MODULE 5: MASS-STORAGE STRUCTURE PROTECTION THE LINUX SYSTEM

MODULE 5: MASS-STORAGE STRUCTURE

5.1 Mass Storage Structures

5.1.1 Hard-Disks

- Hard-disks provide the bulk of secondary-storage for modern computer-systems (Figure 5.1).
- Each disk-platter has a flat circular-shape, like a CD.
- The 2 surfaces of a platter are covered with a magnetic material.
- Information is stored on the platters by recording magnetically.

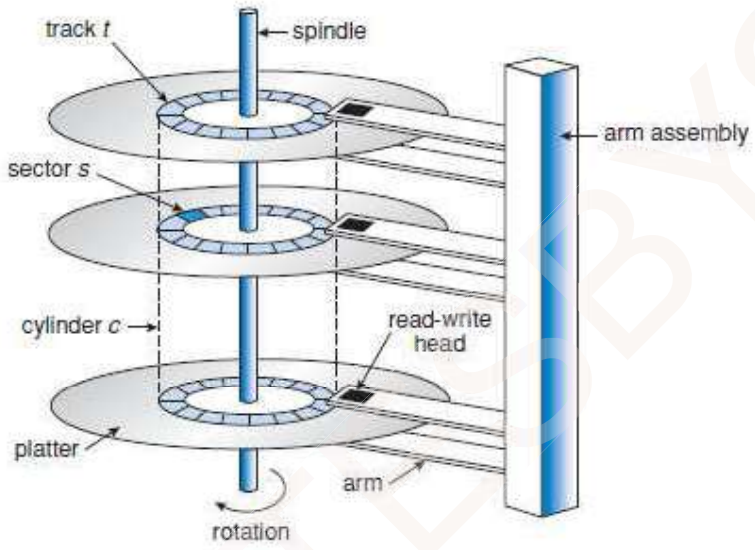


Figure 5.1 Moving-head disk mechanism

- A read–write head "flies" just above the surface of the platter.
- The heads are attached to a disk-arm that moves all the heads as a unit.
- The surface of a platter is logically divided into circular tracks, which are subdivided into sectors.
- The set of tracks that are at one arm position makes up a cylinder.
- There may be thousands of concentric-cylinders in a disk-drive, and each track may contain hundreds of sectors.
- Disk-speed has 2 parts:
    1) The transfer-rate is the rate at which data flow between the drive and the computer.
    2) The positioning-time(or random-access time) consists of 2 parts:
        i) Seek-time refers to the time necessary to move the disk-arm to the desired cylinder.
        ii) Rotational-latency refers to the time necessary for the desired sector to rotate to the disk-head.
- A disk can be removable which allows different disks to be mounted as needed.
- A disk-drive is attached to a computer by an I/O bus.
- Different kinds of buses:
    → advanced technology attachment (ATA)
    → serial ATA (SATA)
    → eSATA, universal serial bus (USB) and
    → fibre channel (FC).

5.1.2 Solid-State Disks

- An SSD is non-volatile memory that is used like a hard-drive.
- For example:
  DRAM with a battery to maintain its state in a power-failure through flash-memory technologies.
- Advantages compared to Hard-disks:
  1) More reliable : SSDs have no moving parts and are faster because they have no seek-time or latency.
  2) Less power consumption.
- Disadvantages:
  1) More expensive
  2) Less capacity and so shorter life spans, so their uses are somewhat limited.
- Applications:
  1) One use for SSDs is in storage-arrays, where they hold file-system metadata that require high performance.
  2) SSDs are also used in laptops to make them smaller, faster, and more energy-efficient.

5.1.3 Magnetic Tapes

- Magnetic tape was used as an early secondary-storage medium.
- Advantages:
  It is relatively permanent and can hold large quantities of data.
- Disadvantages:
  1) Its access time is slow compared with that of main memory and Hard-disk.
  2) In addition, random access to magnetic tape is about a thousand times slower than random access to Hard-disk, so tapes are not very useful for secondary-storage.
- Applications:
  1) Tapes are used mainly for backup, for storage of infrequently used information.
  2) Tapes are used as a medium for transferring information from one system to another.

5.2 Disk Structure

- Modern Hard-disk-drives are addressed as large one-dimensional arrays of logical blocks.
- The logical block is the smallest unit of transfer.
- How one-dimensional array of logical blocks is mapped onto the sectors of the disk sequentially?
  Sector 0 is the first sector of the first track on the outermost cylinder.
  The mapping proceeds in order through that track, then through the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.
- In practice, it is difficult to perform this mapping, for two reasons.
  1) Most disks have some defective sectors, but the mapping hides this by substituting spare sectors from elsewhere on the disk.
  2) The number of sectors per track is not a constant on some drives.

5.3 Disk Attachment

• Computers access disk storage in two ways.
1) via I/O ports (or host-attached storage); this is common on small systems.
2) via a remote host in a distributed file system; this is referred to as network-attached storage.

5.3.1 Host-Attached Storage

• Host-attached storage is storage accessed through local I/O ports.
• These ports use several technologies.
1) The desktop PC uses an I/O bus architecture called IDE or ATA.
➢ This architecture supports a maximum of 2 drives per I/O bus.
2) High-end workstations(and servers) use fibre channel (FC), a high-speed serial architecture that can operate over optical fiber.
➢ It has two variants:
i) One is a large switched fabric having a 24-bit address space.
¤ This variant is the basis of storage-area networks (SANs).
ii) The other FC variant is an arbitrated loop (FC-AL) that can address 126 devices.
• A wide variety of storage devices are suitable for use as host-attached storage.
For ex: Hard-disk-drives, RAID arrays, and CD, DVD, and tape drives.

5.3.2 Network-Attached Storage

Figure 5.2 Network-attached storage

• A network-attached storage (NAS) device is a special-purpose storage system that is accessed remotely over a data network (Figure 5.2).
• Clients access NAS via a remote-procedure-call interface such as
→ NFS for UNIX systems
→ CIFS for Windows machines.
• The remote procedure calls (RPCs) are carried via TCP or UDP over a local area network (LAN).
• Usually, the same local area network (LAN) carries all data traffic to the clients.
• The NAS device is usually implemented as a RAID array with software that implements the RPC interface.
• Advantage:
All computers on a LAN can
→ share a pool of storage with the same ease of naming and
→ access local host-attached storage.
• Disadvantages:
1) NAS is less efficient and have lower performance than some direct-attached storage options.
2) The storage I/O operations consume bandwidth on the data network, thereby increasing the latency of network communication.
• iSCSI is the latest network-attached storage protocol.
• iSCSI uses the IP network protocol to carry the SCSI protocol.
• Thus, networks—rather than SCSI cables—can be used as the interconnects between hosts and their storage.

5.3.3 Storage-Area Network

- A storage-area network (SAN) is a private network connecting servers and storage units (Figure 5.3).
- The power of a SAN lies in its flexibility.
    1) Multiple hosts and multiple storage-arrays can attach to the same SAN.
    2) Storage can be dynamically allocated to hosts.
    3) A SAN switch allows or prohibits access between the hosts and the storage.
    4) SANs make it possible for clusters of servers to share the same storage and for storage arrays to include multiple direct host connections.
    5) SANs typically have more ports than storage-arrays.
- FC is the most common SAN interconnect.
- Another SAN interconnect is InfiniBand — a special-purpose bus architecture that provides hardware and software support for high-speed interconnection networks for servers and storage units.

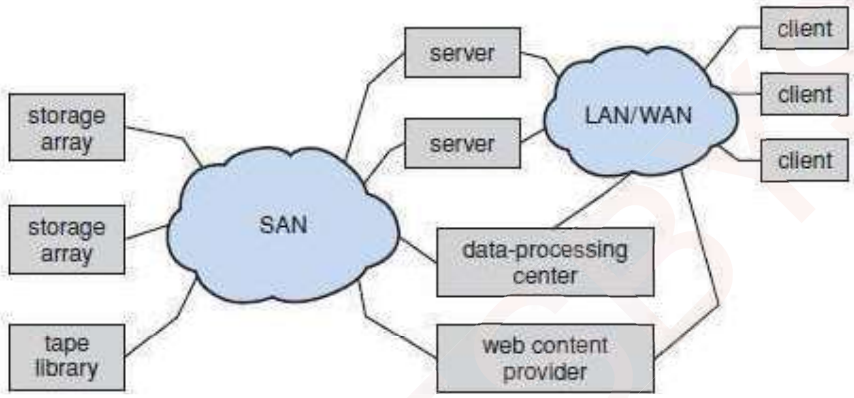


Figure 5.3 Storage-area network

5.4 Disk Scheduling

• Access time = Seek-time + Rotational-latency

1) Seek-time: The seek-time is the time for the disk-arm to move the heads to the cylinder containing the desired sector.

2) Rotational-latency: The Rotational-latency is the additional time for the disk to rotate the desired sector to the disk-head.

• The disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

• We can improve both the access time and the bandwidth by managing the order in which disk I/O requests are serviced.

• Whenever a process needs I/0 to or from the disk, it issues a system call to the operating system.

• The request specifies several pieces of information:

1) Whether this operation is input or output
2) What the disk address for the transfer is
3) What the memory address for the transfer is
4) What the number of sectors to be transferred is

• If the desired disk-drive and controller are available, the request can be serviced immediately.

• If the drive or controller is busy, any new requests for service will be placed in the queue of pending requests for that drive.

• For a multiprogramming system with many processes, the disk queue may often have several pending requests.

• Thus, when one request is completed, the operating system chooses which pending request to service next.

• Any one of several disk-scheduling algorithms can be used.

5.4.1 FCFS Scheduling

• FCFS stands for First Come First Serve.

• The requests are serviced in the same order, as they are received.

• For example:

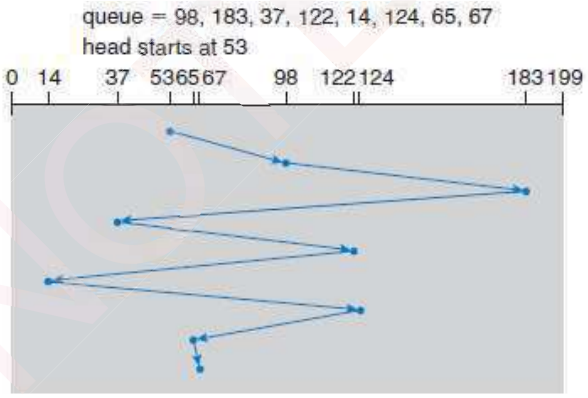


Figure 5.4 FCFS disk scheduling.

• Starting with cylinder 53, the disk-head will first move from 53 to 98, then to 183, 37, 122, 14, 124, 65, and finally to 67 as shown in Figure 5.4.

Head movement from 53 to 98 = 45
Head movement from 98 to 183 = 85
Head movement from 183 to 37 = 146
Head movement from 37 to 122 =85
Head movement from 122 to 14 =108
Head movement from 14 to 124 =110
Head movement from 124 to 65 =59
Head movement from 65 to 67 = 2
Total head movement = 640

• Advantage: This algorithm is simple & fair.

• Disadvantage: Generally, this algorithm does not provide the fastest service.

5.4.2 SSTF Scheduling

• SSTF stands for Shortest Seek-time First.
• This selects the request with minimum seek-time from the current head-position.
• Since seek-time increases with the number of cylinders traversed by head, SSTF chooses the pending request closest to the current head-position.
• Problem: Seek-time increases with the number of cylinders traversed by head.
Solution: To overcome this problem, SSTF chooses the pending request closest to the current head-position.
• For example:

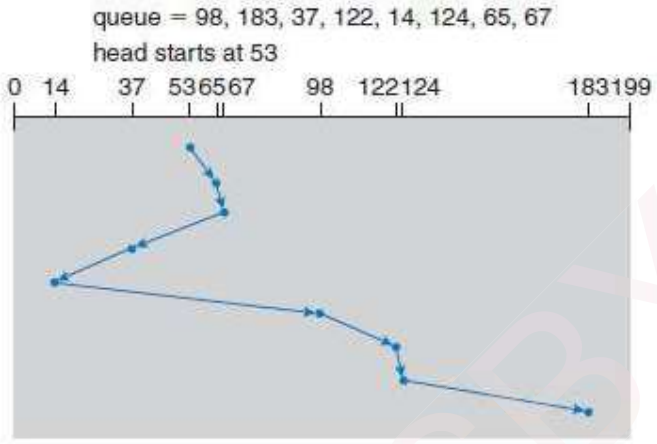


Figure 5.5 SSTF disk scheduling

• The closest request to the initial head position 53 is at cylinder 65. Once we are at cylinder 65, the next closest request is at cylinder 67.
• From there, the request at cylinder 37 is closer than 98, so 37 is served next. Continuing, we service the request at cylinder 14, then 98, 122, 124, and finally 183. It is shown in Figure 5.5.

Head movement from 53 to 65 = 12
Head movement from 65 to 67 = 2
Head movement from 67 to 37 = 30
Head movement from 37 to 14 =23
Head movement from 14 to 98 =84
Head movement from 98 to 122 =24
Head movement from 122 to 124 =2
Head movement from 124 to 183 = 59
Total head movement = 236

• Advantage: SSTF is a substantial improvement over FCFS, it is not optimal.
• Disadvantage: Essentially, SSTF is a form of SJF and it may cause starvation of some requests.

5.4.3 SCAN Scheduling

• The SCAN algorithm is sometimes called the elevator algorithm, since the disk-arm behaves just like an elevator in a building.

• Here is how it works:

1. The disk-arm starts at one end of the disk.
2. Then, the disk-arm moves towards the other end, servicing the request as it reaches each cylinder.
3. At the other end, the direction of the head movement is reversed and servicing continues.

• The head continuously scans back and forth across the disk.

• For example:

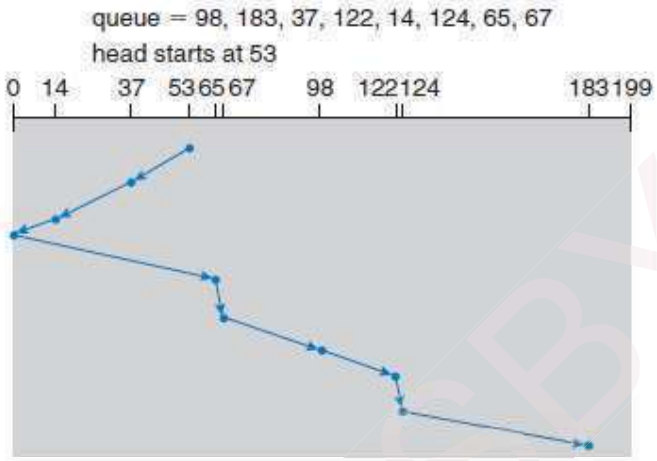


Figure 5.6 SCAN disk scheduling.

• Before applying SCAN algorithm, we need to know the current direction of head movement.

• Assume that disk-arm is moving toward 0, the head will service 37 and then 14.

• At cylinder 0, the arm will reverse and will move toward the other end of the disk, servicing the requests at 65,67,98, 122, 124, and 183. It is shown in Figure 5.6.

Head movement from 53 to 37 = 16
Head movement from 37 to 14 = 23
Head movement from 14 to 0 = 14
Head movement from 0 to 65 =65
Head movement from 65 to 67 =2
Head movement from 67 to 98 =31
Head movement from 98 to 122 =24
Head movement from 122 to 124 = 2
Head movement from 124 to 183 = 59
Total head movement = 236

• Disadvantage: If a request arrives just in from of head, it will be serviced immediately.
On the other hand, if a request arrives just behind the head, it will have to wait until the arms reach other end and reverses direction.

5.4.4 C-SCAN Scheduling

• Circular SCAN (C-SCAN) scheduling is a variant of SCAN designed to provide a more uniform wait time.
• Like SCAN, 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, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip (Figure 5.7).
• The C-SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

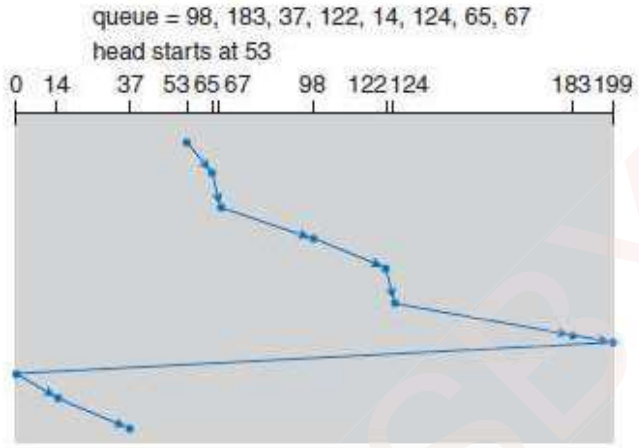


Figure 5.7: C-SCAN disk scheduling

• Before applying C - SCAN algorithm, we need to know the current direction of head movement.
• Assume that disk-arm is moving toward 199, the head will service 65, 67, 98, 122, 124, 183.
• Then it will move to 199 and the arm will reverse and move towards 0.
• While moving towards 0, it will not serve. But, after reaching 0, it will reverse again and then serve 14 and 37. It is shown in Figure 5.7.

Head movement from 53 to 65 = 12
Head movement from 65 to 67 = 2
Head movement from 67 to 98 = 31
Head movement from 98 to 122 =24
Head movement from 122 to 124 =2
Head movement from 124 to 183 =59
Head movement from 183 to 199 =16
Head movement from 199 to 0 = 199
Head movement from 0 to 14 = 14
Head movement from 14 to 37 = 23
Total head movement = 382

5.4.5 LOOK Scheduling

• SCAN algorithm move the disk-arm across the full width of the disk.
In practice, the SCAN algorithm is not implemented in this way.
• The arm goes only as far as the final request in each direction.
Then, the arm reverses, without going all the way to the end of the disk.
• This version of SCAN is called Look scheduling because they look for a request before continuing to move in a given direction.
• For example:

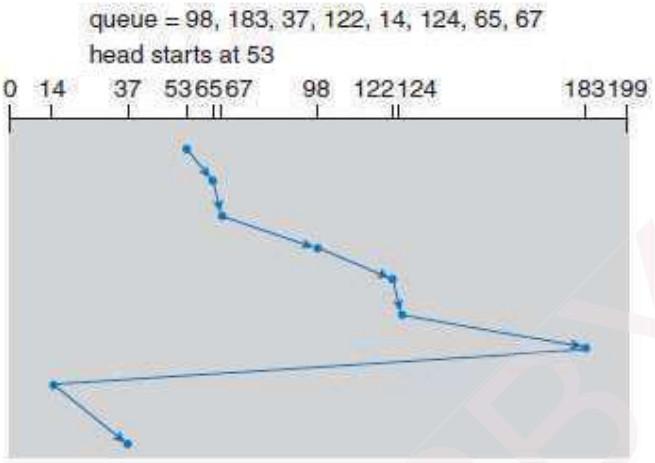


Figure 5.8 C-LOOK disk scheduling.

• Assume that disk-arm is moving toward 199, the head will service 65, 67, 98, 122, 124, 183.
• Then the arm will reverse and move towards 14. Then it will serve 37. It is shown in Figure 5.8.

Head movement from 53 to 65 = 12
Head movement from 65 to 67 = 2
Head movement from 67 to 98 = 31
Head movement from 98 to 122 =24
Head movement from 122 to 124 =2
Head movement from 124 to 183 =59
Head movement from 183 to 14 = 169
Head movement from 14 to 37 = 23
Total head movement = 322

5.5 Disk Management

- The operating system is responsible for several other aspects of disk management.
- For example:
  1) disk initialization
  2) booting from disk
  3) bad-block recovery.

5.5.1 Disk Formatting

- Usually, a new Hard-disk is a blank slate: it is just a platter of a magnetic recording material.
- Before a disk can store data, it must be divided into sectors that the disk controller 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.
- The data structure for a sector typically consists of
  → a header
  → a data area (usually 512 bytes in size), and
  → a trailer.
- The header and trailer contain information used by the disk controller, such as
  → sector number and
  → **error-correcting code (ECC)**.
- Before a disk can store data, the operating system still needs to record its own data structures on the disk.
- It does so in two steps.
  1) Partition the disk into one or more groups of cylinders.
  ➢ The operating system can treat each partition as a separate disk.
  ➢ For example:
  one partition can hold a copy of the operating system's executable code,
  another partition can hold user files.
  2) Logical formatting, or creation of a file system.
  ➢ The operating system stores the initial file-system data structures onto the disk.
  ➢ These data structures may include maps of free and allocated space and an initial empty directory.
- To increase efficiency, most file systems group blocks together into larger chunks, frequently called clusters.
  1) Disk I/O is done via blocks,
  2) File system I/O is done via clusters.

5.5.2 Boot Block

- For a computer to start running, it must have a bootstrap program to run.
- Bootstrap program
  → initializes CPU registers, device controllers and the contents of main memory and
  → then starts the operating system.
- For most computers, the bootstrap is stored in read-only memory (ROM).
- Main Problem: To change the bootstrap code, the ROM hardware chips has to be changed.
- To solve this problem, most systems store a tiny bootstrap loader program in the boot-ROM.
- Job of boot-ROM: Bring in a full bootstrap program from disk.
- The full bootstrap program can be changed easily: "A new version is simply written onto the disk".
- The full bootstrap program is stored in the "boot blocks" at a fixed location on the disk.
- A disk that has a boot partition is called a boot disk or system disk.
- In the boot-ROM, the code
  → instructs the disk-controller to read the boot blocks into memory and
  → then starts executing that code.

5.5.3 Bad Blocks

- Because disks have moving parts and small tolerances, they are prone to failure.
- Sometimes,
  - → The disk needs to be replaced.
  - → The disk-contents need to be restored from backup media to the new disk.
  - → One or more sectors may become defective.
- From the manufacturer, most disks have bad-blocks.
- How to handle bad-blocks?
  - ➢ On simple disks, bad-blocks are handled manually.
  - ➢ One strategy is to scan the disk to find bad-blocks while the disk is being formatted.
  - ➢ Any bad-blocks that are discovered are flagged as unusable. Thus, the file system does not allocate them.
  - ➢ If blocks go bad during normal operation, a special program (such as Linux bad-blocks command) must be run manually
    - → to search for the bad-blocks and
    - → to lock the bad-blocks.
  - ➢ Usually, data that resided on the bad-blocks are lost.
- A typical bad-sector transaction might be as follows:

  1) The operating system tries to read logical block 87.

  2) The controller calculates the ECC and finds that the sector is bad. It reports this finding to the operating system.

  3) The next time the system is rebooted, a special command is run to tell the controller to replace the bad sector with a spare.

  4) After that, whenever the system requests logical block 87, the request is translated into the replacement sector's address by the controller.

5.6 Swap Space Management

• Swap-space management is a low-level task of the operating system.
• Virtual memory uses disk space as an extension of main memory.
• Main goal of swap space: to provide the best throughput for the virtual memory system.
• Here, we discuss about 1) Swap space use 2) Swap space location.

5.6.1 Swap-Space Use

• Swap space can be used in 2 ways.
1) Swapping-Systems may use swap space to hold an entire process image, including the code and data segments.
2) Paging-systems may simply store pages that have been pushed out of main memory.

• The amount of swap space needed on a system can therefore vary from a few megabytes of disk space to gigabytes, depending on
→ amount of physical memory,
→ amount of virtual memory it is backing, and
→ way in which the virtual memory is used.

5.6.2 Swap-Space Location

• A swap space can reside in one of two places:
1) The swap space can be a large file within the file system.
➢ Here, normal file-system routines can be used to create it, name it, and allocate its space.
➢ Advantage: This approach easy to implement,
➢ Disadvantage: This approach is inefficient. This is because
i) Navigating the directory structure and the disk structures takes time and extra disk accesses.
ii) External fragmentation can greatly increase swapping times by forcing multiple seeks during reading or writing of a process image.
2) The swap space can be in a separate raw (disk) partition.
➢ No file system or directory structure is placed in the swap space.
Rather, a separate swap-space storage manager is used to allocate and de-allocate the blocks from the raw partition.
➢ This manager uses algorithms optimized for speed rather than for storage efficiency, because swap space is accessed much more frequently than file system.

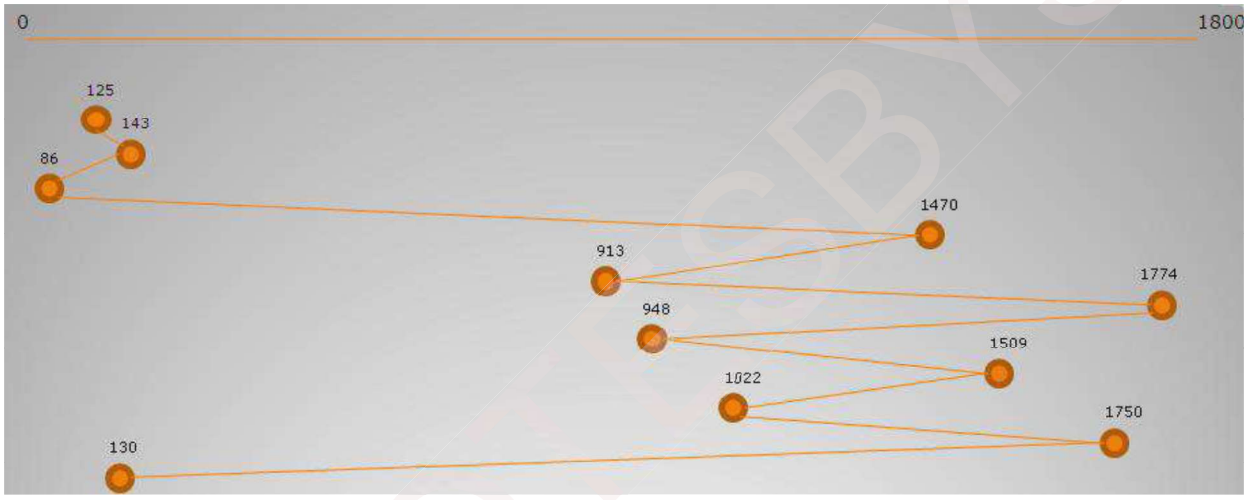
Exercise Problems

1) Suppose that the disk-drive has 5000 cylinders numbered from 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests in FIFO order is 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130. Starting from the current (location) head position, what is the total distance (in cylinders) that the disk-arm moves to satisfy all the pending requests, for each of the following disk-scheduling algorithms?

(i) FCFS
(ii) SSTF
(iii) SCAN
(iv) LOCK
(v) C-SCAN

Solution:

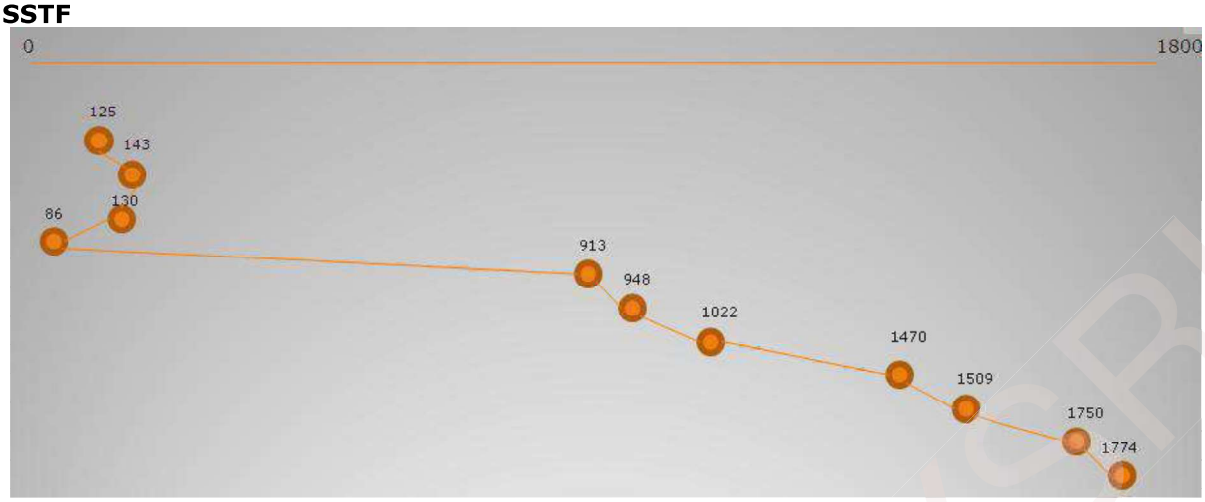
(i) FCFS



From cylinder	To cylinder	Seek Time
143	86	57
86	1470	1384
1470	913	557
913	1774	861
1774	948	826
948	1509	561
1509	1022	487
1022	1750	728
1750	130	1620
Total Seek Time		7081

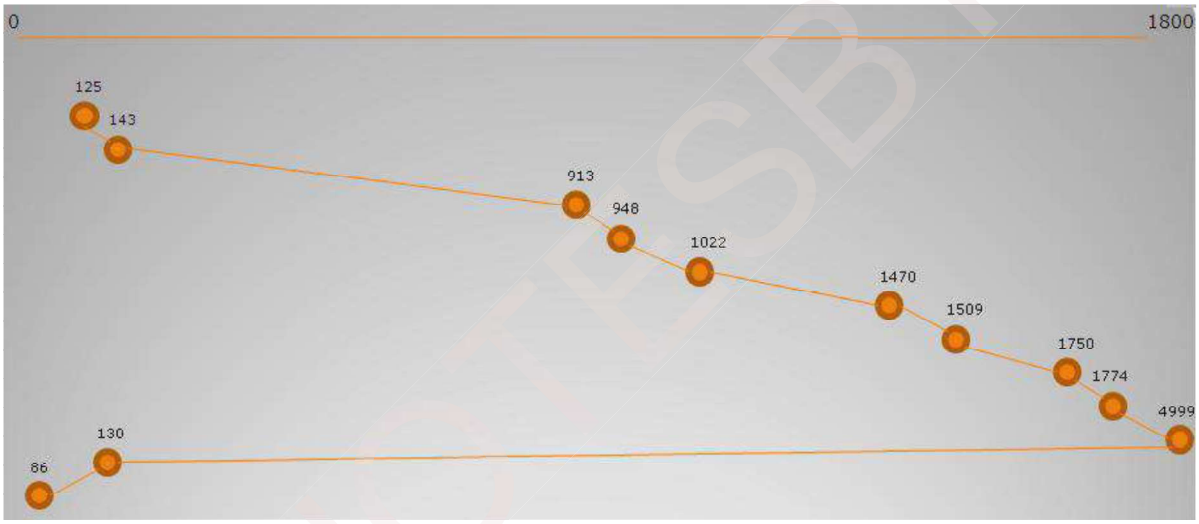
For FCFS schedule, the total seek distance is 7081.

(ii) SSTF



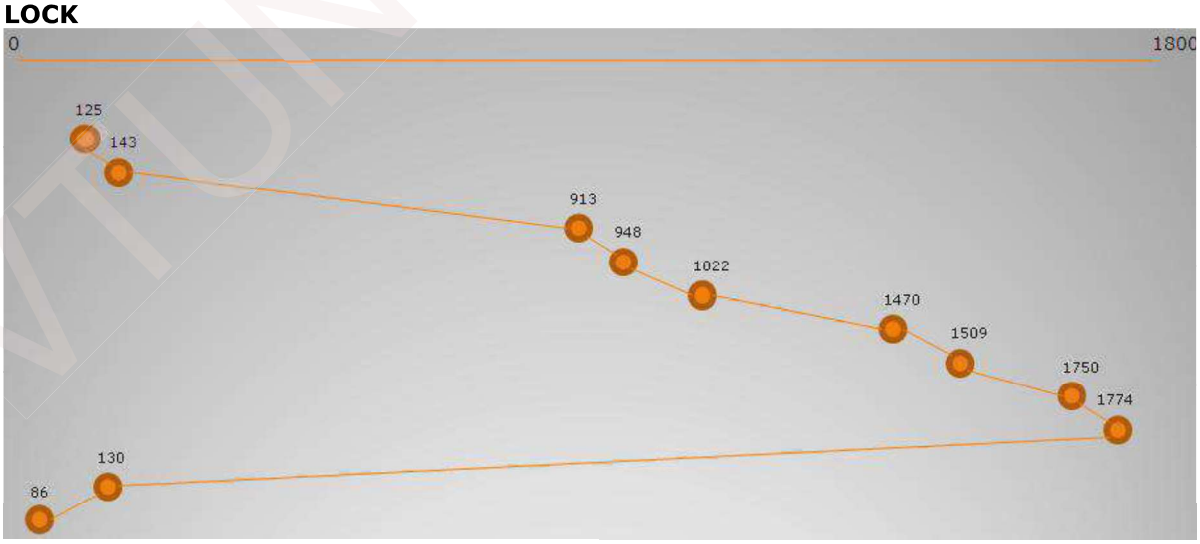
For SSTF schedule, the total seek distance is 1745.

(iii) SCAN



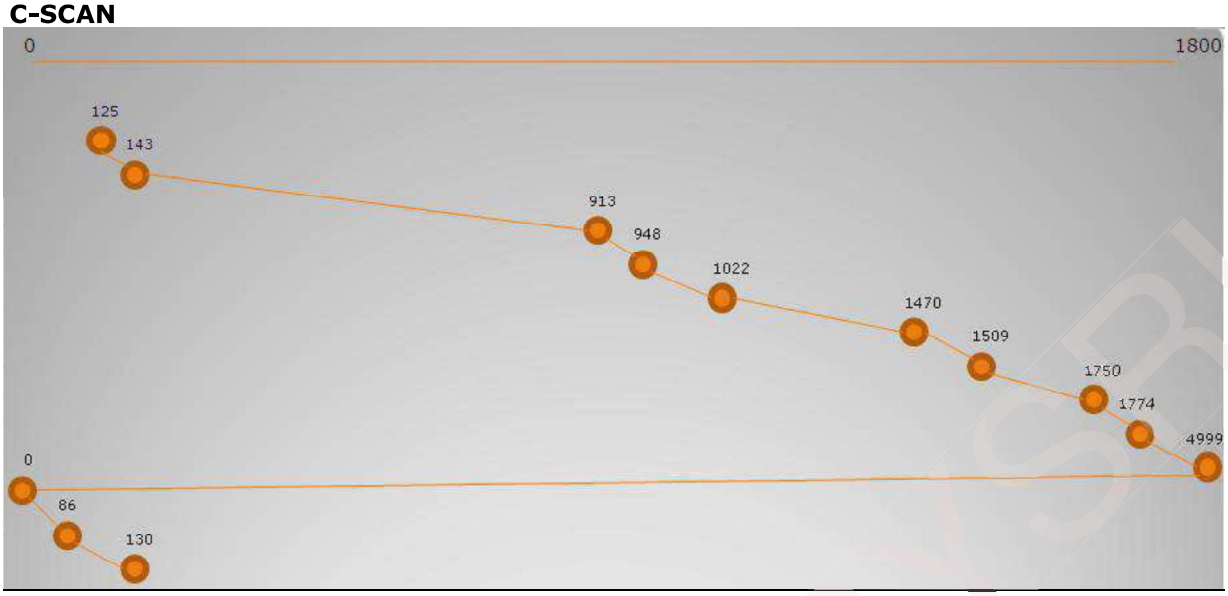
For SCAN schedule, the total seek distance is 9769.

(iv) LOCK



For LOOK schedule, the total seek distance is 3319.

(v) C-SCAN



For C-SCAN schedule, the total seek distance is 9813.

2) Suppose that a disk has 50 cylinder named 0 to 49. The R/W head is currently serving at cylinder 15. The queue of pending request are in order: 4 40 11 35 7 14 starting from the current head position, what is the total distance traveled (in cylinders) by the disk-arm to satisfy the request using algorithms

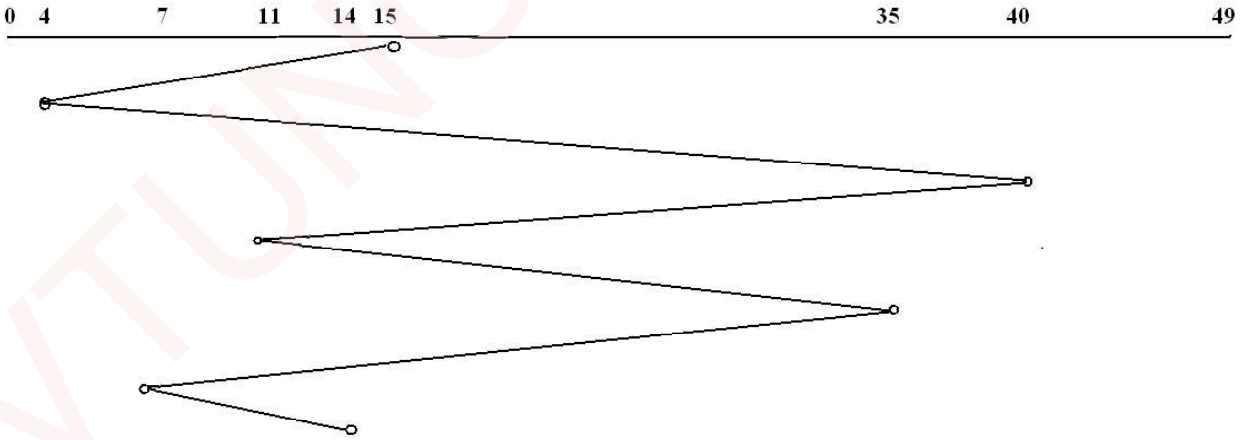
i) FCFS
ii) SSTF and
iii) LOOK.

Illustrate with figure in each case.

Solution:

(i) FCFS

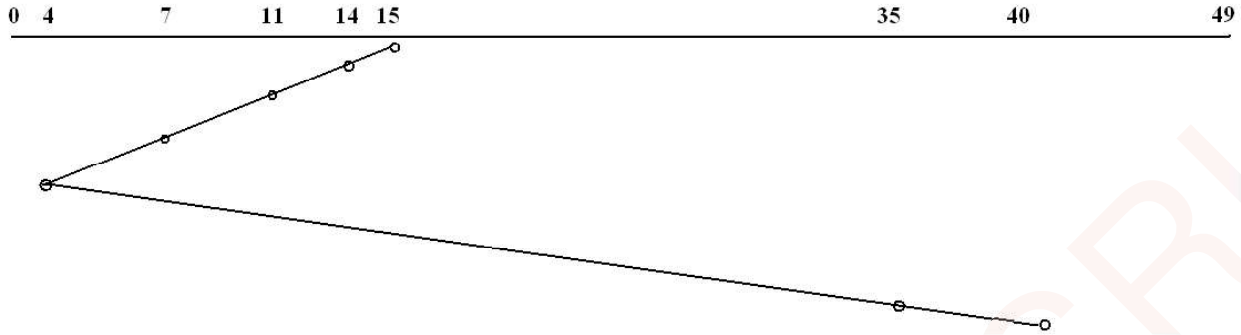
Queue: 4 40 11 35 7 14
Head starts at 15



For FCFS schedule, the total seek distance is 135

(ii) SSTF

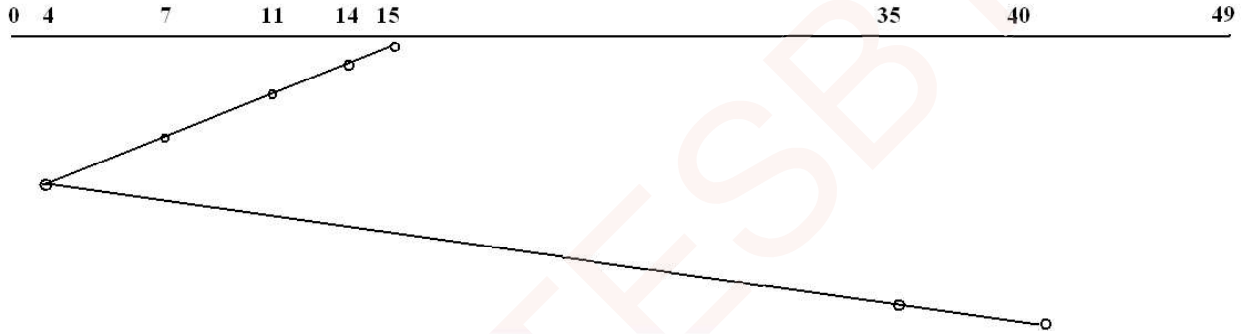
Queue: 4 40 11 35 7 14
Head starts at 15



For SSTF schedule, the total seek distance is 47.

(iii) LOOK

Queue: 4 40 11 35 7 14
Head starts at 15



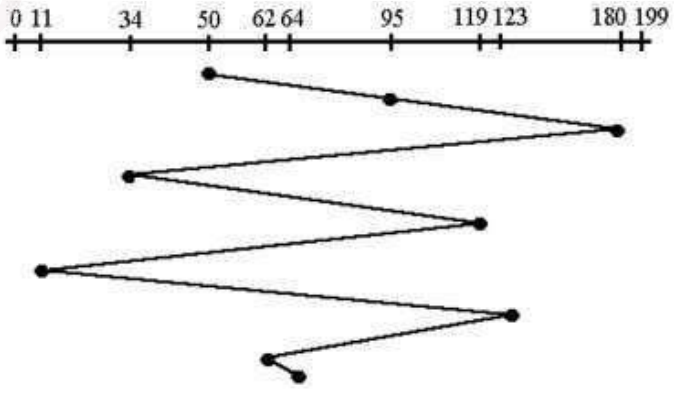
For LOOK schedule, the total seek distance is 47.

3) 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 moves using

i) FCFS
ii) SSTF
iii) Elevator and
iv) C-look.

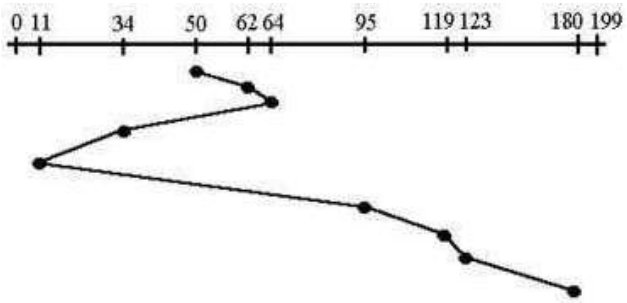
Solution:

(i) FCFS



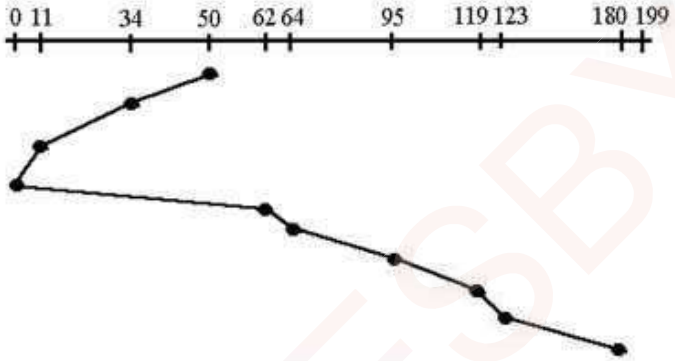
For FCFS schedule, the total seek distance is 640.

(ii) SSTF



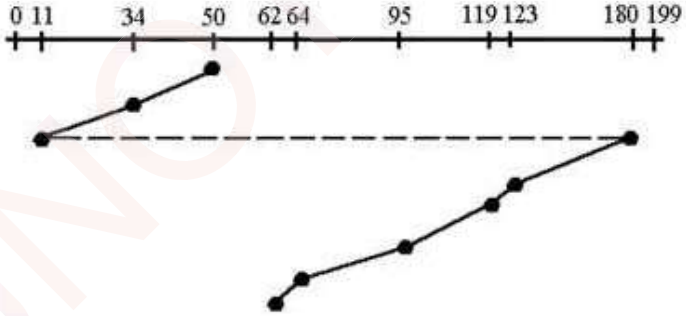
For SSTF schedule, the total seek distance is 236.

(iii) Elevator (SCAN)



For SCAN schedule, the total seek distance is 230.

(iv) C LOOK



For C-LOOK schedule, the total seek distance is 157.

MODULE 5 (CONT.): PROTECTION

5.7 Protection vs. Security

Protection

• Protection controls access to the system-resources by
  → Programs
  → Processes or
  → Users.
• Protection ensures that only processes that have gained proper authorization from the OS can operate on
  → memory-segments
  → CPU and
  → other resources.
• Protection must provide
  → means for specifying the controls to be imposed
  → means of enforcing the controls.
• Protection is an internal problem. Security, in contrast, must consider both the computer-system and the environment within which the system is used.

Security

• Security ensures the authentication of system-users to protect
  → integrity of the information stored in the system (both data and code)
  → physical resources of the computer-system.
• The security-system prevents
  → unauthorized access
  → malicious destruction
  → alteration of data or
  → accidental introduction of inconsistency.

5.8 Goals of Protection

• Operating system consists of a collection of objects, hardware or software.
• Each object has a unique name and can be accessed through a well-defined set of operations.
• Protection problem:
  ensure that each object is accessed correctly & only by those processes that are allowed to do so.
• Reasons for providing protection:
  1) To prevent mischievous violation of an access restriction.
  2) To ensure that each program component active in a system uses system resources only in ways consistent with policies.
• Mechanisms are distinct from policies:
  1) Mechanisms determine how something will be done.
  2) Policies decide what will be done.
• This principle provides flexibility.

5.9 Principles of Protection

• A key principle for protection is the principle of least privilege.
• Principle of Least Privilege:
"Programs, users, and even systems are given just enough privileges to perform their tasks".
• The principle of least privilege can help produce a more secure computing environment.
• An operating system which follows the principle of least privilege implements its features, programs, system-calls, and data structures.
• Thus, failure of a component results in minimum damage.
• An operating system also provides system-calls and services that allow applications to be written with fine-grained access controls.
• Access Control provides mechanisms
→ to enable privileges when they are needed.
→ to disable privileges when they are not needed.
• Audit-trails for all privileged function-access can be created.
• Audit-trail can be used to trace all protection/security activities on the system.
• The audit-trail can be used by
→ Programmer
→ System administrator or
→ Law-enforcement officer.
• Managing users with the principle of least privilege requires creating a separate account for each user, with just the privileges that the user needs.
• Computers implemented in a computing facility under the principle of least privilege can be limited to
→ running specific services
→ accessing specific remote hosts via specific services
→ accessing during specific times.
• Typically, these restrictions are implemented through enabling or disabling each service and through using Access Control Lists.

5.10 Domain of Protection

- A process operates within a protection domain.
- Protection domain specifies the resources that the process may access.
- Each domain defines
  → set of objects and
  → types of operations that may be invoked on each object.
- The ability to execute an operation on an object is an access-right.
- A domain is a collection of access-rights.
- The access-rights are an ordered pair <object-name, rights-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 on file F.
- As shown in Figure 5.9, domains may share access-rights. The access-right <O4, {print}> is shared by D2 and D3.

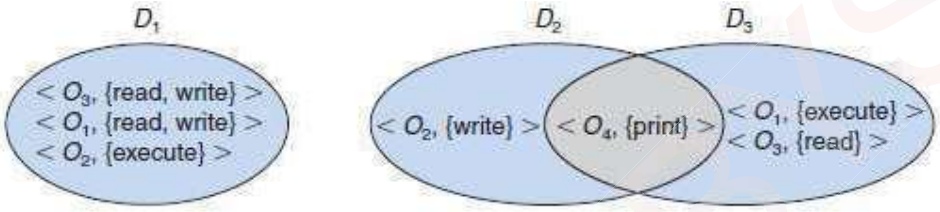


Figure 5.9 System with three protection domains.

- The association between a process and a domain may be either static or dynamic.
  1) If the association between processes and domains is static, then a mechanism must be available to change the content of a domain.
  ➢ Static means the set of resources available to the process is fixed throughout the process's lifetime.
  2) If the association between processes and domains is dynamic, then a mechanism is available to allow domain switching.
  ➢ Domain switching allows the process to switch from one domain to another.
- A domain can be realized in a variety of ways:
  1) Each user may be a domain.
  2) Each process may be a domain.
  3) Each procedure may be a domain.

5.10.1 Domain Structure

- A protection domain specifies the resources a process may access
- A domain is a collection of access rights, each of which is an ordered pair <object-name, rights-set>
- Access right = the ability to execute an operation on an object
  Access-right = <object-name, rights-set>
  where rights-set is a subset of all valid operations that can be performed on the object.
- Domains also define the types of operations that can be invoked.
- The association between a process and a domain may be
  1) Static (if the process' life-time resources are fixed): Violates the need-to-know principle
  2) Dynamic: A process can switch from one domain to another.
- A domain can be realized in several ways:
  1) Each user may be a domain
  ➢ Domain switching occurs when a user logs out.
  2) Each process may be a domain
  ➢ Domain switching occurs when a process sends a message to another process and waits for a response
  3) Each procedure may be a domain
  ➢ Domain switching occurs when a procedure call is made

5.11 Access Matrix

- Access-matrix provides mechanism for specifying a variety of policies.
- The access matrix is used to implement policy decisions concerning protection.
- In the matrix, 1) Rows represent domains.
  2) Columns represent objects.
  3) Each entry consists of a set of access-rights (such as read, write or execute).
- In general, Access(i, j) is the set of operations that a process executing in $Domain_i$ can invoke on $Object_j$
- Example: Consider the access matrix shown in Figure 5.10.
  - There are
    1) Four domains: D_1, D_2, D_3, and D_4
    2) Three objects: F_1, F_2 and F_3
  - A process executing in domain D_1 can read files F_1 and F_3.

domain \ object	F_1	F_2	F_3
D_1	read		read
D_2			
D_3		read	execute
D_4	read write		read write

Figure 5.10 Access matrix

- Domain switching allows the process to switch from one domain to another.
- When we switch a process from one domain to another, we are executing an operation (switch) on an object (the domain)
- We can include domains in the matrix to control domain switching.
- Consider the access matrix shown in Figure 5.11.
  - A process executing in domain D_2 can switch to domain D_3 or to domain D_4.

domain \ object	F_1	F_2	F_3	D_1	D_2	D_3	D_4
D_1	read		read		switch		
D_2						switch	switch
D_3		read	execute				
D_4	read write		read write	switch			

Figure 5.11 Access matrix with domains as objects

- Allowing controlled change in the contents of the access-matrix entries requires 3 additional operations (Figure 5.12):
  1) Copy(*) denotes ability for one domain to copy the access right to another domain.
  2) Owner denotes the process executing in that domain can add/delete rights in that column.
  3) Control in access(D_2,D_4) means: A process executing in domain D_2 can modify row D_4.

domain \ object	F_1	F_2	F_3	D_1	D_2	D_3	D_4
D_1	read		read*		switch		
D_2						switch	switch control
D_3		read owner	execute				
D_4	write		write	switch			

Figure 5.12 Access matrix with Copy rights, Owner rights & Control rights

- The problem of guaranteeing that no information initially held in an object can migrate outside of its execution environments is called the confinement problem.

5.12 Implementation of Access Matrix

5.12.1 Global Table

• A global table consists of a set of ordered triples <domain, object, rights-set>.
• Here is how it works:
  ➢ Whenever an operation M is executed on an object O_j within domain D_i, the global table is searched for a triple < D_i , O_j , R_k >, with M ∈ R_k.
  ➢ If this triple is found,
    Then, we allow the access operation;
    Otherwise, access is denied, and an exception condition occurs.
• Disadvantages:
  1) The table is usually large and can't be kept in main memory.
  2) It is difficult to take advantage of groupings, e.g. if all may read an object, there must be an entry in each domain.

5.12.2 Access Lists for Objects

• In the access-matrix, each column can be implemented as an access-list for one object.
• Obviously, the empty entries can be discarded.
• For each object, the access-list consists of ordered pairs <domain, rights-set>.
• Here is how it works:
  ➢ Whenever an operation M is executed on an object O_j within domain D_i, the access-list is searched for an entry < D_i , R_k>, with M ∈ R_k.
  ➢ If this entry is found,
    Then, we allow the access operation;
    Otherwise, we check the default-set.
      If M is in the default-set, we allow the access operation;
      Otherwise, access is denied, and an exception condition occurs.
• Advantages:
1) The strength is the control that comes from storing the access privileges along with each object
2) This allows the object to revoke or expand the access privileges in a localized manner.
• Disadvantages:
  1) The weakness is the overhead of checking whether the requesting domain appears on the access list.
  2) This check would be expensive and needs to be performed every time the object is accessed.
  3) Usually, the table is large & thus cannot be kept in main memory, so additional I/O is needed.
  4) It is difficult to take advantage of special groupings of objects or domains.

5.12.3 Capability Lists for Domains

• For a domain, a capability list is a list of objects & operations allowed on the objects.
• Often, an object is represented by its physical name or address, called a capability.
• To execute operation M on object O_j, the process executes the operation M, specifying the capability (or pointer) for object O_j as a parameter.
• The capability list is associated with a domain.
  But capability list is never directly accessible by a process.
    Rather, the capability list is maintained by the OS & accessed by the user only indirectly.
• Capabilities are distinguished from other data in two ways:
  1) Each object has a tag to denote whether it is a capability or accessible data.
  2) Program address space can be split into 2 parts.
    i) One part contains normal data, accessible to the program.
    ii) Another part contains the capability list, accessible only to the OS

5.12.4 A Lock–Key Mechanism

• The lock–key scheme is a compromise between 1) Access-lists and 2) Capability lists.
• Each object has a list of unique bit patterns, called locks.
• Similarly, each domain has a list of unique bit patterns, called keys.
• A process executing in a domain can access an object only if that domain has a key that matches one of the locks of the object.

5.13 Access Control

- Protection can be applied to non-file resources (Figure 5.13).
- Solaris 10 provides role-based access control (RBAC) to implement least privilege.
- Privilege is right to execute system call or use an option within a system call.
- Privilege can be assigned to processes.
- Users assigned roles granting access to privileges and programs

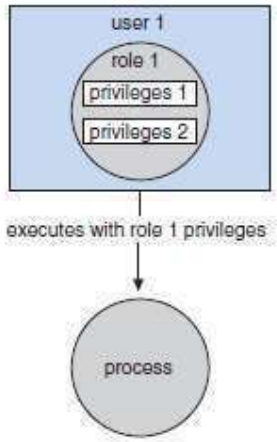


Figure 5.13 Role-based access control in Solaris 10.

5.14 Revocation of Access Rights

- In a dynamic protection system, we may sometimes need to revoke access rights to objects shared by different users.
- Following questions about revocation may arise:

1) Immediate versus Delayed

- Does revocation occur immediately, or is it delayed?
- If revocation is delayed, can we find out when it will take place?

2) 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?

3) 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?

4) 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?

- Schemes that implement revocation for capabilities include the following:

1) Reacquisition

- Periodically, capabilities are deleted from each domain.
- The process may then try to reacquire the capability.

2) Back-Pointers

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

3) Indirection

- 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.

4) Keys

- A key is associated with each capability and can't be modified / inspected by the process owning the capability
- Master key is associated with each object; can be defined or replaced with the set-key operation

MODULE 5 (CONT.): THE LINUX SYSTEM

5.15 Linux History

- Linux is a free OS-based on UNIX standards. (Linus + Unix= Linux)
- Linux is an open source software i.e. source-code is made available free on the Internet.
- Linux was first developed as a small self-contained kernel in 1991 by Linus Torvalds.
- Major design-goal of Linux project: UNIX-compatibility.
- In early days, Linux-development revolved largely around the central OS kernel.
- Kernel
  - → manages all system-resources and
  - → interacts directly with the computer-hardware.
- Linux-Kernel vs Linux-System:
  1) Linux-Kernel is an original piece of software developed from scratch by Linux community
  2) Linux-System includes a large no. of components, where
     - → some components are written from scratch and
     - → some components are borrowed from other development-projects and
- A Linux-Distribution includes
  - → all the standard components of the Linux-System
  - → set of administrative-tools to install/remove other packages on the system.

5.15.1 Linux Kernel

- Linux version 0.01 was released on 1991.
  Main features:
  - → ran only on 80386-compatible Intel processors and PC hardware.
  - → support for extremely limited device-driver.
  - → support for only the Minix file-system.
- Linux 1.0 was released on 1994.
  Main features:
  - → support for UNIX's standard TCP/IP networking-protocols.
  - → support for device-driver to run IP over an Ethernet.
  - → support for a range of SCSI controllers for high-performance disk-access
- Linux 1.2 was released on 1995.
  Main features:
  - → first PC-only Linux-Kernel
  - → support for a new PCI hardware bus architecture.
  - → support for dynamically loadable and unloadable kernel modules.
- Linux 2.0 was released on 1996.
  Main features:
  - → support for multiple architectures.
  - → support for symmetric multiprocessing (SMP).
  - → support for the automatic loading of modules on-demand.
- Linux 2.2 was released on in 1999.
  Main features:
  - → Networking was enhanced with i) firewall, ii) improved routing/traffic management
- Improvements continued with the release of Linux 2.4 and 2.6.
  Main features:
  - → added journaling file-system.
  - → support for pre-emptive kernel, 64-bit memory.
- Linux 3.0 was released in 2011.
  Main features:
  - → support for improved virtualization.
  - → facility for new page write-back
  - → improvement to the memory-management system

5.15.2 Linux-System

• Linux uses many tools developed as part of
  → Berkeley's BSD OS
  → MIT's X Window-System and
  → Free Software Foundation's GNU project.
• Main system-libraries are created by GNU project.
• Linux networking-administration tools are derived from 4.3BSD code.
• Linux-System is maintained by a many developers collaborating over the Internet.
• A small groups or individuals are responsible for maintaining the integrity of specific components.
• Linux community is responsible for maintaining the File-system Hierarchy Standard.
• This standard ensures compatibility across the various system-components.

5.15.3 Linux-Distributions

• Linux-Distributions include
  → system-installation and management utilities
  → ready-to-install packages of common UNIX tools (ex: text-processing, web browser).
• The first distributions managed these packages by simply providing a means of unpacking all the files into the appropriate places.
• Early distributions included SLS and Slackware.
• RedHat and Debian are popular distributions from commercial and non-commercial sources, respectively.
• RPM Package file format permits compatibility among the various Linux-Distributions.

5.15.4 Linux Licensing

• The Linux-Kernel is distributed under the GNU General Public License (GPL).
• Linux is not public-domain software.
• Public domain implies that the authors have waived copyright rights in the software.
• Linux is free software i.e. the people can copy it, modify it, use it.
• Anyone 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.

5.16 Design Principles

- Linux is a multiuser multitasking-system with a full set of UNIX-compatible tools.
- Linux's file-system follows traditional UNIX semantics.
- The standard UNIX networking model is fully implemented.
- Main design-goals are speed, efficiency, and standardization
- Linux is designed to be compliant with the relevant POSIX documents; at least two Linux-Distributions have achieved official POSIX certification.

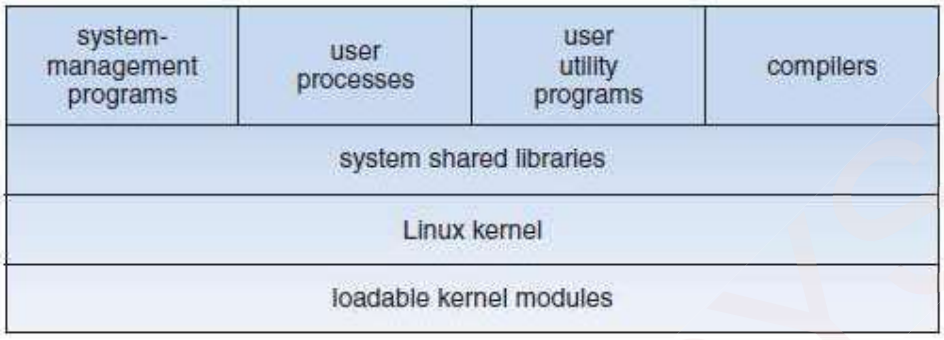


Figure 5.14 Components of the Linux-System

5.16.1 Components of a Linux-System

- The Linux-System is composed of 3 main bodies of code (Figure 5.14):

1) Kernel

- The kernel is responsible for maintaining all the important abstractions of the OS.
- The abstractions include i) virtual-memory and ii) processes.

2) 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.
- The most important system-library is the C library, known as libc.
- libc implements
  - → user-mode side of the Linux-System-call interface, and
  - → other critical system-level interfaces.

3) System Utilities

- The system utilities perform individual, specialized management tasks.
- Some system utilities are invoked just once to initialize and configure some aspect of the system.
- Other daemons run permanently, handling such tasks as
  - → responding to incoming network connections
  - → accepting logon requests from terminals, and
  - → updating log files.

- The system can operate in 2 modes: 1) Kernel mode 2) User-mode.

Sr. No.	Kernel Mode	User-Mode
1	All the kernel-code executes in the processor's privileged mode with full access to all the physical resources of the computer. This privileged mode called as kernel mode	Any operating-system-support code that does not need to run in kernel mode is placed into the system-libraries and runs in user-mode.
2	No user code is built into the kernel.	User-mode has access only to a controlled subset of the system's resources.

5.17 Kernel Modules

• A kernel module can implement a device-driver, a file-system, or a networking protocol.
• The kernel's 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.
• The module support has 3 components:
1) Module Management
2) Driver Registration
3) Conflict Resolution

5.17.1 Module Management

• Allows modules to be loaded into memory.
• Allows modules to communicate with the rest of the kernel.
• Module loading is split into 2 separate sections:
1) The management of sections of module code in kernel-memory.
2) The handling of symbols that modules are allowed to reference.
• Linux maintains an internal symbol table in the kernel.
• This symbol table
→ does not contain the full set of symbols defined in the kernel.
→ contains a set of symbol that must be explicitly exported.
• The set of exported symbols constitutes a well-defined interface by which a module can interact with the kernel.
• The loading of the module is performed in two stages.
1) Module Loader Utility asks the kernel to reserve a continuous area of virtual kernel memory for the module.
➢ The kernel returns the address of the memory allocated.
➢ The loader utility can use this address to relocate the module's machine code to the correct loading address.
2) 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.
➢ It will unload the module when it is no longer actively needed.

5.17.2 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.
• The kernel calls a module's startup routine when that module is loaded.
The kernel calls the module's cleanup routine before that module is unloaded.
• Registration tables include the following items:
1) Device-Drivers
➢ These drivers include character devices (such as printers, terminals, and mice), block devices (including all disk drives), and network interface devices.
2) File-systems
➢ The file-system implements Linux's virtual file-system-calling routines.
3) Network Protocols
➢ A module may implement an entire networking protocol, such as TCP or simply a new set of packet-filtering rules for a network firewall.
4) Binary Format
➢ This format specifies a way of recognizing, loading, and executing a new type of executable file.

5.17.3 Conflict Resolution

- Allows different device-drivers to
  - → reserve hardware resources and
  - → protect the resources from accidental use by another driver.
- Its aims are as follows:
  1) To prevent modules from clashing over access to hardware resources.
  2) To prevent autoprobes from interfering with existing device-drivers.
  3) To resolve conflicts among multiple drivers trying to access the same hardware.

5.18 Process management

5.18.1 The fork() and exec() Process Model

• UNIX process management separates the creation of processes and the running of a new program into two distinct operations.
• A new process is created by the fork() system-call. A new program is run after a call to exec().
• Process properties fall into 3 groups: 1) Process identity 2) Environment and 3) Context.

5.18.1.1 Process Identity

• A process identity consists mainly of the following items:

1) Process ID (PID)

• Each process has a unique identifier.
• The PID is used to specify the process to the OS when an application makes a system-call to signal, modify, or wait for the process.

2) 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.

3) Personality

• 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.

4) Namespace

• Each process is associated with a specific view of the file-system hierarchy, called its namespace.
• Most processes share a common namespace & thus operate on a shared file-system hierarchy.
• However, processes and their children can have different namespaces.

5.18.1.2 Process Environment

• Process's environment is inherited from its parent & is composed of 2 null-terminated vectors:

1) Argument vector simply lists the command-line arguments used to invoke the running program.

2) Environment vector is a list of "NAME=VALUE" pairs that associates named environment variables with arbitrary textual values.

5.18.1.3 Process Context

• Process context is the state of the running program at any one time; it changes constantly.
• Process context includes the following parts:

1) Scheduling Context

• Scheduling context refers to the info. scheduler needs to suspend & restart the process.
• This information includes saved copies of all the process's registers.
• The scheduling context also includes information about
  → scheduling priority and
  → any outstanding signals waiting to be delivered to the process.

2) Accounting

• The kernel maintains accounting information about
  → resources currently being consumed by each process and
  → total resources consumed by the process in its entire lifetime.

3) File Table

• The file table is an array of pointers to kernel file structures representing open files.
• When making file-I/O system-calls, processes refer to files by a file descriptor(fd) that the kernel uses to index into this table.

4) File-System Context

• File-system context includes the process's root directory, current working directory, and namespace.

5) Signal-Handler Table

• The signal-handler table defines the action to take in response to a specific signal.
• Valid actions include ignoring the signal, terminating the process, and invoking a routine in the process's address-space.

6) Virtual-Memory Context

• Virtual-memory context describes the full contents of a process's private address-space.

5.18.2 Processes and Threads

• Linux provides the ability to create threads via the clone() system-call.
• The clone() system-call behaves identically to fork(), except that it accepts as arguments a set of flags.
• The flags dictate what resources are shared between the parent and child.
• The flags include:

flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

• If clone() is passed the above flags, the parent and child tasks will share
→ same file-system information (such as the current working directory)
→ same memory space
→ same signal handlers and
→ same set of open files.
However, if none of these flags is set when clone() is invoked, the associated resources are not shared
• A separate data-structures is used to hold information of process. Information includes:
→ file-system context
→ file-descriptor table
→ signal-handler table and
→ virtual-memory context
• The process data-structure contains pointers to these other structures.
• So any number of processes can easily share a sub-context by
→ pointing to the same sub-context and
→ incrementing a reference count.
• The arguments to the clone() system-call tell it
→ which sub-contexts to copy and
→ which sub-contexts to share.
• The new process is always given a new identity and a new scheduling context

5.19 Scheduling

- Scheduling is a process of allocating CPU-time to different tasks within an OS.
- Like all UNIX systems, Linux supports preemptive multitasking.
- In such a system, the process-scheduler decides which process runs and when.

5.19.1 Process Scheduling

- Linux uses 2 scheduling-algorithms:
  1) A time-sharing algorithm for fair preemptive-scheduling between multiple processes.
  2) A real-time algorithm where absolute priorities are more important than fairness.
- A scheduling-class defines which algorithm to apply
- The scheduler is a preemptive, priority-based algorithm with 2 separate priority ranges:
  1) Real-time range from 0 to 99 and
  2) Nice-value ranging from −20 to 19.
- Smaller nice-values indicate higher priorities.
- Thus, by increasing the nice-value, the priority is decreased and being "nice" to the rest of the system.
- Linux implements fair scheduling.
- All processes are allotted a proportion of the processor's time.

5.19.2 Real-Time Scheduling

- Linux implements the two real-time scheduling-classes:
  1) FCFS (First-Come, First Served) and
  2) Round-robin.
- In both cases, each process has a priority in addition to its scheduling-class.
- The scheduler always runs the process with the highest priority.
- Among processes of equal priority, the scheduler runs the process that has longest waiting-time.
- Difference between FCFS and round-robin scheduling:
  1) In FCFS, processes continue to run until they either exit or block.
  2) In round-robin, process will be preempted after a while and moved to the end of the scheduling-queue. Thus, processes of equal priority will automatically time-share among themselves.

5.19.3 Kernel Synchronization

• Two ways of requesting for kernel-mode execution:
  1) A running program may request an OS service, either
    → explicitly via a system-call or
    → implicitly when a page-fault occurs
  2) A device-driver may deliver a hardware-interrupt.
    The interrupt causes the CPU to start executing a kernel-defined handler.

• Two methods to protect critical-sections: 1) spinlocks and 2) semaphores.
  1) Spinlocks are used in the kernel only when the lock is held for short-durations.
    i) On SMP machines, spinlocks are the main locking mechanism used.
    ii) On single-processor machines, spinlocks are not used, instead kernel pre-emption are enabled and disabled.

single processor	multiple processors
Disable kernel preemption.	Acquire spin lock.
Enable kernel preemption.	Release spin lock.

  2) Semaphores are used in the kernel only when a lock must be held for longer periods.
  ➢ The second protection technique applies to critical-sections that occur in ISR (interrupt service routine).
  ➢ The basic tool is the processor's interrupt-control hardware.
  ➢ By disabling interrupts during a critical-section, the kernel guarantees that it can proceed without the risk of concurrent-access to shared data-structures.

• Kernel uses a synchronization architecture that allows long critical-sections to run for their entire duration without interruption.

• ISRs are separated into a top half and a bottom half (Figure 5.15):
  1) The top half is a normal ISR, and runs with recursive interrupts disabled.
  2) The bottom half is run, with all interrupts enabled, by a miniature-scheduler that ensures that bottom halves never interrupt themselves.

• This architecture is completed by a mechanism for disabling selected bottom halves while executing normal, foreground kernel-code.

• Each level may be interrupted by code running at a higher level, but will never be interrupted by code running at the same or a lower level.

• User-processes can always be preempted by another process when a time-sharing scheduling interrupt occurs.

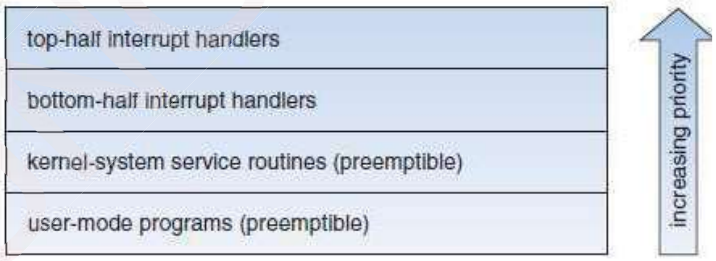


Figure 5.15 Interrupt protection levels

5.19.4 Symmetric Multiprocessing (SMP)

• Linux 2.0 kernel was the first stable Linux-Kernel to support SMP hardware,

• Separate processes can execute in parallel on separate processors.

• In Linux 2.2, a single kernel spinlock (called BKL for "big kernel lock") was created to allow multiple processes to be active in the kernel concurrently.

• However, the BKL provided a very coarse level of locking granularity. This resulted in poor scalability.

• Linux 3.0 provided additional SMP enhancements such as
  → ever-finer locking
  → processor affinity and
  → load-balancing.

5.20 Memory-Management

- Memory-management has 2 components:
  1) The first component is used for allocating and freeing physical-memory such as
     → groups of pages and
     → small blocks of RAM.
  2) The second component is used for handling virtual-memory.
     A virtual-memory is a memory-mapped into the address-space of running processes.

5.20.1 Management of Physical-Memory

- The memory is divided into 3 different zones (Figure 5.16):
  1) ZONE DMA
  2) ZONE NORMAL
  3) ZONE HIGHMEM

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

Figure 5.16 Relationship of zones and physical addresses in Intel x86-32.

- Page-allocator is used to
  → allocate and free all physical-pages.
  → allocate a ranges of physically-contiguous pages on-demand.
- Page-allocator uses a buddy-heap algorithm to keep track of available physical-pages (Figure 5.17).
- Each allocatable memory-region is paired with an adjacent partner (hence, the name buddy-heap).
  1) When 2 allocated partners regions are freed up, they are combined to form a larger region (called as a buddy heap).
  2) Conversely, if a small memory-request cannot be satisfied by allocation of an existing small free region, then a larger free region will be subdivided into two partners to satisfy the request.
- Memory allocations occur either
  → statically (drivers reserve a contiguous area of memory during system boot time) or
  → dynamically (via the page-allocator).

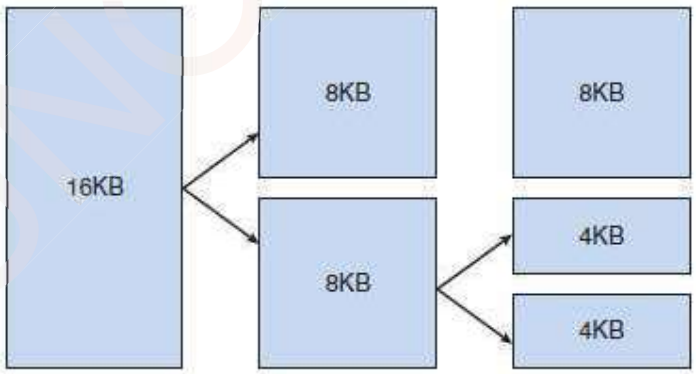


Figure 5.17 Splitting of memory in the buddy system.

- Slab-allocator is another strategy for allocating kernel-memory.
- A slab is made up of one or more physically contiguous pages.
- A cache consists of one or more slabs.
- In Linux, a slab will be in one of 3 possible states:
  1) Full: All objects in the slab are marked as used.
  2) Empty: All objects in the slab are marked as free.
  3) 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(Figure 5.18)
  1) If none exists, a free object is assigned from an empty slab.
  2) 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.

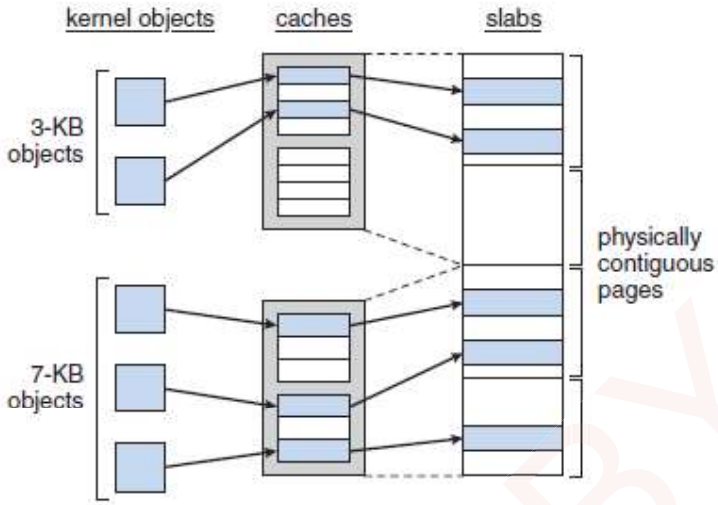


Figure 5.18 Slab-allocator in Linux.

5.20.2 Virtual-memory

• VM system
→ maintains the address-space visible to each process.
→ creates pages of virtual-memory on-demand and
→ loads those pages from disk & swaps them back out to disk as required.
• The VM manager maintains 2 separate views of a process's address-space:
1) Logical-view and 2) Physical-view

1) Logical-Vew

• Logical-view of a address-space refers to a set of separate regions.
• The address-space consists of a set of non-overlapping regions.
• Each region represents a continuous, page-aligned subset of the address-space.
• The regions are linked into a balanced binary-tree to allow fast lookup of the region.

2) Physical-View

• Physical-view of a address-space refers to a set of pages.
• This view is stored in the hardware page-tables for the process.
• The page-table entries identify the exact current location of each page of virtual-memory.
• Each page of virtual-memory may be on disk or in physical-memory.
• A set of routines manages the Physical-view.
• The routines are invoked whenever a process tries to access a page that is not currently present in the page-tables.

5.20.2.1 Virtual-Memory-Regions

• Virtual-memory-regions can be classified by backing-store.
• Backing-store defines from where the pages for the region come.
• Most memory-regions are backed either 1) by a file or 2) by nothing.

1) By Nothing

➢ Here, a region is backed by nothing.
➢ The region represents demand-zero memory.
➢ When a process reads a page in the memory, the process is returned a page-of-memory filled with zeros.

2) By File

➢ A region backed by a file acts as a viewport onto a section of that file.
➢ When the process tries to access a page within that region, the page-table is filled with the address of a page within the kernel's page-cache.
➢ The same page of physical-memory is used by both the page-cache and the process's page tables.

• A virtual-memory-region can also be classified by its reaction to writes. 1) Private or 2) Shared.
1) If a process writes to a private-region, then the pager detects that a copy-on-write is necessary to keep the changes local to the process.
2) If a process writes to a shared-region, the object mapped is updated into that region.
➢ Thus, the change will be visible immediately to any other process that is mapping that object.

5.20.2.2 Lifetime of a Virtual Address-Space

• Under following 2 situations, the kernel creates a new virtual address-space:
1) When a process runs a new program with the exec() system-call.
➢ When a new program is executed, the process is given a new, completely empty virtual address-space.
➢ It is up to the routines to populate the address-space with virtual-memory-regions.
2) When a new process is created by the fork() system-call.
➢ Here, a complete copy of the existing process's virtual address-space is created.
➢ The parent's page-tables are copied directly into the child's page-tables.
➢ Thus, after the fork, the parent and child share the same physical-pages of memory in their address-spaces.

5.20.2.3 Swapping and Paging

• A VM system relocates pages of memory from physical-memory out to disk when that memory is needed.
• Paging refers to movement of individual pages of virtual-memory between physical-memory & disk.
• Paging-system is divided into 2 sections:
    1) Policy algorithm decides
        → which pages to write out to disk and
        → when to write those pages.
    2) Paging mechanism
        → carries out the transfer and
        → pages data back into physical-memory when they are needed again.
• Linux's pageout policy uses a modified version of the standard clock algorithm.
• A multiple pass clock is used, and every page has an age that is adjusted on each pass of the clock.
• The age is 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. (LFU → least frequently used)
• This age valuing allows the pager to select pages to page out based on a LFU policy.
• The paging mechanism supports paging both to
    1) dedicated swap devices and partitions and
    2) normal files
• Blocks are allocated from the swap devices according to a bitmap of used blocks, which is maintained in physical-memory at all times.
• The allocator uses a next-fit algorithm to try to write out pages to continuous runs of disk blocks for improved performance.

5.20.2.4 Kernel Virtual-Memory

• Linux reserves a constant, architecture-dependent region of the virtual address-space of every process for its own internal use.
• The page-table entries that map to these kernel pages are marked as protected.
• Thus, the pages are not visible or modifiable when the processor is running in user-mode.
• The kernel virtual-memory area contains two regions.
    1) A static area contains page-table references to every available physical-page of memory in the system.
    ➢ Thus, a simple translation from physical to virtual addresses occurs when kernel-code is run.
    ➢ The core of the kernel, along with all pages allocated by the normal page-allocator, resides in this region.
    2) The remainder of the 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.

5.20.3 Execution and Loading of User Programs

• Linux
    → maintains a table of possible loader-functions
    → gives loader-function the opportunity to try loading the given file when an exec() system-call is made.
• The registration of multiple loader routines allows Linux to support both the ELF and a.out binary formats.
• ELF has a number of advantages over a.out:
    1) Flexibility and extendability.
    2) New sections can be added to ELF w/o causing the loader routines to become confused.

5.20.3.1 Mapping of Programs into Memory

- Initially, the pages of the binary-file are mapped into regions of virtual-memory.
- Only when a program tries to access a given page, a page-fault occurs.
- Page-fault results in loading the requested-page into physical-memory.
- An ELF-format binary-file consists of a header followed by several page-aligned sections.
- The ELF loader
  - → reads the header and
  - → maps the sections of the file into separate regions of virtual-memory.
- As shown in Figure 5.19
  - ➢ Kernel VM is not accessible to normal user-mode programs.
  - ➢ Job of loader: To set up the initial memory mapping to begin the execution of the program.
  - ➢ The regions to be initialized include 1) stack and 2) program's text/data regions.
  - ➢ The stack is created at the top of the user-mode virtual-memory.
  - ➢ The stack includes copies of the arguments given to the program.
  - ➢ In the binary-file,
    - ¤ Firstly, program-text or read-only data are mapped into a write-protected region.
    - ¤ Then, writable initialized data are mapped.
    - ¤ Then, any uninitialized data are mapped in as a private demand-zero region.
    - ¤ Finally, we have a variable-sized region that programs can expand as needed to hold data allocated at run time.
  - ➢ Each process has a pointer brk that points to the current extent of this data region,

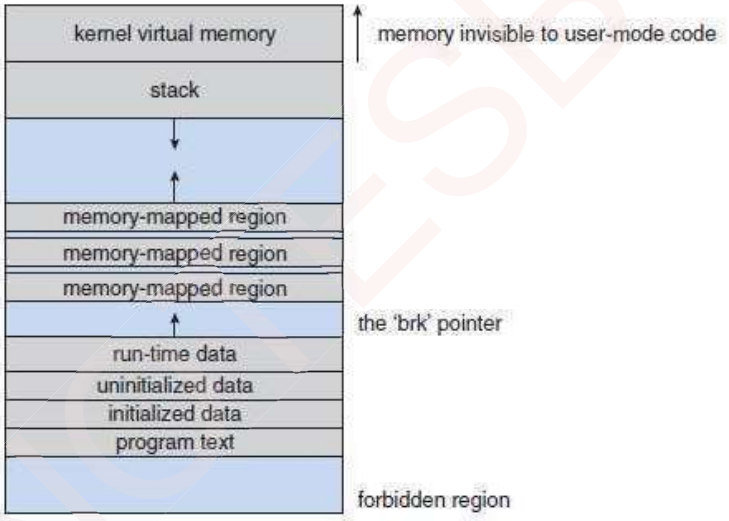


Figure 5.19 Memory layout for ELF programs

5.20.3.2 Static and Dynamic Linking

- A program is statically linked to its libraries if the necessary library-functions are embedded directly in the program's executable binary-file.
- Disadvantage of Static Linking:
  Every program generated must contain copies of exactly the same common system-library functions.
- Advantage of Dynamic Linking:
  Dynamic linking is more efficient in terms of both physical-memory and disk-space usage. This is because the system-libraries are loaded into memory only once.
- 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
  - → 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

5.21 File-Systems

5.21.1 Virtual File-System

• The Linux VFS is designed around object-oriented principles.
• It has two components:
    1) A set of definitions that specify the file-system objects.
    2) A layer of software to manipulate the objects.
• The VFS defines 4 main object types:
    1) An inode object represents an individual file.
    2) A file-object represents an open file.
    3) A superblock object represents an entire file-system.
    4) A dentry object represents an individual directory entry.
• For each object type, the VFS defines a set of operations.
• Each object 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.
• Example of 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 located in the file /usr/include/linux/fs.h.
• An implementation of the file-object is required to implement each function specified in the definition of the file-object.
• The VFS software layer can perform an operation on the file-objects by calling the appropriate function from the object's function-table.
• The VFS does not know whether an inode represents
    → networked file
    → disk file
    → network socket, or
    → directory file.
• The inode and file-objects are the mechanisms used to access files.
• An inode object is a data-structure containing pointers to the disk blocks that contain the actual file contents.
• The inode also maintains standard information about each file, such as
    → owner
    → size and
    → time most recently modified.
• A file-object represents a point of access to the data in an open file.
• A process cannot access an inode's contents without first obtaining a file-object pointing to the inode.
• The file-object keeps track of where in the file the process is currently reading/writing.
• File-objects typically belong to a single process, but inode objects do not.
• There is one file-object for every instance of an open file, but always only a single inode object.
• Directory files are dealt with slightly differently from other files.
• The UNIX programming interface defines a number of operations on directories, such as
    → creating file
    → deleting file and
    → renaming file.

5.21.2 Linux ext3 File-system

- Similar to BSD FFS, ext3 File-system locates the data blocks belonging to a specific file.
- The main differences between ext3 and FFS lie in their disk-allocation policies.
  1) In FFS, the disk is allocated to files in blocks of 8 KB. (FFS → Fast File-system)
  - The 8KB-blocks are further subdivided into fragments of 1 KB for storage of small files.

  2) In ext3, fragments are not used.
  - Allocations are performed in smaller units.
  - Supported block sizes are 1, 2, 4, and 8 KB.
- ext3 uses allocation policies designed to place logically adjacent blocks of a file into physically adjacent blocks on disk.
- Thus, ext3 can submit an I/O request for several disk blocks as a single operation.
- The allocation-policy works as follows (Figure 5.20):
  - An ext3 file-system is divided into multiple segments. These are called block-groups.
  - When allocating a file, ext3 first selects the block-group for that file.
  - Within a block-group, ext3 keeps the allocations physically contiguous to reduce fragmentation.
  - ext3 maintains a bitmap of all free blocks in a block-group.
    i) When allocating the first blocks for a new file, ext3 starts searching for a free block from the beginning of the block-group.
    ii) When extending a file, ext3 continues the search from the block most recently allocated to the file. The search is performed in 2 stages:
    1) First, ext3 searches for an entire free byte in the bitmap; if it fails to find one, it looks for any free bit.
    - The search for free bytes aims to allocate disk-space in chunks of at least 8 blocks.

    2) After a free block is found, the search is extended backward until an allocated block is encountered.
    - The backward extension prevents ext3 from leaving a hole.
  - The preallocated blocks are returned to the free-space bitmap when the file is closed.

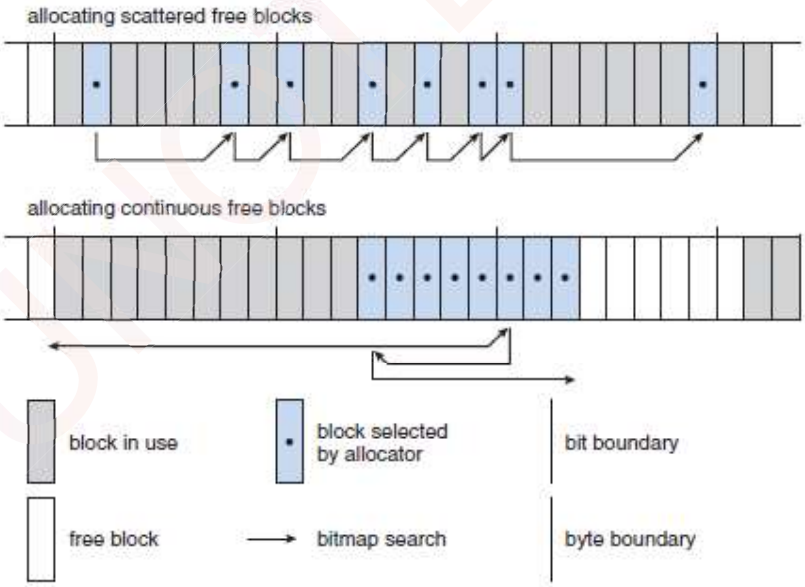


Figure 5.20 ext3 block-allocation policies.

5.21.3 Journaling

• ext3 file-system supports a popular feature called journaling.
• Here, modifications to the file-system are written sequentially to a journal.
• A set of operations that performs a specific task is a transaction.
• Once a transaction is written to the journal, it is considered to be committed.
• The journal entries relating to the transaction are replayed across the actual file-system structures.
• When an entire committed transaction is completed, it is removed from the journal.
• If the system crashes, some transactions may remain in the journal.
• If those transactions were never completed, then they must be completed once the system recovers.
• The only problem occurs when a transaction has been aborted i.e. it was not committed before the system crashed.
• Any changes from those transactions that were applied to the file-system must be undone, again preserving the consistency of the file-system.

5.21.4 Linux Process File-system (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.
• The /proc file-system must implement two things: a directory structure and the file contents within.
• proc must define a unique and persistent inode number for each directory and the associated files.
• proc uses this inode number to identify 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 are read from these files, proc will collect the appropriate information, format it into textual form, and place it into the requesting process's read buffer.
• The kernel can allocate new /proc inode mappings dynamically, maintaining a bitmap of allocated inode numbers.
• The kernel also maintains a tree data-structure of registered global /proc file-system entries.
• Each entry contains
  → file's inode number
  → file name and
  → access permissions
  → special functions used to generate the file's contents.
• Drivers can register and deregister entries in this tree at any time, and a special section of the tree is reserved for kernel variables.

5.22 Input and Output

• Three types of devices (Figure 5.21): 1) Block device
2) Character device and
3) Network device.

1) Block Devices

• Block devices allow random access to completely independent, fixed-sized blocks of data.
• For example: hard disks and floppy disks, CD-ROMs and Blu-ray discs, and flash memory.
• Block devices are typically used to store file-systems.

2) Character Devices

• A character-device-driver does not offer random access to fixed blocks of data.
• For example: mice and keyboards.
• Character devices include mice and keyboards.

3) Network Devices

• Users cannot directly transfer data to network devices.
• Instead, they must communicate indirectly by opening a connection to the kernel's networking subsystem.

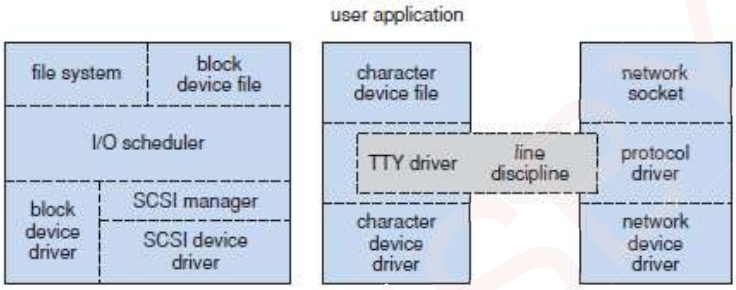


Figure 5.21 Device-driver block structure.

5.22.1 Block Devices

• Block devices allow random access to completely independent, fixed-sized blocks of data.
• For example: hard disks and floppy disks, CD-ROMs and Blu-ray discs, and flash memory.
• Block devices are typically used to store file-systems.
• Block devices provide the main interface to all disk devices in a system.
• A block represents the unit with which the kernel performs I/O.
• When a block is read into memory, it is stored in a buffer.
• The request manager is the layer of software that manages the reading and writing of buffer contents to and from a block-device-driver.
• A separate list of requests is kept for each block-device-driver.
• These requests are scheduled according to a C-SCAN algorithm.
• C-SCAN algorithm exploits the order in which requests are inserted in and removed from the lists.
• The request lists are maintained in sorted order of increasing starting-sector number.

5.22.2 Character Devices

• A character-device-driver does not offer random access to fixed blocks of data.
• For example: mice and keyboards.
• Difference between block and character devices:
i) block devices are accessed randomly, ii) character devices are accessed serially.
• A character device-driver must register a set of functions which implement the driver's various file I/O operations
• The kernel performs almost no preprocessing of a file read or write request to a character device.
• The kernel simply passes the request to the device.
• The main exception to this rule is the special subset of character device-drivers which implement terminal devices.
• The kernel maintains a standard interface to these drivers.
• A line discipline is an interpreter for the information from the terminal device.
• The most common line discipline is the tty discipline, which glues the terminal's data stream onto the standard input and output streams of a user's running processes.
• This allows the processes to communicate directly with the user's terminal.

5.23 Inter-Process Communication

• In some situations, one process needs to communicate with another process.
• Three methods for IPC:
    1) Synchronization and Signals
    2) Message Passing Data between Processes
    3) Shared Memory Object

5.23.1 Synchronization and Signals

• Linux informs processes that an event has occurred via signals.
• Signals can be sent from any process to any other process.
• There are a limited number of signals, and they cannot carry information.
• Only the fact that a signal has occurred is available to a process.
• The kernel also generates signals internally.
• The Linux-Kernel does not use signals to communicate with processes running in kernel mode.
    Rather, communication within the kernel is accomplished via scheduling states and wait_queue structures.
• Whenever a process wants to wait for some event to complete, the process
    → 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, every process on the wait queue will be awoken.
• This procedure allows multiple processes to wait for a single event.

5.23.2 Passing of Data among Processes

• The standard UNIX pipe mechanism allows a child process to inherit a communication channel from its parent.
• Data written to one end of the pipe can be read at the other.
• Shared memory offers an extremely fast way to communicate large or small amounts of data.
• Any data written by one process to a shared memory-region can be read immediately by any other process.
• Main disadvantage of shared memory:
    1) It offers no synchronization.
    2) A process cannot
        → ask the OS whether a piece of shared memory has been written or
        → suspend execution until the data is written.

5.23.3 Shared Memory Object

• The shared-memory object acts as a backing-store for shared-memory-regions, just as a file can act as a backing-store for a memory-mapped memory-region.
• Shared-memory mappings direct page-faults to map in pages from a persistent shared memory object.
• Also, shared memory objects remember their contents even if no processes are currently mapping them into virtual-memory.