How fast is main memory?

- Typical time for getting info from:

Main memory: ~12 nanosec = 120×10^{-8} sec

Magnetic disks: ~30 milisec = 30×10^{-3} sec

An analogy keeping same time proportion as above:

Looking at the index of a book : 20 sec

versus

Going to the library: 58 days

Normal Arrangement

- Secondary storage (SS) provides reliable, long-term storage for large volumes of data
- At any given time, we are usually interested in only a small portion of the data
- This data is loaded temporarily into main memory, where it can be rapidly manipulated and processed.
- As our interests shift, data is transferred automatically between MM and SS, so the data we are focused on is always in MM.

Goal of the file structures

- Minimize the number of trips to the disk in order to get desired information
- Grouping related information so that we are likely to get everything we need with only one trip to the disk.

Physical Files and Logical Files

- **physical file:** a collection of bytes stored on a disk or tape
- **logical file:** a "channel" (like a telephone line) that connects the program to a physical file

C++ streams

- The program (application) sends (or receives) bytes to (from) a file through the logical file. The program knows nothing about where the bytes go (came from)
- The operating system is responsible for associating a logical file in a program to a physical file in disk or tape. Writing to or reading from a file in a program is done through the operating system.

Files

- The physical file has a name, for instance `myfile.txt`
- The logical file has a logical name (a variable) inside the program.
  - In C :

```
FILE * outfile;
```

  - In C++:

```
fstream outfile;
```

Basic File Processing Operations

- Opening
- Closing
- Reading
- Writing
- Seeking

File Systems

contains

- Stored data is organized into **files**.
- Files are organized into **records**.
- Records are organized into **fields**.

13

Example

- A student file may be a collection of student records, one record for each student
- Each student record may have several fields, such as
  - Name
  - Address
  - Student number
  - Gender
  - Age
  - GPA
- Typically, each record in a file has the same fields.

14

Properties of Files

1) Persistance: Data written into a file persists after the program stops, so the data can be used later.
2) Sharability: Data stored in files can be shared by many programs and users simultaneously.
3) Size: Data files can be very large. Typically, they cannot fit into main memory.

15

Secondary Storage Devices

16

Secondary Storage Devices

> Two major types of storage devices:

1. Direct Access Storage Devices (DASDs)
   - Magnetic Disks
     Hard disks (high capacity, low cost per bit)
     Floppy disks (low capacity, slow, cheap)
   - Optical Disks
     CD-ROM = (Compact disc, read-only memory)
     DVD
2. Serial Devices
   - Magnetic tapes (very fast sequential access)

17

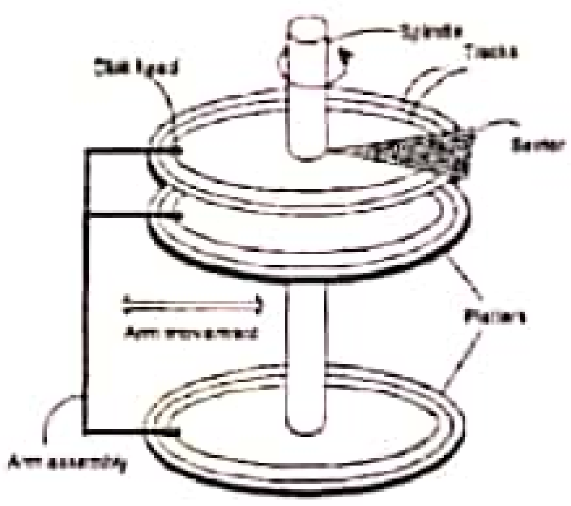
Magnetic Disks

- Bits of data (0's and 1's) are stored on circular magnetic platters called disks.
- A disk rotates rapidly (& never stops).
- A disk head reads and writes bits of data as they pass under the head.
- Often, several platters are organized into a disk pack (or disk drive).

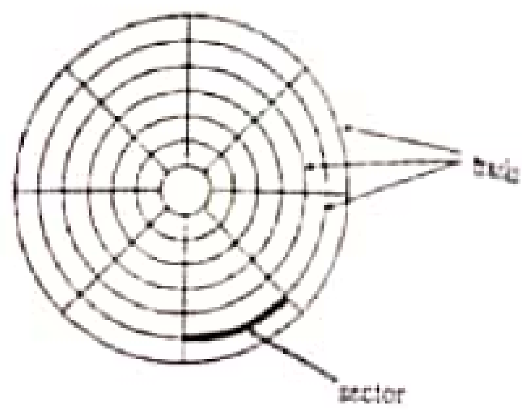
18

Top view of a 36 GB, 10,000 RPM, IBM SCSI server hard disk, with its top cover removed.
Note the height of the drive and the 10 stacked platters.
(The IBM Ultrastar 36ZX.)

Components of a Disk



Looking at a surface



Surface of disk showing tracks and sectors

concentric = ortak merkezli

Organization of Disks

- Disk contains concentric **tracks**.
- Tracks are divided into **sectors**
- A sector is the smallest addressable unit in a disk.
- Sectors are addressed by:
  - surface #
  - cylinder (track) #
  - sector #

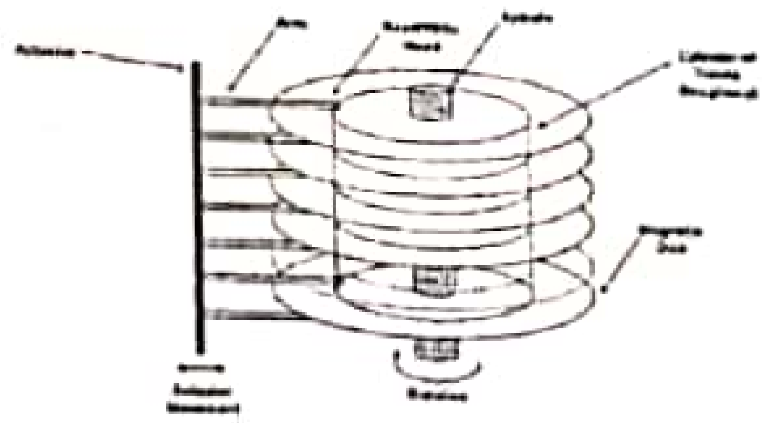
Accessing Data

- When a program reads a byte from the disk, the operating system locates the surface, track and sector containing that byte, and reads the entire sector into a special area in main memory called **buffer**.
- The bottleneck of a disk access is moving the read/write arm. So it makes sense to store a file in tracks that are below/above each other in different surfaces, rather than in several tracks in the same surface.

Cylinders

- A **cylinder** is the set of tracks at a given radius of a disk pack.
  - i.e. a cylinder is the set of tracks that can be accessed without moving the disk arm.
- All the information on a cylinder can be accessed without moving the read/write arm.

Cylinders



cluster = multiple sectors grouped together

Clusters

- Another view of sector organization is the one maintained by the O.S.'s **file manager**.
- It views the file as a series of **clusters** of sectors.
- File manager uses a **file allocation table (FAT)** to map logical sectors of the file to the physical clusters.

FAT is not efficient enough for bigger capacities. NTFS is mostly used for newer versions of Windows and flash drives uses exFAT.

Extents

- If there is a lot of room on a disk, it may be possible to make a file consist entirely of contiguous clusters. Then we say that the file is one **extent**. (very good for sequential processing)
- If there isn't enough contiguous space available to contain an entire file, the file is divided into two or more noncontiguous parts. Each part is an extent.

Extent: serious of clusters

Fragmentation

➢ Internal fragmentation: loss of space within a sector or a cluster.

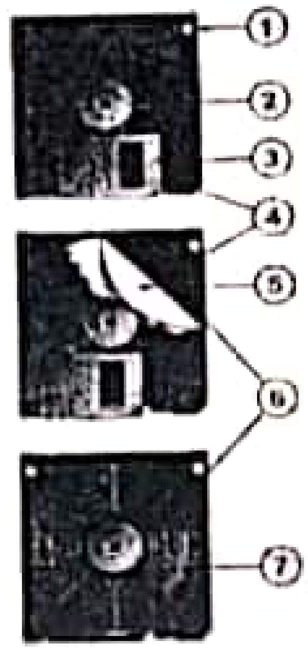
1) Due to records not fitting exactly in a sector: e.g. Sector size is 512 and record size is 300 bytes. Either-
   - store one record per sector, or
   - allow records *span* sectors.
2) Due to the use of clusters: If the file size is not a multiple of the cluster size, then the last cluster will be partially used.

Secondary Storage Devices: Floppy Disks

Floppy Disks

A floppy disk is a disk storage medium composed of a disk of thin and flexible magnetic storage medium.

Developed by IBM
3.5-inch, 5.24-inch and 8-inch forms



Internal parts of a 3½-inch floppy disk

1)A hole that indicates a high capacity disk.
2)The hub that engages with the drive motor
3)A shutter that protects the surface when removed from the drive
4)The plastic housing
5)A polyester sheet reducing friction against the disk media as it rotates within the housing
6)The magnetic coated plastic disk
7)A schematic representation of one sector of data on the disk; the tracks and sectors are not visible on actual disks.

Floppy Disks

A spindle motor in the drive rotates the magnetic medium at a certain speed

A stepper motor-operated mechanism moves the magnetic read/write head(s) along the surface of the disk

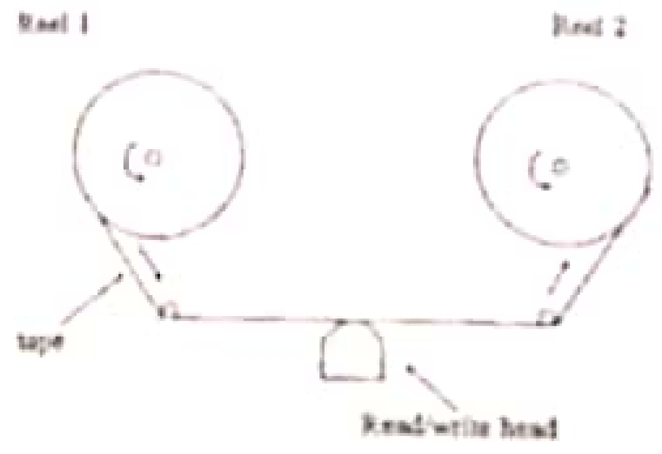
Secondary Storage Devices: Magnetic Tapes

Characteristics

- No direct access, but very fast sequential access.
- Resistant to different environmental conditions.
- Easy to transport, store, cheaper than disk.
- Before it was widely used to store application data; nowadays, it's mostly used for backups or archives

Magnetic tapes

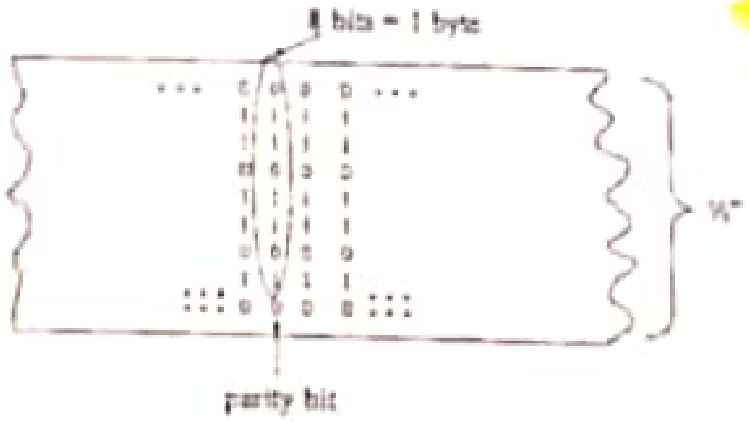
- A sequence of bits are stored on magnetic tape.
- For storage, the tape is wound on a reel.
- To access the data, the tape is unwound from one reel to another.
- As the tape passes the head, bits of data are read from or written onto the tape.



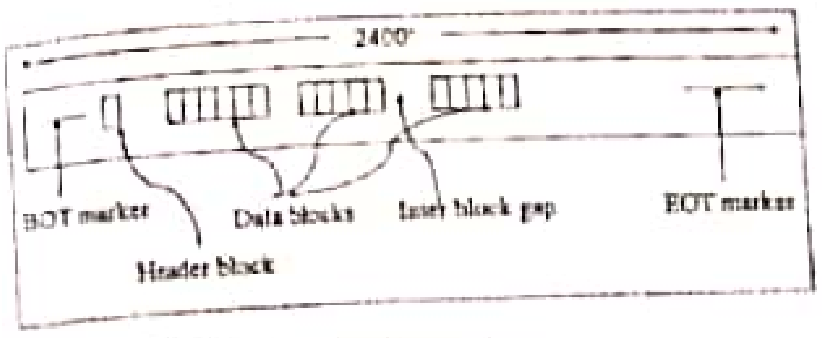
Tracks

- Typically data on tape is stored in 9 separate bit streams, or tracks.
- Each track is a sequence of bits.
- Recording density = # of bits per inch (bpi). Typically 800 or 1600 bpi. 30000 bpi on some recent devices.

In detail



Tape Organization



BOT – beginning of tape, EOT – end of tape
Header block: describes data blocks
Inter block gap: For acceleration and deceleration of tape
Blocking factor: # records per block

Data Blocks and Records

- Each data block is a sequence of contiguous records.
- A record is the unit of data that a user's program deals with.
- The tape drive reads an entire block of records at once.
- Unlike a disk, a tape starts and stops.
- When stopped, the read/write head is over an interblock gap.

Example: tape capacity

- Given the following tape:
  - Recording density = 1600 bpi
  - Tape length = 2400 ' (feet)
  - Interblockgap = ½ " (inch)
  - 512 bytes per record
  - Blocking factor = 25
- How many records can we write on the tape? (ignoring BOT and EOT markers and the header block for simplicity)

Solution

- #bytes/block = (512 bytes/record) * (25 records/block)
  = 12,800 bytes/block
- Block length = (#bytes/block) / (#bytes/inch)
  = 12,800/1600 inches = 8 inches
- Block + gap = 8" + 1/2" = 8.5"
- Tape length = 2400 ft * 12 in/ft = 28,800 in
- #blocks = (tape length) / (block + gap)
  = 28,800/8.5 = 3388 blocks
- #records = (#blocks) * (#records/block)
  = 3388 * 25 = 84,700 records

Secondary Storage Devices: CD-ROM

Physical Organization of CD-ROM

- Compact Disk – read only memory (write once)
- Data is encoded and read optically with a laser
- Can store around 600MB data
- Digital data is represented as a series of Pits and **Lands:**
  - Pit = a little depression, forming a lower level in the track
  - Land = the flat part between pits, or the upper levels in the track

Organization of data

- Reading a CD is done by shining a laser at the disc and detecting changing reflections patterns.
  - 1 = change in height (land to pit or pit to land)
  - 0 = a "fixed" amount of time between 1's

```
 LAND      PIT     LAND PIT     LAND
...--------.       .------------.        .----...
           |_______|            |________|
..0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 ..
```

- Note ; we cannot have two 1's in a row!
  => uses Eight to Fourteen Modulation (EFM) encoding table

Properties

- Note that: Since 0's are represented by the length of time between transitions, we must travel at **constant linear velocity (CLV)** on the tracks.
- Sectors are organized along a spiral
- Sectors have same linear length
- Advantage: takes advantage of all storage space available.
- Disadvantage: has to change rotational speed when seeking (slower towards the outside)

Addressing

- 1 second of play time is divided up into 75 **sectors**.
- Each sector holds 2KB
- 60 min CD:
  60min * 60 sec/min * 75 sectors/sec = 270,000 sectors = 540,000 KB ~ 540 MB
- A sector is addressed by:
  Minute:Second:Sector
  e.g. 16:22:34

DVD (Digital Video Disc) Characteristics

- A DVD disc has the same physical size as a CD disc, but it can store from 4,7 to 17 GB of data.
- Like a CD disc, data is recorded on a DVD disc in a spiral trail of tiny pits separated by lands.
- The DVD's larger capacity is achieved by making the pits smaller and the spiral tighter, and by recording the data as many as four layers, two on each side of the disc.
- To read these tightly packed discs, lasers that produce a shorter wavelength beam of light are required to achieve more accurately aiming and focusing mechanism. In fact, the focusing mechanism is the technology that allows data to be recorded on two layers. To read the second layer, the reader simply focuses the laser a little deeper into the disc, where the second layer of data is recorded.

Secondary Storage: Flash Memory

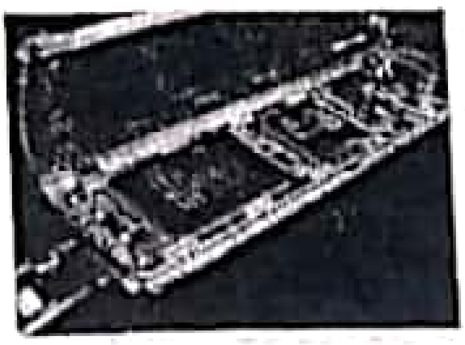
Flash Memory

- Non-volatile computer storage chip that can be electrically erased and reprogrammed.
- It was developed from EEPROM (electrically erasable programmable read-only memory)
- The NAND type: primarily used in memory cards, USB flash drives, for general storage and transfer of data.
- The NOR type: used as a replacement for the older EPROM and as an alternative to certain kinds of ROM applications.

Flash Memory

Replacement for hard disks:

- Adv: Flash memory does not have the mechanical limitations and latencies of hard drives
- Disadv: The cost per gigabyte of flash memory remains significantly higher than that of hard disks.

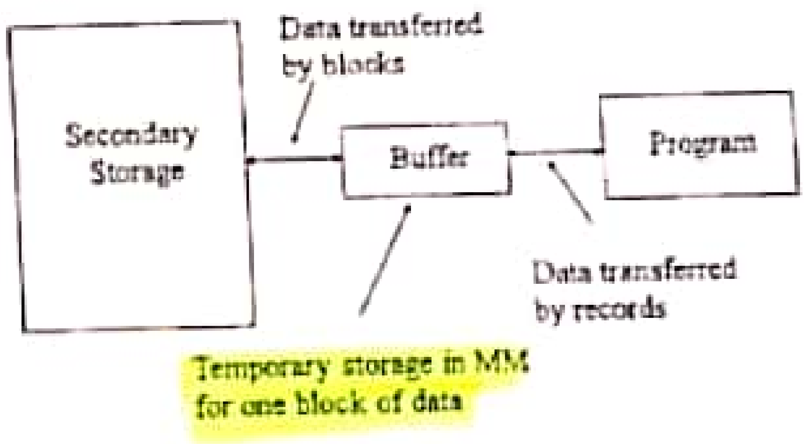


Buffer Management

Buffer Management

- Buffering means working with large chunks of data in main memory so the number of accesses to secondary storage is reduced.
- System I/O buffers: These are beyond the control of application programs and are manipulated by the O.S.

System I/O Buffer



Buffer Bottlenecks

- Consider the following program segment:

```
while (1) {
  infile >> ch;
   if (infile.fail()) break;
   outfile << ch;
}
```

- What happens if the O.S. used only one I/O buffer?
  ⇒ Buffer bottleneck
- Most O.S. have an input buffer and an output buffer.

Buffering Strategies

- **Double Buffering:** Two buffers can be used to allow processing and I/O to overlap.
  - Suppose that a program is only writing to a disk.
  - CPU wants to fill a buffer at the same time that I/O is being performed.
  - If two buffers are used and I/O-CPU overlapping is permitted, CPU can be filling one buffer while the other buffer is being transmitted to disk.
  - When both tasks are finished, the roles of the buffers can be exchanged.
- The actual management is done by the O.S.

Other Buffering Strategies

- Multiple Buffering: instead of two buffers any number of buffers can be used to allow processing and I/O to overlap.
- Buffer pooling:
  - There is a pool of buffers.
  - When a request for a sector is received, O.S. first looks to see that sector is in some buffer.
  - If not there, it brings the sector to some free buffer. If no free buffer exists, it must choose an occupied buffer. (usually LRU strategy is used)

2

Fundamental File Structure Concepts

Outline

- Field and record organization
- Sequential search and direct access
- Sequential Files
  - Heap (unsorted pile) file
  - Sorted Sequential Files
- Co-sequential processing

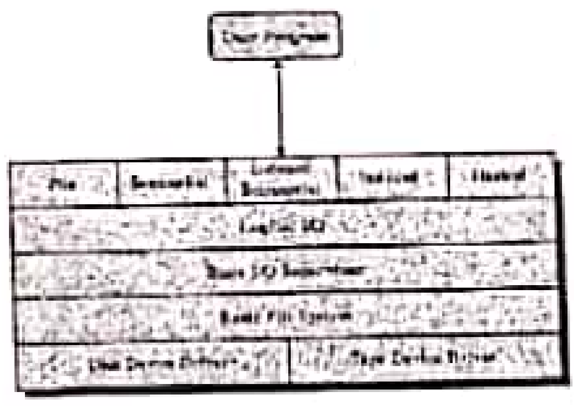


Figure 12.2 File System Software Architecture

Files

A file can be seen as

1. A stream of bytes (no structure, data semantics is lost),
2. A collection of records with fields

Field and Record Organization

Definitions

__Record:__ a collection of related fields.

__Field__: the smallest logically meaningful unit of information in a file.

__Key__: a subset of the fields in a record used to identify (uniquely) the record.

e.g. In the example file of books:
- Each line corresponds to a record.
- Fields in each record: ISBN, Author, Title

Record Keys

- **Primary key**: a key that uniquely identifies a record.
- **Secondary key**: other keys that may be used for search
  - Author name
  - Book title
  - Author name + book title
- Note that in general not every field is a key (keys correspond to fields, or a combination of fields, that may be used in a search).

Record Id (rid)

- Each record in a file has a unique identifier called record id (rid)
- We can identify the disk address of the page containing the record by using the rid
- Indices include data entries including <key, rid>

Field Structures

- Fixed-length fields

```
87359Carroll   Alice in wonderland
38180Folk      File Structures
```

- Begin each field with a length indicator

```
058735907Carroll19Alice in wonderland
053818004Folk15File Structures
```

- Place a delimiter at the end of each field

```
87359|Carroll|Alice in wonderland|
38180|Folk|File Structures|
```

- Store field as keyword = value

```
ISBN=87359|AU=Carroll|TI=Alice in wonderland|
ISBN=38180|AU=Folk|TI=File Structures
```

Type	Advantages	Disadvantages
Fixed	Easy to read/write	Waste space with padding
Length-based	Easy to jump ahead to the end of the field	Long fields require more than 1 byte
Delimited	May waste less space than with length-based	Have to check every byte of field against the delimiter
Keyword	Fields are self describing. Allows for missing fields	Waste space with keywords

Record Structures

1. Fixed-length records.
2. Variable-length records.

-Öğrenme süreçleri
Yazı ve dil iletişimi
-İletişimdeki hatalar

Fixed-length records

Two ways of making fixed-length records:

1. Fixed-length records with fixed-length fields.

```
87359   Carroll       Alice in wonderland
38180   Folk          File Structures
```

2. Fixed-length records with variable-length fields.

```
87359|Carroll|Alice in wonderland|    unused
38180|Folk|File Structures|           unused
```

Variable-length records

- Fixed number of fields:

```
87359|Carroll|Alice in wonderland|38180|Folk|File Structures| ...
```

- Record beginning with length indicator:

```
3387359|Carroll|Alice in wonderland|2638180|Folk|File Structures| ..
```

- Use an index file to keep track of record addresses:
  - The index file keeps the byte offset for each record; this allows us to search the index (which have fixed length records) in order to discover the beginning of the record.
- Placing a delimiter: e.g. end-of-line char

Type	Advantages	Disadvantages
Fixed length record	Easy to jump to the i-th record	Waste space with padding
Variable-length record	Saves space when record sizes are diverse	Cannot jump to the i-th record, unless through an index file

13

File Organization

- Four basic types of organization:
  1. **Sequential** ← today
  2. Indexed
  3. Indexed Sequential
  4. Hashed
- In all cases we view a file as a sequence of records.
- A record is a list of fields. Each field has a data type.

14

File Operations

- Typical Operations:
  - Retrieve a record
  - Insert a record
  - Delete a record
  - Modify a field of a record
- In direct files:
  - Get a record with a given field value
- In sequential files:
  - Get the next record

Hız avantajı açısından evet

Sequential files

- Records are stored contiguously on the storage device.
- Sequential files are read from beginning to end.
- Some operations are very efficient on sequential files (e.g. finding averages)
- Organization of records:
  1. Unordered sequential files (pile files/heap files)
  2. Sorted sequential files (records are ordered by some field)

16

Öğrenci agno'su örneğinde dosyanın iki versiyonu da tutulur mu?

ordered unordered

Pile Files

- A pile file is a succession of records, simply placed one after another with no additional structure.
- Records may vary in length.
- Typical Request:
  - Print all records with a given field value
    - e.g. print all books by *Folk*.
  - We must examine each record in the file, in order, starting from the first record.

17

Searching Pile Files

- To look-up a record, given the value of one or more of its fields, we must search the whole file.
- In general, (b is the total number of blocks in file):
  - At least 1 block is accessed
  - At most b blocks are accessed.
  - On average $1/b * b(b+1)/2 \rightarrow b/2$
- Thus, time to find and read a record in a pile file is approximately :

$$T_F = (b/2) * btt$$

Time to fetch one record

18

Exhaustive Reading of the File

Read and process all records (reading order is not important)

$$T_X = b * btt$$

(approximately twice the time to fetch one record)

- e.g. Finding averages, min or max, or sum.
  - Pile file is the best organization for this kind of operations.
  - They can be calculated using double buffering as we read though the file once.

11

Inserting a new record

- Just place the new record at the end of the file (assuming that we don't worry about duplicates)
  - Read the last block
  - Put the record at the end.

  $\Rightarrow T_I = s + r + btt + \underbrace{(2r - btt)}_{\text{Time to wait}} + btt$

  $\Rightarrow T_I = s+r+btt+2r$

- What if the last block is full?

Updating a record

- For fixed length records:

  T_U (fixed length) $= T_F + 2r$

- For variable length records update is treated as a combination of delete and insert.

  T_U (variable length) $= T_D + T_I$

21

Strategies for record deletion

1. **Record deletion and Storage compaction:**
   - Deletion can be done by "marking" a record as deleted.
     - e.g. Place '*' at the beginning of the record
   - The space for the record is not released, but the program must include logic that checks if record is deleted or not.
   - After a lot of records have been deleted, a special program is used to squeeze the file - this is called **Storage Compaction**.

Strategies for record deletion (cont.)

2. **Deleting fixed-length records and reclaiming space dynamically.**
   - How to use the space of deleted records for storing records that are added later?
     - Use an "AVAIL LIST", a linked list of available records.
     - A header record (at the beginning of the file) stores the beginning of the AVAIL LIST
     - When a record is deleted, it is marked as deleted and inserted into AVAIL LIST

21

Strategies for record deletion (cont.)

3. **Deleting variable length records**
   - Use AVAIL LIST as before, but take care of the variable-length difficulties.
   - The records in AVAIL LIST must store its size as a field. Exact byte offset must be used
   - Addition of records must find a large enough record in AVAIL LIST

Placement Strategies

- There are several placement strategies for selecting a record in AVAIL LIST when adding a new record:

1. First-fit Strategy
   - AVAIL LIST is not sorted by size; first record large enough is used for the new record
2. Best-fit Strategy
   - List is sorted by size. Smallest record large enough is used
3. Worst-fit strategy
   - List is sorted by decreasing order of size, largest record is used; unused space is placed in AVAIL LIST again

Exercise 1

- Estimate the time required to reorganize a file for storage compaction. (Derive a formula in terms of the number of blocks, b, in the original file; btt; number of records n in the new file and blocking factor Bfr)
   i. Reorganize the file with one disk drive
   ii. Reorganize the file with two disk drives

Solution

i) Reorganizing with one disk drive:
   - ✓ Read records into memory eliminating deleted records
   - ✓ When memory fills, write reorganized blocks to the disk
   - ✓ Continue in this fashion until the end of the file

- Time to read the old file = b * btt
- Time to write the new file = (n/Bfr) * btt
- Total time = (b + n/Bfr) * btt

Solution (cont.)

ii) Reorganizing with two disk drives
   - ✓ Read in the old file with one disk and simultaneously write the reorganized file on the other disk.
   - ✓ Input and output speeds may not be the same, so slower process can use larger buffers

- Writing out will be overlapped by reading in:

Total time = b * btt

Exercise 2

- Given two pile files A, B with
  - n=100,000
  - R=400 bytes each,
- Create an <u>intersection</u> file
- Assume that 70% of the records are in common and the available memory for this operation is 10M.
- Calculate a timing estimate for deriving and writing the intersection file
  - Use s = 16 ms,
  - r = 8.3ms,
  - btt = 0.84ms,
  - B=2400bytes)

Solution

Procedure:
- Read file A (10 M)
- Compare each record with file B, marking common records
- Write out 7 M to intersection file
- Read in next 10 M from A

Calculations:
- T_a(file A) = 4 * (s + r) + b * btt (reading A in 4 segments)

b = 40 M/2400 = 16,667 blocks

T_a = 4 * (16+8.3) + 16,667 * 0.84

T_a = 14 sec

- Writing intersection file : 70% of 14 sec = 10 sec
- Comparisons:

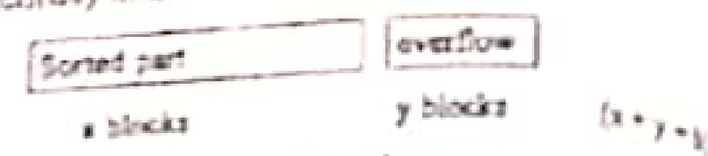
70000 * 7 sec + 30000 * 14 sec = 910000 sec = 10.5 days

Sorted Sequential Files

- Sorted files are usually read sequentially for processing.
- A sorted file cannot stay in order after additions (usually it is used as a temporary file).
- A sorted file will have an overflow area of added records. Overflow area is not sorted.
- To find a record:
  - First look at sorted area
  - Then search overflow area
  - If there are too many overflows, the access time degenerates to that of a sequential file

Searching for a record

- We can do binary search (assuming fixed-length records) in the sorted part.



- Worst case to fetch a record :

$$T_F = \log_2 x * (s + r + btt).$$

- If the record is not found, search the overflow area too. Thus total time is:

$$T_F = \log_2 x * (s + r + btt) + s + r + (y/2) * btt$$

Exercise 3

➢ Given the following:
- Block size = 2400
- File size = 40M
- Block transfer time (btt) = 0.84ms
- s = 16ms
- r = 8.3 ms

I) Calculate T_F for a certain record
  a) in a pile file
  b) in a sorted file (no overflow area)

II) Calculate the time to look up 10000 names.

Solution

I) a) T_F for pile file = b/2 * 0.84 = 7 sec.
b) T_F for sorted file = log b * (s +r+btt)

40 M file => b = 16,667 => log b = 13

T_F for sorted file = 13 * (16+8.3+0.84) = 326 ms

II) 10000 names:

Pile file => 19 hrs.

Sorted file => 54 min.

Co-sequential Processing

- Co-sequential processing involves the coordinated processing of two or more sequential files to produce a single output file.
- Two main types of resulting file are:
  - Matching (intersection) of the records in the files
  - Merging (union) of the records in the files.

Examples of applications

1. Matching:
   i. Finding the intersection file
   ii. Batch Update
      - Master file – bank account info (account number, person name, balance) - sorted by account number
      - Transaction file – updates on accounts (account number, credit/debit info) - sorted by account number
2. Union (Merging):
   i. Merging two class lists keeping alphabetic order.
   ii. Sorting large files (break into small pieces, sort each piece and them merge them)

Exercise 4: Intersection file

- Solve Problem 2 with two sorted files.
- Solution:
  - Read a segment from A
  - Start reading file B and compare records, marking common records
  - As soon as we find a record in B whose key is greater than the largest value in memory write out the matched records from A
  - Read the next segment from A
- Thus we read through each file once:
  - Read file A => 14 sec.
  - Read file B => 14 sec.
  - Write intersection => 10 sec.
  - Total = 38 sec.

Algorithm for Co-sequential Batch Update

1. Initialize pointers to the first records in the master file and transaction file.
2. Do until pointers reach end of file:
   i. Compare keys of the current records in both files
   ii. Take appropriate action *

Appropriate Action

if master key < transaction key
- copy master file record to the end of the new master file
- advance master file pointer

if master key > transaction key
- if transaction is an insert
  copy transaction file record to the end of new master file
  else log an error
- advance the transaction file pointer

if master key = transaction key
- If transaction is modify
  copy modified master file record to the end of the new file
- If transaction is insert log an error
- If transaction is delete, do nothing
- In all three cases, advance both the master and transaction file pointer

Reorganization of a sorted file

- Merging the sorted part of a file with its overflow area
  - Read in the overflow area blocks,
  - sort the records in memory,
  - if possible merge the sorted records with sorted part of the file into another sorted file.

Anlamadim

Reorganization of a sorted file

- Assume that there are x blocks in the sorted area and y blocks and z records in the overflow area.
- $T_Y = y * btt + T_{sort} + x * btt + (n/m) * btt$
  - Where n is total number of records, m is the blocking factor
  - Tsort is the time it takes to sort z records in the overflow area, say (zlogz)*t, where t is the time each iteration of the sort routine takes. This time is usually in microsecond domain and can be ignored.

disregarded : hice saymak
concentric : ortak merkezli
housekeeping : optimization of a harddrive

rigidly : katı
revolve : dönmek
compromise : uyuşma

FILE ORGANISATIONS

Introduction

Magnetic disk storage is available in many forms, including floppies, hard-disks, cartridge, exchangeable multi-platter, and fixed disks. The following deals with the concepts which are applied, in many different ways, to all of the above methods.

A typical disk pack comprises of 6 disks held on a central spindle. As the top and bottom are disregarded with recording information, only 10 surfaces are used, each with 200 concentric recording tracks. A diagrammatic representation is shown in Figure 1.

Each of the 200 tracks will be capable of holding equal amounts of information as this is a feature that is provided by the special software (housekeeping) that is used in conjunction with the handling of disk files. When the unit is in position in the drive, the tracks are accessed by a comb of 10 read/write heads held rigidly on an arm assembly which moves in and out between the disk as illustrated. In this way 10 tracks may be referenced without further movement of the heads once they are positioned over the first track. For the sake of economy it is therefore usual to record over a 'cylinder' of data (see Figure 1) and avoid unnecessary movement of the heads.

The disk pack is loaded onto the drive and revolves at normal speed of approximately 3600 rpm when being accessed. Each track can hold upwards of 4000 bytes. Sometimes the sectors on a disk are used in a similar manner to inter-block gaps on a tape file but in other cases they are ignored as far as data handling is concerned.

Because this device has the ability to locate an area of data directly it is known as a Direct Access Device and is popular for the versatility that this affords. Direct access devices provide a storage medium which is a satisfactory compromise between main store and magnetic tape.

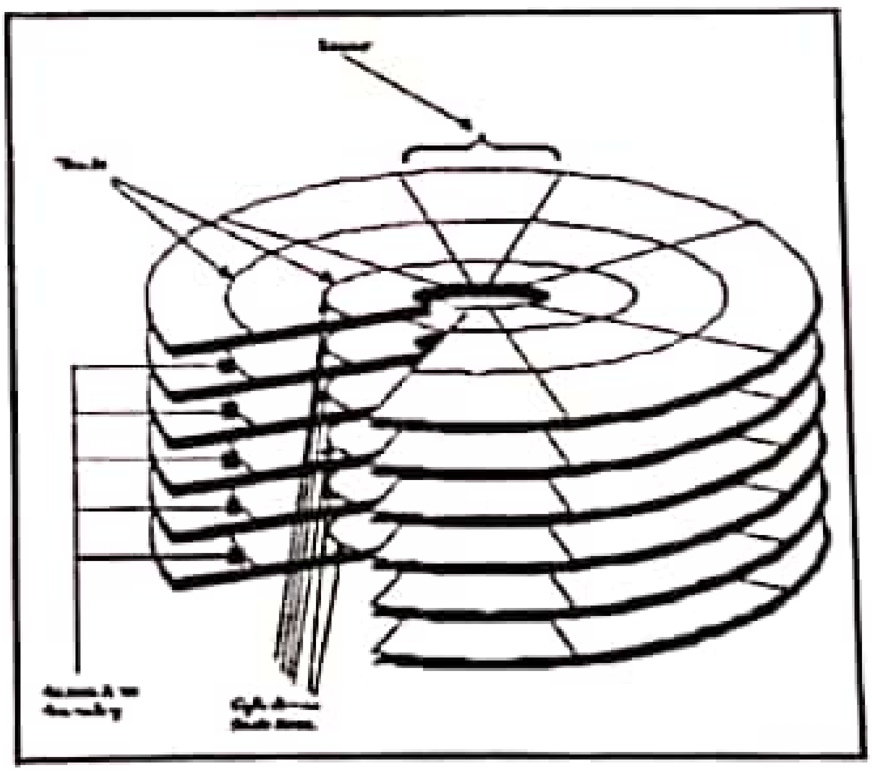


Figure 1 - Disk Pack

There are a large number of ways records can be organised on disk or tape. The main methods of file organisation used for files are:

Serial
Sequential
Indexed Sequential
Random (or Direct)

a) ***Serial Organisation***

Serial files are stored in chronological order, that is as each record is received it is stored in the next available storage position. In general it is only used on a serial medium such as magnetic tape. This type of file organisation means that the records are in no particular order and therefore to retrieve a single record the whole file needs to be read from the begging to end. Serial organisation is usually the method used for creating Transaction files (unsorted), Work and Dump files.

b) ***Sequential Organisation***

Sequential files are serial files whose records are sorted and stored in an ascending or descending on a particular key field. The physical order of the records on the disk is not necessarily sequential, as most manufacturers support an organisation where certain records (inserted after the file has been set up) are held in a logical sequence but are physically placed into an overflow area. They are no longer physically contiguous with the preceding and following logical records, but they can be retrieved in sequence.

c) ***Indexed Sequential Organisation***

Indexed Sequential file organisation is logically the same as sequential organisation, but an index is built indicating the block containing the record with a given value for the Key field. This method combines the advantages of a sequential file with the possibility of direct access using the Primary Key (the primary Key is the field that is used to control the sequence of the records). These days manufacturers providing Indexed Sequential Software allow for the building of indexes using fields other than the primary Key. These additional fields on which indexes are built are called Secondary Keys.

There are three major types of indexes used:

Basic Index: This provides a location for each record (key) that exists in the system.

Implicit Index: This type of index gives a location of all possible records (keys) whether they exist or not.

Anlomodim

Limit Index: This index groups the records (keys) and only provides the location of the highest key in the group. Generally they form a hierarchical index.

Data records are blocked before being written to disk. An index may consist of the highest key in each block, (or on each track).

Index	Data	
1 A0025	A0012	
2 A0053	A0017	Block 1
3 A0075	A0025	
	A0037	
	A0038	Block 2
	A0053	
	A0064	
	A0073	Block 3
	A0075	

Figure 2 The Block Index

In the above example, data records are shown as being 3 to a block. The index, then, holds the key of the highest record in each block. (An essential element of the index, which has been omitted from the diagram for simplicity, is the physical address of the block of data records). Should we wish to access record 5, whose key is A0038, we can quickly determine from the index that the record is held m block 2, since this key is greater than the highest key in block 1, A0025, and less than the highest key in block 2, A0053. By way of the index we can go directly to the record we wish to retrieve, hence the term "direct access".

d) ***Random (or Direct)*** Hashed

A randomly organised file contains records arranged physically without regard to the sequence of the primary key. Records are loaded to disk by establishing a direct relationship between the Key of the record and its address on the file, normally by use of a formula (or algorithm) that converts the primary Key to a physical disk address. This relationship is also used for retrieval.

The use of a formula (or algorithm) is known as 'Key Transformation' and there are several techniques that can be used:

Division Taking Quotient

Provision : 25%mark

Division Taking Remainder
Truncation
Folding
Squaring
Radix Conversion

These methods are often mixed to produce a unique address (or location) for each record (key). The production of the same address for two different records is known as a synonym.

Random files show a distinct advantage where:

Hit Rate is low
Data cannot be batched or sorted
Fast response is required.

Catering for expansion

A normal tendency of master files is to expand. Records may be increased in size or may be added. Even if the total size or number of records does not increase, there will almost inevitably be changes.

Although it is usual to update files on disk by overlay there must be provision for additions and preferably some means of re-utilising storage arising from deletion.

Overflow arises from:

i) A record being assigned to a block that is already full.

ii) A record being expanded so that it can no longer be accommodated in the block.

There are a number of methods for catering for expansion:

i) Specifying less than 100% block packing density on initial load.

ii) Specifying less than 100% cylinder packing density.

iii) Specifying extension blocks (usually at the end of the file).

The first method is only effective and efficient if the expansion is regular. Where localised expansion occurs to any extent, even in one block, this system will fail.

The other two method have the result of allowing space for first and second level overflow, respectively. Extension blocks are used when all other overflow facilities have been exhausted.

Ideally, first level overflow is situated on the same cylinder as the overflowed block hence there is no penalty incurred in terms of head movement. Second level overflow normally consists of one or more blocks at the end of the file. If a significant number of accesses to second level overflow are made, run-time will increase considerably. There is the a need to reorganise the file to bring record back from overflow.

Each file organisation can be accessed or processed in different ways, often combing the advantages of one organisation with the advantages of another.

Summary of file organisation and access methods:

ACCESS METHODS			
FILE ORGANISATION	**Serial**	**Sequential**	**Random**
Serial	X		
Sequential		X	
Indexed Sequential		X	X
Random	X		X

The transfer time of data from a direct storage device such as a disk drive can be calculated, however the formulae needed for the different types of file organisations differ. An example of these formula are shown on the following pages.

SEQUENTIAL FILE TRANSFER TIMINGS:

This time taken to transfer ALL records will equal

The transfer time for the file
+
Total SEEK time for the file
+
Total Latency for the file

In order to calculate the above the following are required:

1. $$\frac{\text{Size of file in characters}}{\text{Transfer Rate}} = \textbf{Transfer time for file}$$

2. $$\frac{\text{Bytes per Track}}{\text{Characters per record}} = \textbf{Number of records per track}$$

Bytes

3. $$\frac{\text{Number of Records}}{\text{Number of records per track}} = \textbf{Number tracks required}$$

4. $$\frac{\text{Number of tracks required}}{\text{Number of surfaces}} = \textbf{Number of cylinders required}$$

5. $$\text{Number of cylinders} \times \text{Minimum SEEK Time} = \textbf{Total seek time for file}$$

6. $$\text{Number of cylinders} \times \text{Average Latency} = \textbf{Total Latency for file}$$

For Ransom or Direct file organisations both the SEEK time and Latency between each record transferred needs to be included in the calculation:

Dosya

1. $$\frac{\text{Size of file in Characters}}{\text{Transfer Rate}}$$ **= Transfer time for file**

→ random olduğu için

2. Number of records in file X Average SEEK time **= Total SEEK time for file**

3. Number of Records in file X Average Latency **= Total Latency for file**

The sum of these three calculations will give the transfer time for All records.

In order to calculate the Latency (Rotational Delay) the time for one rotation of the disk needs to be expressed in milliseconds.

$$\frac{\text{RPM}}{60}$$ **= R.P.Seconds**

$$\frac{1}{\text{R.P.Seconds}}$$ **= Latency express as M. seconds**

Access time **= SEEK + Average Latency.**

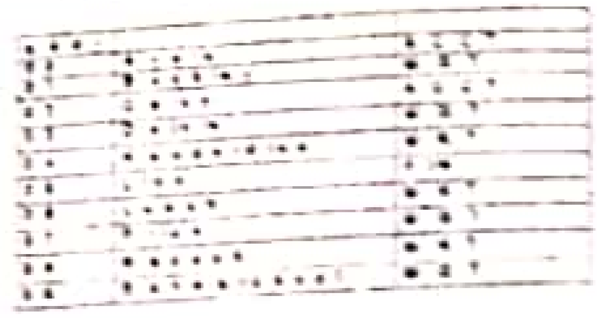
File Organization

Physical arrangement of the records of a file on secondary storage devices

- Sequential
- Linked List
- Indexed
- Hashed

Sequential File

Sequential file sorted in alphabetical order
Sequential files are usually sorted in ID sequence order to facilitate batch processing.

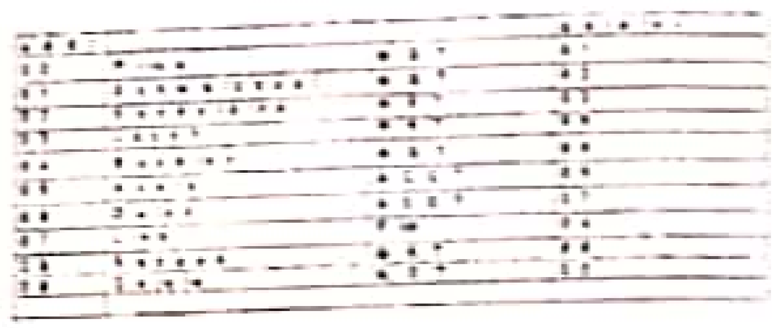


Sequential File Processing

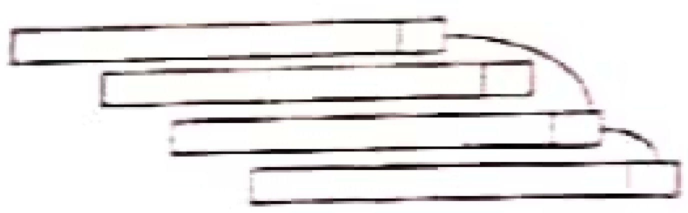
Sequential files must be recopied from the point of any insertion or deletion to the end of the file. They are commonly used in batch processing where a new master file will be generated each time the file is updated.

Linked List

Linked list to sort data alphabetically within department.
An external reference must point to the start record (06).



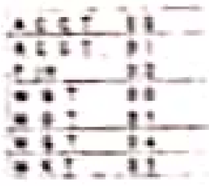
Linked List File Processing



The next record in a linked list is found at the address stored in the record. Records are added at any location in the DASD and pointers adjusted to include them. Deletions are not erased, but pointers changed to omit the deleted record.

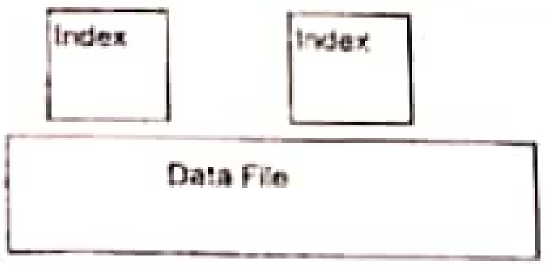
Indexed File (sequential index)

Index to access data by department abbreviation.



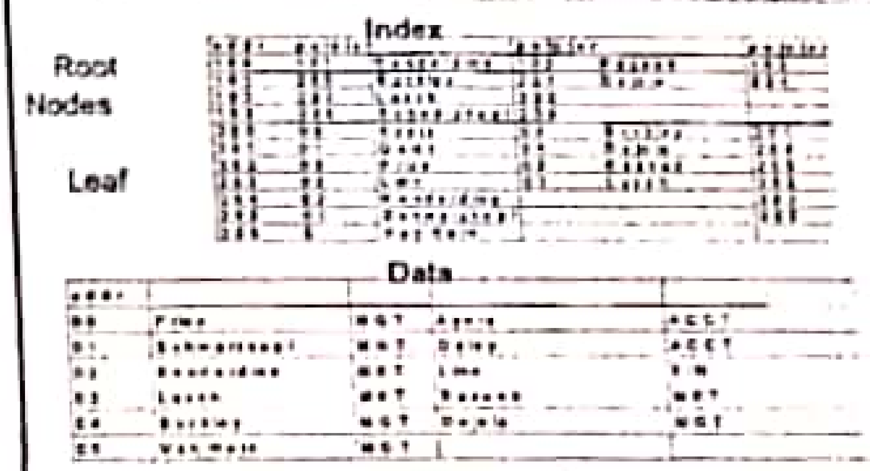
Scanned with CamScanner

Indexed File Processing

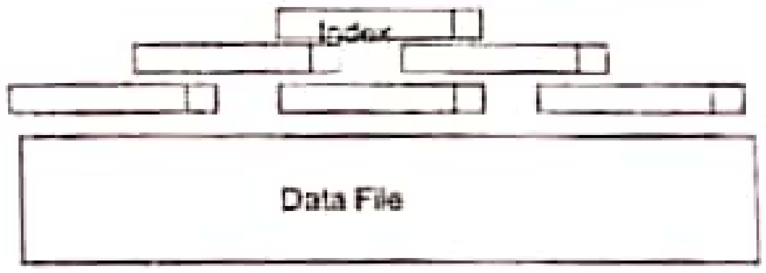


When a record is inserted or deleted in a file the data can be added at any location in the data file. Each index must also be updated to reflect the change. For a simple sequential index this may mean rewriting the index for each insertion.

Segmented Index



Indexed File Processing (segmented index)



Data can be inserted or deleted at any location in the data file. The index(es) must be updated for each change, but only the affected segments need to be rewritten.

Physical Design

- ⌘Volume and Usage analysis
- ⌘Distribution Strategy
- ⌘File Organizations
- ⌘Indexes and Access Methods
- ⌘Integrity Constraints

Source: Bilkent University

11.3 Files and Streams

- Read/Write functions in standard library
  - `fgetc`
    - Reads one character from a file
    - Takes a `FILE` pointer as an argument
    - `fgetc( stdin )` equivalent to `getchar()`
  - `fputc`
    - Writes one character to a file
    - Takes a `FILE` pointer and a character to write as an argument
    - `fputc( 'a', stdout )` equivalent to `putchar( 'a' )`
  - `fgets`
    - Reads a line from a file
  - `fputs`
    - Writes a line to a file
  - `fscanf` / `fprintf`
    - File processing equivalents of `scanf` and `printf`

11.4 Creating a Sequential-Access File

- C imposes no file structure
  - No notion of records in a file
  - Programmer must provide file structure
- Creating a File
  - `FILE *cfPtr;`
  - Or `FILE* cfPtr;`
    - Creates a `FILE` pointer called `cfPtr`
  - `cfPtr = fopen("clients.dat", "w");`
    - Function `fopen` returns a `FILE` pointer to file specified
    - Takes two arguments – file to open and file open mode
    - If open fails, `NULL` returned

Personally, I like cfPtrW. W is a reminder for "w".

11.4 Creating a Sequential-Access File

- `fprintf`
  - Used to print to a file
  - Like `printf`, except first argument is a `FILE` pointer (pointer to the file you want to print in)
- `feof( FILE pointer )`
  - Returns true if end-of-file indicator (no more data to process) is set for the specified file
- `fclose( FILE pointer )`
  - Closes specified file
  - Performed automatically when program ends
  - Good practice to close files explicitly
- Details
  - Programs may process no files, one file, or many files
  - Each file must have a unique name and should have its own pointer


```
/* Fig. 11.3: fig11_03.c
   Create a sequential file */
#include <stdio.h>

int main( void )
{
   int account;     /* account number */
   char name[ 30 ]; /* account name */
   double balance;  /* account balance */

   FILE *cfPtr;     /* cfPtr = clients.dat file pointer */

   /* fopen opens file. Exit program if unable to create file  */
   if ( ( cfPtr = fopen( "clients.dat", "w" ) ) == NULL ) {
      printf( "File could not be opened\n" );
   } /* end if */
   else {
      printf( "Enter the account, name, and balance.\n" );
      printf( "Enter EOF to end input.\n" );
      printf( "? " );
      scanf( "%d%s%lf", &account, name, &balance );
```

FILE pointer definition creates new file pointer

fopen function opens a file; w argument means the file is opened for writing

11.7 Creating a Random-Access File

- Data in random access files
  - Unformatted (stored as "raw bytes")
    - All data of the same type (`int`s, for example) uses the same amount of memory
    - All records of the same type have a fixed length
    - Data not human readable.
- What is human unreadable?
  - Use nodepad to open a pdf file, you will know.

11.7 Creating a Random-Access File

- Unformatted I/O functions
  - `fwrite`
    - Transfer bytes from a location in memory to a file
  - `fread`
    - Transfer bytes from a file to a location in memory
  - Example:

    `fwrite( &number, sizeof( int ), 1, myPtr );`
    - `&number` – Location to transfer bytes from
    - `sizeof( int )` – Number of bytes to transfer
    - `1` – For arrays, number of elements to transfer

      In this case, "one element" of an array is being transferred
    - `myPtr` – File to transfer to or from

11.7 Creating a Random-Access File

- Writing `struct`s

  `fwrite( &myObject, sizeof (struct myStruct), 1, myPtr );`
  - `sizeof` – returns size in bytes of object in parentheses
- To write several array elements
  - Pointer to array as first argument
  - Number of elements to write as third argument

Outline

```c
/* Fig. 11.11: fig11_11.c
   Creating a random-access file sequentially */
#include <stdio.h>

/* clientData structure definition */
struct clientData {
   int acctNum;           /* account number */
   char lastName[ 15 ];   /* account last name */
   char firstName[ 10 ];  /* account first name */
   double balance;        /* account balance */
}; /* end structure clientData */

int main( void )
{
   int i; /* counter used to count from 1-100 */

   /* create clientData with default information */
   clientData blankClient = { 0, "", "", 0.0 };
```

fig11_11.c

(1 of 2)

11.3 Files and Streams

- C views each file as a sequence of bytes
  - File ends with the *end-of-file marker*
    - Or, file ends at a specified byte

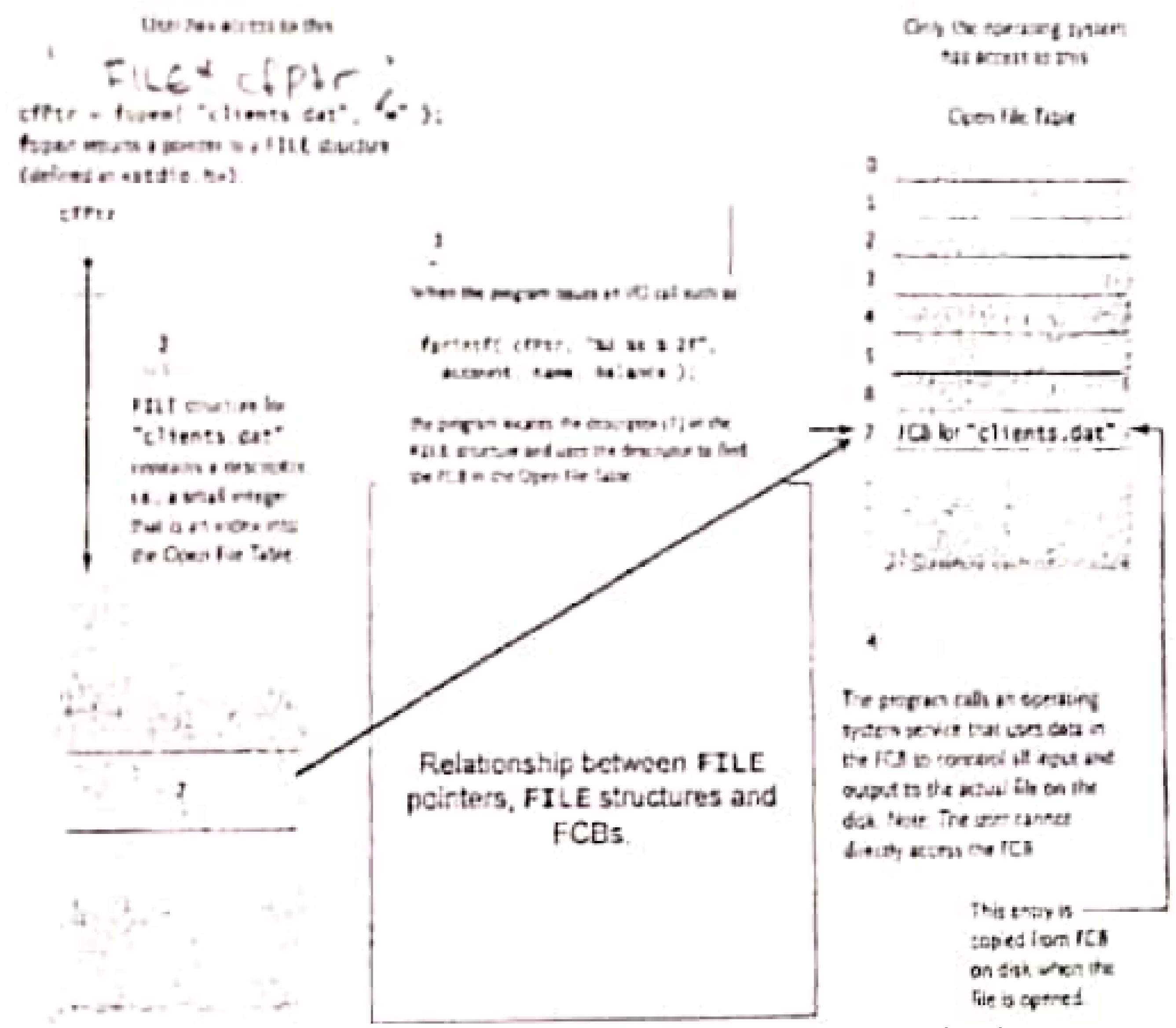
Operating system	Key combination
Linux/Mac OS X/UNIX	<Ctrl> d
Windows	<Ctrl> z

11.3 Files and Streams

- Stream created when a file is opened
  - Provide communication channel between files and programs
  - Opening a file returns a pointer to a FILE structure
    - Example file pointers:
    - `stdin` - standard input (keyboard) This is a file
    - `stdout` - standard output (screen) This is a file
    - `stderr` - standard error (screen) This is a file

11.3 Files and Streams

- FILE structure
  - File descriptor
    - Index into operating system array called the open file table
- File Control Block (FCB)
  - Found in every array element, system uses it to administer the file



Relationship between FILE pointers, FILE structures and FCBs.

```
Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 1

Accounts with zero balances:
300         White              0.00

? 2

Accounts with credit balances:
400         Stone            -42.16

? 3

Accounts with debit balances:
100         Jones             24.98
200         Doe              345.67
500         Rich             224.62

? 4
End of run.
```


11.5 Reading Data from a Sequential-Access File

- Sequential access file
  - Cannot be modified without the risk of destroying other data
  - Fields can vary in size
    - Different representation in files and screen than internal representation
    - 1, 34, -890 are all ints, but have different sizes on disk
  - Note, int 1, char '1', and string "1" have no difference on disk.

Anlamadım

11.6 Random-Access Files

- Random access files
  - Access individual records without searching through other records
  - Instant access to records in a file
  - Data can be inserted without destroying other data
  - Data previously stored can be updated or deleted without overwriting
- *Implemented using fixed length records*
  - *Sequential files do not have fixed length records*

Anlamadım

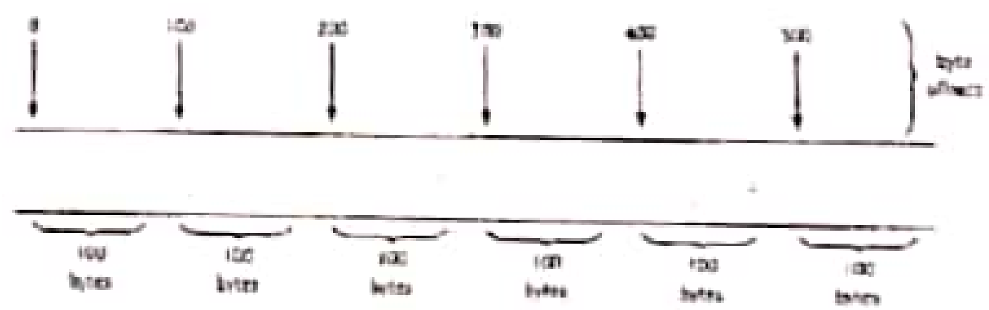


Fig. 11.10 | C's view of a random-access file