Module 1 Part A

INTRODUCTION

File: A data structure in a file system which maps to names to file system objects such as files or other directories.

File structure: A pattern for arranging data in a file.

Data structure: A pattern for organizing data in a program.

Volatile storage: Storage which loses its contents when power is removed.

Non-volatile storage: Storage which retains its contents when power is removed.

Persistent data: Information which is retained after execution of the program which created it.

- Speed: RAM (main memory) access time, about 100 ns
- Speed: File (disk) access time, about 10 ms
- Cost: RAM (main memory), about $0.05 / Mbit
- Cost: File (disk), about $0.0002 / Mbit
- Capacity: RAM (main memory), maximum above 1 Gigabyte
- Capacity: File (disk), maximum above 1 Terabyte
- Volatility: RAM (main memory) is volatile.
- Volatility: File (disk) is nonvolatile.
- Persistence: Information stored in RAM (main memory) are typically retained only as long as the program which "owns" the variables is actually running.
- Persistence: Information stored in files (disk) can be persistent.

Goals for Design of File Structures and Algorithms

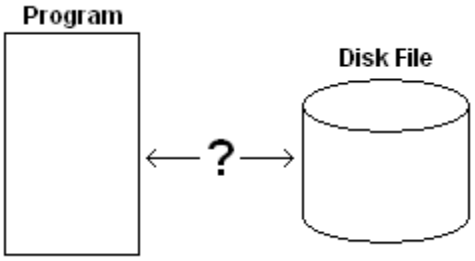
- Minimize the number of disk accesses.
- If possible, transfer all information needed in one access.
- Group related information physically so it can be accessed together.

Problems and Concerns

- File data is frequently dynamic - that is, it changes from time to time.
- Designing file structures for changes adds complexity.
- Typical file sizes are growing.
- Solutions which work for small files may be inadequate for large files.
- File structures, algorithms, and data structures must work together.

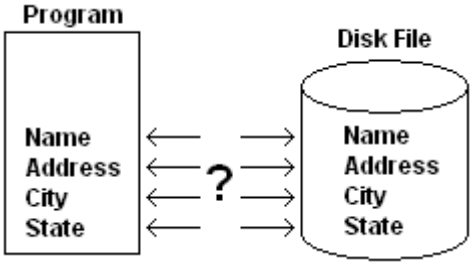
Fundamental File Processing Operations

- How is data transferred between a program and a disk file?



Secondary Storage and System Software

- How are files implemented by computer systems?
- How is information transferred between program variables and file structures?



Managing Files of Records

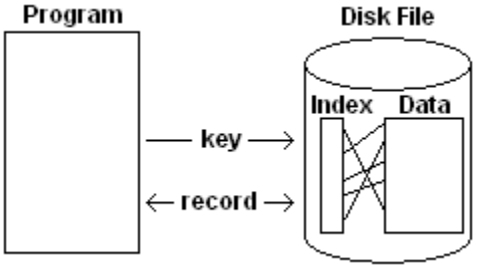
- How can records be accessed according to their contents?
- How can files be made self-describing?

Organizing Files for Performance

- How can data be stored compactly?
- How can file record space be reused?
- How can a file be searched?

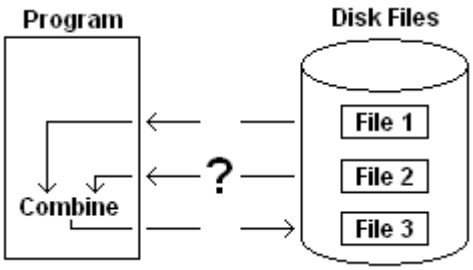
Indexing

- How can a single record in a file be accessed?



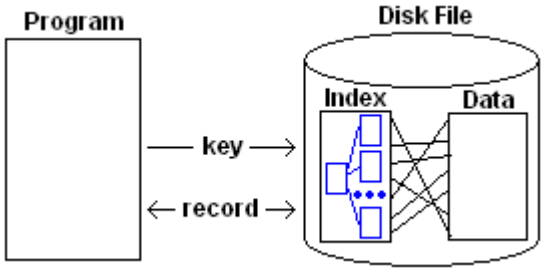
Cosequential Processing and the Sorting of Large Files

- How can information from separate files be combined into a single file?



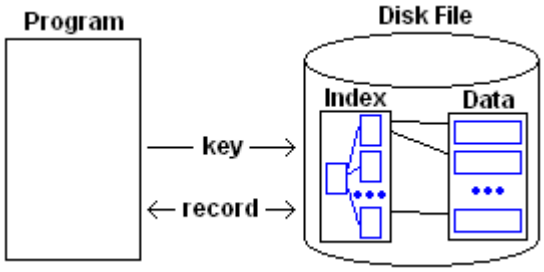
Multilevel Indexing and B-Tree

- How can a single record in a large file be accessed?



Indexed Sequential File Access and Prefix B$^+$Trees

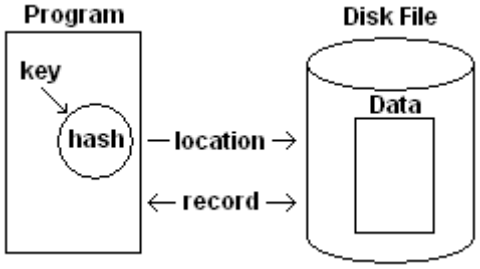
- How can files be organized for both indexed and sequential access?



Hashing

The transformation of a search key into a number by means of mathematical calculations.

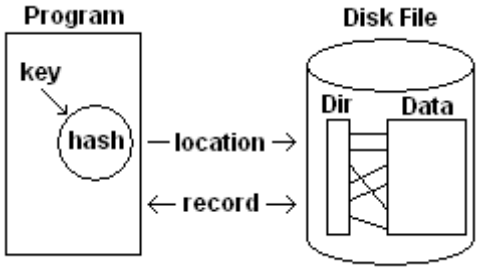
- How else can a single record in a large file be accessed?



Extendible Hashing

An application of hashing that works well with files that over time undergo substantial changes in size.

- How can hashing be applied go large, growing, files?



Disks

- Data is stored on a magnetic disk by controlling the direction in which small areas of the disk surface are magnetized.
- Data is stored on a disk serially, that is, one bit at a time.

Organization of Disks

Block: A physical data record, separated on the medium from other blocks by inter-block gaps.

Sector: A fixed size physical data block on a disk drive.

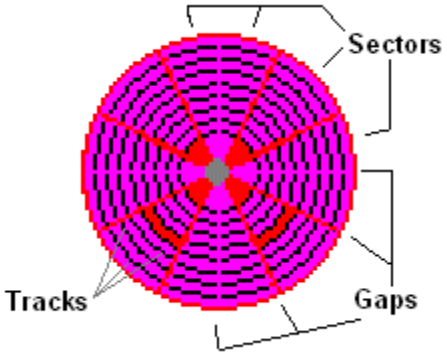
Interblock gap: An area between data blocks which contains no data and which separates the blocks.

Cylinder: The set of tracks of a disk drive which can be accessed without changing the position of the access arm.

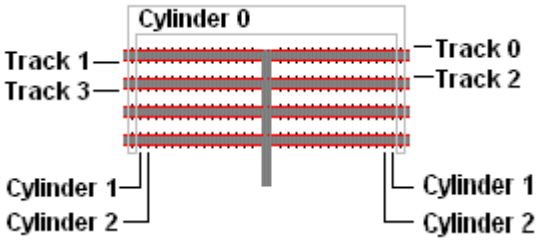
Track: The (circular) area on a disk platter which can be accessed without moving the access arm of the drive.

Disk Drive Physical Structure

- Data is recorded on each surface of the disk in sectors, which are located in concentric circles.
- The sectors are separated by gaps which contain no data.
- On PC disk drives, each sector contains 512 bytes of data.

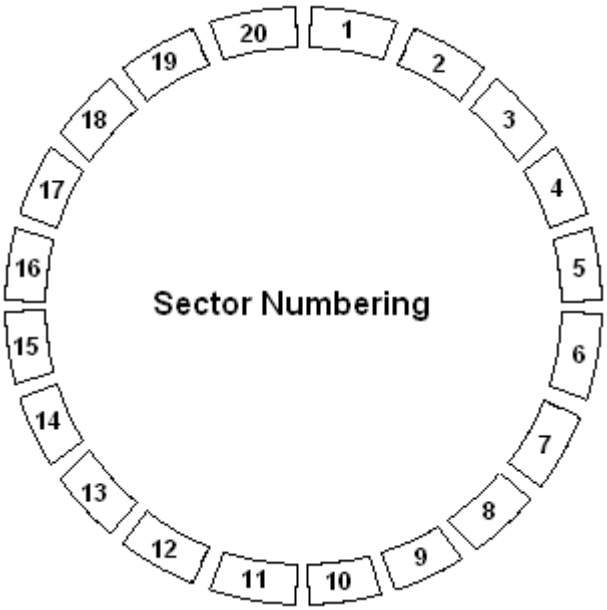


- Track and cylinder locations are determined by the physical geometry of the drive.
- Track and cylinder numbers begin with 0.
- Tracks are often referred to as heads.



Organizing Tracks by Sector

- Sector locations are determined by the electronics of the drive, and are identified by recorded address marks.
- Sector numbers begin with 1.
- Today, logically adjacent sectors are typically physically adjacent.



Estimating Capacities and Space Needs

- bytes/track = sectors/track * bytes/sector
- bytes/cylinder = tracks/cylinder * bytes/track
- bytes/drive = cylinders/drive * bytes/cylinder
- bytes/drive = cylinders/drive * tracks/cylinder * sectors/track * bytes/sector

File allocation

Cluster: A group of sectors handled as a unit of file allocation.

Extent: A physical section of a file occupying adjacent clusters.

Fragmentation: Unused space within a file.

- Clusters are also referred to as allocation units (ALUs).
- Space is allocated to files as integral numbers of clusters.
- A file can have a single extent, or be scattered in several extents.
- Access time for a file increases as the number of separate extents increases, because of seeking.
- Defragmentation utilities physically move files on a disk so that each file has a single extent.
- Allocation of space in clusters produces fragmentation.
- A file of one byte is allocated the space of one cluster.
- On average, fragmentation is one-half cluster per file.

Organizing Tracks by Block

- Mainframe computers typically use variable size physical blocks for disk drives.
- Track capacity is dependent on block size, due to fixed overhead (gap and address block) per block.

The Cost of a Disk Access

Direct access device: A data storage device which supports direct access.

Direct access: Accessing data from a file by record position with the file, without accessing intervening records.

Access time: The total time required to store or retrieve data.

Transfer time: The time required to transfer the data from a sector, once the transfer has begun.

Seek time: The time required for the head of a disk drive to be positioned to a designated cylinder.

Rotational delay: The time required for a designated sector to rotate to the head of a disk drive.

- Access time of a disk is related to physical movement of the disk parts.
- Disk access time has three components: seek time, rotational delay, and transfer time.
- Seek time is affected by the size of the drive, the number of cylinders in the drive, and the mechanical responsiveness of the access arm.
- Average seek time is approximately the time to move across 1/3 of the cylinders.
- Rotational delay is also referred to as *latency*.
- Rotational delay is inversely proportional to the rotational speed of the drive.
- Average rotational delay is the time for the disk to rotate 180°.
- Transfer is inversely proportional to the rotational speed of the drive.
- Transfer time is inversely proportional to the physical length of a sector.
- Transfer time is roughly inversely proportional to the number of sectors per track.
- Actual transfer time may be limited by the disk interface.

Current Technology

- Seagate 750 GB
- Western Digital 750 GB
- IBM 320 TB

Effect of Block Size

- Fragmentation waste increases as cluster size increases.
- Average access time decreases as cluster size increases.

Disk as a bottleneck

Striping: The distribution of single files to two or more physical disk drives.

Redundant Array of Inexpensive Disks: An array of multiple disk drives which appears as a single drive to the system.

RAM disk: A virtual disk drive which actually exists in main memory.

solid state disk

A solid state memory: array with an interface which responds as a disk drive.

Cache: Solid state memory used to buffer and store data temporarily.

- Several techniques have been developed to improve disk access time.
- Striping allows disk transfers to be made in parallel.
- There are 6 versions, or levels, of RAID technology.
- RAID-0 uses striping.
- RAID-0 improves access time, but does not provide redundancy.
- RAID-1 uses mirroring, in which two drives are written with the same data.

- RAID-1 provides complete redundancy. If one drive fails, the other provides data backup.
- RAID-1 improves read access time, but slows write access time.
- RAM disks appear to programs as fast disk drives.
- RAM disks are volatile.
- Solid state disks appear to computer systems as fast disk drives.
- Solid state disks are used on high performance data base systems.
- Caching improves average access time.
- Disk caching can occur at three levels: in the computer main memory, in the disk controller, and in the disk drive.
- Windows operating systems use main memory caching.
- Disk controller caching requires special hardware.
- Most disk drives now contain caching memory.
- With caching, writes are typically reported as complete when the data is in the cache.
- The physical write is delayed until later.
- With caching, reads typically read more data than is requested, storing the unrequested data in the cache.
- If a read can be satisfied from data already in the cache, no additional physical read is needed.
- Read caching works on average because of program locality.

File System Organization

- File Allocation Table
- A table on a disk volume containing chained lists of the physical locations of all files on the volume.
- index node
- A data structure, associated with a file, which describes the file.

I/O in UNIX

Block device: A device which transfers data in blocks (as opposed to character by character.)

Block I/O: Input or output performed in blocks

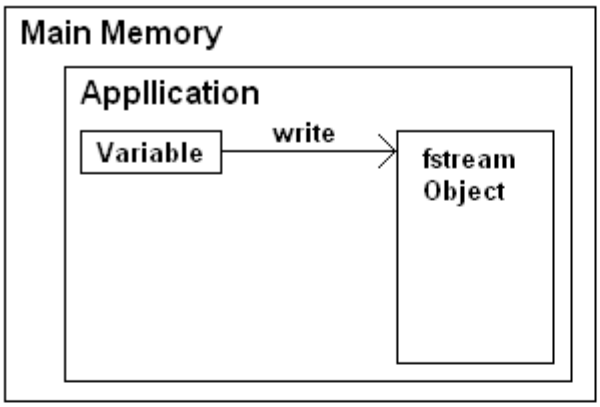
Character device: A device which transfers data character by character (as opposed to in blocks.)

Character I/O: Input or output performed character by character.

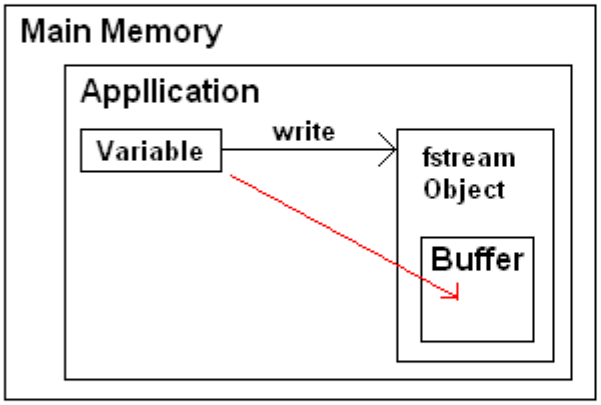
- Disks are block devices.
- Keyboards, displays, and terminals are character devices.

A Journey of a Byte

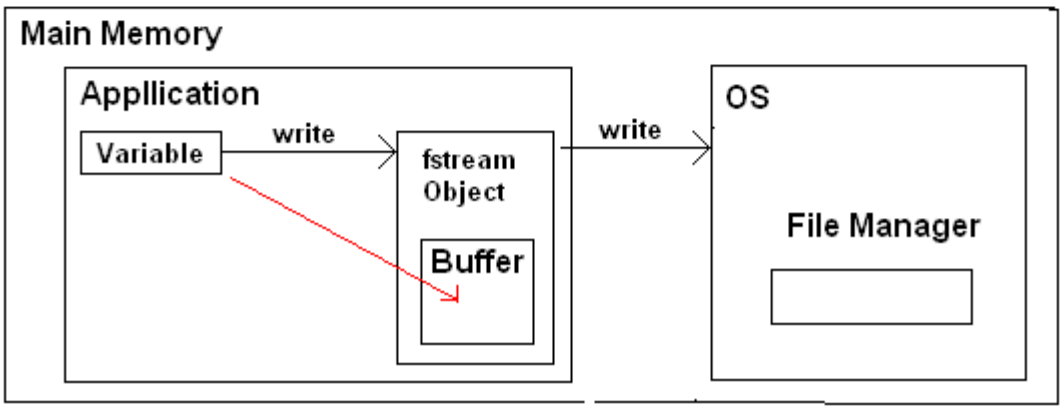
1. An application in main memory issues a write command to an fstream object.



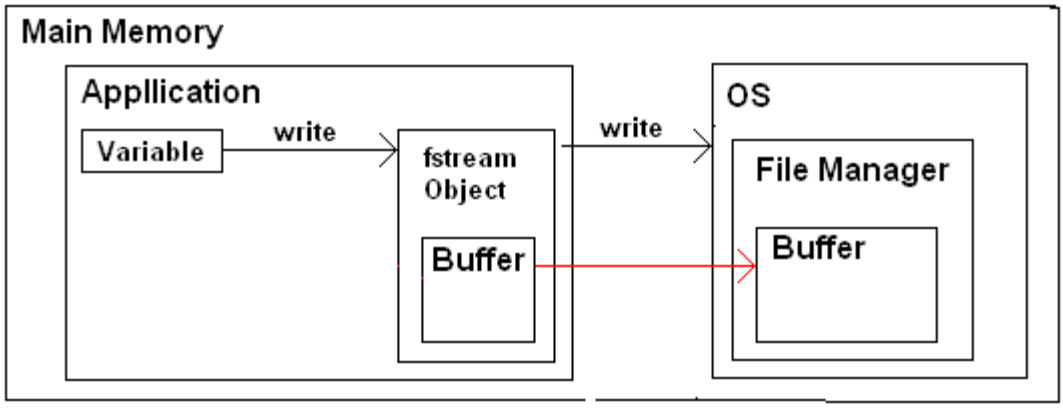
2. The fstream object copies the variable into an internal buffer.



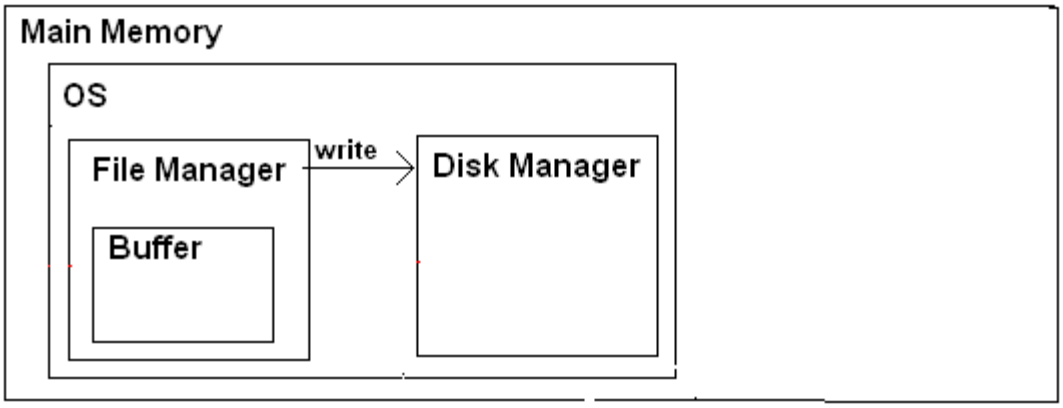
3. After repeated write, the fstream buffer fills up and the fstream object issues a write request to the operating system



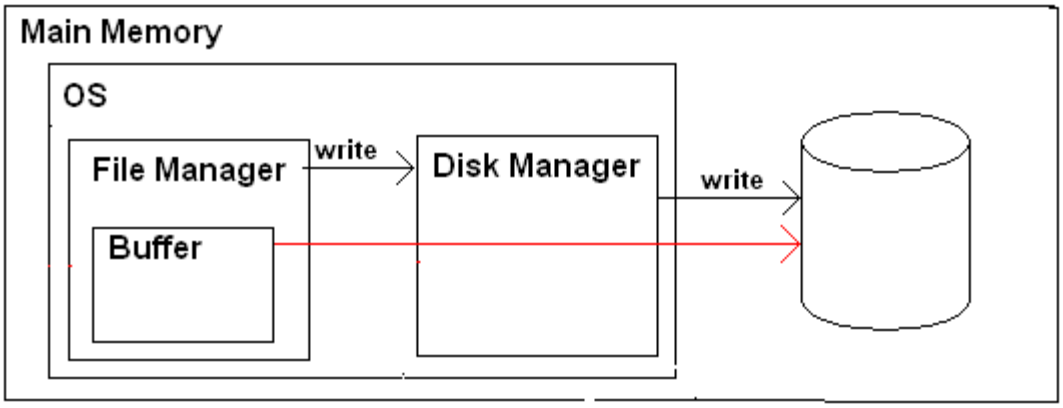
4. The operating system file manager copies the contents of the fstream buffer into an OS buffer.



5. After repeated writes, the OS buffer fills up, and the File Manager issues a write request to the OS disk manager.



6. The disk manager issues a write request to the disk drive

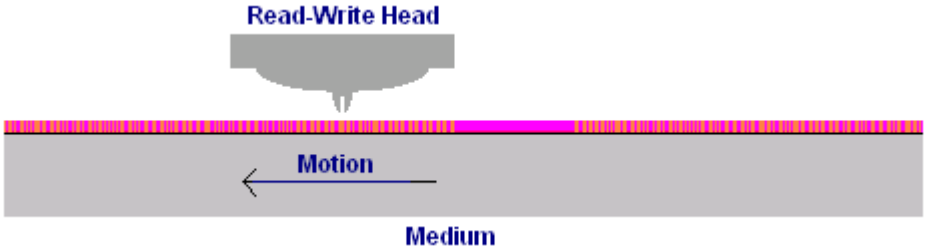


Magnetic Technology

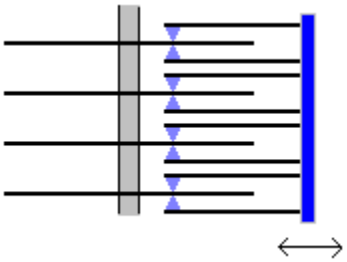
- Data is stored on a magnetic disk by controlling the direction in which small areas of the disk surface are magnetized.
- Data is stored on a disk serially, that is, one bit at a time.

- Data is recorded on magnetic disks as a series of magnitized areas on the surface of the disk:

- The data is read by a head with a coil which is sensitive to changes in magnetism on the data track as the disk rotates.

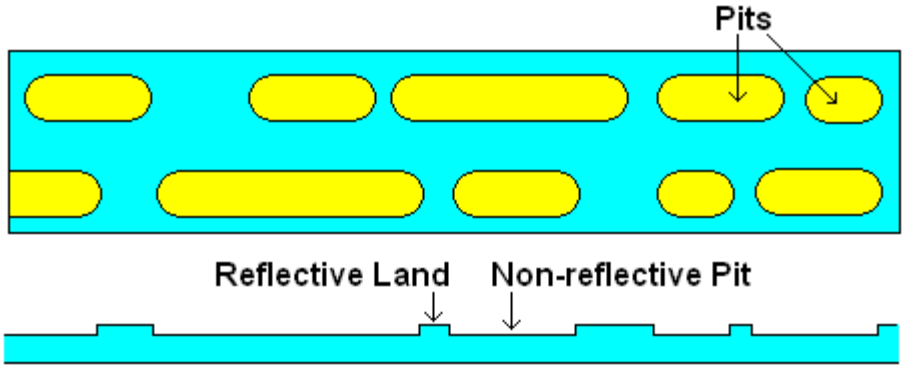


- The heads are attached to a common shuttle, which moves them in and out, to different redial positions, together:

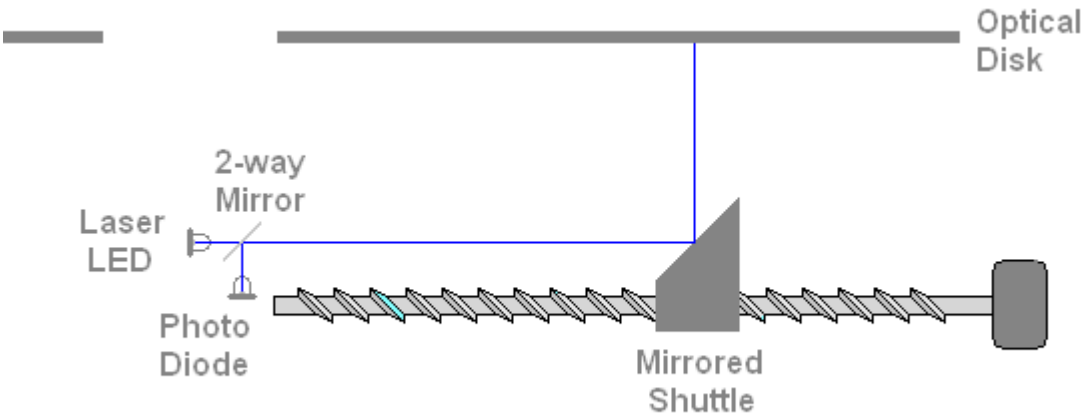


Optical Technology

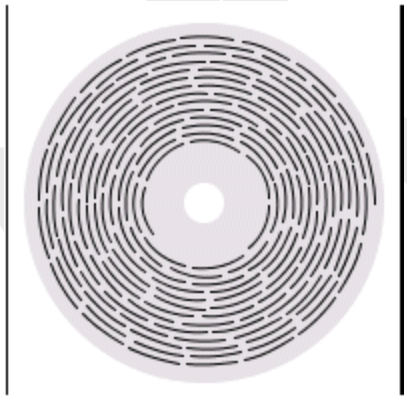
- Data is recorded on optical disks as a series of reflective and non-reflective areas.



- The data is read by focusing a laser beam on the data track as the disk rotates.



- The data is blocked into physical sectors, which are arranged in a single spiral track.



File Access with Handles

File descriptor: A cardinal number used as the identifier for a logical file by operating systems such as UNIX and PC-DOS.

- For handle level access, the logical file is declared as an *int*.
- The handle is also known as a *file descriptor*.

Opening

- The *open* function is used to open a file for handle level access.
- The handle open function must be supplied with (as arguments):
  - The name of the physical file
  - The access mode
  - For new files, the protection mode
- The value returned by the *open* is the handle, and is assigned to the file variable.
- Prototypes:

  int open (const char* Filename, int Access);

  int open (const char* Filename, int Access, int Protection);

- Example:

  int Input;

  Input = open ("Daily.txt", O_RDONLY);
- The following flags can be bitwise *or*ed together for the access mode:

  O_RDONLY - Read only

  O_WRONLY - Write only

  O_RDWR - Read or write

  O_BINARY - Binary mode

  O_TEXT - Text (ASCII) mode

  O_CREAT - Create file if it does not exist

  O_EXCL - Do not open existing file (used only with O_CREAT)

  O_APPEND - Leave any existing file contents and set file pointer at end of file

  O_TRUNC - Delete any prior file contents
- The following flags can be bitwise *or*ed together for the protection mode (which is required if the O_CREAT flag is set):

  S_IREAD - Read permission

  S_IWRITE - Write permission

Closing

- The *close* function is used to close a file for handle level access.
- The handle close function must be supplied with (as an argument):
  - The handle of the logical file
- The value returned by the *close* is 0 if the close succeeds, and -1 if the close fails..

Prototypes:

int close (int Handle);

Example:

close (Input);

Reading

- The *read* function is used to read data from a file for handle level access.
- The handle read function must be supplied with (as an arguments):
  - The handle of the logical file
  - The address of the buffer into which the data will be read
  - The number of bytes to be read
- The value returned by the *read* function is the number of bytes read.

Prototypes:

int read (int Handle, void * Buffer, unsigned Length);

Example:

read (Input, &C, 1);

Writing

- The *write* function is used to write data to a file for handle level access.
- The handle write function must be supplied with (as an arguments):
  - The handle of the logical file
  - The address of the buffer from which the data will be written
  - The number of bytes to be write
- The value returned by the *write* function is the number of bytes written.

Prototypes:

int write (int Handle, void * Buffer, unsigned Length);

Example:

write (Output, &C, 1);

Seeking

- The *lseek* function is used to move the file pointer of a file identified by its handle.
- The handle lseek function must be supplied with (as an arguments):
  - The handle of the logical file
  - The reference point of the seek
  - The offset from the designated reference point
- The value returned by the *lseek* function is the new offset of the file pointer from the beginning of the file

Prototypes:

long lseek (int Handle, long Offset, int Origin);

- Example:

  lseek (Output, 100, SEEK_BEG);

- The Origin argument should be one of the folloing, to designate the reference point:

  SEEK_SET - Beginning of file

  SEEK_CUR - Current file position

  SEEK_END - End of file