **two ways**-
    - Each object has a **tag** to denote its type as either a capability or as accessible data.
    - The **address space** associated with a program can be split into two parts. One part is accessible to the program and contains the programs normal data and instructions. The other part containing the capability list is accessible only by the operating system.

- **A Lock –Key Mechanism**

  - ✓ The lock key scheme is a compromise between accesss list and capability list.

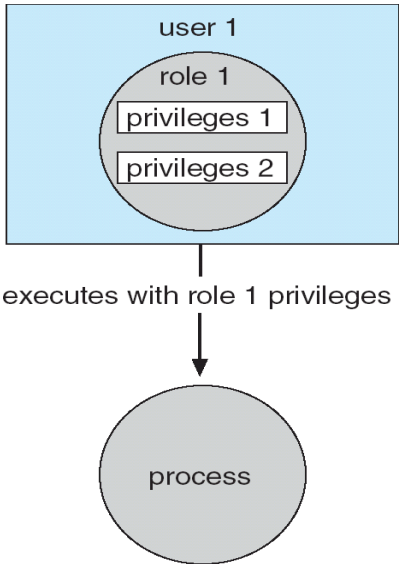
- ✓ Each object has a list of **unique bit** patterns called **locks** and each domain has a list of unique bit patterns called **keys**.
- ✓ A process executing in a domain can access an object only if the domain has a key that matches one of the locks of the object.
- ✓ Users are not allowed to examine or to modify the list of keys directly.

- **Comparison**

- ✓ Global table is simple but table can be quite large and cannot take advantage of special groupings of objects or domains.
- ✓ Access lists corresponds directly to the needs of users. But determining the set of access rights of a particular domain is difficult.
- ✓ Capability lists do not correspond directly to the needs of users. They are useful for localizing information for a given process.
- ✓ Lock-Key mechanism is a compromise between access lists and capability lists. The mechanism can be effective and flexible depending on the length of the keys.

➢ Access Control

- ✓ **Role-based Access control (RBAC)** 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 **process or 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 shown in the below **figure**.



➢ Revocation of Access Rights

- ✓ Revocation of access rights to objects in shared environment is possible. Following **parameters** are considered for revocation of access rights
  - **Immediate versus delayed.** Does revocation occur immediately, or is it delayed? If revocation is delayed, can we find out when it will take place?
  - **Selective versus general.** When an access right to an object is revoked, does it affect all the users who have an access right to that object, or can we specify a **select group of users whose access rights should be revoked?**
  - **Partial versus total.** Can a subset of the rights associated with an object be revoked, or must we revoke all access rights for this object?
  - **Temporary versus permanent.** Can access be revoked permanently (that is, the revoked access right will never again be available), or can access be revoked and later be obtained again?
- ✓ Revocation is **easy** for access list and complex **for capabilities list**. The access list is searched for any access rights to be revoked, and they are deleted from the list.
- ✓ **Schemes** that implement revocation for capabilities include the following:
  - **Reacquisition.** Periodically, 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.
  - **Back-pointers.** A list of pointers is maintained with each object, pointing to all capabilities associated with that object. When revocation is required, changing the capabilities as necessary. This scheme was adopted in the MULTICS system. It is quite general, but its implementation is costly.
  - **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. Table entries can be reused for other capabilities without difficulty, since both the capability and the table entry contain the unique name of the object. The object for a capability and its table entry must match. This scheme was adopted in the CAL system. It does not allow selective revocation.
  - **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 that owns 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. If we associate a list of keys with each object, then selective revocation can be implemented. Finally, we can group all keys into one **global table** of keys. A capability is valid only if its key matches some key in the global table. 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 Systems

Capability based protection systems are of two types,

- **An example - Hydra**
  - ✓ Hydra provides a fixed set of possible access rights that are 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.
  - ✓ Operations an object are defined procedurally. The procedures that implement such operations are themselves a form of object and they are accessed indirectly by capabilities. When the definition of an object is made known to hydra, the names of operations on the type become **auxiliary right.**
  - ✓ Hydra also provides **rights amplification.** This scheme allows certification of a procedure as trust worthy to act on an formal parameter of a specified type, on behalf of any process that holds a right to execute the procedure.
- **An example - Cambridge Cap System**
  - ✓ CAP system is simpler and superficially less powerful than that of hydra. CAP has two kinds of capabilities-
    - o **Data capability** can be used to provide access to objects, but the only rights provided are the standard read, write execute of the individual storage segments associated with the object. Data capabilities are interpreted by microcode in the CAP machine.

    Software capability is protected, but not interpreted by the CAP microcode. It is interpreted by a protected procedure, which may be written by an application programmer as part of a subsystem. The interpretation of a software capability is left completely to the subsystem, through the protected procedures it contains.

The Linux System

➢ Linux History

- ✓ Linux is a modern, free operating system based on UNIX standards.
- ✓ First developed as a small but self-contained kernel in 1991 by Linus Torvalds, with the major design goal of UNIX compatibility.
- ✓ Its history has been one of collaboration by many users from all around the world, corresponding almost exclusively over the Internet.
- ✓ It has been designed to run efficiently and reliably on common PC hardware, but also runs on a variety of other platforms.
- ✓ The core Linux operating system kernel is entirely original, but it can run much existing free UNIX software, resulting in an entire UNIX-compatible operating system free from proprietary code.
- ✓ A **linux distribution** includes all the standard components of the Linux system, plus *a* set of administrative tools to simplify the initial installation and subsequent upgrading of Linux and to manage installation and removal of other packages on the system.

▪ The Linux Kernel

- ✓ Version 0.01 (May 1991) had no networking, ran only on 80386-compatible Intel processors and on PC hardware, had extremely limited device-drive support, and supported only the Minix file system
- ✓ Linux 1.0 (March 1994) included these new features:
  - o Support for UNIX's standard TCP/IP networking protocols
  - o BSD-compatible socket interface for networking programming
  - o Device-driver support for running IP over an Ethernet
  - o Enhanced file system
  - o Support for a range of SCSI controllers for high-performance disk access
  - o Extra hardware support
- ✓ Version 1.2 (March 1995) was the final PC-only Linux kernel.
- ✓ The Linux kernel forms the core of the Linux project, but other components make up the complete Linux operating system.
- ✓ Linux uses many tools developed as part of Berkeley's BSD operating system, MIT's X Window System, and the Free Software Foundation's GNU project.
- ✓ The **GNU C compiler(gcc)**, were already of sufficiently high quality to be used directly in Linux.
- ✓ The networking administration tools under Linux were derived from code first developed for 4.3 BSD, but more recent BSD derivatives, such as FreeBSD, have borrowed code from Linux in return.
- ✓ Examples include the Intel floating-point-emulation math library and the PC sound-hardware device drivers.
- ✓ The Linux system as a whole is maintained by a loose network of developers collaborating over the Internet.

- ✓ The **file system hierarchy standard** document is also maintained by the Linux community as a means of ensuring compatibility across the various system components.

Released in June 1996, 2.0 added two major new capabilities:

- Support for multiple architectures, including a fully 64-bit native Alpha port
- Support for multiprocessor architectures

- ✓ Other new features included:
  - Improved memory-management code
  - Improved TCP/IP performance
  - Support for internal kernel threads, for handling dependencies between loadable modules, and for automatic loading of modules on demand
  - Standardized configuration interface
- ✓ 2.4 and 2.6 increased SMP support, added journaling file system, preemptive kernel, 64-bit memory support

- **The Linux System**
  - ✓ Linux uses many tools developed as part of Berkeley's BSD operating system, MIT's X Window System, and the Free Software Foundation's GNU project
  - ✓ The min system libraries were started by the GNU project, with improvements provided by the Linux community.
  - ✓ Linux networking-administration tools were derived from 4.3BSD code; recent BSD derivatives such as Free BSD have borrowed code from Linux in return.
  - ✓ The Linux system is maintained by a loose network of developers collaborating over the Internet, with a small number of public ftp sites acting as de facto standard repositories.

- **Linux Distributions**
  - ✓ Standard, precompiled sets of packages, or *distributions*, include the basic Linux system, system installation and management utilities, and ready-to-install packages of common UNIX tools.
  - ✓ The first distributions managed these packages by simply providing a means of unpacking all the files into the appropriate places; modern distributions include advanced package management.
  - ✓ Early distributions included SLS and Slackware.
  - ✓ *Red Hat* and *Debian* are popular distributions from commercial and noncommercial sources, respectively.
  - ✓ The RPM Package file format permits compatibility among the various Linux distributions.

- **Linux Licensing**
  - ✓ The Linux kernel is distributed under the GNU General Public License (GPL), the terms of which are set out by the Free Software Foundation.
  - ✓ Anyone using Linux, or creating their own derivative of Linux, may not make the derived product proprietary; software released under the GPL may not be redistributed as a binary-only product.

➢ Design Principles

- ✓ Linux resembles any other traditional, nonmicrokernel UNIX implementation. It is a multiuser, multitasking system with a full set of UNIX-compatible tools.
- ✓ Linux's file system adheres to traditional UNIX semantics, and the standard UNIX networking model is implemented fully.
- ✓ Linux can run happily on a multiprocessor machine with hundreds of megabytes of main memory and many gigabytes of disk space, but it is still capable of operating usefully in under 4 MB of RAM.
- ✓ Speed and efficiency are still important design goals, but much recent and current work on Linux has concentrated on a third major design goal: standardization.
- ✓ There are POSIX documents for common operating-system functionality and for extensions such as process threads and real-time operations.
- ✓ The Linux programming interface adheres to SVR4 UNIX semantics.
- ✓ A separate set of libraries is available to implement BSD semantics in places where the two behaviors differ significantly.
- ✓ Linux currently supports the POSIX threading extensions-Pthreads -and a subset of the POSIX extensions for real-time process control.

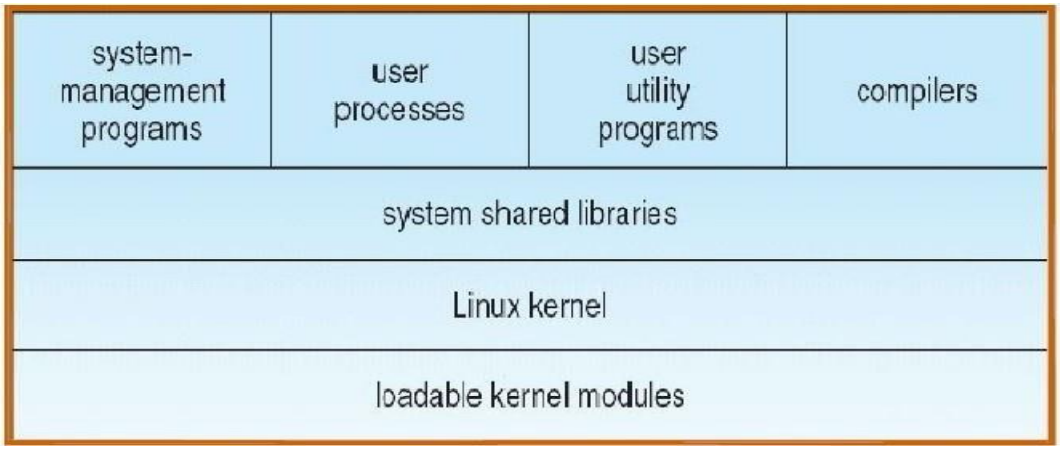
▪ Components of a Linux System

- ✓ The Linux system is composed of three main bodies of code, in line with mosttraditional UNIX implementations:
  - **Kernel-** The kernel is responsible for maintaining all the important abstractions of the operating system, including such things as virtual memory and processes.
  - **System libraries**- The system libraries define a standard set of functions through which applications can interact with the kernel. These functions implement much of the operating-system functionality that does not need the full privileges of kernel code.
  - **System utilities**-The system utilities are programs that perform individual, specialized management tasks. Some system utilities may be invoked just once to initialize and configure some aspect of the system; others known as *daemons* in UNIX terminology -may run permanently, handling such tasks as responding to incoming network connections, accepting logon requests from terminals, and updating log files.

All the kernel code executes in the processor's privileged mode with full access to all the physical resources of the computer.

- ✓ Linux refers to this privileged mode as **kernel mode.**
- ✓ The kernel is created as a single, monolithic binary inorder to improve performance.
- ✓ Because all kernel code and data structures are kept in a single address space, no context switches are necessary when a process calls an operating-system function or when a hardware interrupt is delivered.
- ✓ Not only the core scheduling and virtual memory code but *all* kernel code, including all device drivers, file systems, and networking code.

- ✓ The kernel does not necessarily need to know in advance which modules may be loaded-they are truly independent loadable components.
- ✓ The Linux kernel forms the core of the Linux operating system.



- ✓ The **system libraries,** allow applications to make kernel-system service requests.
- ✓ The libraries take care of collecting the system-call arguments and, if necessary, arranging those arguments in the special form necessary to make the system call.
- ✓ The libraries may also provide more complex versions of the basic system calls. The libraries also provide routines that do not correspond to system calls at all, such as sorting algorithms, mathematical functions, and string-manipulation routines.
- ✓ All the functions necessary to support the running of UNIX or POSIX applications are implemented here in the system libraries.
- ✓ The Linux system includes a wide variety of user-mode programs-both system utilities and user utilities.
- ✓ The system utilities include all the programs necessary to initialize the system, such as those to configure network devices and to load kernel modules.

➢ Kernel Modules

- ✓ Kernel modules are the sections of kernel code that can be compiled, loaded, and unloaded independent of the rest of the kernel.
- ✓ A kernel module may typically implement a device driver, a file system, or a networking protocol.
- ✓ The module interface allows third parties to write and distribute, on their own terms, device drivers or file systems that could not be distributed under the GPL
- ✓ Kernel modules allow a Linux system to be set up with a standard, minimal kernel, without any extra device drivers built in.
- ✓ Three components to Linux module support:
  - o module management
  - o driver registration
  - o conflict resolution

- **Module Management**

  - ✓ It supports loading modules into memory and letting them talk to the rest of the kernel.
  - ✓ Module loading is split into two separate sections:
    - Managing sections of module code in kernel memory
    - Handling symbols that modules are allowed to reference
  - ✓ The module requestor manages loading requested, but currently unloaded, modules; it also regularly queries the kernel to see whether a dynamically loaded module is still in use, and will unload it when it is no longer actively needed

- **Driver Registration**

  - ✓ Driver Registration allows modules to tell the rest of the kernel that a new driver has become available.
  - ✓ The kernel maintains dynamic tables of all known drivers, and provides a set of routines to allow drivers to be added to or removed from these tables at any time.
  - ✓ A module may register many types of drivers and may register more than one driver if it wishes.
  - ✓ Registration tables include the following items:
    - **Device drivers**- These drivers include character devices (such as printers/terminals/ and mice), block devices (including all disk drives) and network interface devices.
    - **File systems**- The file system may be anything that implements Linux's virtual-file-system calling routines. It might implement a format for storing files on a disk, but it might equally well be a network file system, such as NFS1 or a virtual file system whose contents are generated on demand/ such as Linux's /proc file system.
    - **Network protocols**- A module may implement an entire networking protocol such as IPX1 or simply a new set of packet-filtering rules for a network firewall.
    - **Binary format**- This format specifies a way of recognizing/ and loading/ a new type of executable file.

- **Conflict Resolution**

  - ✓ Conflict Resolution is a mechanism that allows different device drivers to reserve hardware resources and to protect those resources from accidental use by another driver
  - ✓ The conflict resolution module aims to:
    - Prevent modules from clashing over access to hardware resources
    - Prevent *autoprobes* from interfering with existing device drivers
    - Resolve conflicts with multiple drivers trying to access the same hardware

➢ Process management

- ✓ A process is the basic context within which all user-requested activity is serviced within the operating system.
- ✓ To be compatible with other UNIX systems, Linux must use a process model similar

to those of other versions of UNIX.
- ✓ The traditional UNIX process model.

- **The fork() and exec() Process Model**

  - ✓ The basic principle of UNIX process management is to separate two operations: the creation of a process and the running of a new program.
  - ✓ A new process is created by the fork() system call, and a new program is run after a call to exec().
  - ✓ A new process may be created with fork() without a new program being run-the new sub process simply continues to execute exactly the same program that the first (parent) process was running.
  - ✓ Running a new program does not require that a new process be created first: any process may call exec() at any time.
  - ✓ The currently running program is immediately terminated, and the new program starts executing in the context of the existing process.
  - ✓ This model has the advantage of great simplicity.
  - ✓ It is not necessary to specify every detail of the environment of a new program in the system call that runs that program; the new program simply runs in its existing environment.
  - ✓ If a parent process wishes to modify the environment in which a new program is to be run, it can fork and then, still running the original program in a child process, make any system calls it requires to modify that child process before finally executing the new program.

Process properties fall into three groups: the process identity, environment, and context.

- **Process Identity**

  - ✓ A process identity consists mainly of the following items:
    - **Process ID (PID)**- Each process has a Lmique identifier. The PID is used to specify the process to the operating system when an application makes a system call to signal, modify, or wait for the process. Additional identifiers associate the process with a process group (typically, a tree of processes forked by a single user command) and login session.
    - **Credentials**- Each process must have an associated user ID and one or more group IDs that determine the rights of a process to access system resources and files.
    - **Personality**- Process personalities are not traditionally found on UNIX systems, but under Linux each process has an associated personality identifier that can slightly modify the semantics of certain system calls. Personalities are primarily used by emulation libraries to request that system calls be compatible with certain varieties of UNIX.

- **Process Environment**

  - ✓ The process's environment is inherited from its parent, and is composed of two null-terminated vectors:
    - **The argument vector lists** the command-line arguments used to invoke the running program; conventionally starts with the name of the program itself
    - **The environment vector** is a list of "NAME=VALUE" pairs that associates named environment variables with arbitrary textual values
  - ✓ Passing environment variables among processes and inheriting variables by a process's children are flexible means of passing information to components of the user-mode system software
  - ✓ The environment-variable mechanism provides a customization of the operating system that can be set on a per-process basis, rather than being configured for the system as a whole

- **Process Context**

  - ✓ process context is the state of the running program at any one time; it changes constantly. Process context includes the following parts:
    - **Scheduling context** The most important part of the process context is its scheduling context-the information that the scheduler needs to suspend and restart the process. This information includes saved copies of all the process's registers.
    - **Accounting** The kernel maintains accounting information about the resources currently being consumed by each process and the total resources consumed by the process in its entire lifetime so far.
    - **File table** The file table is an array of pointers to kernel file structures. When making file-I/O system calls, processes refer to files by their index into this table.
    - **File-system context** File table lists the existing open files and the file-system context applies to requests to open new files. The current root and default directories to be used for new file searches are stored here.
    - **Signal-handler table** UNIX systems can deliver asynchronous signals to a process in response to various external events. The signal-handler table defines the routine in the process's address space to be called when a specific signals arrive.
    - **Virtual memory context** The virtual memory context describes the full contents of a process's private address space.

- **Processes and Threads**

  - ✓ Linux provides the fork() system call with the traditional functionality of duplicating a process.
  - ✓ Linux also provides the ability to create threads using the clone() system call.
  - ✓ However, Linux does not distinguish between processes and threads. Linux uses the term task.
  - ✓ When clone() is invoked, it is passed a set of flags that determine how much sharing is to take place between the parent and child tasks.
  - ✓ Some of these flags are:

- CLONE_FS - file system information is shared.
- CLONE_VM - the same memory space is shared.
- CLONE_SIGHAND - Signals handlers are shared.
- CLONE_FILES - the set of open files are shared.

✓ Thus, if clone() is passed the flags CLONE_FS, CLONE_VM, CLONE_SIGHAND, and CLONE_FILES, the parent and child tasks will share the same file-system information (such as the current working directory), the same memory space, the same signal handlers, and the same set of open files.

✓ If none of these flags is set when clone () is invoked, no sharing takes place, resulting in functionality similar to the fork() system call.

✓ The arguments to the clone () system call tell it which sub contexts to copy, and which to share, when it creates a new process.

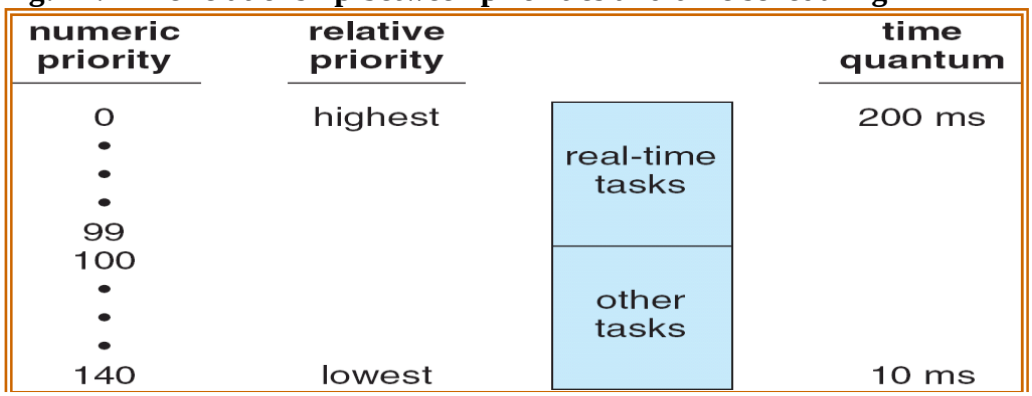
✓ The new process always is given a new identity and a new scheduling context according to the arguments passed.

➢ Scheduling

✓ Scheduling is the job of allocating CPU time to different tasks within an operating system.

- **Process Scheduling**
  - ✓ Linux has two separate process-scheduling algorithms.
  - ✓ One is a time-sharing algorithm for fair, preemptive scheduling among multiple processes; the other is designed for real-time tasks, where absolute priorities are more important than fairness.
  - ✓ Scheduler also provides increased support for SMP, including processor affinity and load balancing, as well as maintaining fairness and support for interactive tasks.
  - ✓ The Linux scheduler is a preemptive, priority-based algorithm with two separate priority ranges: a real-time range from 0 to 99 and a nice value ranging from 100 to 140.
  - ✓ These two ranges map into a global priority scheme whereby numerically lower values indicate higher priorities.
  - ✓ The Linux scheduler assigns higher-priority tasks longer time quanta and lower-priority tasks shorter time quanta and vice-versa.

Fig: 21.2 The relationship between priorities and time scheduling



- ✓ When a task has exhausted its time slice, it is considered expired and is not eligible for execution again until all other tasks have also exhausted their time quanta.
- ✓ The kernel maintains a list of all runnable tasks in a run queue data structure.
- ✓ Because of its support for SMP, each processor maintains its own run queue and schedules itself independently. Each run queue contains two priority arrays-active and expired.
- ✓ The active array contains all tasks with time remaining in their time slices, and the expired array contains all expired tasks. Each of these priority arrays includes a list of tasks indexed according to priority (Figure 21.3).
- ✓ When all tasks have exhausted their time slices (that is, the active array is empty), the two priority arrays are exchanged as the expired array becomes the active array and vice-versa.

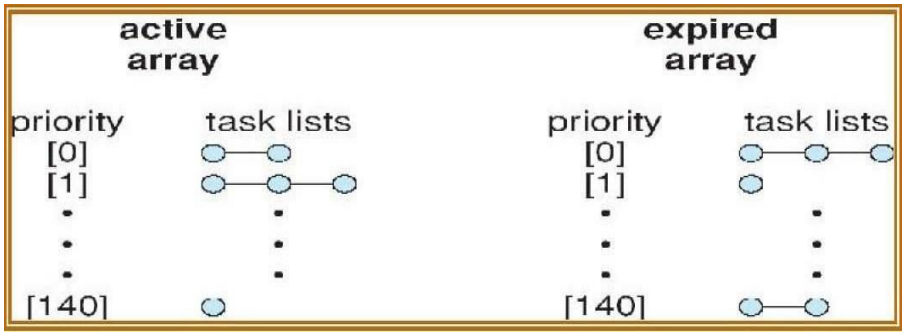


Fig: 21.3 list of tasks indexed according to priority

- ✓ Tasks are assigned dynamic priorities that are based on the *nice* value plus or minus a value up to the value 5 based upon the interactivity of the task.
- ✓ A task's interactivity is determined by how long it has been sleeping while waiting for I/0.
- ✓ Linux implements the two real time scheduling classes required by POSIX.lb: **first-come, first-served (FCFS) and round-robin**.
- ✓ In real-time scheduling, scheduler always runs the process with the highest priority. Among processes of equal priority, it runs the process that has been waiting longest.
- ✓ Unlike routine time-sharing tasks, real-time tasks are assigned static priorities.

- **Kernel Synchronization**

- ✓ A request for kernel-mode execution can occur in two ways:
  - A running program may request an operating system service, either explicitly via a system call, or implicitly, for example, when a page fault occurs
  - A device driver may deliver a hardware interrupt that causes the CPU to start executing a kernel-defined handler for that interrupt
- ✓ Kernel synchronization requires a framework that will allow the kernel's critical sections to run without interruption by another critical section

  The kernel is not pre emptible if a kernel-mode task is holding a lock.
- ✓ To enforce this rule, each task in the system has a thread-info structure that includes the field preempt_count, which is a counter indicating the number of locks being held by the task.
- ✓ The counter is incremented when a lock is acquired and decremented when a lock is released.

- ✓ If the value of preempt_count for the task currently running is greater than zero, it is not safe to preempt the kernel as this task currently holds a lock.
- ✓ If the count is zero, the kernel can safely be interrupted, assuming there are no outstanding calls to preempt_disable ().
- ✓ When the lock is held for short durations. When a lock must be held for longer periods, semaphores are used.

The second protection technique used by Linux applies to critical sections that occur in interrupt service routines.

- ✓ The basic tool is the processor's interrupt-control hardware.
- ✓ By disabling interrupts (or using spinlocks) during a critical section, the kernel guarantees that it can proceed without the risk of concurrent access to shared data structures.
- ✓ The Linux kernel uses a synchronization architecture that allows long critical sections to run for their entire duration without having interrupts disabled.
- ✓ This ability is especially useful in the networking code.

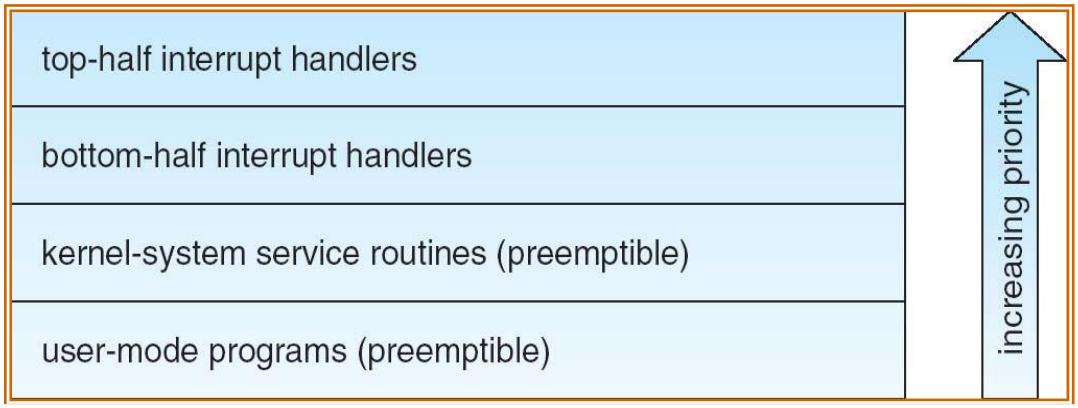


Fig: 21.4 interrupt protection levels

- ✓ Linux implements this architecture by separating interrupt service routines into two sections: the top half and the bottom half.
- ✓ The **Top half** is a normal interrupt service routine that runs with recursive interrupts disabled; interrupts of a higher priority may interrupt the routine, but interrupts of the same or lower priority are disabled.
- ✓ The **Bottom half** of a service routine is run, with all interrupts enabled, by a miniature scheduler that ensures that bottom halves never interrupt themselves.
- ✓ The bottom-half scheduler is invoked automatically whenever an interrupt service routine exits.
- ✓ This separation means that the kernel can complete any complex processing that has to be done in response to an interrupt without worrying about being interrupted itself.
- ✓ If another interrupt occurs while a bottom half is executing, then that interrupt can request that the same bottom half execute, but the execution will be deferred until the one currently running completes.
- ✓ Each execution of the bottom half can be interrupted by a top half but can never be interrupted by a similar bottom half.
- ✓ Figure 21.4 summarizes the various levels of interrupt protection within the kernel.

- **Symmetric Multiprocessing**

  - ✓ The Linux 2.0 kernel was the first stable Linux kernel to support **Symmetric Multiprocessor (SMP)** hardware, allowing separate processes to execute in parallel on separate processors.
  - ✓ In Version 2.2 of the kernel, a single kernel spinlock (sometimes termed BKL for "big kernel lock") was created to allow multiple processes (running on different processors) to be active in the kernel concurrently.
  - ✓ However, the BKL provided a very coarse level of locking granularity. Later releases of the kernel made the SMP implementation more scalable by splitting this single kernel spinlock into multiple locks, each of which protects only a small subset of the kernel's data structures.

➢ Memory Management

- ✓ Memory management under Linux has two components.
- ✓ The first deals with allocating and freeing physical memory.
- ✓ The second handles virtual memory, which is memory mapped into the address space of running processes.

- **Management of Physical Memory**

  - ✓ Linux separates physical memory into three different zones, or regions:
    - ZONE_DMA
    - ZONE_NORMAL
    - ZONE_HIGHMEM
  - ✓ These zones are architecture specific. The relationship of zones and physical addresses on the Intel80x86 architecture is shown in Figure 21.5.
  - ✓ The kernel maintains a list of free pages for each zone. When a request for physical memory arrives, the kernel satisfies the request using the appropriate zone.
  - ✓ The primary physical-memory manager in the Linux kernel is the **page allocator.** Each zone has its own allocator, which is responsible for allocating and freeing all physical pages for the zone and is capable of allocating ranges of physically contiguous pages on request.
  - ✓ The allocator uses a buddy system to keep track of available physical pages. In this scheme, adjacent units of allocatable memory are paired together (hence its name). Each allocatable memory region has an adjacent partner (or buddy).

zone	physical memory
ZONE_DMA	< 16 MB
ZONE_NORMAL	16 .. 896 MB
ZONE_HIGHMEM	> 896 MB

Fig: 21.5 relationship of zones and physical addresses on the intel 80*86

- ✓ Whenever two allocated partner regions are freed up, they are combined to form a larger region-a *buddy heap.*
- ✓ That larger region also has a partner, with which it can combine to form a still larger free region.
- ✓ Conversely, if a small memory request cannot be satisfied by allocation of an existing small free region, then a large free region will be subdivided into two partners to satisfy the request.
- ✓ Separate linked lists are used to record the free memory regions of each allowable size;
- ✓ Figure 21.6 shows an example of buddy-heap allocation. A 4-KB region is being allocated, but the smallest available region is 16 KB. The region is broken up recursively until a piece of the desired size is available.
- ✓ The kmalloc () variable-length allocator; the slab allocator, used for allocating memory for kernel data structures; and the page cache, used for caching pages belonging to files.

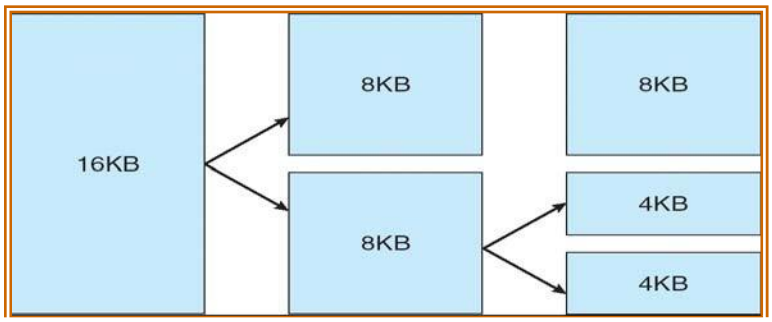


Fig: 21.6 Splitting of memory in the buddy system

- ✓ Many components of the Linux operating system need to allocate entire pages on request, but often smaller blocks of memory are required.
- ✓ The kernel provides an additional allocator for arbitrary-sized requests, where the size of a request is not known in advance and may be only a few bytes.
- ✓ kmalloc () service allocates entire pages on demand but then splits them into smaller pieces.
- ✓ The kernel maintains lists of pages in use by the kmalloc () service.
- ✓ Another strategy adopted by Linux for allocating kernel memory is known as slab allocation.
- ✓ A **slab** is used for allocating memory for kernel data structures and is made up of one or more physically contiguous pages.
- ✓ A **Cache** consists of one or more slabs.
- ✓ There is a single cache for each unique kernel data structure.
- ✓ Each cache is populated with that are instantiations of the kernel data structure the cache represents.
- ✓ The relationship among slabs, caches, and objects is shown in Figure 21.7.

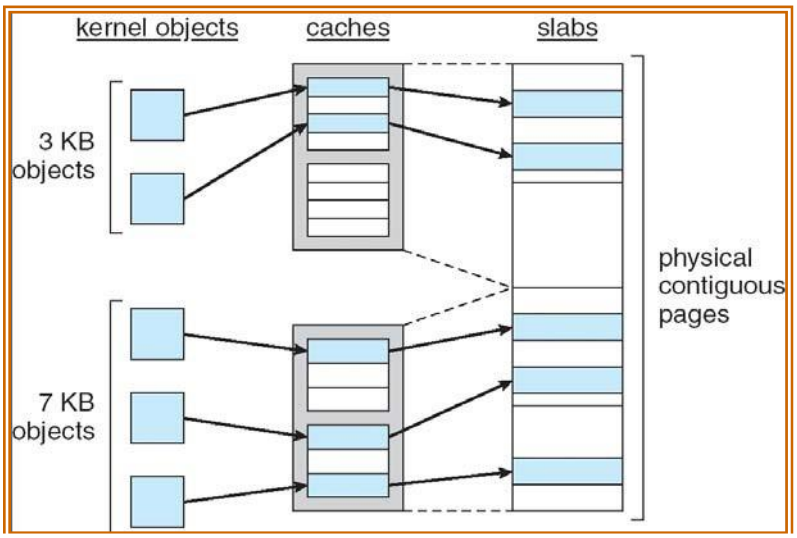


Fig: 21.7 Slab allocator in Linux

- ✓ The figure shows two kernel objects 3 KB in size and three objects 7 KB in size.
- ✓ These objects are stored in the respective caches for 3-KB and 7-KB objects.
- ✓ The slab-allocation algorithm uses caches to store kernel objects. When a cache is created, a number of objects are allocated to the cache. The number of objects in the cache depends on the size of the associated slab.
- ✓ Let's consider a scenario in which the kernel requests memory from the slab allocator for an object representing a process descriptor.
- ✓ In Linux, a slab may be in one of three possible states:
  - o Full- All objects in the slab are marked as used.
  - o Empty-All objects in the slab are marked as free.
  - o Partial-The slab consists of both used and free objects.
- ✓ The slab allocator first attempts to satisfy the request with a free object in a partial slab. If none exist, a free object is assigned from an empty slab.
- ✓ If no empty slabs are available, a new slab is allocated from contiguous physical pages and assigned to a cache; memory for the object is allocated from this slab.

- **Virtual Memory**

  - ✓ The VM system maintains the address space visible to each process: It creates pages of virtual memory on demand, and manages the loading of those pages from disk or their swapping back out to disk as required
  - ✓ The VM manager maintains two separate views of a process's address space:
    - o A logical view describing instructions concerning the layout of the address space. The address space consists of a set of non overlapping regions, each representing a continuous, page-aligned subset of the address space
    - o A physical view of each address space which is stored in the hardware page tables for the process
  - ✓ Virtual memory regions are characterized by:

- The backing store, which describes from where the pages for a region come; regions are usually backed by a file or by nothing (*demand-zero* memory)
- The region's reaction to writes (page sharing or copy-on-write)

✓ The kernel creates a new virtual address space
- When a process runs a new program with the exec system call
- Upon creation of a new process by the fork system call

✓ On executing a new program, the process is given a new, completely empty virtual-address space; the program-loading routines populate the address space with virtual-memory regions

✓ Creating a new process with **fork** involves creating a complete copy of the existing process's virtual address space
- The kernel copies the parent process's VMA descriptors, then creates a new set of page tables for the child
- The parent's page tables are copied directly into the child's, with the reference count of each page covered being incremented
- After the fork, the parent and child share the same physical pages of memory in their address spaces
-

▪ **Swapping and Paging**

✓ An important task for a virtual memory system is to relocate pages of memory from physical memory out to disk when that in memory is needed.

✓ The VM paging system relocates pages of memory from physical memory out to disk when the memory is needed for something else

✓ The VM paging system can be divided into two sections:
- The pageout-policy algorithm decides which pages to write out to disk, and when
- The paging mechanism actually carries out the transfer, and pages data back into physical memory as needed

✓ Linux's pageout policy uses a modified version of the standard clock (or second-chance) algorithm.

✓ a multiplepass clock is used, and every page has an *age* that is adjusted on each pass of the clock.

✓ The age is more precisely a measure of the page's youthfulness, or how much activity the page has seen recently.

✓ Frequently accessed pages will attain a higher age value, but the age of infrequently accessed pages will drop toward zero with each pass.

✓ This age valuing allows the pager to select pages to page out based on a least frequently used (LFU) policy.

▪ **Kernel Virtual Memory**

✓ Linux reserves for its own internal use a constant, architecture-dependent region of the virtual address space of every process.

✓ The page-table entries that map to these kernel pages are marked as protected, so that the pages are not visible or modifiable when the processor is running in user mode. This kernel virtual memory area contains two regions.

- ✓ The first is a static area that contains page-table references to every available physical page of memory in the system, so that a simple translation from physical to virtual addresses occurs when kernel code is run.
- ✓ The remainder of the kernel's reserved section of address space is not reserved for any specific purpose.
- ✓ Page-table entries in this address range can be modified by the kernel to point to any other areas of memory.
- ✓ The kernel provides a pair of facilities that allow processes to use this virtual memory.
- ✓ The vmalloc () function allocates an arbitrary number of physical pages of memory that may not be physically contiguous into a single region of virtually contiguous kernel memory.
- ✓ The vremap () function maps a sequence of virtual addresses to point to an area of memory used by a device driver form memory-mapped I/0.

- **Execution and Loading of User Programs**

- ✓ The Linux kernel's execution of user programs is triggered by a call to the exec() system call.
- ✓ This exec() call commands the kernel to run a new program within the current process, completely overwriting the current execution context with the initial context of the new program.
- ✓ The first job of this system service is to verify that the calling process has permission rights to the file being executed.
- ✓ Once that matter has been checked, the kernel invokes a loader routine to start running the program. The loader does not necessarily load the contents of the program file into physical memory, but it does at least set up the mapping of the program into virtual memory.
- ✓ There is no single routine in Linux for loadil1.g a new program. Instead, Linux maintains a table of possible loader functions, and it gives each such function the opportunity to try loading the given file when an exec() system call is made.
- ✓ Newer Linux systems use the more modern ELF format, now supported by most current UNIX implementations.
- ✓ ELF has a number of advantages over a. out, including flexibility and extensibility.
- ✓ New sections can be added to an ELF binary (for example, to add extra debugging information) without causing the loader routines to become confused.
- ✓ By allowing registration of multiple loader routines, Linux can easily support the ELF and a. out binary formats in a single running system.
- ✓ Figure 21.8 shows the typical layout of memory regions set up by the ELF loader. In a reserved region at one end of the address space sits the kernel in its own privileged region of virtual memory inaccessible to normal user-mode programs.

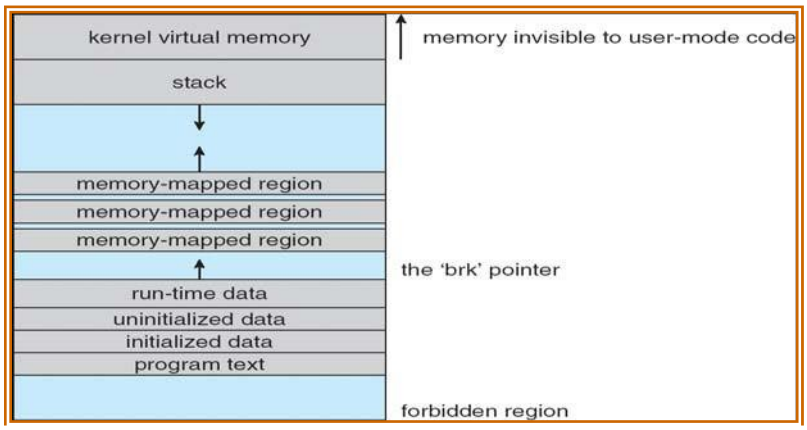


Fig: 21.8 Memory layout for ELF programs

- **Static and dynamic Linking**
  - ✓ The main disadvantage of static linking is that every program generated must contain copies of exactly the same common system library functions.
  - ✓ It is much more efficient, in terms of both physical memory and disk-space usage, to load the system libraries into memory only once.
  - ✓ Dynamic linking allows this single loading to happen.
  - ✓ Linux implements dynamic linking in user mode through a special linker library.
  - ✓ Every dynamically linked program contains a small, statically linked function that is called when the program starts.
  - ✓ This static function just maps the link library into memory and runs the code that the function contains.
  - ✓ The link library determines the dynamic libraries required by the program and the names of the variables and functions needed from those libraries by reading the information contained in sections of the ELF binary.
  - ✓ It then maps the libraries into the middle of virtual memory and resolves the references to the symbols contained in those libraries i.e., It does not matter exactly where in memory these shared libraries are mapped: they are compiled into position-independent code (PIC), which can run at any address in memory.

➢ File System

- ✓ To the user, Linux's file system appears as a hierarchical directory tree obeying UNIX semantics
- ✓ Internally, the kernel hides implementation details and manages the multiple different file systems via an abstraction layer, that is, the virtual file system (VFS)

- **The Virtual File System**

- ✓ The Linux VFS is designed around object-oriented principles.
- ✓ It has two components: a set of definitions that specify what file-system objects are allowed to look like and a layer of software to manipulate the objects.

- ✓ The VFS defines four main object types:
  - An **inode object** represents an individual file.
  - A **file object** represents an open file.
  - A **superblock object** represents an entire file system.
  - A **dentry object** represents an individual directory entry.
- ✓ For each of these four object types, the VFS defines a set of operations.
- ✓ Every object of one of these types contains a pointer to a function table.
- ✓ The function table lists the addresses of the actual functions that implement the defined operations for that object.
- ✓ For example, an abbreviated API for some of the file object's operations includes:
  - int open (. . .) - Open a file.
  - ssize_t read(. . .) -Read from a file.
  - ssize_t write (. . .) - Write to a file.
  - int mmap (. . .) - Memory-map a file.
- ✓ The complete definition of the file object is specified in the struct file_operations which is located in the file /usr/include/linux/fs.h.

- **The Linux ext2fs File System**

  - ✓ Ext2fs uses a mechanism similar to that of BSD Fast File System (ffs) for locating data blocks belonging to a specific file
  - ✓ The main differences between ext2fs and ffs concern their disk allocation policies
    - In ffs, the disk is allocated to files in blocks of 8Kb, with blocks being subdivided into fragments of 1Kb to store small files or partially filled blocks at the end of a file
    - Ext2fs does not use fragments; it performs its allocations in smaller units
    - The default block size on ext2fs is 1Kb, although 2Kb and 4Kb blocks are also supported
    - Ext2fs uses allocation policies designed to place logically adjacent blocks of a file into physically adjacent blocks on disk, so that it can submit an I/O request for several disk blocks as a single operation

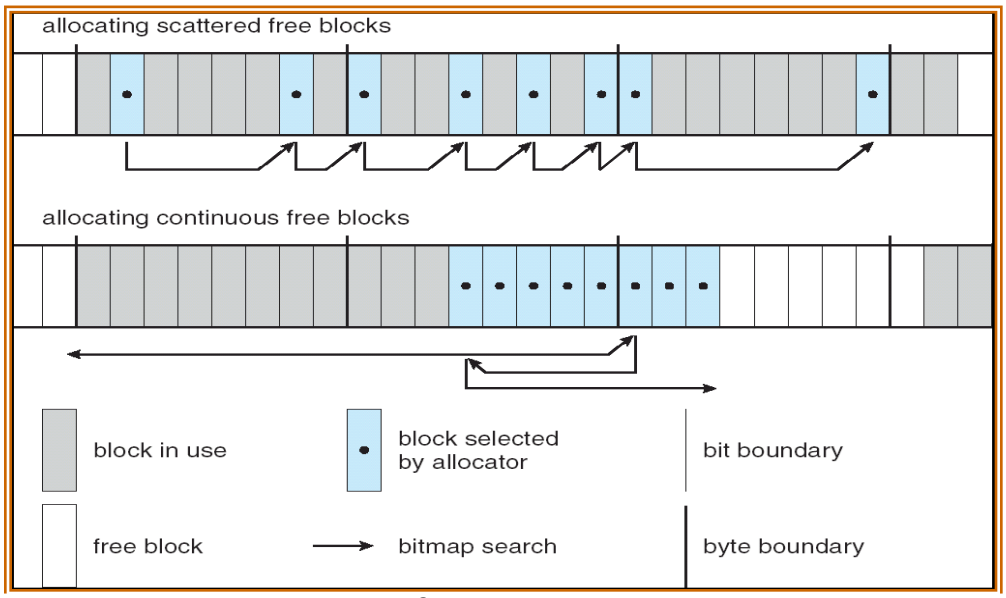


Fig: 21.9 ext2fs block-allocation policies

- **Journaling**

  - ✓ One popular feature in a file system is journaling, whereby modifications to the file system are sequentially written to a journal.
  - ✓ Once a transaction is written to the journal it is considered to be committed, and the system call modifying the file system (write()) can return to the user process, allowing it to continue execution.
  - ✓ If the system crashes, some transactions may remain in the journal.
  - ✓ Those transactions were never completed to the file system even though they were committed by the operating system, so they must be completed.
  - ✓ Journaling file systems are also typically faster than non-journaling systems, as updates proceed much faster when they are applied to the in-memory journal rather than directly to the on-disk data structures.
  - ✓ Journaling is not provided in ext2fs. It is provided in another common file system available for Linux systems, ext3, which is based on ext2fs.

- **The Linux Proc File System**

  - ✓ The **proc** file system does not store data, rather, its contents are computed on demand according to user file I/O requests.
  - ✓ **proc** must implement a directory structure, and the file contents within; it must then define a unique and persistent inode number for each directory and files it contains
    - It uses this inode number to identify just what operation is required when a user tries to read from a particular file inode or perform a lookup in a particular directory inode
    - When data is read from one of these files, **proc** collects the appropriate information, formats it into text form and places it into the requesting process's read buffer

➢ Input and Output

- ✓ To the user, the I/O system in Linux looks much like as normal files.
- ✓ Users can open an access channel to a device in the same way they opens any other file-devices can appear as objects within the file system.
- ✓ The system administrator can create special files within a file system that contain references to a specific device driver, and a user opening such a file will be able to read from and write to the device referenced.
- ✓ By using the normal file-protection system, which determines who can access which file, the administrator can set access permissions for each device.

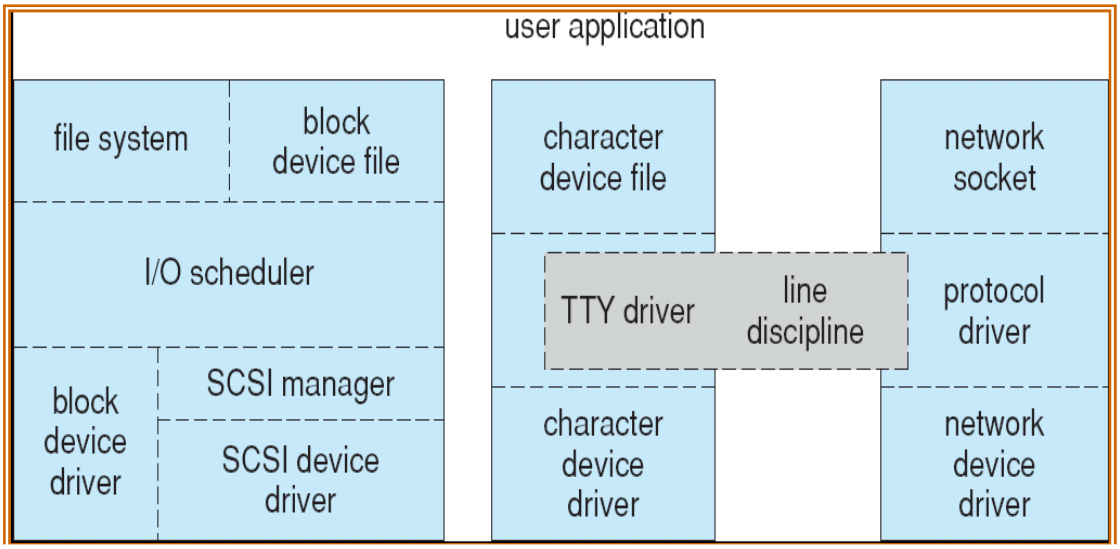


Fig: 21.10 Device driver block structure

- ✓ **Linux splits all devices into three classes: block devices, character devices, and network devices.**
- ✓ Figure 21.10 illustrates the overall structure of the device-driver system.
- ✓ **Block Devices** include all devices that allow random access to completely independent, fixed-sized blocks of data, including hard disks and floppy disks, CD-ROMs, and flash memory.
- ✓ Block devices are typically used to store file systems, but direct access to a block device is also allowed so that programs can create and repair the file system that the device contains.
- ✓ Applications can also access these block devices directly if they wish; for example, a database application may prefer to perform its own, fine-tuned laying out of data onto the disk, rather than using the general-purpose file system.
- ✓ **Character Devices** include most other devices, such as mice and keyboards.
- ✓ The fundamental difference between block and character devices is random access-block devices may be accessed randomly, while character devices are only accessed serially.
- ✓ For example, seeking to a certain position in a file might be supported for a DVD but makes no sense to a pointing device such as a mouse.
- ✓ **Network Devices** are dealt with differently from block and character devices.
- ✓ Users cannot directly transfer data to network devices; instead, they must communicate indirectly by opening a connection to the kernel's networking subsystem.

➢ Interprocess Communication

- ✓ Linux provides a rich environment for processes to communicate with each other.
- ✓ Communication may be just a matter of letting another process know that some event has occurred, or it may involve transferring data from one process to another.

- **Synchronization and Signals**

  - ✓ The standard Linux mechanism for informing a process that an event has occurred is the Signals can be sent from any process to any other process, with restrictions on signals sent to processes owned by another user.
  - ✓ Signals are not generated only by processes. The kernel also generates signals internally.
  - ✓ The Linux kernel does not use signals to communicate with processes with are running in kernel mode, rather, communication within the kernel is accomplished via scheduling states and **wait.queue** structures.
  - ✓ These mechanisms allow kernel-mode processes to inform one another about relevant events, and they also allow events to be generated by device drivers or by the networking system.
  - ✓ Whenever a process wants to wait for some event to complete, it places itself on a wait queue associated with that event and tells the scheduler that it is no longer eligible for execution.
  - ✓ Once the event has completed, it will wake up every process on the **wait Queue**.
  - ✓ This procedure allows multiple processes to wait for a single event.
  - ✓ Although signals have always been the main mechanism for communicating asynchronous events among processes, Linux also implements the semaphore mechanism of System V UNIX.
  - ✓ A process can wait on a semaphore as easily as it can wait for a signal, but semaphores have two advantages: Large numbers of semaphores can be shared among multiple independent processes, and operations on multiple semaphores can be performed atomically.

- **Passing of Data Among Processes**

  - ✓ The pipe mechanism allows a child process to inherit a communication channel to its parent, data written to one end of the pipe can be read a the other
  - ✓ Shared memory offers an extremely fast way of communicating; any data written by one process to a shared memory region can be read immediately by any other process that has mapped that region into its address space
  - ✓ To obtain synchronization, however, shared memory must be used in conjunction with another Interprocess-communication mechanism
  - ✓ The shared-memory object acts as a backing store for shared-memory regions in the same way as a file can act as backing store for a memory-mapped memory region.

- ✓ Shared-memory mappings direct page faults to map in pages from a persistent shared-memory object.
- ✓ Shared-memory objects remember their contents even if no processes are currently mapping them into virtual memory.

Question Bank

1. Explain the structure of magnetic disks.
2. Explain host attached storage, NAS and SAN.
3. What is disk scheduling? Explain various disk scheduling algorithms?
4. What is boot block and bad block? Explain techniques used for handling bad blocks.
5. Explain how logical & physical formatting is done on a disk.
6. What is swap space management. Explain.
7. Explain Goals and principles of Protection.
8. Explain the difference between protection and security.
9. What is access matrix? Explain access matrix with domain as objects. Explain the methods for implementing access matrix.
10. Explain the parameters considered for revocation of access matrix.
11. Explain the capability based systems : Hydra & **Cambridge Cap System**
12. Discuss various components of a linux os.
13. Write a short note on Design principle of Linux system.
14. Explain different IPC mechanisms available in Linux System .
15. What are the three classes of devices in LINUX? Explain the overall structure of the device-driver system in LINUX
16. Explain the process scheduling and kernel synchronization in Linux in detail.
17. Explain file systems implementation in Linux.
18. How does Linux manage authentication and access control mechanisms.
19. Explain kernel modules in detail.
20. Explain process context in Linux
21. Discuss the memory management with reference to Linux.
22. Explain the Linux device driver block structure.
23. Given the following queue 95, 180, 34,119,11,123,62,64 with head initially at track 50 and ending at track 199 calculate the number of moves using FCFS, SSTF, Elevator and C look algorithm.
24. Discuss the strengths and weakness of implementing an access matrix using access lists that are associated with objects.

VIVA QUESTIONS – OPERATING SYSTEMS

1. What is an operating system?

An operating system is a program that acts as an intermediary between the user and the computer hardware. The purpose of an OS is to provide a convenient environment in which user can execute programs in a convenient and efficient manner. It is a resource allocator responsible for allocating system resources and a control program which controls the operation of the computer h/w.

2. What are the different operating systems?

1. Batched operating systems
2. Multi-programmed operating systems
3. timesharing operating systems
4. Distributed operating systems
5. Real-time operating systems

3. What is a boot-strap program?

Bootstrapping is a technique by which a simple computer program activates a more complicated system of programs. It comes from an old expression "to pull oneself up by one's bootstraps."

4. What is BIOS?

A BIOS is software that is put on computers. This allows the user to configure the input and output of a computer. A BIOS is also known as firmware.

5. Explain the concept of the batched operating systems?

In batched operating system the users gives their jobs to the operator who sorts the programs according to their requirements and executes them. This is time consuming but makes the CPU busy all the time.

6. Explain the concept of the multi-programmed operating systems?

A multi-programmed operating systems can execute a number of programs concurrently. The operating system fetches a group of programs from the job-pool in the secondary storage which contains all the programs to be executed, and places them in the main memory. This process is called job scheduling. Then it chooses a program from the ready queue and gives them to CPU to execute. When a executing program needs some I/O operation then the operating system fetches another program and hands it to the CPU for execution, thus keeping the CPU busy all the time.

7. Explain the concept of the timesharing operating systems?

It is a logical extension of the multi-programmed OS where user can interact with the program. The CPU executes multiple jobs by switching among them, but the switches occur so frequently that the user feels as if the operating system is running only his program.

8. Explain the concept of the multi-processor systems or parallel systems?

They contain a no. of processors to increase the speed of execution, and reliability, and economy. They are of two types:

1. Symmetric multiprocessing
2. Asymmetric multiprocessing

In Symmetric multi processing each processor run an identical copy of the OS, and these copies communicate with each other as and when needed. But in Asymmetric multiprocessing each processor is assigned a specific task.

9. Explain the concept of the Distributed systems?

Distributed systems work in a network. They can share the network resources, communicate with each other

10. Explain the concept of Real-time operating systems?

A real time operating system is used when rigid time requirement have been placed on the operation of a processor or the flow of the data; thus, it is often used as a control device in a dedicated application. Here the sensors bring data to the computer. The computer must analyze the data and possibly adjust controls to
modify the sensor input.

They are of two types:

1. Hard real time OS
2. Soft real time OS

Hard-real-time OS has well-defined fixed time constraints. But soft real time operating systems have less stringent timing constraints.

11. What is dual-mode operation?

In order to protect the operating systems and the system programs from the malfunctioning programs the two mode operations were evolved:

1. System mode.
2. User mode.

Here the user programs cannot directly interact with the system resources, instead they request the operating system which checks the request and does the required task for the user programs-DOS was written for / intel 8088 and has no dual-mode. Pentium provides dual-mode operation.

12. What are the operating system components?

1. Process management
2. Main memory management
3. File management
4. I/O system management
5. Secondary storage management
6. Networking
7. Protection system
8. Command interpreter system

13. What are operating system services?

1. Program execution
2. I/O operations
3. File system manipulation
4. Communication
5. Error detection
6. Resource allocation
7. Accounting
8. Protection

14. What are system calls?

System calls provide the interface between a process and the operating system. System calls for modern Microsoft windows platforms are part of the win32 API, which is available for all the compilers written for Microsoft windows.

15. What is a layered approach and what is its advantage?

Layered approach is a step towards modularizing of the system, in which the operating system is broken up into a number of layers (or levels), each built on top of lower layer. The bottom layer is the hard ware and the top most is the user interface. The main advantage of the layered approach is modularity. The layers are
selected such that each uses the functions (operations) and services of only lower layer. This approach simplifies the debugging and system verification.

16. What is micro kernel approach and site its advantages?

Micro kernel approach is a step towards modularizing the operating system where all nonessential components from the kernel are removed and implemented as system and user level program, making the kernel smaller. The benefits of the micro kernel approach include the ease of extending the operating system. All new services are added to the user space and consequently do not require modification of the kernel. And as kernel is smaller it is easier to upgrade it. Also this approach provides more security and reliability since most services are running as user processes rather than kernel's keeping the kernel intact.

17. What are a virtual machines and site their advantages?

It is the concept by which an operating system can create an illusion that a process has its own processor with its own (virtual) memory. The operating system implements virtual machine concept by using CPU scheduling and virtual memory.

18. What is a process?

A program in execution is called a process. Or it may also be called a unit of work. A process needs some system resources as CPU time, memory, files, and i/o devices to accomplish the task. Each process is represented in the operating system by a process control block or task control block (PCB).Processes are of two types:

1. Operating system processes
2. User processes

19.What are the states of a process?

1. New
2. Running
3. Waiting
4. Ready
5. Terminated

20.What are various scheduling queues?

1. Job queue
2. Ready queue
3. Device queue

21. What is a job queue?

When a process enters the system it is placed in the job queue.

22. What is a ready queue?

The processes that are residing in the main memory and are ready and waiting to execute are kept on a list called the ready queue.

23. What is a device queue?

A list of processes waiting for a particular I/O device is called device queue.

24. What is a long term scheduler & short term schedulers?

Long term schedulers are the job schedulers that select processes from the job queue and load them into memory for execution. The short term schedulers are the CPU schedulers that select a process form the ready queue and allocate the CPU to one of them.

25. What is context switching?

Transferring the control from one process to other process requires saving the state of the old process and loading the saved state for new process. This task is known as context switching.

26. What are the disadvantages of context switching?

Time taken for switching from one process to other is pure over head. Because the system does no useful work while switching. So one of the solutions is to go for threading when ever possible.

27. What are co-operating processes?

The processes which share system resources as data among each other. Also the processes can communicate with each other via interprocess communication facility generally used in distributed systems. The best example is chat program used on the www.

28. What is a thread?

Thread sometimes called a light-weight process, is a basic unit of CPU utilization; it comprises a thread id, a program counter, a register set, and a stack.

29. What are types of threads?

1. User thread
2. Kernel thread

User threads are easy to create and use but the disadvantage is that if they perform a blocking system calls the kernel is engaged completely to the single user thread blocking other processes. They are created in user space. Kernel threads are supported directly by the operating system. They are slower to create and manage. Most of the OS like Windows NT, Windows 2000, Solaris2, BeOS, and Tru64 Unix support kernel threading.

30. What are multithreading models?

Many OS provide both kernel threading and user threading. They are called multithreading models. They are of three types:

1. Many-to-one model (many user level thread and one kernel thread).
2. One-to-one model
3. Many-to –many

31. What is a P-thread?

P-thread refers to the POSIX standard (IEEE 1003.1c) defining an API for thread creation and synchronization. This is a specification for thread behavior, not an implementation. The windows OS have generally not supported the P-threads.

32. What is process synchronization?

A situation, where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called race condition. To guard against the race condition we need to ensure that only one process at a time can be manipulating
the same data. The technique we use for this is called process synchronization.

33. What is critical section problem?

Critical section is the code segment of a process in which the process may be changing common variables, updating tables, writing a file and so on. Only one process is allowed to go into critical section at any given time (mutually exclusive).The critical section problem is to design a protocol that the processes can use to
co-operate. The three basic requirements of critical section are:

1. Mutual exclusion
2. Progress
3. bounded waiting

Bakery algorithm is one of the solutions to CS problem.

34. What is a semaphore?

It is a synchronization tool used to solve complex critical section problems. A semaphore is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: Wait and Signal.

35. What is a deadlock?

Suppose a process request resources; if the resources are not available at that time the process enters into a wait state. A waiting process may never again change state, because the resources they have requested are held by some other waiting processes. This situation is called deadlock.

36. What are necessary conditions for dead lock?

1. Mutual exclusion (where at least one resource is non-sharable)
2. Hold and wait (where a process hold one resource and waits for other resource)
3. No preemption (where the resources can't be preempted)
4. circular wait (where p[i] is waiting for p[j] to release a resource. i= 1,2,…n
j=if (i!=n) then i+1
else 1)

37. What are deadlock prevention techniques?

1. Mutual exclusion : Some resources such as read only files shouldn't be mutually exclusive. They should be sharable. But some resources such as printers must be mutually exclusive.
2. Hold and wait : To avoid this condition we have to ensure that if a process is requesting for a resource it should not hold any resources.
3. No preemption : If a process is holding some resources and requests another resource that cannot be immediately allocated to it (that is the process must wait), then all the resources currently being held are preempted(released autonomously).
4. Circular wait : the way to ensure that this condition never holds is to impose a total ordering of all the resource types, and to require that each process requests resources in an increasing order of enumeration.

38. What is a safe state and a safe sequence?

A system is in safe state only if there exists a safe sequence. A sequence of processes is a safe sequence for the current allocation state if, for each Pi, the resources that the Pi can still request can be satisfied by the currently available resources plus the resources held by all the Pj, with j

39. What are the deadlock avoidance algorithms?

A dead lock avoidance algorithm dynamically examines the resource-allocation state to ensure that a circular wait condition can never exist. The resource allocation state is defined by the number of available and allocated resources, and the maximum demand of the process. There are two algorithms:

1. Resource allocation graph algorithm
2. Banker's algorithm
   a. Safety algorithm
   b. Resource request algorithm

40. Why paging is used?

Paging is solution to external fragmentation problem which is to permit the logical address space of a process to be noncontiguous, thus allowing a process to be allocating physical memory wherever the latter is available.

41. What is Dispatcher?

->Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
Switching context
Switching to user mode
Jumping to the proper location in the user program to restart that program
Dispatch latency – time it takes for the dispatcher to stop one process and start another running.

42. What is Throughput, Turnaround time, waiting time and Response time?

Throughput – number of processes that complete their execution per time unit
Turnaround time – amount of time to execute a particular process
Waiting time – amount of time a process has been waiting in the ready queue
Response time – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

43. What is starvation and aging?

Starvation: Starvation is a resource management problem where a process does not get the resources it needs for a long time because the resources are being allocated to other processes.
Aging: Aging is a technique to avoid starvation in a scheduling system. It works by adding an aging factor to the priority of each request. The aging factor must increase the request's priority as time passes and must ensure that a request will eventually be the highest priority request (after it has waited long enough)

44. Recovery from Deadlock?

Process Termination:
Resource Preemption:

45.Difference between Logical and Physical Address Space?

->The concept of a logical address space that is bound to a separate physical address space is central to proper memory management.
Logical address – generated by the CPU; also referred to as virtual address.
Physical address – address seen by the memory unit.
->Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme

46. Binding of Instructions and Data to Memory?

Address binding of instructions and data to memory addresses can happen at three different stages
Compile time: If memory location known a priori, absolute code can be generated; must recompile code if starting location changes.

Load time: Must generate relocatable code if memory location is not known at compile time. Execution time: Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., base and limit registers).

47. What is Memory-Management Unit (MMU)?

Hardware device that maps virtual to physical address.
In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.
->The user program deals with logical addresses; it never sees the real physical addresses

48. What is fragmentation? Different types of fragmentation?

Fragmentation occurs in a dynamic memory allocation system when many of the free blocks are too small to satisfy any request.
External Fragmentation: External Fragmentation happens when a dynamic memory allocation algorithm allocates some memory and a small piece is left over that cannot be effectively used. If too much external fragmentation occurs, the amount of usable memory is drastically reduced.Total memory space exists to satisfy a request, but it is not contiguous
Internal Fragmentation: Internal fragmentation is the space wasted inside of allocated memory blocks because of restriction on the allowed sizes of allocated blocks. Allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used Reduce external fragmentation by compaction
->Shuffle memory contents to place all free memory together in one large block.
->Compaction is possible only if relocation is dynamic, and is done at execution time.

49. Define Demand Paging, Page fault interrupt, and Thrashing?

Demand Paging: Demand paging is the paging policy that a page is not read into memory until it is requested, that is, until there is a page fault on the page.
Page fault interrupt: A page fault interrupt occurs when a memory reference is made to a page that is not in memory.The present bit in the page table entry will be found to be off by the virtual memory hardware and it will signal an interrupt.
Thrashing: The problem of many page faults occurring in a short time, called “page thrashing,”

50. Explain Segmentation with paging?

Segments can be of different lengths, so it is harder to find a place for a segment in memory than a page. With segmented virtual memory, we get the benefits of virtual memory but we still have to do dynamic storage allocation of physical memory. In order to avoid this, it is possible to combine segmentation and paging into a two-level virtual memory system. Each segment descriptor points to page table for that segment.This give some of the advantages of paging (easy placement) with some of the advantages of segments (logical division of the program).

51. Under what circumstances do page faults occur? Describe the actions taken by the operating system when a page fault occurs?

A page fault occurs when an access to a page that has not been brought into main memory takes

place. The operating system verifies the memory access, aborting the program if it is invalid. If it is valid, a free frame is located and I/O is requested to read the needed page into the free frame. Upon completion of I/O, the process table and page table are updated and the instruction is restarted.

52. Expalin different page replacement algorithms.

1. FIFO
2. OPTIMAL algorithm
3. LRU

53. What is Belady's anamoly?

Bélády's anomaly is the phenomenon in which the number of page frames results in an increase in the number of increasing page faults for certain memory access patterns. This phenomenon is commonly experienced when using the first-in first-out (FIFO) page replacement algorithm.