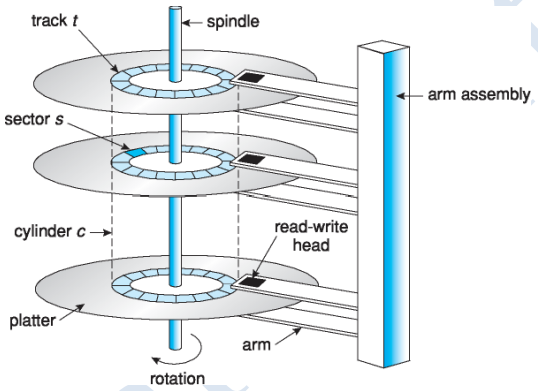
UNIT IV

I/O SYSTEMS

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.
- Common platter diameters range from 1.8 to 3.5 inches.
- The two surfaces of a platter are covered with a magnetic material.
- Information is stored by recording it magnetically, on the platters



Moving- head disk mechanism

- 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.
- The surface of a platter is logically divided into circular tracks
  - tracks are subdivided into sectors.
- The set of tracks that are at one arm position makes up a cylinder.
- The storage capacity of common disk drives is measured in gigabytes.
- When the disk is in use, a drive motor spins it at high speed.
  - Drives rotate 60 to 250 times per second
  - Specified in terms of rotations per minute (RPM).
  - Common drives spin at 5,400,7,200, 10,000, and 15,000RPM.
- Disk speed has two parts.
  - Transfer rate - rate at which data flow between the drive and the computer.
  - Positioning time, or random-access time, consists of two parts:
    - Seek time - time necessary to move the disk arm to the desired cylinder
    - Rotational latency - time necessary for the desired sector to rotate to disk head
- Head crash - as the disk head flies on an extremely thin cushion of air (measured in microns), there is a danger that the head will make contact with the disk surface, the head will sometimes damage the magnetic surface. This accident is called a head crash.
  - A head crash normally cannot be repaired; the entire disk must be replaced.

- A disk can be removable, allowing different disks to be mounted as needed.
- Removable magnetic disks generally consist of one platter, held in a plastic case to prevent damage while not in the disk drive.
- Other forms of removable disks include CDs, DVDs, and Blu-ray discs as well as removable flash-memory devices known as flash drives.
- 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 advanced technology attachment (ATA), serial ATA (SATA), eSATA, universal serial bus (USB), and fibre channel (FC).
- 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/O operation, the computer places a command into the host controller, typically using memory-mapped I/O ports.

Solid-State Disks

- SSDs or solid state have the same characteristics as traditional hard disks but can be more reliable because they have no moving parts and faster because they have no seek time or latency.
- But they consume less power and are more expensive per megabyte, have less capacity, and may have shorter life spans than hard disks, so their uses are somewhat limited.
- One use for SSDs is in storage arrays, where they hold file-system meta data that require high performance.
- SSDs are also used in some laptop computers to make them smaller, faster, and more energy-efficient.
- Because SSDs can be much faster than magnetic disk drives, standard bus interfaces can cause a major limit on throughput.
- Some SSDs are designed to connect directly to the system bus(PCI, for example).
- SSDs are changing other traditional aspects of computer design as well.
    - Some systems use them as a direct replacement for disk drives, while others use them as a new cache tier, moving data between magnetic disks, SSDs, and memory to optimize performance.

Magnetic Tapes

- **Magnetic tape** was used as an early secondary-storage medium.
- Tapes are used mainly for backup, for storage of infrequently used information, and as a medium for transferring information from one system to another.
- A tape is kept in a spool and is wound or rewound past a read–write head.
- Moving to the correct spot on a tape can take minutes, but once positioned, tapedrives can write data at speeds comparable to disk drives.
- Tape capacities vary greatly, depending on the particular kind of tape drive, with current capacities exceeding several terabytes.
- Some tapes have built-in compressions that can more than double the effective storage.

Disk Scheduling

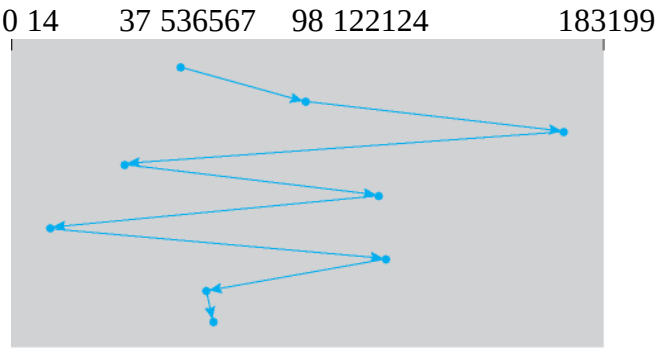
- One of the responsibilities of the operating system is to use the hardware efficiently.
- For the disk drives, meeting this responsibility entails having fast access time and large disk bandwidth.
- For magnetic disks, the access time has two major components.
- The **seek time** is the time for the disk arm to move the heads to the cylinder containing the desired sector.
- 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.
- Whenever a process needs I / O to or 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 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.

FCFS Scheduling

- Simplest form of disk scheduling - first-come, first-served (F C F S) algorithm.
- Fair, but it generally does not provide the fastest service.

Example: A disk queue with requests for I / O to blocks on cylinders in that order.

98, 183, 37, 122, 14, 124, 65, 67,
queue~ 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



FCFS disk scheduling

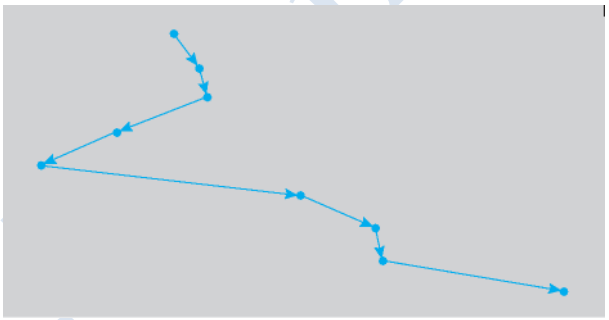
- If the disk head is initially at cylinder 53, it will first move from 53 to 98, then to 183, 37, 122, 14, 124, 65, and finally to 67, for a total head movement of 640 cylinders.
- The wild swing from 122 to 14 and then back to 124 illustrates the problem with this schedule.
- If the requests for cylinders 37 and 14 could be serviced together, before or after the requests for 122 and 124, the total head movement could be decreased substantially, and performance could be thereby improved.

SSTF Scheduling

- It seems reasonable to service all the requests close to the current head position before moving the head far away to service other requests.
- This assumption is the basis for the **shortest-seek-time-first (SSTF) algorithm.**
- The SSTF algorithm selects the request with the least seek time from the current head position.
- SSTF chooses the pending request closest to the current head position.
  Example: Consider the above request queue, the closest request to the initial head position (53) is at cylinder 65.

queue~ 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53

0 14 37 536567 98 122124 183199



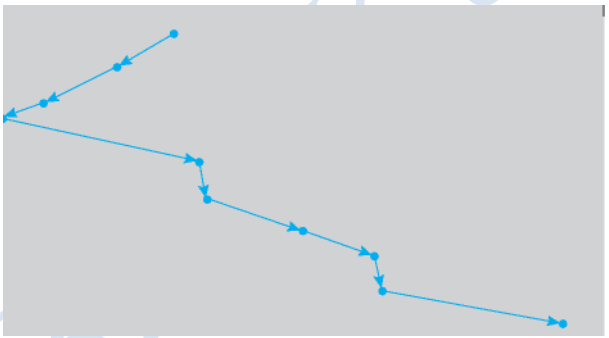
SSTF disk scheduling

- Once we are at cylinder 65, the next closest request is at cylinder 67.
- From there, the request at cylinder 37 is closer than the one at 98, so37 is served next.
- Continuing, it services the request at cylinder 14, then 98, 122, 124, and finally 183. This scheduling method results in a total head movement of only 236 cylinders—little more than one-third of the distance needed for FCFS scheduling of this request queue.
- This algorithm gives a substantial improvement in performance.
- Although the S S T F algorithm is a substantial improvement over the FC FS algorithm, it is not optimal.

SCAN Scheduling

- In the **S C A N 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 gets to the other end of the disk.
- At the other end, the direction of head movement is reversed, and servicing continues.
- The head continuously scans back and forth across the disk.
- The S C A N algorithm is sometimes called the **elevator algorithm.**
- Before applying S C A N to schedule the requests on cylinders, it is necessary to know the direction of head movement in addition to the head's current position.
- Assuming that the disk arm is moving toward 0 and that the initial head position is again 53, the head will next 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.
- If a request arrives in the queue just in front of the head, it will be serviced almost immediately; a request arriving just behind the head will have to wait until the arm moves to the end of the disk, reverses direction, and comes back.

queue ~ 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0 14 37 536567 98 122124 183199

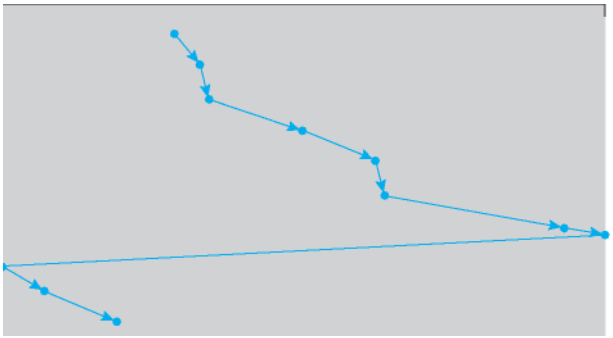


SCAN disk scheduling

CAN Scheduling

- **Circular S C A N (C - S C AN) scheduling** is a variant of S C A N designed to provide a more uniform wait time.
- Like S C A N , C - S C A N 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.
- The C - S C A N scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0 14 37 536567 98 122124 183199



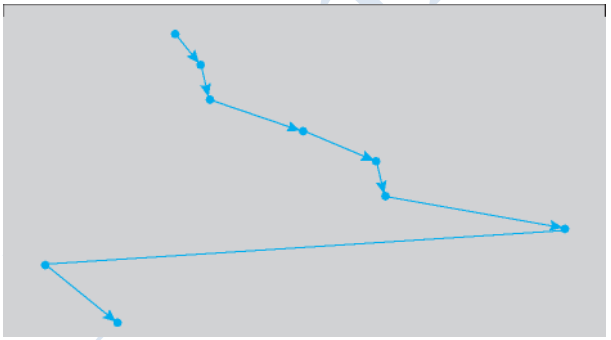
C- SCAN disk scheduling

LOOK and C-LOOK Scheduling

- Both SCAN and C-SCAN move the disk arm across the full width of the disk.
- In practice, neither algorithm is often implemented this way.
- More commonly, the arm goes only as far as the final request in each direction.
- It reverses direction immediately, without going all the way to the end of the disk.
- Versions of SCAN and C-SCAN that follow this pattern are called **L O O K** and **C - L O O K scheduling,** because they ***look*** for a request before continuing to move in a given direction.

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53

0 14 37 536567 98 122124 183199



C- LOOK disk scheduling

Selection of a Disk- Scheduling Algorithm

- With any scheduling algorithm, performance depends heavily on the number and types of requests.
- If the queue usually has just one outstanding request, then all scheduling algorithms behave the same.
- Requests for disk service can be greatly influenced by the file-allocation method.
- A program reading a contiguously allocated file will generate several requests that are close together on the disk, resulting in limited head movement
- A linked or indexed file, in contrast, may include blocks that are widely scattered on the disk, resulting in greater head movement.

- The location of directories and index blocks is also important. Since every file must be opened to be used, and opening a file requires searching the directory structure, the directories will be accessed frequently.
- Caching the directories and index blocks in main memory can also help to reduce disk-arm movement, particularly for read requests.
- The scheduling algorithms described here consider only the seek distances. For modern disks, the rotational latency can be nearly as large as the average seek time.
- It is difficult for the operating system to schedule for improved rotational latency, though, because modern disks do not disclose the physical location of logical blocks.

Disk Management

The operating system is also responsible for several other aspects of disk management.

Disk Formatting

- A new magnetic 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 a sector number and an **error-correcting code (ECC)**
- When the controller writes a sector of data during normal I / O , the E C C is updated with a value calculated from all the bytes in the data area. When the sector is read, the E C C is recalculated and compared with the stored value.
- If the stored and calculated numbers are different, this mismatch indicates that the data area of the sector has become corrupted and that the disk sector may be bad
- The ECC is an error-correcting code because it contains enough information, if only a few bits of data have been corrupted, to enable the controller to identify which bits have changed and calculate what their correct values should be.
- It then reports a recoverable soft error.
- The operating system still needs to record its own data structures on the disk.
- It does so in two steps.
- The first step is to **partition** the disk into one or more groups of cylinders.
- The operating system can treat each partition as though it were a separate disk. For instance, one partition can hold a copy of the operating system's executable code, while another holds user files.
- The second step is **logical formatting,** or creation of a file system. In this step, 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.**
- Some operating systems give special programs the ability to use a disk partition as a large sequential array of logical blocks, without any file-system data structures.
- This array is sometimes called the raw disk, and I/O to this array is termed raw I/O.

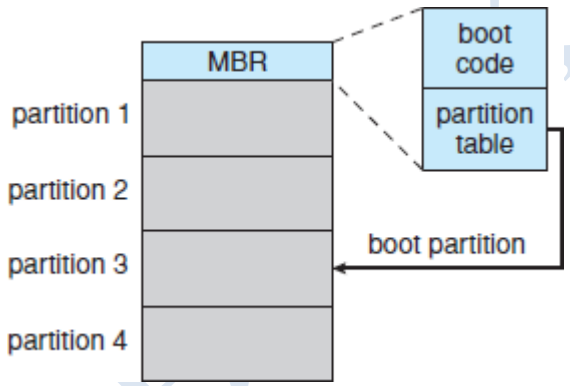
Boot Block

- For a computer to start running—for instance, when it is powered up or rebooted—it must have an initial program to run.
- This initial **bootstrap** program tends to be simple. It initializes all aspects of the system, from C P U registers to device controllers and the contents of main memory, and then starts the operating system.
- To do its job, the bootstrap program finds the operating-system kernel on disk, loads that kernel into memory, and jumps to an initial address to begin the operating-system execution.
- For most computers, the bootstrap is stored in **read-only memory (R O M) .** This location is convenient, because R O M needs no initialization and is at a fixed location that the processor can start executing when powered up or reset.
- The problem is that changing this bootstrap code requires changing the R O M hardware chips. For this reason, most systems store a tiny bootstrap loader program in the boot R O M whose only job is to bring in a full bootstrap program from disk.
- The full bootstrap program can be changed easily: a new version is simply written onto the disk. The full bootstrap program is stored in the "boot blocks" at a fixed location on the disk. A disk that has a boot partition is called a **boot disk** or **system disk.**
- Example: the boot process in Windows.
  - ¬Windows allows a hard disk to be divided into partitions, and one partition —identified as the boot partition—contains the operating system and device drivers.
  - ¬The Windows system places its boot code in the first sector on the hard disk, which it terms the master boot record, or MBR. Booting begins by running code that is resident in the system's ROM memory.
  - ¬This code directs the system to read the boot code from the MBR. In addition to containing boot code, the MBR contains a table listing the partitions for the hard disk and a flag indicating which partition the system is to be booted from.
  - ¬Once the system identifies the boot partition, it reads the first sector from that partit ion (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 (recall that the disk head flies just above the disk surface), they are prone to failure.
- Sometimes the failure is complete; in this case, the disk needs to be replaced and its contents restored from backup media to the new disk.
- More frequently, one or more sectors become defective.

- Most disks even come from the factory with **bad blocks.**
- Depending on the disk and controller in use, these blocks are handled in a variety of ways.
- More sophisticated disks are smarter about bad-block recovery.
- The controller maintains a list of bad blocks on the disk. The list is initialized during the low-level formatting at the factory and is updated over the life of the disk.
- Low-level formatting also sets aside spare sectors not visible to the operating system.
- The controller can be told to replace each bad sector logically with one of the spare sectors.
- This scheme is known as **sector sparing** or **forwarding.**
- A typical bad-sector transaction might be as follows:
  - The operating system tries to read logical block 87.
  - The controller calculates theE C C and finds that the sector is bad. It reports this finding to the operating system.
  - The next time the system is rebooted, a special command is run to tell the controller to replace the bad sector with a spare.
  - After that, whenever the system requests logical block 87, the request is translated into the replacement sector's address by the controller.



Booting from disk in window

- As an alternative to sector sparing, some controllers can be instructed to replace a bad block by **sector slipping.**

File System Storage

File Concept

- A file is a named collection of related information that is recorded on secondary storage.
- Files represent programs (both source and object forms) and data.
- Data files may be numeric, alphabetic, alphanumeric, ordinary.
- Files may be free form, such as text files, or may be formatted rigidly.
- In general, a file is a sequence of bits, bytes, lines, or records, the meaning of which is defined by the file's creator and user.
- Many different types of information may be stored in a file—source or executable programs, numeric or text data, photos, music, video, and so on.

- A file has a certain defined structure, which depends on its type.
- A **text file** is a sequence of characters organized into lines (and possibly pages).
- A **source file** is a sequence of functions, each of which is further organized as declarations followed by executable statements.
- An **executable file** is a series of code sections that the loader can bring into memory and execute.

F i l e Attributes

A file's attributes vary from one operating system to another but typically consist of these:

- ✓ **Name** - The symbolic file name is the only information kept in human-readable form.
- ✓ **Identifier** - This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
- ✓ **Type** - This information is needed for systems that support different types of files.
- ✓ **Location** - This information is a pointer to a device and to the location of the file on that device.
- ✓ **Size** - The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.
- ✓ **Protection** - Access-control information determines who can do reading, writing, executing, and so on.
- ✓ **Time, date, and user identification** - This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

- Some newer file systems also support **extended file attributes,** including character encoding of the file and security features such as a file checksum
- The information about all files is kept in the directory structure, which also resides on secondary storage.
- Typically, a directory entry consists of the file's name and its unique identifier. The identifier in turn locates the other file attributes.
- It may take more than a kilobyte to record this information for each file. In a system with many files, the size of the directory itself may be megabytes.
- Because directories, like files, must be nonvolatile, they must be stored on the device and brought into memory piecemeal, as needed.

File Operations

- ✓ A file is an abstract data type.
- ✓ The operations that can be performed on files are:
  - **Creating a file.** Two steps are necessary to create a file. First, space in the file system must be found for the file.
  - **Writing a file.**
    - To write a file, we make a system call specifying both the name of the file and the information to be written to the file.
    - Given the name of the file, the system searches the directory to find the file's location.
    - The system must keep a **write pointer** to the location in the file where the next write is to take place.

- The write pointer must be updated whenever a write occurs.

- **Reading a file.**
  - To read from a file, we use a system call that specifies the name of the file and where (in memory) the next block of the file should be put.
  - Again, the directory is searched for the associated entry, and the system needs to keep a **read pointer** to the location in the file where the next read is to take place.
  - Once the read has taken place, the read pointer is updated.
  - Because a process is usually either reading from or writing to a file, the current operation location can be kept as a per-process **current file-position pointer.**
  - Both the read and write operations use this same pointer, saving space and reducing system complexity.
- **Repositioning within a file.** The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value. Repositioning within a file need not involve any actual I / O . This file operation is also known as a file **seek.**
- **Deleting a file.** To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase the directory entry.
- **Truncating a file.** The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and thenrecreate it, this function allows all attributes to remain unchanged—except for file length—but lets the file be reset to length zero and its file space released.

✓ The operating system can provide system calls to create, write, read, reposition, delete, and truncate files.
✓ These six basic operations comprise the minimal set of required file operations.
- **File pointer.** This pointer is unique to each process operating on the file and therefore must be kept separate from the on-disk file attributes.
- **File-open count.** The file-open count tracks the number of opens and closes and reaches zero on the last close. The system can then remove the entry.
- **Disk location of the file.** The information needed to locate the file on disk is kept in memory so that the system does not have to read it from disk for each operation.
- **Access rights.** Each process opens a file in an access mode. This information is stored on the per-process table so the operating system can allow or deny subsequent I / O requests.

✓ File locks provide functionality similar to reader–writer locks.
✓ A **shared lock** is akin to a reader lock in that several processes can acquire the lock concurrently.
✓ An **exclusive lock** behaves like a writer lock; only one process at a time can acquire such a lock.

File Types

- A common technique for implementing file types is to include the type as part of the file name.
- The name is split into two parts—a name and an extension, usually separated by a period.

- Example: resume.docx, server. c, and ReaderThread. cpp.

- The system uses the extension to indicate the type of the file and the type of operations that can be done on that file.
- Only a file with a.com, .exe, or.sh extension can be executed.

File type	Usual	Function
Executable	exe, com, bin or or none	machine-ready-to-run machine-language program
Object	obj, o	compiled, machine language, not linked
Source code	c, cc, java, perl, asm	source code in various languages
Batch	bat, sh	commands to the command interpreter
Markup	xml, html, tex	textual data, documents
Word Processor	xml, rtf, docx	various word-processor formats
Library	lib, a, so, dll	libraries of routines for programmers
Print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
Archive	rar, zip, tar	related files grouped into one file, sometimes com-pressed, for archiving or storage
Multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

Common types of Files

File Structure

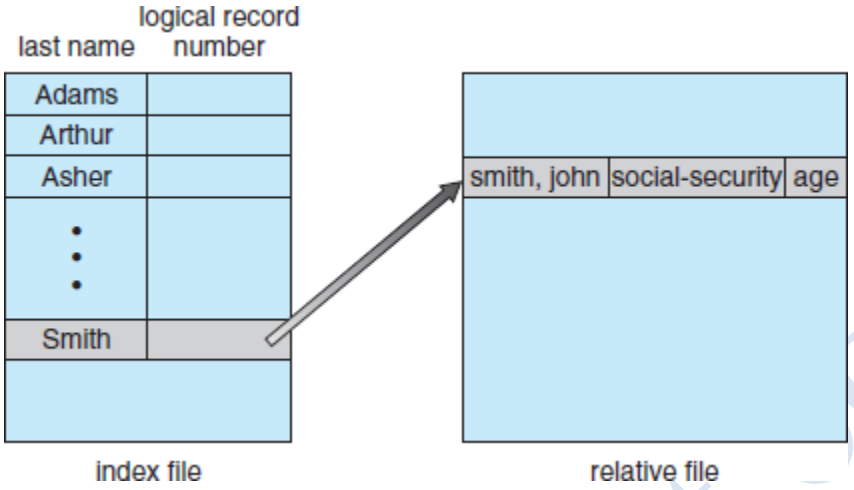
- File types also can be used to indicate the internal structure of the file.
- Example:
  The operating system requires that an executable file have a specific structure so that it can determine where in memory to load the file and what the location of the first instruction is.
- Some operating systems extend this idea into a set of system-supported file structures, with sets of special operations for manipulating files with those structures.

➢ **Internal File Structure**

- Disk systems typically have a well-defined block size determined by the size of a sector.
- All disk I / O is performed in units of one block (physical record), and all blocks are the same size.
- It is unlikely that the physical record size will exactly match the length of the desired logical record. Logical records may even vary in length. Packing a number of logical records into physical blocks is a common solution to this problem.
  Example: the U N I X operating system defines all files to be simply streams of bytes.
- Each byte is individually addressable by its offset from the beginning (or end) of the file.
- In this case, the logical record size is 1byte. The file system automatically packs and unpacks bytes into physical disk blocks— say, 512 bytes per block—as necessary.

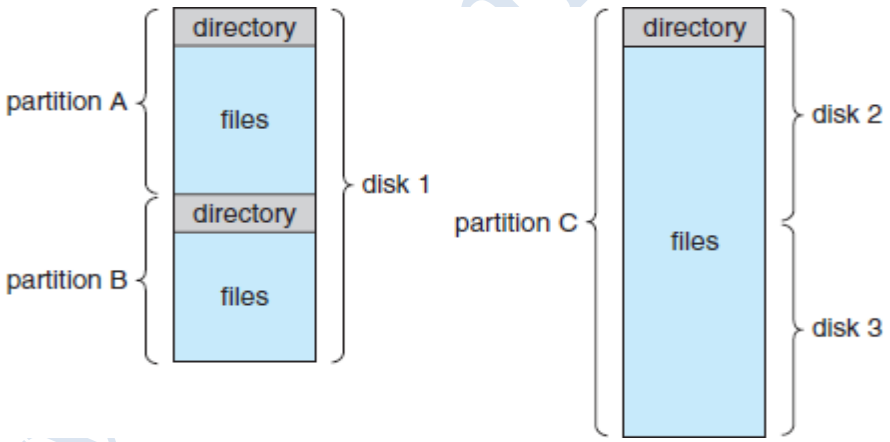
Directory and Disk Structure

- Files are stored on random-access storage devices, including hard disks, optical disks, and solid-state (memory-based) disks.
- A storage device can be used in its entirety for a file system. It can also be subdivided for finer-grained control.
  Example: a disk can be **partitioned** into quarters, and each quarter can hold a separate file system.
- Partitioning is useful for limiting the sizes of individual file systems, putting multiple file-system types on the same device, or leaving part of the device available for other uses, such as swap space or unformatted (raw) disk space.
- A file system can be created on each of these parts of the disk. Any entity containing a file system is generally known as a **volume.**
- The volume may be a subset of a device, a whole device, or multiple devices linked together into a R A ID set.
- Each volume can be thought of as a virtual disk.
- Volumes can also store multiple operating systems, allowing a system to boot and run more than one operating system.



Example of index and relative files

- Each volume that contains a file system must also contain information about the files in the system.
- This information is kept in entries in a **device directory** or **volume table of contents.** The device directory (more commonly known simply as the **directory)** records information—such as name, location, size, and type—for all files on that volume



A Typical file-system organization

Storage Structure

- A general-purpose computer system has multiple storage devices, and those devices can be sliced up into volumes that hold file systems.
- Computer systems may have zero or more file systems, and the file systems may be of varying types.
  Example: Types of file systems in the Solaris:
  - **tmpfs –** a "temporary" file system that is created in volatile main memory and has its contents erased if the system reboots or crashes
  - **objfs –** a "virtual" file system (essentially an interface to the kernel that looks like a

file system) that gives debuggers access to kernel symbols

- **ctfs** – a virtual file system that maintains "contract" information to manage which processes start when the system boots and must continue to run during operation
- **lofs –** a "loop back" file system that allows one file system to be accessed in place of another one
- **procfs –** a virtual file system that presents information on all processes as a file system
- **ufs, zfs -** general-purpose file systems

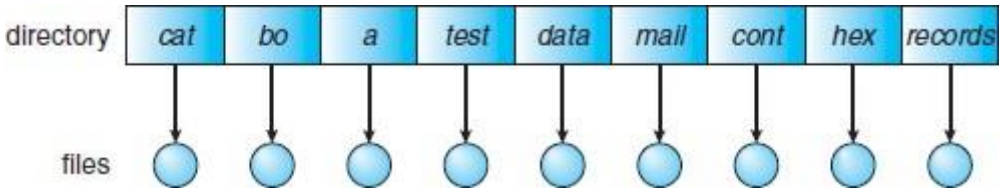
.

Directory Overview

- The directory can be viewed as a symbol table that translates file names into their directory entries.
- The organization must allow us to insert entries, to delete entries, to search for a named entry, and to list all the entries in the directory.
- When considering a particular directory structure, we need to keep in mind the operations that are to be performed on a directory:
  - **Search for a file.** We need to be able to search a directory structure to find the entry for a particular file. Since files have symbolic names, and similar names may indicate a relationship among files, we may want to be able to find all files whose names match a particular pattern.
  - **Create a file.** New files need to be created and added to the directory.
  - **Delete a file.** When a file is no longer needed, we want to be able to remove it from the directory.
  - **List a directory.** We need to be able to list the files in a directory and the contents of the directory entry for each file in the list.
  - **Rename a file.** Because the name of a file represents its contents to its users, we must be able to change the name when the contents or use of the file changes. Renaming a file may also allow its position within the directory structure to be changed.
  - **Traverse the file system.** We may wish to access every directory and every file within a directory structure. For reliability, it is a good idea to save the contents and structure of the entire file system at regular intervals.

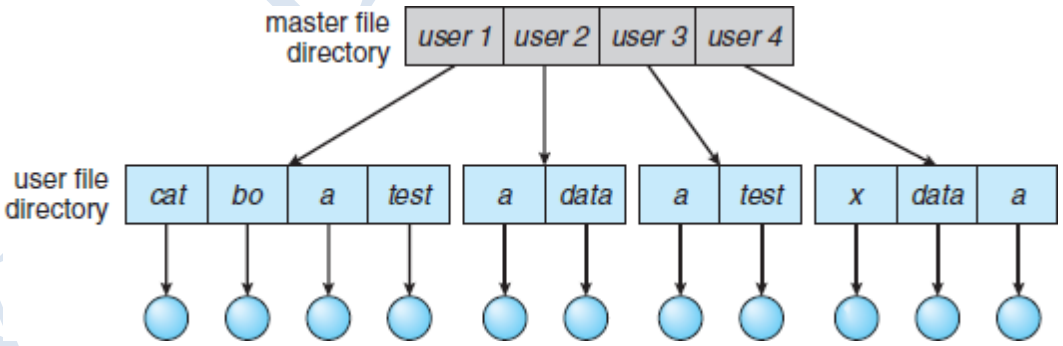
➢ **Single-Level Directory**

- Simplest.
- Has significant limitations - when the number of files increases or when the system has more than one user.
- Since all files are in the same directory, they must have unique names. If two users call their data filet e s t . t x t , then the unique-name rule is violated.
- Even a single user on a single-level directory may find it difficult to remember the names of all the files as the number of files increases.
- It is not uncommon for a user to have hundreds of files on one computer system and an equal number of additional files on another system.
- Keeping track of so many files is a daunting task.



Single-level Directory

- **Two-Level Directory**
  - As, a single-level directory often leads to confusion of file names among different users.
  - The standard solution is to create a separate directory for each user.
  - In the two-level directory structure, each user has own **user file directory (UFD).**
  - The U F D s have similar structures, but each lists only the files of a single user. When a user job starts or a user logs in, the system's **master file directory (M F D)** is searched.
  - When a user refers to a particular file, only his own U F D is searched.
  - Thus, different users may have files with the same name, as long as all the file names within each U F D are unique.
  - To create a file for a user, the operating system searches only that user's U F D to ascertain whether another file of that name exists.
  - To delete a file, the operating system confines its search to the local U F D ; thus, it cannot accidentally delete another user's file that has the same name.
  - A two-level directory can be thought of as a tree, or an inverted tree, of height 2. The root of the tree is the M F D . Its direct descendants are the U F Ds.
  - The descendants of the U F D s are the files themselves. The files are the leaves of the tree. Specifying a user name and a file name defines a path in the tree from the root (the M F D) to a leaf (the specified file).



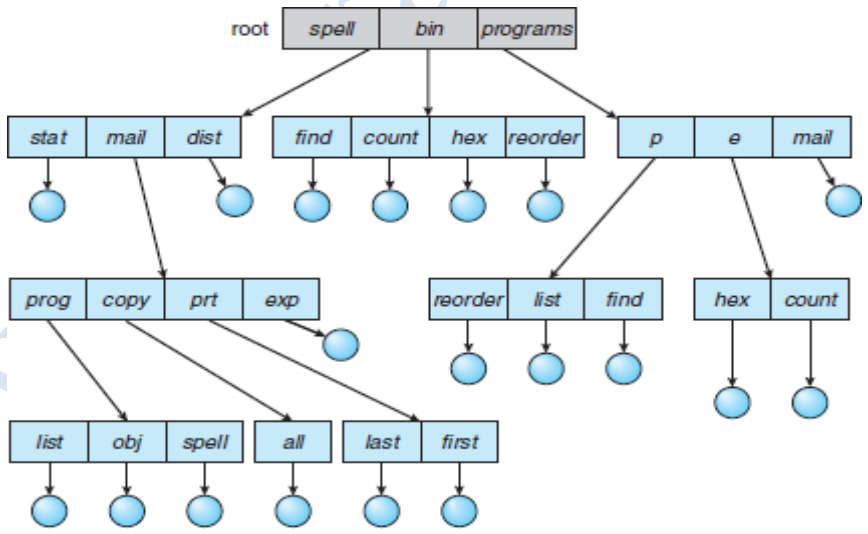
T w o - L e v e l D i r e c t o r y S t r u c t u r e

  - Thus, a user name and a file name define a **path name**
  - Every file in the system has a path name. To name a file uniquely, a user must know the path name of the file desired.
  - If the file is found, it is used.

- If it is not found, the system automatically searches the special user directory that contains the system files.
- The sequence of directories searched when a file is named is called the **search path.**
- The search path can be extended to contain an unlimited list of directories to search when a command name is given. This method is the one most used in U N I X and Windows.
- System scan also be designed so that each user has his own search path.

➢ **Tree – Structured Directories**

- A tree is the most common directory structure.
- The tree has a root directory, and every file in the system has a unique path name.
- A directory (or subdirectory) contains a set of files or subdirectories.
- All directories have the same internal format. One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1).
- Special system calls are used to create and delete directories.
- The **current directory** should contain most of the files that are of current interest to the process.
- When reference is made to a file, the current directory is searched. If a file is needed that is not in the current directory, then the user usually must either specify a path name or change the current directory to be the directory holding that file.
- To change directories, a system call is provided that takes a directory name as a parameter and uses it to redefine the current directory.
- Path names can be of two types: absolute and relative.
  - An **absolute path name** begins at the root and follows a path down to the specified file, giving the directory names on the path.
  - A **relative path name** defines a path from the current directory.

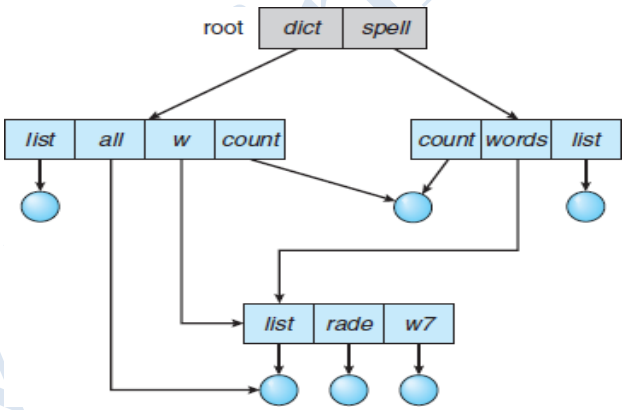


T r e e S t r u c t u r e d d i r e c t o r y s t r u c t u r e

- If the current directory is r o o t / s p e l l / m a i l , then the relative path name p r t / f i r s t refers to the same file as does the absolute path name r o o t / s p e l l / m a i l / p r t / f i r s t .

Acyclic-Graph Directories

- An **acyclic graph** —that is, a graph with no cycles—allows directories to share subdirectories and files.
- The same file or subdirectory may be in two different directories. The acyclic graph is a natural generalization of the tree-structured directory scheme.
- It is important to note that a shared file (or directory) is not the same as two copies of the file.
- With two copies, each programmer can view the copy rather than the original, but if one programmer changes the file, the changes will not appear in the other's copy.
- With a shared file, only one actual file exists, so any changes made by one person are immediately visible to the other.
- Sharing is particularly important for subdirectories; a new file created by one person will automatically appear in all the shared subdirectories.
- Shared files and subdirectories can be implemented in several ways.
- A common way, exemplified by many of the U N I X systems, is to create a new directory entry called a link.
- A **link** is effectively a pointer to another file or subdirectory.
- An acyclic-graph directory structure is more flexible than a simple tree structure, but it is also more complex.
- A file may now have multiple absolute path names. Distinct file names may refer to the same file.

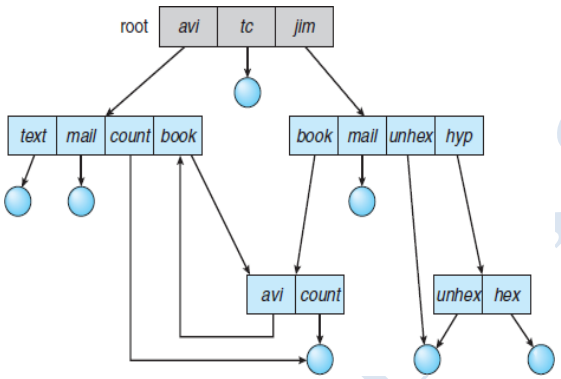


Acyclic-graph directory structure

General Graph Directory

- A serious problem with using an acyclic-graph structure is ensuring that there are no cycles.
- If we start with a two-level directory and allow users to create subdirectories, a tree-structured directory results.
- It should be fairly easy to see that simply adding new files and subdirectories to an existing tree-structured directory preserves the tree-structured nature
- However, when we add links, the tree structure is destroyed, resulting in a simple graph structure.

- The primary advantage of an acyclic graph is the relative simplicity of the algorithms to traverse the graph and to determine when there are no more references to a file.
- A similar problem exists when we are trying to determine when a file can be deleted.
- With acyclic-graph directory structures, a value of 0 in the reference count means that there are no more references to the file or directory, and the file can be deleted.
- However, when cycles exist, the reference count may not be 0 even when it is no longer possible to refer to a directory or file. This anomaly results from the possibility of self-referencing (or a cycle) in the directory structure.
- In this case, we generally need to use a garbage collection scheme to determine when the last reference has been deleted and the disk space can be reallocated.
- Garbage collection involves traversing the entire file system, marking everything that can be accessed.
- Then, a second pass collects everything that is not marked onto a list of free space



General Graph directory

File Sharing and Protection

File Sharing

- File sharing is very desirable for users who want to collaborate and to reduce the effort required to achieve a computing goal.
- User-oriented operating systems must accommodate the need to share files in spite of the inherent difficulties.

➢ **Multiple Users**

- ✓ When an operating system accommodates multiple users, the issues of file sharing, file naming, and file protection become preeminent.
- ✓ Given a directory structure that allows files to be shared by users, the system must mediate the file sharing.
- ✓ The system can either allow a user to access the files of other users by default or require that a user specifically grant access to the files.
- ✓ To implement sharing and protection, the system must maintain more file and directory attributes than are needed on a single-user system.

- ✓ Most systems have evolved to use the concepts of file (or directory) **owner** (or **user)** and **group.**
- ✓ The owner is the user who can change attributes and grant access and who has most control over the file.
- ✓ The group attribute defines a subset of users who can share access to the file.
- ✓ The owner and group ID s of a given file (or directory) are stored with the other file attributes.
- ✓ When a user requests an operation on a file, the user ID can be compared with the owner attribute to determine if the requesting user is the owner of the file.

➢ **Remote File Systems**

- Networking allows the sharing of resources spread across a campus or even around the world.
- One obvious resource to share is data in the form of files.
- Through the evolution of network and file technology, remote file-sharing methods have changed.
- The first implemented method involves manually transferring files between machines via programs like FTP.
- The second major method uses a **distributed file system (D FS)** in which remote directories are visible from a local machine.
- In some ways, the third method, the **World Wide Web,** is a reversion to the first.
- A browser is needed to gain access to the remote files, and separate operations (essentially a wrapper for ftp) are used to transfer files.
- ftp is used for both anonymous and authenticated access.
- **Anonymous access** allows a user to transfer files without having an account on the remote system.

✓ **The Client–Server Model**

- Remote file systems allow a computer to mount one or more file systems from one or more remote machines.
- In this case, the machine containing the files is the **server,** and the machine seeking access to the files is the **client.**
- The client–server relationship is common with networked machines.
- Generally, the server declares that a resource is available to clients and specifies exactly which resource (in this case, which files) and exactly which clients.
- A server can serve multiple clients, and a client can use multiple servers, depending on the implementation details of a given client–server facility. The server usually specifies the available files on a volume or directory level.
- Client identification is more difficult. A client can be specified by a network name or other identifier, such as an IP address, but these can be **spoofed,** or imitated.
- As a result of spoofing, an unauthorized client could be allowed access to the server. More secure solutions include secure authentication of the client via encrypted keys.

- ✓ **Distributed Information Systems**
  - To make client–server systems easier to manage, **distributed information systems,** also known as **distributed naming services,** provide unified access to the information needed for remote computing.
  - The **domain name system (D N S)** provides host-name-to-network-address translations for the entire Inter-net.
  - Before D N S became widespread, files containing the same information were sent via e-mail or f t p between all networked hosts.
  - Other distributed information systems provide **user name/password/user ID/group ID** space for a distributed facility.
  - UNIX systems have employed a wide variety of distributed information methods. Sun Microsystems (now part of Oracle Corporation) introduced yellow pages (since renamed network information service, or NIS), and most of the industry adopted its use.
  - In the case of Microsoft's common Internet file system (CIFS), network information is used in conjunction with user authentication (user name and password) to create a network login.
  - Microsoft uses active directory as a distributed naming structure to provide a single name space for users. Once established, the distributed naming facility is used by all clients and servers to authenticate users.
  - The industry is moving toward use of the lightweight directory-access protocol (LDAP) as a secure distributed naming mechanism. In fact, active directory is based on LDAP.

Failure Modes

- Local file systems can fail for a variety of reasons, including failure of the disk containing the file system, corruption of the directory structure or other disk-management information (collectively called **metadata),** disk-controller failure, cable failure, and host-adapter failure.
- User or system-administrator failure can also cause files to be lost or entire directories or volumes to be deleted.
- Many of these failures will cause a host to crash and an error condition to be displayed, and human intervention will be required to repair the damage.
- Remote file systems have even more failure modes.
- Because of the complexity of network systems and the required interactions between remote machines, many more problems can interfere with the proper operation of remote file systems.
- To implement this kind of recovery from failure, some kind of **state information** may be maintained on both the client and the server.
- If both server and client maintain knowledge of their current activities and open files, then they can seamlessly recover from a failure.
- In the situation where the server crashes but must recognize that it has remotely mounted exported file systems and opened files, NFS takes a simple approach, implementing a stateless DFS.

Consistency semantics

- **Consistency semantics** represent an important criterion for evaluating any file system that supports file sharing.
- In particular, they specify when modifications of data by one user will be observable by other users.
  Example: performing an atomic transaction to a remote disk could involve several network communications, several disk reads and writes, or both.
- Systems that attempt such a full set of functionalities tend to perform poorly.
- A successful implementation of complex sharing semantics can be found in the Andrew file system.
- For the following discussion, we assume that a series of file accesses (that is, reads and writes) attempted by a user to the same file is always enclosed between the `open()` and `close()` operations.
- The series of accesses between the `open()` and `close()` operations makes up a **file session.**

UNIX Semantics

- The UNIX file system uses the following consistency semantics:
  - Writes to an open file by a user are visible immediately to other users who have this file open.
  - One mode of sharing allows users to share the pointer of current location into the file.
  - Thus, the advancing of the pointer by one user affects all sharing users. Here, a file has a single image that interleaves all accesses, regardless of their origin.
- In the UNIX semantics, a file is associated with a single physical image that is accessed as an exclusive resource. Contention for this single image causes delays in user processes.

Session Semantics

- The Andrew file system (OpenAFS) uses the following consistency semantics:
  - Writes to an open file by a user are not visible immediately to other users that have the same file open.
  - Once a file is closed, the changes made to it are visible only in sessions starting later. Already open instances of the file do not reflect these changes.
- According to these semantics, a file may be associated temporarily with several(possibly different) images at the same time.
- Consequently, multiple users areallowed to perform both read and write accesses concurrently on their imagesof the file, without delay.

Immutable-Shared-Files Semantics

- A unique approach is that of **immutable shared files.**
- Once a file is declared as shared by its creator, it cannot be modified.

- An immutable file has two key properties: its name may not be reused, and its contents may not be altered.
- Thus, the name of an immutable file signifies that the contents of the file are fixed.

Protection

- When information is stored in a computer system, we want to keep it safe from physical damage (the issue of reliability) and improper access (the issue of protection).
- Reliability is generally provided by duplicate copies of files.
- Many computers have systems programs that automatically (or through computer-operator intervention) copy disk files to tape at regular intervals (once per day or week or month) to maintain a copy should a file system be accidentally destroyed.
- File systems can be damaged by hardware problems (such as errors in reading or writing), power surges or failures, head crashes, dirt, temperature extremes, and vandalism.
- Files may be deleted accidentally.
- Protection can be provided in many ways. For a single-user laptop system, we might provide protection by locking the computer in a desk drawer or file cabinet.
- In a larger multiuser system, however, other mechanisms are needed.

Types of Access

- Protection mechanisms provide controlled access by limiting the types of file access that can be made.
- Access is permitted or denied depending on several factors, one of which is the type of access requested. Several different types of operations may be controlled:
  - **Read.** Read from the file.
  - **Write.** Write or rewrite the file.
  - **Execute.** Load the file into memory and execute it.
  - **Append.** Write new information at the end of the file.
  - **Delete.** Delete the file and free its space for possible reuse.
  - **List.** List the name and attributes of the file.

Access Control

- The most general scheme to implement identity-dependent access is to associate with each file and directory an **access-control list (A C L)** specifying user names and the types of access allowed for each user.
- When a user requests access to a particular file, the operating system checks the access list associated with that file.
- If that user is listed for the requested access, the access is allowed. Otherwise, a protection violation occurs, and the user job is denied access to the file.

- This approach has the advantage of enabling complex access methodologies.
- The main problem with access lists is their length. If we want to allow everyone to read a file, we must list all users with read access.
- This technique has two undesirable consequences:
  - Constructing such a list may be a tedious and unrewarding task, especially if we do not know in advance the list of users in the system.
  - The directory entry, previously of fixed size, now must be of variable size, resulting in more complicated space management.

- These problems can be resolved by use of a condensed version of the access list.
- To condense the length of the access-control list,many systemsrecognize three classifications of users in connection with each file:
  - **Owner.** The user who created the file is the owner.
  - **Group.** A set of users who are sharing the file and need similar access is a group, or work group.
  - **Universe.** All other users in the system constitute the universe.

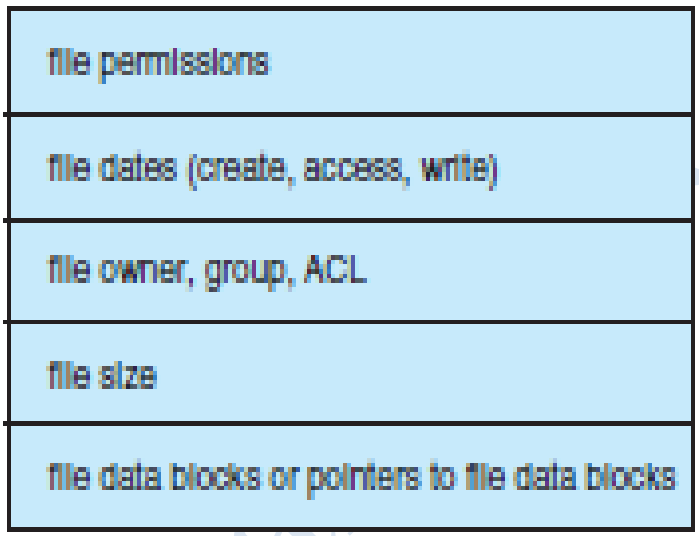
Other Protection Approaches

- Another approach to the protection problem is to associate a password with each file**.**
- If the passwords are chosen randomly and changed often, this scheme may be effective in limiting access to a file. The use of passwords has a few disadvantages.
- First, the number of passwords that a user needs to remember may become large, making the scheme impractical.
- Second, if only one password is used for all the files, then once it is discovered, all files are accessible.

File System Implementation

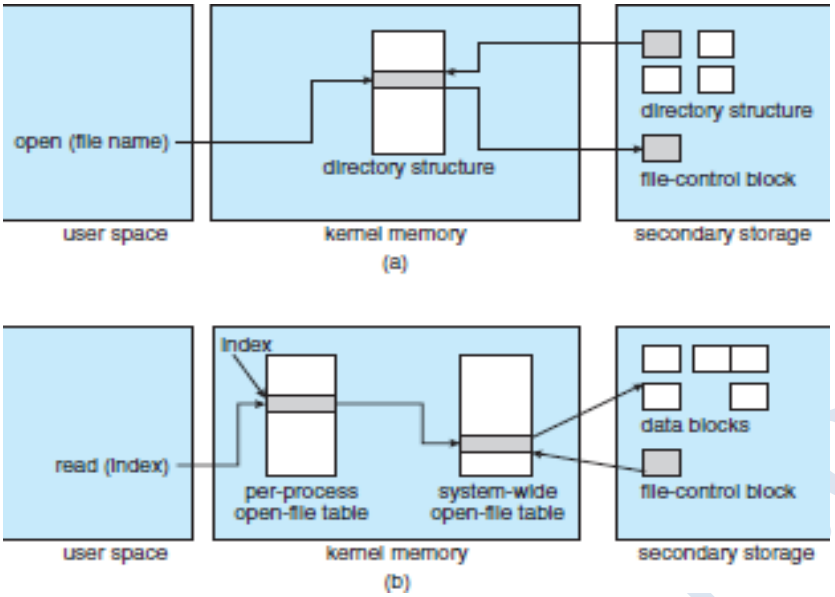
- A **boot control block** (per volume) can contain information needed by the system to boot an operating system from that volume. If the disk does not contain an operating system, this block can be empty. It is typically the first block of a volume. In UFS, it is called the **boot block**. In NTFS, it is the **partition boot sector**.
- A **volume control block** (per volume) contains volume (or partition) details, such as the number of blocks in the partition, the size of the blocks, a free-block count and free-block pointers, and a free-FCB count and FCB pointers. In UFS, this is called a **superblock**. In NTFS, it is stored in the **master file table**.
- A directory structure (per file system) is used to organize the files. In UFS, this includes file names and associated inode numbers. In NTFS, it is stored in the master file table.
- A per-file FCB contains many details about the file. It has a unique identifier number to allow association with a directory entry. In NTFS, this information is actually stored within the master file table, which uses a relational database structure, with a row per file.
- The in-memory information is used for both file-system management and performance improvement via caching. The data are loaded at mount time, updated during file-system operations, and discarded at dismount. Several types of structures may be included.

- An in-memory **mount table** contains information about each mounted volume.
- An in-memory directory-structure cache holds the directory information of recently accessed directories.
- The **system-wide open-file table** contains a copy of the FCB of each open file, as well as other information.
- The **per-process open-file table** contains a pointer to the appropriate entry in the system-wide open-file table, as well as other information.
- Buffers hold file-system blocks when they are being read from disk or written to disk.
- To create a new file, an application program calls the logical file system. The logical file system knows the format of the directory structures.



A typical file-control block

- To create a new file, it allocates a new FCB. (Alternatively, if the file-system implementation creates all FCBs at file-system creation time, an FCB is allocated from the set of free FCBs.) The system then reads the appropriate directory into memory, updates it with the new file name and FCB, and writes it back to the disk.



In memory file system structures (a) File open (b) File read

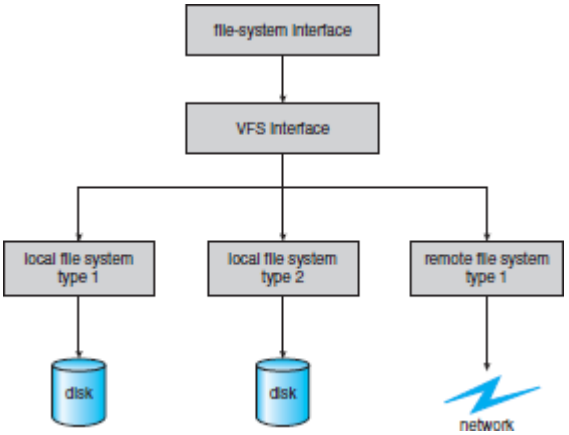
- When a file has been created, it can be used for I/O. First, it must be opened.
- The open() call passes a file name to the logical file system.
- The open() system call first searches the system-wide open-file table to see if the file is already in use by another process. If it is, a per-process open-file table entry is created pointing to the existing system-wide open-file table.
- This algorithm can save substantial overhead. If the file is not already open, the directory structure is searched for the given file name. Parts of the directory structure are usually cached in memory to speed directory operations.
- Once the file is found, the FCB is copied into a system-wide open-file table in memory.
- This table not only stores the FCB but also tracks the number of processes that have the file open.
- Next, an entry is made in the per-process open-file table, with a pointer to the entry in the system-wide open-file table and some other fields. These other fields may include a pointer to the current location in the file (for the next read() or write() operation) and the access mode in which the file is open.
- The open() call returns a pointer to the appropriate entry in the per-process file-system table. All file operations are then performed via this pointer.
- The file name may not be part of the open-file table, as the system has no use for it once the appropriate FCB is located on disk.
- It could be cached, though, to save time on subsequent opens of the same file. The name given to the entry varies.
- UNIX systems refer to it as a **file descriptor**; Windows refers to it as a **file handle**.
- When a process closes the file, the per-process table entry is removed, and
- the system-wide entry's open count is decremented.
- When all users that have opened the file close it, any updated metadata is copied back to the disk-based directory structure, and the system-wide open-file table entry is removed.

Partitions and Mounting

- The layout of a disk can have many variations, depending on the operating system. A disk can be sliced into multiple partitions, or a volume can span multiple partitions on multiple disks.
- Each partition can be either "raw," containing no file system, or "cooked," containing a file system.
- **Raw disk** is used where no file system is appropriate. Raw disk can also hold information needed by disk RAID systems, such as bit maps indicating which blocks are mirrored and which have changed and need to be mirrored.
- Execution of the image starts at a predefined location, such as the first byte. This boot loader in turn knows enough about the file-system structure to be able to find and load the kernel and start it executing.
- Many systems can be dual-booted, allowing us to install multiple operating systems on a single system.
- The **root partition**, which contains the operating-system kernel and sometimes other system files, is mounted at boot time.
- Mounting is implemented by setting a flag in the in-memory copy of the inode for that directory. The flag indicates that the directory is a mount point.
- A field then points to an entry in the mount table, indicating which device is mounted there.
- The mount table entry contains a pointer to the superblock of the file system on that device. This scheme enables the operating system to traverse its directory structure, switching seamlessly among file systems of varying types.

Virtual File Systems

- The first layer is the file-system interface, based on the open(), read(), write(), and close() calls and on file descriptors.
- The second layer is called the **virtual file system (VFS)** layer. The VFS layer serves two important functions:
  1. It separates file-system-generic operations from their implementation by defining a clean VFS interface. Several implementations for the VFS interface may coexist on the same machine, allowing transparent access to different types of file systems mounted locally.
  2. It provides a mechanism for uniquely representing a file throughout a network. The VFS is based on a file-representation structure, called a **vnode**, which contains a numerical designator for a network-wide unique file. (UNIX inodes are unique within only a single file system.)
- The VFS activates file-system-specific operations to handle local requests according to their file-system types and calls the NFS protocol procedures for remote requests.
- File handles are constructed from the relevant vnodes and are passed as arguments to these procedures.
- The layer implementing the file-system type or the remote-file-system protocol is the third layer of the architecture.



Schematic view of a virtual file system

- The four main object types defined by the Linux VFS are:
  - The **inode object**, which represents an individual file
  - The **file object**, which represents an open file
  - The **superblock object**, which represents an entire file system
  - The **dentry object**, which represents an individual directory entry
- For each of these four object types, the VFS defines a set of operations that may be implemented.
- 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 particular object.
  Example: Some of the operations for the file object includes:
  - int open(...) - Open a file.
  - int close(...) - Close an already-open file.
  - ssize t read(...) - Read from a file.
  - ssize t write(...) - Write to a file.
  - int mmap(...) - Memory-map a file.

Directory Implementation

a) Linear List

- The simplest method of implementing a directory is to use a linear list of file names with pointers to the data blocks. This method is simple to program but time-consuming to execute.
- To create a new file, we must first search the directory to be sure that no existing file has the same name. Then, we add a new entry at the end of the directory.
- To delete a file, we search the directory for the named file and then release the space allocated to it.
- To reuse the directory entry, we can do one of several things. We can mark the entry as unused (by assigning it a special name, such as an all-blank name, or by including a used– unused bit in each entry), or we can attach it to a list of free directory entries.
- A third alternative is to copy the last entry in the directory into the freed location and to decrease the length of the directory.

- A linked list can also be used to decrease the time required to delete a file. The real disadvantage of a linear list of directory entries is that finding a file requires a linear search.
- A sorted list allows a binary search and decreases the average search time.
- An advantage of the sorted list is that a sorted directory listing can be produced without a separate sort step.

b) Hash Table

- The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list. Therefore, it can greatly decrease the directory search time.
- Insertion and deletion are also fairly straightforward, although some provision must be made for collisions—situations in which two file names hash to the same location.
- The major difficulties with a hash table are its generally fixed size and the dependence of the hash function on that size.

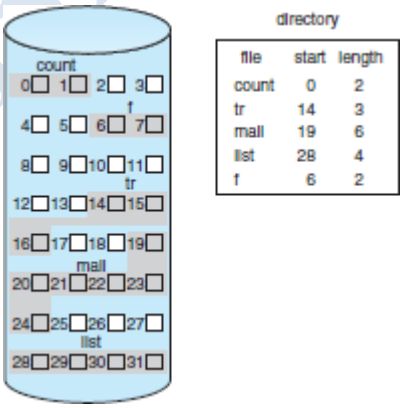
Allocation Methods

Three major methods of allocating disk space are in wide use: contiguous, linked, and indexed.

a) Contiguous Allocation

- **Contiguous allocation** requires that each file occupy a set of contiguous blocks on the disk.
- Disk addresses define a linear ordering on the disk. With this ordering, assuming that only one job is accessing the disk, accessing block *b* +1 after block *b* normally requires no head movement.
- When head movement is needed (from the last sector of one cylinder to the first sector of the next cylinder), the head need only move from one track to the next. Thus, the number of disk seeks required for accessing contiguously allocated files is minimal, as is seek time when a seek is finally needed.

Contiguous allocation has some problems, however. One difficulty is finding space for a new file.



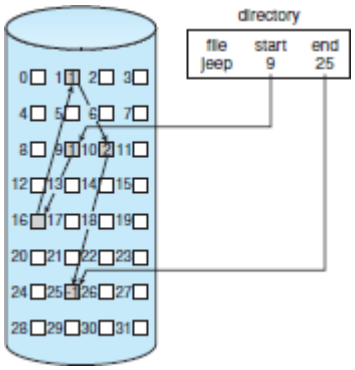
Contiguous allocation of disk space

- The contiguous-allocation problem can be seen as a particular application of the general **dynamic storage-allocation** which involves how to satisfy a request of size *n* from a list of free holes.
- First fit and best fit are the most common strategies used to select a free hole from the set of available holes.

- All these algorithms suffer from the problem of **external fragmentation**. As files are allocated and deleted, the free disk space is broken into little pieces.
- External fragmentation exists whenever free space is broken into chunks. It becomes a problem when the largest contiguous chunk is insufficient for a request; storage is fragmented into a number of holes, none of which is large enough to store the data.
- Depending on the total amount of disk storage and the average file size, external fragmentation may be a minor or a major problem.
- One strategy for preventing loss of significant amounts of disk space to external fragmentation is to copy an entire file system onto another disk.
- The original disk is then freed completely, creating one large contiguous free space. We then copy the files back onto the original disk by allocating contiguous space from this one large hole.
- This scheme effectively **compacts** all free space into one contiguous space, solving the fragmentation problem.
- Some systems require that this function be done off-line, with the file system unmounted.
- During this down time, normal system operation generally cannot be permitted, so such compaction is avoided at all costs on production machines. Most modern systems that need defragmentation can perform it on-line during normal system operations, but the performance penalty can be substantial.
- Another problem with contiguous allocation is determining how much space is needed for a file. When the file is created, the total amount of space it will need must be found and allocated.
- To prevent them, the user will normally overestimate the amount of space needed, resulting in considerable wasted space.

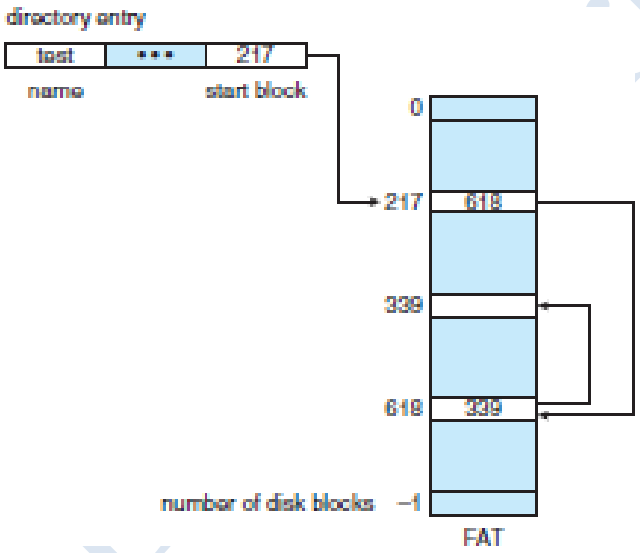
b) Linked Allocation

- **Linked allocation** solves all problems of contiguous allocation. With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk.
- The directory contains a pointer to the first and last blocks of the file.
  Example:
  A file of five blocks might start at block 9 and continue at block 16, then block 1, then block 10, and finally block 25
- Each block contains a pointer to the next block. These pointers are not made available to the user. Thus, if each block is 512 bytes in size, and a disk address (the pointer) requires 4 bytes, then the user sees blocks of 508 bytes.



Linked allocation of disk space

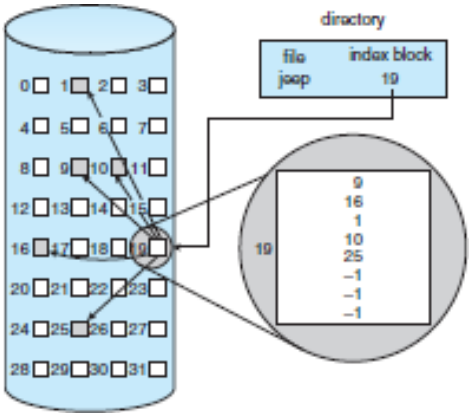
- To create a new file, we simply create a new entry in the directory. There is no external fragmentation with linked allocation, and any free block on the free-space list can be used to satisfy a request.
- The size of a file need not be declared when the file is created. A file can continue to grow as long as free blocks are available.
- Linked allocation does have disadvantages.
- The major problem is that it can be used effectively only for sequential-access files.
- The usual solution to this problem is to collect blocks into multiples, called **clusters**, and to allocate clusters rather than blocks.
- An important variation on linked allocation is the use of a **file-allocation table (FAT)**.
- The FAT is used in much the same way as a linked list. The directory entry contains the block number of the first block of the file. The table entry indexed by that block number contains the block number of the next block in the file.



File-allocation table

c) Indexed Allocation

- Linked allocation solves the external-fragmentation and size-declaration problems of contiguous allocation
- Linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all over the disk and must be retrieved in order.
- **Indexed allocation** solves this problem by bringing all the pointers together into one location: the **index block**.
- Each file has its own index block, which is an array of disk-block addresses. The *ith* entry in the index block points to the *ith* block of the file.
- The directory contains the address of the index block .To find and read the *ith* block, we use the pointer in the *ith* index-block entry.
- When the file is created, all pointers in the index block are set to null.When the *ith* block is first written, a block is obtained from the free-space manager, and its address is put in the *i*th index-block entry.



Indexed allocation of disk space

- If the index block is too small, however, it will not be able to hold enough pointers for a large file, and a mechanism will have to be available to deal with this issue.
- Mechanisms for this purpose include the following:

• **Linked scheme** - An index block is normally one disk block. Thus, it can be read and written directly by itself. To allow for large files, we can link together several index blocks.

• **Multilevel index –** A variant of linked representation uses a first-level index block to point to a set of second-level index blocks, which in turn point to the file blocks. To access a block, the operating system uses the first-level index to find a second-level index block and then uses that block to find the desired data block.

Performance

- For any type of access, contiguous allocation requires only one access to get a disk block. Since we can easily keep the initial address of the file in memory, we can calculate immediately the disk address of the *ith* block (or the next block) and read it directly.
- For linked allocation, we can also keep the address of the next block in memory and read it directly. This method is fine for sequential access; for direct access, however, an access to the *ith* block might require *i* disk reads
- As a result, some systems support direct-access files by using contiguous allocation and sequential-access files by using linked allocation. For these systems, the type of access to be made must be declared when the file is created.
- The performance of indexed allocation depends on the index structure, on the size of the file, and on the position of the block desired.
- Some systems combine contiguous allocation with indexed allocation by using contiguous allocation for small files (up to three or four blocks) and automatically switching to an indexed allocation if the file grows large.
- Since most files are small, and contiguous allocation is efficient for small files, average performance can be quite good.

.

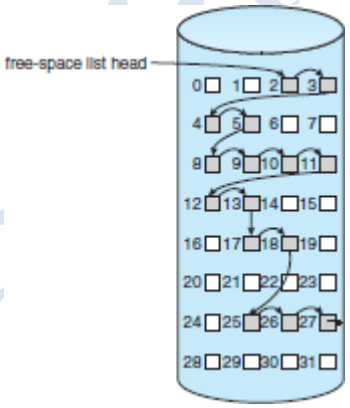
Free-Space Management

- To keep track of free disk space, the system maintains a **free-space list**.
- The free-space list records all free disk blocks—those not allocated to some file or directory. To create a file, we search the free-space list for the required amount of space and allocate that space to the new file.
- This space is then removed from the free-space list. When a file is deleted, its disk space is added to the free-space list.

a) Bit Vector

- Frequently, the free-space list is implemented as a **bit map** or **bit vector**.
- Each block is represented by 1 bit. If the block is free, the bit is 1; if the block is allocated, the bit is 0.
  Example:
  Consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free and the rest of the blocks are allocated. The free-space bit map would be 00111100111111000110000001110000 0...
- The main advantage of this approach is its relative simplicity and its efficiency in finding the first free block or *n* consecutive free blocks on the disk
- The first non-0 word is scanned for the first 1 bit, which is the location of the first free block.
- The calculation of the block number is
  (number of bits per word) × (number of 0-value words) + offset of first 1 bit.

b) Linked List



Linked free-space list on disk

- In a Linked List keeping a pointer to the first free block in a special location on the disk and caching it in memory. This first block contains a pointer to the next free disk block, and so on.
- 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 were free and the rest of the blocks were allocated. In this situation, we would keep a pointer to block 2 as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on.

c) Grouping

- A modification of the free-list approach stores the addresses of *n* free blocks in the first free block.

- The first $n-1$ of these blocks are actually free.
- The last block contains the addresses of another n free blocks, and so on.
- The addresses of a large number of free blocks can now be found quickly, unlike the situation when the standard linked-list approach is used.

d) Counting

- Another approach takes advantage of the fact that, generally, several contiguous blocks may be allocated or freed simultaneously, particularly when space is allocated with the contiguous-allocation algorithm or through clustering.
- Thus, rather than keeping a list of n free disk addresses, we can keep the address of the first free block and the number (n) of free contiguous blocks that follow the first block. Each entry in the free-space list then consists of a disk address and a count.

e) Space Maps

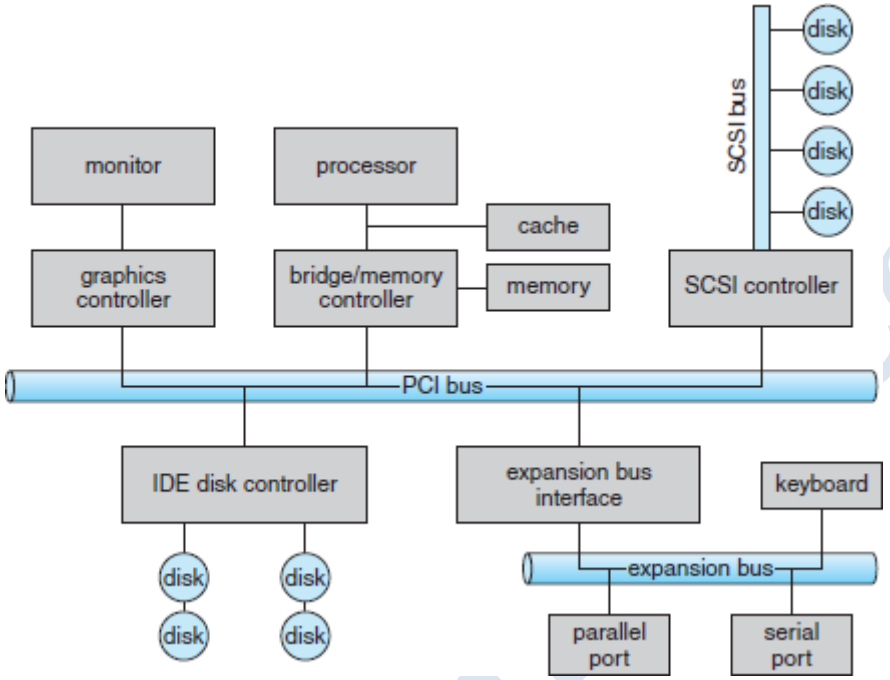
- Oracle's ZFS file system (found in Solaris and other operating systems) was designed to encompass huge numbers of files, directories, and even file systems (in ZFS, we can create file-system hierarchies).
- On these scales, metadata I/O can have a large performance impact.
  Example:
  If the free-space list is implemented as a bit map, bit maps must be modified both when blocks are allocated and when they are freed.
- Freeing 1 GB of data on a 1-TB disk could cause thousands of blocks of bit maps to be updated, because those data blocks could be scattered over the entire disk.
- Clearly, the data structures for such a system could be large and inefficient.
- In its management of free space, ZFS uses a combination of techniques to control the size of data structures and minimize the I/O needed to manage those structures.
- First, ZFS creates meta slabs to divide the space on the device into chunks of manageable size. A given volume may contain hundreds of meta slabs. Each meta slab has an associated space map.
- ZFS uses the counting algorithm to store information about free blocks. Rather than write counting structures to disk, it uses log-structured file-system techniques to record them.
- The space map is a log of all block activity (allocating and freeing), in time order, in counting format.
- When ZFS decides to allocate or free space from a meta slab, it loads the associated space map into memory in a balanced-tree structure (for very efficient operation), indexed by offset, and replays the log into that structure.
- The in-memory space map is then an accurate representation of the allocated and free space in the meta slab.

I/O Systems

I/O Hardware

- Computers operate a great many kinds of devices.

- Most fit into the general categories of storage devices (disks, tapes), transmission devices (network connections, Bluetooth), and human-interface devices (screen, keyboard, mouse, audio in and out).



A typical PC bus structure

- The device communicates with the machine via a connection point, or **port**—for example, a serial port. If devices share a common set of wires, the connection is called a bus.
- A **bus** is a set of wires and a rigidly defined protocol that specifies a set of messages that can be sent on the wires.
- In terms of the electronics, the messages are conveyed by patterns of electrical voltages applied to the wires with defined timings. When device *A* has a cable that plugs into device *B*, and device *B* has a cable that plugs into device *C*, and device *C* plugs into a port on the computer, this arrangement is called a **daisy chain**.
- In the figure, a **PCI bus** (the common PC system bus) connects the processor–memory subsystem to fast devices, and an **expansion bus** connects relatively slow devices, such as the keyboard and serial and USB ports.
- In the upper-right portion of the figure, four disks are connected together on a **Small Computer System Interface (SCSI)** bus plugged into a SCSI controller.
- Other common buses used to interconnect main parts of a computer include **PCI Express (PCIe)**, with throughput of up to 16 GB per second, and **Hyper Transport**, with throughput of up to 25 GB per second.
- A **controller** is a collection of electronics that can operate a port, a bus, or a device. A serial-port controller is a simple device controller.
- It is a single chip (or portion of a chip) in the computer that controls the signals on the wires of a serial port.

- By contrast, a SCSI bus controller is not simple. Because the SCSI protocol is complex, the SCSI bus controller is often implemented as a separate circuit board (or a host adapter) that plugs into the computer.
- It typically contains a processor, microcode, and some private memory to enable it to process the SCSI protocol messages. Some devices have their own built-in controllers.
- If you look at a disk drive, you will see a circuit board attached to one side. This board is the disk controller. It implements the disk side of the protocol for some kind of connection—SCSI or Serial Advanced Technology Attachment (SATA), for instance.
- It has microcode and a processor to do many tasks, such as bad-sector mapping, prefetching, buffering, and caching.
- The I/O instruction triggers bus lines to select the proper device and to move bits into or out of a device register. Alternatively, the device controller can support **memory-mapped I/O**.
- In this case, the device-control registers are mapped into the address space of the processor.
- The CPU executes I/O requests using the standard data-transfer instructions to read and write the device-control registers at their mapped locations in physical memory.
- Some systems use both techniques.

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

Device I/O port locations on PCs (partial)

- An I/O port typically consists of four registers, called the status, control, data-in, and data-out registers.
  - The **data-in register** is read by the host to get input.
  - The **data-out register** is written by the host to send output.
  - The **status register** contains bits that can be read by the host.
- These bits indicate states, such as whether the current command has completed, whether a byte is available to be read fromthe data-in register, and whether a device error has occurred.

• The **control register** can be written by host to start a command or to change the mode of a device.

Example:

A certain bit in the control register of a serial port chooses between full-duplex and half-duplex communication, another bit enables parity checking, a third bit sets the word length to 7 or 8 bits, and other bits select one of the speeds supported by the serial port.

a) Polling

- The complete protocol for interaction between the host and a controller can be intricate, but the basic handshaking notion is simple.
- Handshaking with an example. Assume that 2 bits are used to coordinate the producer–consumer relationship between the controller and the host
- The controller indicates its state through the busy bit in the status register.
- The controller sets the busy bit when it is busy working and clears the busy bit when it is ready to accept the next command.
- The host signals its wishes via the command-ready bit in the command register.
- The host sets the command-ready bit when a command is available for the controller to execute.

Example: The host writes output through a port, coordinating with the controller by handshaking as follows:

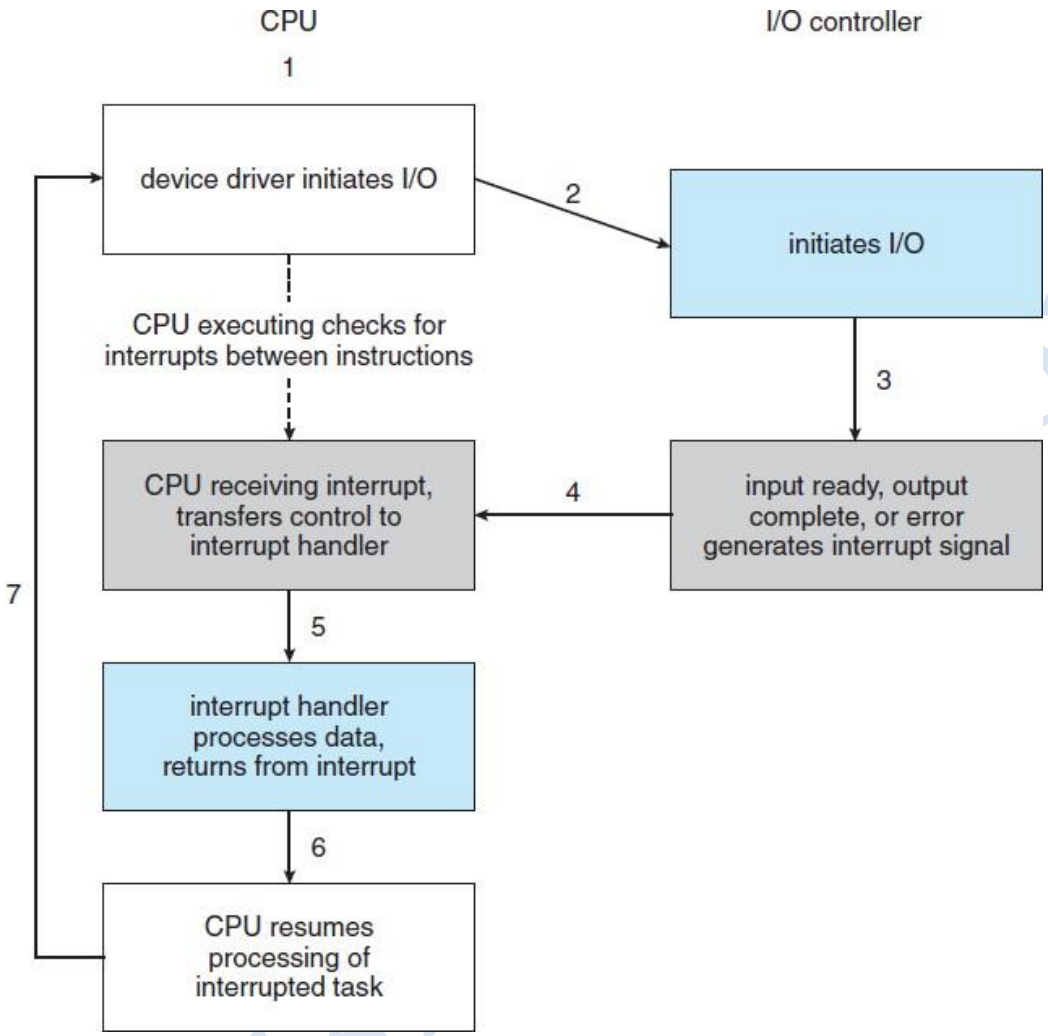
1. The host repeatedly reads the busy bit until that bit becomes clear.
2. The host sets the write bit in the command register and writes a byte into the data-out register.
3. The host sets the command-ready bit.
4. When the controller notices that the command-ready bit is set, it sets the busy bit.
5. The controller reads the command register and sees the write command. It reads the data-out register to get the byte and does the I/O to the device.
6. The controller clears the command-ready bit, clears the error bit in the status register to indicate that the device I/O succeeded, and clears the busy bit to indicate that it is finished. This loop is repeated for each byte.

In step 1, the host is **busy-waiting** or **polling**:

- In many computer architectures, three CPU-instruction cycles are sufficient to poll a device: read a device register, logical--and to extract a status bit, and branch if not zero.
- Clearly, the basic polling operation is efficient. But polling becomes inefficient when it is attempted repeatedly yet rarely finds a device ready for service, while other useful CPU processing remains undone.
- In such instances, it may be more efficient to arrange for the hardware controller to notify the CPU when the device becomes ready for service, rather than to require the CPU to poll repeatedly for an I/O completion. The hardware mechanism that enables a device to notify the CPU is called an **interrupt**.

b) Interrupts

- The basic interrupt mechanism works as follows. The CPU hardware has a wire called the **interrupt-request line** that the CPU senses after executing every instruction.
- When the CPU detects that a controller has asserted a signal on the interrupt-request line, the CPU performs a state save and jumps to the **interrupt-handler routine** at a fixed address in memory.
- The interrupt handler determines the cause of the interrupt, performs the necessary processing, performs a state restore, and executes a return from interrupt instruction to return the CPU to the execution state prior to the interrupt.
- The device controller raises an interrupt by asserting a signal on the interrupt request line, the CPU catches the interrupt and dispatches it to the interrupt handler, and the handler clears the interrupt by servicing the device.



Interrupt-driven I/O cycle

- The basic interrupt mechanism just described enables the CPU to respond to an asynchronous event, as when a device controller becomes ready for service.
- In a modern operating system, however, we need more sophisticated interrupt-handling features.
  1. We need the ability to defer interrupt handling during critical processing.
  2. We need an efficient way to dispatch to the proper interrupt handler for a device without first polling all the devices to see which one raised the interrupt.
  3. We need multilevel interrupts, so that the operating system can distinguish between high- and low-priority interrupts and can respond with the appropriate degree of urgency.
- In modern computer hardware, these three features are provided by the CPU and by the interrupt-controller hardware.
- Most CPUs have two interrupt request lines. One is the **nonmaskable interrupt**, which is reserved for events such as unrecoverable memory errors.
- The second interrupt line is **maskable**: it can be turned off by the CPU before the execution of critical instruction sequences that must not be interrupted.
- The maskable interrupt is used by device controllers to request service.

- The interrupt mechanism accepts an **address**—a number that selects a specific interrupt-handling routine from a small set. In most architectures, this address is an offset in a table called the **interrupt vector**.
- This vector contains the memory addresses of specialized interrupt handlers. The purpose of a vectored interrupt mechanism is to reduce the need for a single interrupt handler to search all possible sources of interrupts to determine which one needs service. In practice, however, computers have more devices (and, hence, interrupt handlers) than they have address elements in the interrupt vector.
- A common way to solve this problem is to use **interrupt chaining**, in which each element in the interrupt vector points to the head of a list of interrupt handlers.

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Intel Pentium processor event-vector table

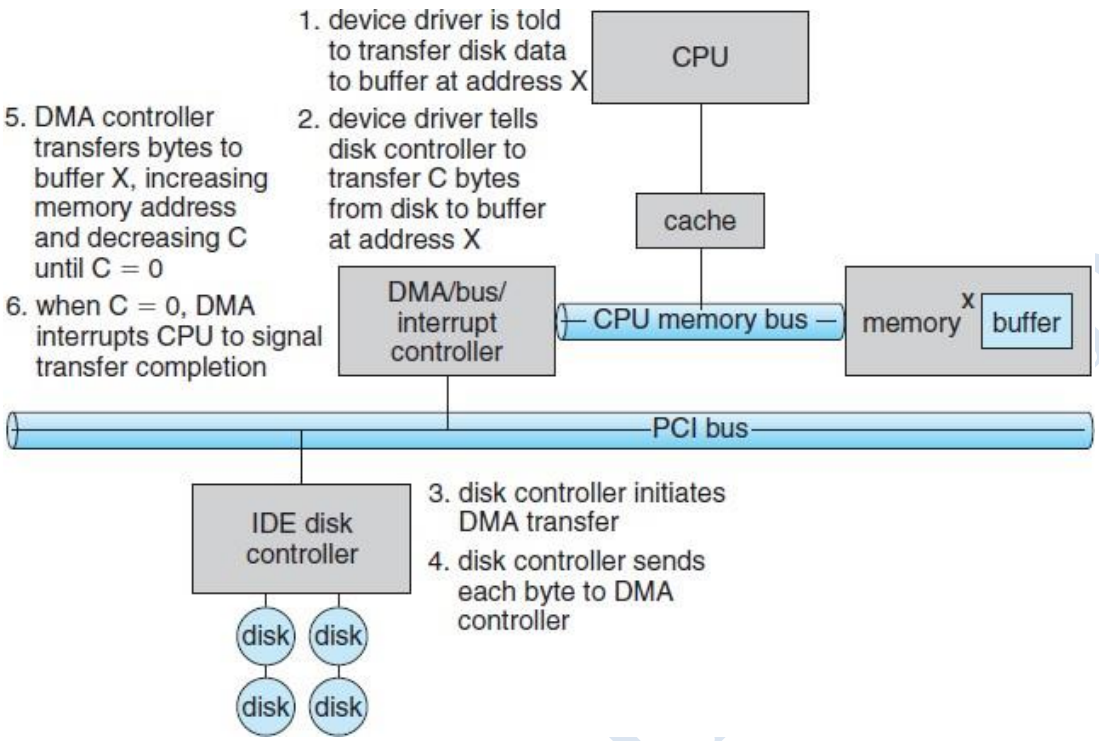
- The events from 0 to 31, which are nonmaskable, are used to signal various error conditions.
- The events from 32 to 255, which are maskable, are used for purposes such as device- generated interrupts.

- The interrupt mechanism also implements a system of **interrupt priority levels**.

- These levels enable the CPU to defer the handling of low-priority interrupts without masking all interrupts and makes it possible for a high priority interrupt to preempt the execution of a low-priority interrupt.
- A modern operating system interacts with the interrupt mechanism in several ways.
- At boot time, the operating system probes the hardware buses to determine what devices are present and installs the corresponding interrupt handlers into the interrupt vector.
- During I/O, the various device controllers raise interrupts when they are ready for service. These interrupts signify that output has completed, or that input data are available, or that a failure has been detected.
- The interrupt mechanism is also used to handle a wide variety of **exceptions**, such as dividing by 0, accessing a protected or nonexistent memory address, or attempting to execute a privileged instruction from user mode.
- Another example is found in the implementation of system calls.
- Usually, a program uses library calls to issue system calls.
- The library routines check the arguments given by the application, build a data structure to convey the arguments to the kernel, and then execute a special instruction called a **software interrupt**, or **trap**.
- This instruction has an operand that identifies the desired kernel service.

c) Direct Memory Access

- For a device that does large transfers, such as a disk drive, it seems wasteful to use an expensive general-purpose processor to watch status bits and to feed data into a controller register one byte at a time—a process termed programmed I/O (PIO)
- Many computers avoid burdening the main CPU with PIO by offloading some of this work to a special-purpose processor called a **direct-memory-access (DMA)** controller.
- To initiate a DMA transfer, the host writes a DMA command block into memory. This block contains a pointer to the source of a transfer, a pointer to the destination of the transfer, and a count of the number of bytes to be transferred.
- The CPU writes the address of this command block to the DMA controller, then goes on with other work.
- The DMA controller proceeds to operate the memory bus directly, placing addresses on the bus to perform transfers without the help of the main CPU. A simple DMA controller is a standard component in all modern computers, fromsmartphones to mainframes.



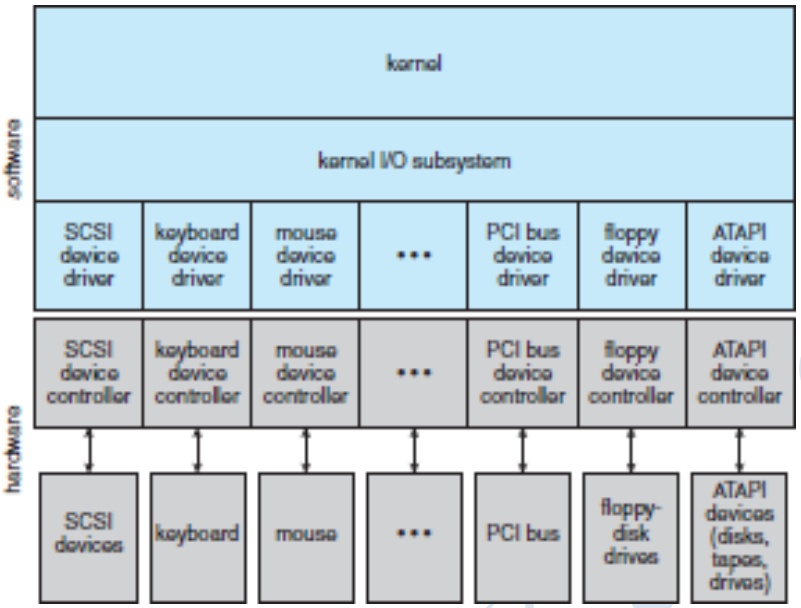
Steps in a DMA transfer

- Handshaking between the DMA controller and the device controller is performed via a pair of wires called **DMA-request** and **DMA-acknowledge**.
- The device controller places a signal on the DMA-request wire when a word of data is available for transfer.
- This signal causes the DMA controller to seize the memory bus, place the desired address on the memory-address wires, and place a signal on the DMA-acknowledge wire.
- When the device controller receives the DMA-acknowledge signal, it transfers the word of data to memory and removes the DMA-request signal.
- When the entire transfer is finished, the DMA controller interrupts the CPU.
- When the DMA controller seizes the memory bus, the CPU is momentarily prevented from accessing main memory.
- Although this **cycle stealing** can slow down the CPU computation, offloading the data transfer work to a DMA controller generally improves the total system performance.
- Some computer architectures use physical memory addresses for DMA, but others perform **direct virtual memory access (DVMA)**, using virtual addresses that undergo translation to physical addresses.
- On protected-mode kernels, the operating system generally prevents processes from issuing device commands directly.

Application I/O Interface

- Each general kind is accessed through a standardized set of functions—an **interface**.

- The differences are encapsulated in kernel modules called device drivers that internally are custom-tailored to specific devices but that export one of the standard interfaces



A kernel I/O structure

- The purpose of the device-driver layer is to hide the differences among device controllers from the I/O subsystem of the kernel, much as the I/O system calls encapsulate the behavior of devices in a few generic classes that hide hardware differences from applications.
- Making the I/O subsystem independent of the hardware simplifies the job of the operating-system developer.

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

Characteristics of I/O devices

- Devices vary on many dimensions.

• **Character-stream or block**. A character-stream device transfers bytes one by one, whereas a block device transfers a block of bytes as a unit.

• **Sequential or random access**. A sequential device transfers data in a fixed order determined by the device, whereas the user of a random-access device can instruct the device to seek to any of the available data storage locations.

• **Synchronous or asynchronous**. A synchronous device performs data transfers with predictable response times, in coordination with other aspects of the system. An asynchronous device exhibits irregular or unpredictable response times not coordinated with other computer events.

• **Sharable or dedicated**. A sharable device can be used concurrently by several processes or threads; a dedicated device cannot.

• **Speed of operation**. Device speeds range from a few bytes per second to a few gigabytes per second.

• **Read–write, read only, or write only**. Some devices perform both input and output, but others support only one data transfer direction.

- Most operating systems also have an escape (or back door) that transparently passes arbitrary commands from an application to a device driver.

a) Block and Character Devices

- The **block-device interface** captures all the aspects necessary for accessing disk drives and other block-oriented devices
- The operating system itself, as well as special applications such as database management systems, may prefer to access a block device as a simple linear array of blocks. This mode of access is sometimes called **raw I/O**.

- if an application provides its own locking of file blocks or regions, then any operating-system locking services would be redundant at the least and contradictory at the worst.
- To avoid these conflicts, raw-device access passes control of the device directly to the application, letting the operating system step out of the way.
- A keyboard is an example of a device that is accessed through a **character stream interface**. The basic system calls in this interface enable an application to get() or put() one character.

b) Network Devices

- Because the performance and addressing characteristics of network I/O differ significantly from those of disk I/O, most operating systems provide a network I/O interface that is different from the read()–write()–seek() interface used for disks.
- One interface available in many operating systems, including UNIX and Windows, is the network **socket** interface.
- Many other approaches to interprocess communication and network communication have been implemented.
  Example: Windows provides one interface to the network interface card and a second interface to the network protocols.

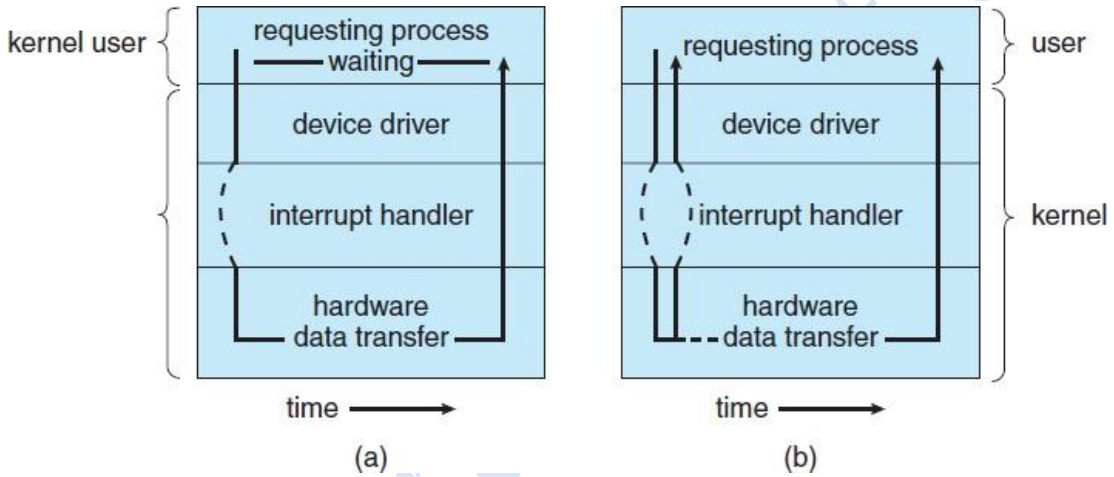
c)Clocks and Timers

- Most computers have hardware clocks and timers that provide three basic functions:
  - Give the current time.
  - Give the elapsed time.
  - Set a timer to trigger operation *X* at time *T*.
- These functions are used heavily by the operating system, as well as by time sensitive applications.
- The hardware to measure elapsed time and to trigger operations is called a **programmable interval timer**. It can be set to wait a certain amount of time and then generate an interrupt, and it can be set to do this once or to repeat the process to generate periodic interrupts.
- The scheduler uses this mechanism to generate an interrupt that will preempt a process at the end of its time slice.
- The disk I/O subsystem uses it to invoke the periodic flushing of dirty cache buffers to disk, and the network subsystem uses it to cancel operations that are proceeding too slowly because of network congestion or failures.

d) Non-blocking and Asynchronous I/O

- When an application issues a **blocking** system call, the execution of the application is suspended.
- The application is moved from the operating system's run queue to a wait queue. After the system call completes, the application is moved back to the run queue, where it is eligible to resume execution.
- When it resumes execution, it will receive the values returned by the system call. The physical actions performed by I/O devices are generally asynchronous—they take a varying or unpredictable amount of time.
- Nevertheless, most operating systems use blocking system calls for the application interface, because blocking application code is easier to understand than non-blocking application code. Some user-level processes need **non-blocking** I/O.

- One way an application writer can overlap execution with I/O is to write
- a multithreaded application.
- A non-blocking call does not halt the execution of the application for an extended time.
- But, it returns quickly, with a return value that indicates how many bytes were transferred.
- An alternative to a non-blocking system call is an asynchronous system call.
- An asynchronous call returns immediately, without waiting for the I/O to complete.
- The difference between non-blocking and asynchronous system calls is that a non-blocking read() returns immediately with whatever data are available—the full number of bytes requested, fewer, or none at all.
- An asynchronous read() call requests a transfer that will be performed in its entirety but will complete at some future time.
- These two I/O methods are shown:
-



Two I/O methods (a) Synchronous (b) Asynchronous

e) Vectored I/O

- **Vectored I/O** allows one system call to perform multiple I/O operations involving multiple locations.
- Multiple separate buffers can have their contents transferred via one system call, avoiding context-switching and system-call overhead.
- Without vectored I/O, the data might first need to be transferred to a larger buffer in the right order and then transmitted, which is inefficient.

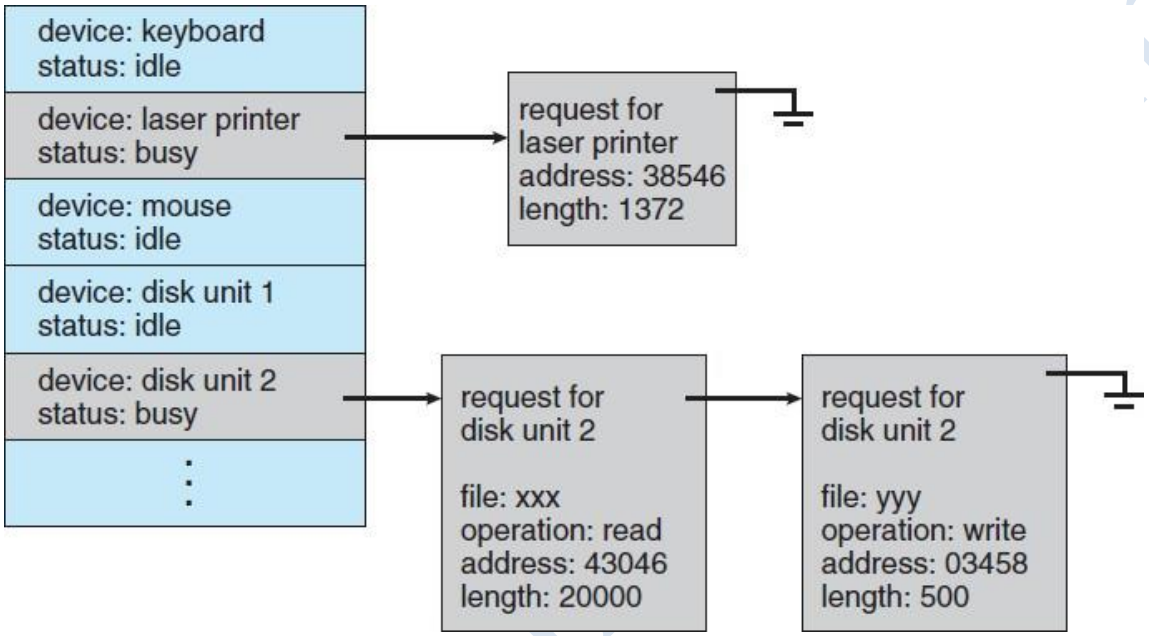
Kernel I/O Subsystem

- Kernels provide many services related to I/O.
- Several services—scheduling, buffering, caching, spooling, device reservation, and error handling

a) I/O Scheduling

- To schedule a set of I/O requests means to determine a good order in which to execute them.

- Scheduling can improve overall system performance, can share device access fairly among processes, and can reduce the average waiting time for I/O to complete.
- The I/O scheduler rearranges the order of the queue to improve the overall system efficiency and the average response time experienced by applications.
- When a kernel supports asynchronous I/O, it must be able to keep track of many I/O requests at the same time.



Device-status table

- For this purpose, the operating system might attach the wait queue to a device-status table. The kernel manages this table, which contains an entry for each I/O device
- Each table entry indicates the device's type, address, and state (not functioning, idle, or busy).
- If the device is busy with a request, the type of request and other parameters will be stored in the table entry for that device.
- Scheduling I/O operations is one way in which the I/O subsystem improves the efficiency of the computer. Another way is by using storage space in main memory or on disk via buffering, caching, and spooling.

b) Buffering

- A **buffer** is a memory area that stores data being transferred between two devices or between a device and an application. Buffering is done for three reasons.
- One reason is to cope with a speed mismatch between the producer and consumer of a data stream.
  Example: A file is being received via modem for storage on the hard disk.
- The modem is about a thousand times slower than the hard disk. So a buffer is created in main memory to accumulate the bytes received from the modem.

- The modem then starts to fill the second buffer while the first buffer is written to disk. By the time the modem has filled the second buffer, the disk write from the first one should have completed, so the modem can switch back to the first buffer while the disk writes the second one.
- This **double buffering** decouples the producer of data from the consumer, thus relaxing timing requirements between them.
- A second use of buffering is to provide adaptations for devices that have different data-transfer sizes.
- A third use of buffering is to support copy semantics for application I/O.

d) Caching

- A **cache** is a region of fast memory that holds copies of data.
- Access to the cached copy is more efficient than access to the original.
- For instance, the instructions of the currently running process are stored on disk, cached in physical memory and copied again in the C P U 's secondary and primary caches.
- The difference between a buffer and a cache is that a buffer may hold the only existing copy of a data item, whereas a cache, by definition, holds a copy on faster storage of an item that resides elsewhere.

e) Spooling and Device Reservation

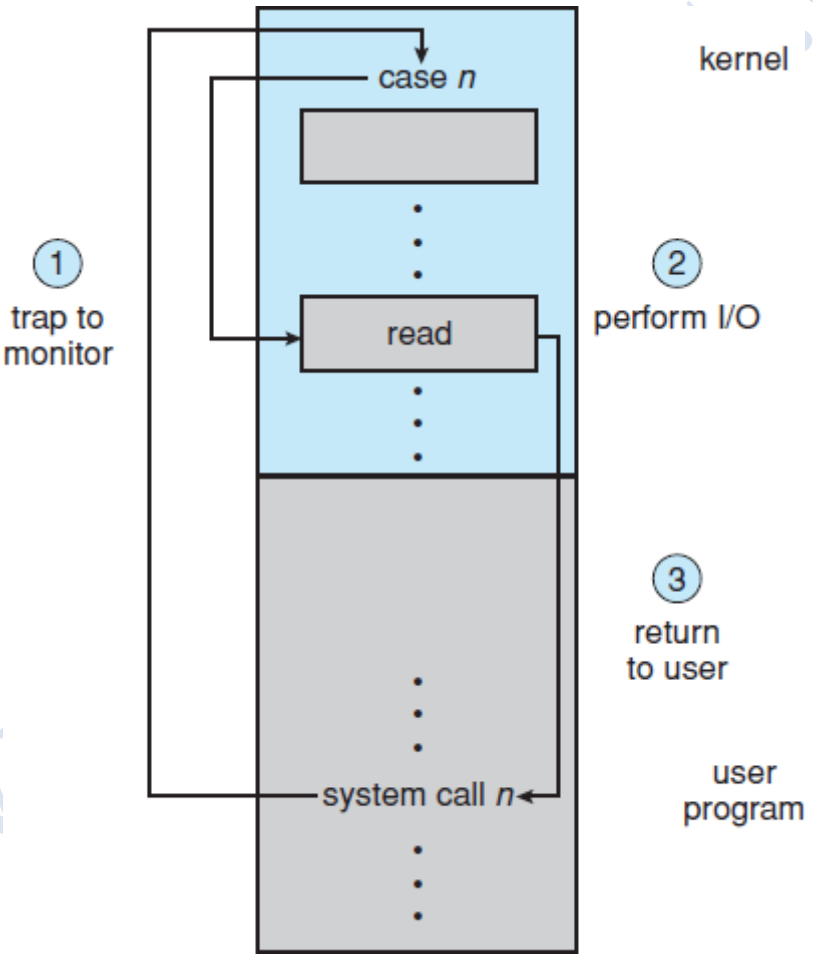
- A **spool** is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams.
- Although a printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixed together.
- The operating system solves this problem by intercepting all output to the printer.
- Each application's output is spooled to a separate disk file.
- When an application finishes printing, the spooling system queues the corresponding spool file for output to the printer.

f) Error Handling

- An operating system that uses protected memory can guard against many kinds of hardware and application errors, so that a complete system failure is not the usual result of each minor mechanical malfunction.
- Devices and I/O transfers can fail in many ways, either for transient reasons, as when a network becomes overloaded, or for "permanent" reasons, as when a disk controller becomes defective.
- Example:
  A failure of a SCSI device is reported by the SCSI protocol in three levels of detail: a **sense key** that identifies the general nature of the failure, such as a hardware error or an illegal request
    - An **additional sense code** that states the category of failure, such as a bad command parameter or a self-test failure
    - An **additional sense-code qualifier** that gives even more detail, such as which command parameter was in error or which hardware subsystem failed its self-test.

g) I/O Protection

- A user process may accidentally or purposely attempt to disrupt the normal operation of a users cannot issue I/O instructions directly; they must do it through the operating system.
- The operating system, executing in monitor mode, checks that the request is valid and, if it is, does the I/O requested.
- The operating system then returns to the user.
- To prevent users from performing illegal I/O, define all I/O instructions to be privileged instructions. Thus, users cannot issue I/O instructions directly; they must do it through the operating system.
- To do I/O, a user program executes a system call to request that the operating system perform I/O on its behalf



Use of a system call to perform I/O

- In addition, any memory-mapped and I/O port memory locations must be protected from user access by the memory-protection system.

- Most graphics games and video editing and playback software need direct access to memory-mapped graphics controller memory to speed the performance of the graphics, for example.
- The kernel might in this case provide a locking mechanism to allow a section of graphics memory (representing a window on screen) to be allocated to one process at a time.

h) Kernel Data Structures

- The kernel needs to keep state information about the use of I/O components.
- It does so through a variety of in-kernel data structures, such as the open-file table structure.
- The kernel uses many similar structures to track network connections, character-device communications, and otherI / O activities.
- Some operating systems use object-oriented methods even more extensively. For instance, Windows uses a message-passing implementation for I / O .
- An I / O request is converted into a message that is sent through the kernel to the I / O manager and then to the device driver, each of which may change the message contents.
- For output, the message contains the data to be written.
- For input, the message contains a buffer to receive the data. The message-passing approach can add overhead, by comparison with procedural techniques that use shared data structures, but it simplifies the structure and design of theI / O system and adds flexibility.

i) Kernel I/O Subsystem Summary

- The I/O subsystem coordinates an extensive collection of services that are available to applications and to other parts of the kernel.
- The I/O subsystem supervises these procedures:
    - Management of the name space for files and devices
    - Access control to files and devices
    - Operation control (for example, a modem cannot seek())
    - File-system space allocation
    - Device allocation
    - Buffering, caching, and spooling
    - I/O scheduling
    - Device-status monitoring, error handling, and failure recovery
    - Device-driver configuration and initialization

Transforming I / O Requests to Hardware Operations

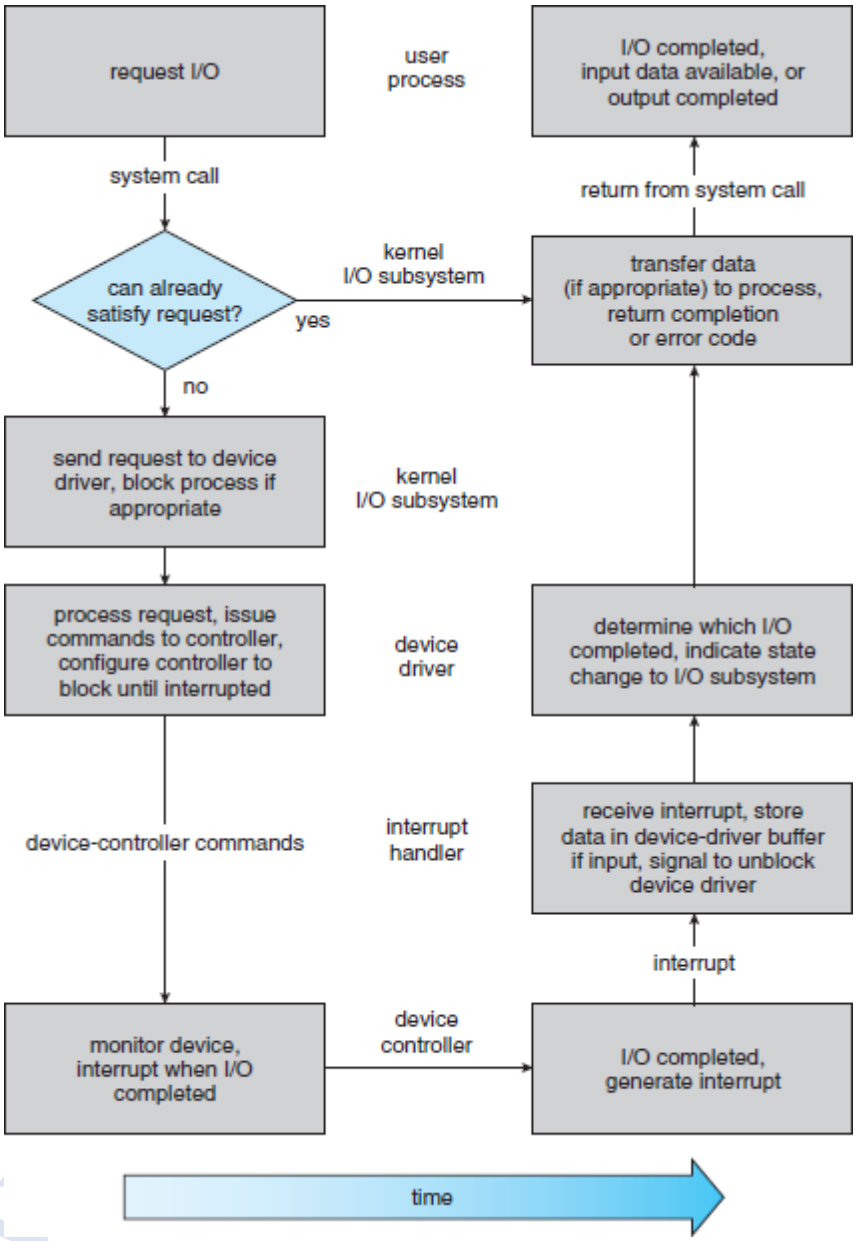
- How the operating system connects an application request to a set of network wires or to a specific disk sector.
  Example:
  Reading a file from disk. The application refers to the data by a file name. Within a disk, the file system maps from the file name through the file-system directories to obtain the space allocation of the file.
- For instance, in MS-DOS, the name maps to a number that indicates an entry in the file-access table, and that table entry tells which disk blocks are allocated to the file.

- In UNIX, the name maps to an inode number, and the corresponding inode contains the space-allocation information. But how is the connection made from the file name to the disk controller.
- One method is that used by MS-DOS, a relatively simple operating system.
- The first part of an MS-DOS file name, preceding the colon, is a string that identifies a specific hardware device.
- UNIX represents device names in the regular file-system name space. Unlike an MS-DOS file name, which has a colon separator, a UNIX path name has no clear separation of the device portion.
- In fact, no part of the path name is the name of a device. UNIX has a mount table that associates prefixes of path names with specific device names.
- Modern operating systems gain significant flexibility from the multiple stages of lookup tables in the path between a request and a physical device controller.
- The mechanisms that pass requests between applications and drivers are general.
- Thus, we can introduce new devices and drivers into a computer without recompiling the kernel.
- In fact, some operating systems have the ability to load device drivers on demand.
- At boot time, the system first probes the hardware buses to determine what devices are present. It then loads in the necessary drivers, either immediately or when first required by an I / O request.

1. A process issues a blocking r e a d ( ) system call to a file descriptor of a file that has been opened previously.

2. The system-call code in the kernel checks the parameters for correctness. In the case of input, if the data are already available in the buffer cache, the data are returned to the process, and the I / O request is completed. Otherwise, a physical I/O must be performed. The process is removed from the run queue and is placed on the wait queue for the device, and the I/O request is scheduled. Eventually, the I/O subsystem sends the request to the device driver. Depending on the operating system, the request is sent via a subroutine call or an in-kernel message.
3. The device driver allocates kernel buffer space to receive the data and schedules the I/O. Eventually, the driver sends commands to the device controller by writing into the device-control registers.
4. The device controller operates the device hardware to perform the data transfer.
5. The driver may poll for status and data, or it may have set up a DMA transfer into kernel memory. We assume that the transfer is managed by a DMA controller, which generates an interrupt when the transfer completes.
6. The correct interrupt handler receives the interrupt via the interrupt-vector table, stores any necessary data, signals the device driver, and returns from the interrupt.
7. The device driver receives the signal, determines which I/O request has completed, determines the request's status, and signals the kernel I/O subsystem that the request has been completed.
8. The kernel transfers data or return codes to the address space of the requesting process and moves the process from the wait queue back to the ready queue.
9. Moving the process to the ready queue unblocks the process. When the scheduler

assigns the process to the CPU, the process resumes execution at the completion of the system call.



The life cycle of an I/O request

Streams

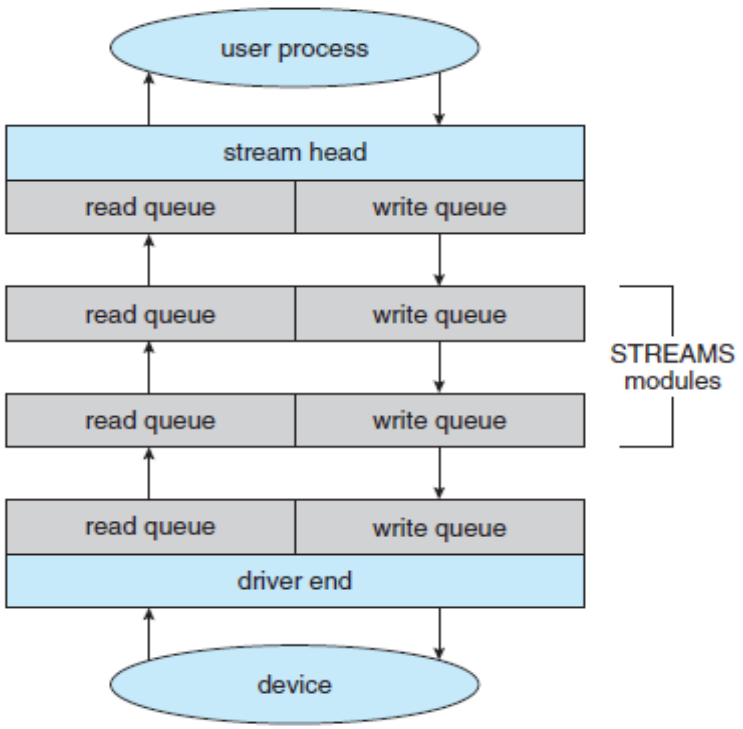
- A stream is a full-duplex connection between a device driver and a user-level process.
- That enables an application to assemble pipelines of driver code dynamically.
- It consists of a **stream head** that interfaces with the user process, a **driver end** that controls the device, and zero or more **stream modules** between the stream head and the driver end.

- Each of these components contains a pair of queues —a read queue and a write queue. Message passing is used to transfer data between queues.
- Modules provide the functionality of STREAMS processing; they are ***pushed*** onto a stream by use of the loctl() system call.

Example:

A process can open a serial-port device via a stream and can push on a module to handle input editing.

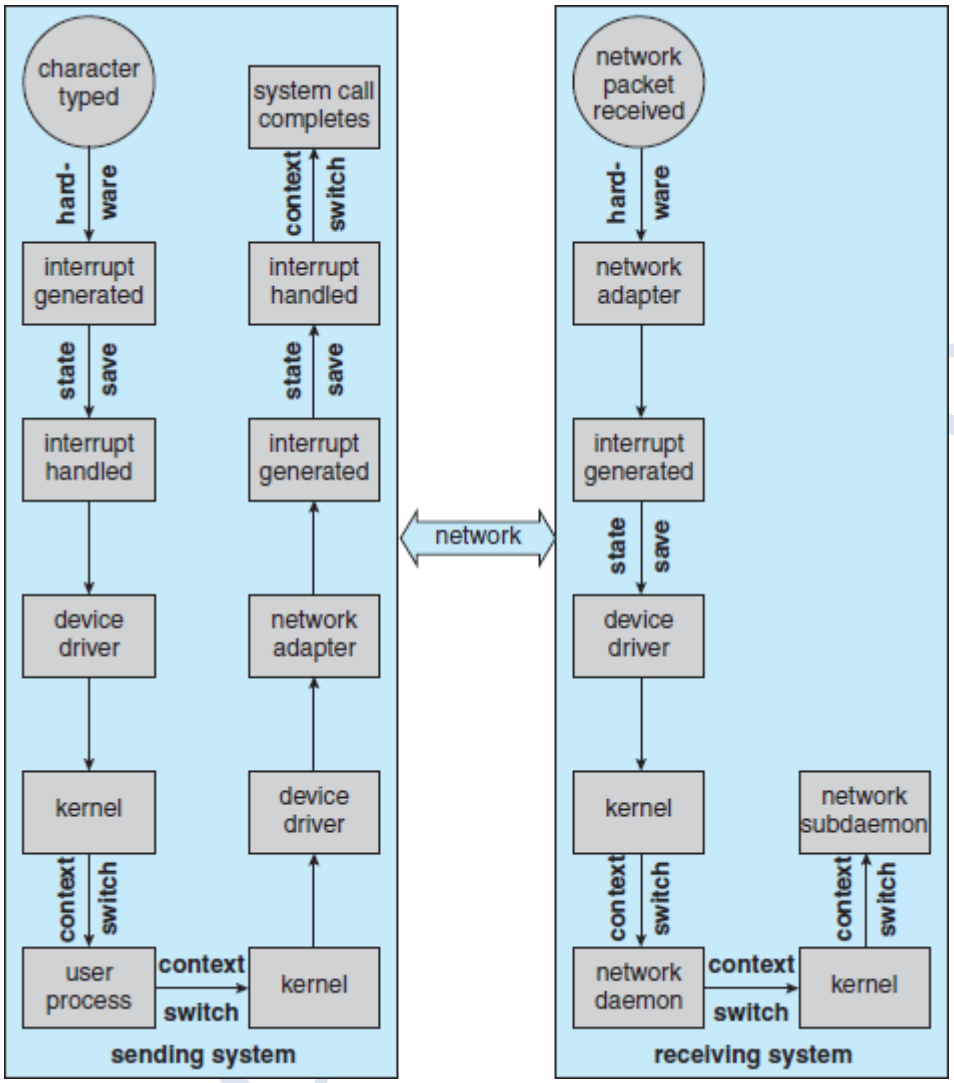
- Because messages are exchanged between queues in adjacent modules, a queue in one module may overflow an adjacent queue.
- To prevent this from occurring, a queue may support **flow control.** Without flow control, a queue accepts all messages and immediately sends them on to the queue in the adjacent module without buffering them.
- A queue that supports flow control buffers messages and does not accept messages without sufficient buffer space. This process involves exchanges of control messages between queues in adjacent modules.
- S T R E A M S I / O is asynchronous (or non-blocking) except when the user process communicates with the stream head.
- When writing to the stream, the user process will block, assuming the next queue uses flow control, until there is room to copy the message.
- Likewise, the user process will block when reading from the stream until data are available.
- Unlike the stream head, which may block if it is unable to copy a message to the next queue in line, the driver end must handle all incoming data.
- Drivers must support flow control as well. However, if a device's buffer is full, the device typically resorts to dropping incoming messages. Consider a network card whose input buffer is full. The network card must simply drop further messages until there is enough buffer space to store incoming messages.
- The benefit of using STREAMS is that it provides a framework for a modular and incremental approach to writing device drivers and network protocols.



The STREAMS structure.

P e r f o r m a n c e

- I/O is a major factor in system performance. It places heavy demands on the CPU to execute device-driver code and to schedule processes fairly and efficiently as they block and unblock.
- The resulting context switches stress the CPU and its hardware caches.
- I/O also exposes any inefficiencies in the interrupt-handling mechanisms in the kernel.
- I/O loads down the memory bus during data copies between controllers and physical memory and again during copies between kernel buffers and application data space.
- Coping gracefully with all these demands is one of the major concerns of a computer architect.
- Although modern computers can handle many thousands of interrupts per second, interrupt handling is a relatively expensive task.
- Each interrupt causes the system to perform a state change, to execute the interrupt handler, and then to restore state.
- Programmed I/O can be more efficient than interrupt-driven I/O, if the number of cycles spent in busy waiting is not excessive.
- An I/O completion typically unblocks a process, leading to the full overhead of a context switch.
- To eliminate the context switches involved in moving each character between daemons and the kernel, the Solaris developers re-implemented the telnet daemon using in-kernel threads.
- Sun estimated that this improvement increased the maximum number of network logins from a few hundred to a few thousand on a large server.
- Other systems use separate front-end processors for terminal I/O to reduce the interrupt burden on the main CPU. For instance, a terminal concentrator can multiplex the traffic from hundreds of remote terminals into one port on a large computer.
- An I/O channel is a dedicated, special-purpose CPU found in mainframes and in other high-end systems. The job of a channel is to offload I/O work fromthe main CPU.



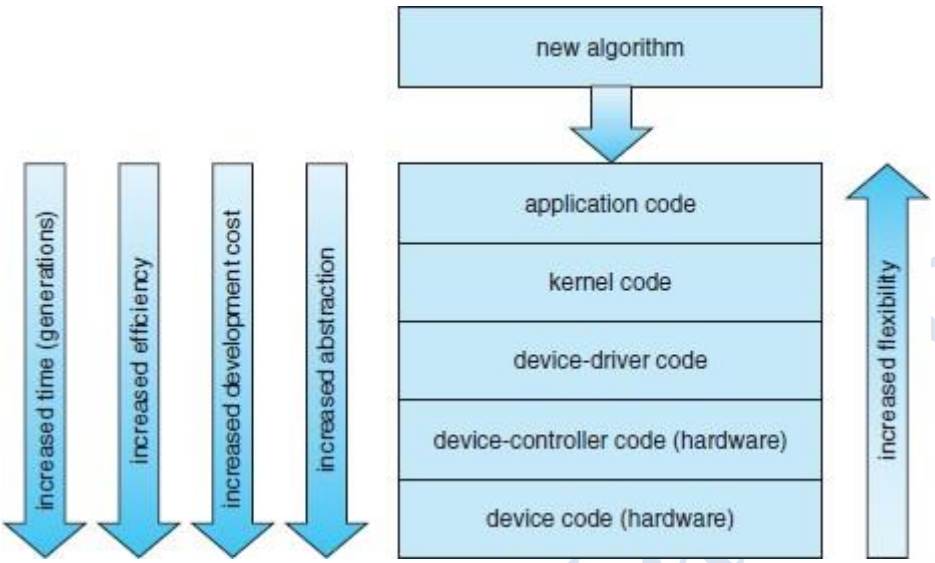
Intercomputer communication

Several principles to improve the efficiency of I / O :

- Reduce the number of context switches.
- Reduce the number of times that data must be copied in memory while passing between device and application.
- Reduce the frequency of interrupts by using large transfers, smart controllers, and polling (if busy waiting can be minimized).
- Increase concurrency by using D M A -knowledgeable controllers or channels to offload simple data copying from the C P U .
- Move processing primitives into hardware, to allow their operation in device controllers to be concurrent with C P U and bus operation.
- Balance C P U , memory subsystem, bus, and I / O performance, because an overload in any one area will cause idleness in others.

Where should the I/O functionality be implemented—in the device hardware,in the device driver,

or in application software?



Device functionality progression

- Initially, we implement experimental I/O algorithms at the application level, because application code is flexible and application bugs are unlikely to cause system crashes.
- Furthermore, by developing code at the application level, we avoid the need to reboot or reload device drivers after every change to the code.
- An application-level implementation can be inefficient, however, because of the overhead of context switches and because the application cannot take advantage of internal kernel data structures and kernel functionality (such as efficient in-kernel messaging, threading, and locking).
- When an application-level algorithm has demonstrated its worth, we may re-implement it in the kernel.
- This can improve performance, but the development effort is more challenging, because an operating-system kernel is a large, complex software system. Moreover, an in-kernel implementation must be thoroughly debugged to avoid data corruption and system crashes.
- The highest performance may be obtained through a specialized implementation in hardware, either in the device or in the controller.
- The disadvantages of a hardware implementation include the difficulty and expense of making further improvements or of fixing bugs, the increased development time (months rather than days), and the decreased flexibility.
- For instance, a hardware RAID controller may not provide any means for the kernel to influence the order or location of individual block reads and writes, even if the kernel has special information about the workload that would enable it to improve the I/O performance.

PART A

1. What is meant by Seek Time? (May/June 2012)

It is the time taken for the disk arm to move the heads to the cylinder containing the desired sector.

2. What is meant by Rotational Latency? (May/June 2012)(Nov/Dec 2010)

It is defined as the additional time waiting for the disk to rotate the desired sector to the disk head.

3. What is meant by Low-level 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 and a trailer.

4. What is meant by Swap-Space Management?

It is a low-level task of the operating system .Efficient management of the swap space is called as Swap space management. This Swap space is the space needed for the entire process image including code and Data segments.

5. What is meant by Disk Scheduling?

Disk scheduling is a process of allocation of the disk to one process at a time. In multi-programmed system, many processes try to read or write the records on disks as the same time. To avoid disk arbitration, it is necessary.

6. Why Disk Scheduling necessary? (April/May 2010)

To avoid Disk arbitration which occurs when many processes try to read or write the records on disks at the same time, Disk Scheduling is necessary.

7. What are the characteristics of Disk Scheduling?

- Throughput
- Mean Response Time
- Variance of Response time

8. What are the different types of Disk Scheduling? (May/June 2014)

Some of the Disk Scheduling are (i) SSTF Scheduling (ii) FCFS Scheduling (iii) SCAN Scheduling (iv) C-SCAN Scheduling (v) LOOK Scheduling (vi) C-LOOK Scheduling

9. What is meant by SSTF Scheduling?

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

10. What is meant by FCFS Scheduling?

It is simplest form of disk scheduling. This algorithm serves the first come process always and is does not provide fast service.

11. What is meant by SCAN scheduling?

In the SCAN algorithm, the disk arm starts at one end of the disk and moves toward the other end of the disk. At the other end, the direction of head movement is reversed and servicing continues across the disk.

12. What is meant by C-SCAN Scheduling?

C-SCAN means Circular SCAN algorithm. This Scheduling is a variant of SCAN designed to provide a more waiting time. This essentially treats the cylinder as a circular list that wraps around from the final cylinder to the first one.

13. Define Throughput.

It is defined as the number of requests serviced per unit time.

14. What is meant by Data Striping?

Data Stripping means splitting the bits of each byte across multiple disks. It is also called as Bit -level Striping.

15. What is meant by Boot Disk?

A Disk that has a boot partition is called as Boot Disk.

16. What are the Components of a Linux System?

Linux System composed of three main modules. They are:

(i) Kernel (ii) System libraries (iii) System utilities

17. What are the main supports for the Linux modules?

The Module support under Linux has three components. They are:

(i) Module Management
(ii) Driver Registration.
(iii) Conflict Resolution mechanism.

18. What do you meant by Buffer cache?

It is the kernel's main cache for block-oriented devices such as disk drives and is the main mechanism through which I/O to these devices is performed.

19. What is meant by Kernel in Linux system?

Kernel is responsible for maintaining all the important abstractions of the operating system including such things as virtual memory and processes.

20. What are System Libraries?

System Libraries define a standard set of functions through which applications can interact with the kernel and that implement much of the operating -system functionality that doesn't need the full privileges of kernel code.

21. What are System Utilities?

System Utilities are system programs that perform individual, specialized management tasks. Some of the System utilities may be invoked just to initialize and configure some aspect of the system and others may run permanently, handling such tasks as responding to incoming network connections, accepting logon requests from terminals or updating log files.

22. What is the function of Conflict Resolution mechanism?

This mechanism allows different device drivers to reserve hardware resources and to protect those resources from accidental use by another driver.

23. What are Device drivers? (Nov/Dec 2013)

Device drivers include (i) Character devices such as printers, terminals (ii) Block devices (including all disk drives) and network interface devices.

24. How do you improve I/O performance? (May/June 2012)

Principles to improve the efficiency of I/O:

a. Reduce the no. of content switches.

b. Reduce the no. of time that data must be copied in memory while passing between device and application.

25. Which scheduling algorithm would be best to optimize the performance of a RAM disk? (Nov/Dec 2011)

Shortest seek time first algorithm.

26. Write the three basic functions which are provided by the hardware clocks and timers. (April/May 2011)

a. Provide current time, elapsed time, timer

b. Programmable interval timer used for timings, periodic interrupts.

c. loctl (on unix) covers odd aspects of I/O such as clocks and timers.

27. What is a file? List some operations on it. (Nov/Dec 2010)

It is a collection of records. The operations performed in it are Open, Close, Read & Write.

28. What is the content of a typical file control block? (April/May 2011)

- File permissions
- File dates (create, access, write)
- File owner, group
- File size
- File data blocks

29. What are File Attributes? (May/June 2012)(April/May 2011)

- Identifier
- Name
- Type, Size

- Location, protection
- Time, Date & User Identification

30. What is meant by stream?

Streams

- A stream is a full-duplex connection between a device driver and a user-level process.
- That enables an application to assemble pipelines of driver code dynamically.
- It consists of a **stream head** that interfaces with the user process, a **driver end** that controls the device, and zero or more **stream modules** between the stream head and the driver end.

31. State three advantages and disadvantages of placing functionality in a device controller, rather than in the kernel.

Three advantages:

a. Bugs are less likely to cause an operating system crash

b. Performance can be improved by utilizing dedicated hardware and hard-coded algorithms

c. The kernel is simplified by moving algorithms out of it

Three disadvantages:

a. Bugs are harder to fix—a new firmware version or new hardware is needed

b. Improving algorithms likewise require a hardware update rather than just a kernel or device-driver update

c. Embedded algorithms could conflict with application's use of the device, causing decreased performance.

32. How does DMA increase system concurrency? How does it complicate hardware design?

DMA increases system concurrency by allowing the CPU to perform tasks while the DMA system transfers data via the system and memory buses. Hardware design is complicated because the DMA controller must be integrated into the system, and the system must allow the DMA controller to be a bus master. Cycle stealing may also be necessary to allow the CPU and DMA controller to share use of the memory bus.

33. Distinguish between a STREAMS driver and a STREAMS module.

The STREAMS driver controls a physical device that could be involved in a STREAMS operation. The STREAMS module modifies the flow of data between the head (the user interface) and the driver.

34. What problems could occur if a system allowed a file system to be mounted simultaneously at more than one location?

There would be multiple paths to the same file, which could confuse users or encourage mistakes (deleting a file with one path deletes the file in all the other paths).

35. Why is the bitmap for file allocation be kept on mass storage, rather than in main memory?

In case of system crash (memory failure) the free-space list would not be lost as it would be if the bit map had been stored in main memory.

36. Consider a system that supports the strategies of contiguous, linked, and indexed allocation. What criteria should be used in deciding which strategy is best utilized for a particular file?

- **Contiguous**—if file is usually accessed sequentially, if file is relatively small.
- **Linked**—if file is large and usually accessed sequentially.
- **Indexed**—if file is large and usually accessed randomly.

PART B

1. Explain the various disk scheduling techniques (May/June 2014).
2. Write notes about disk management.
3. Write notes about disk swap-space management.
4. Suppose that a disk drive has 5000 cylinders, numbered 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, is86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130
   Starting from the current 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
   a. FCFS
   b. SSTF
   c. SCAN
   d. LOOK
   e. C-SCAN
   f. C- LOOK
   Compare the performance of C-SCAN and SCAN scheduling, assuming a uniform distribution of requests. Consider the average response time (the time between the arrival of a request and the completion of that request's service), the variation in response time, and the effective bandwidth. How does performance depend on the relative sizes of seek time and rotational latency?
5. Explain the various I/O data transfer schemes.
6. Explain the various application I/O interface.
7. Write note on (i) Disk attachment (May/June 2014).
   (ii) Streams
   (iii) Tertiary Storage