Operating System
Module-6 Notes

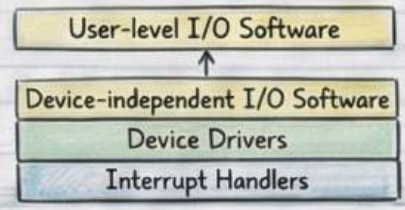
by pyqfort.com

Contents Covered:

- I/O Devices

- Device Controllers

- DMA Data Transfer Process
- Principles of I/O Software

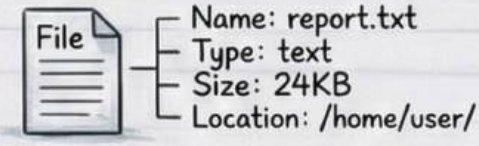


- I/O Software Layers

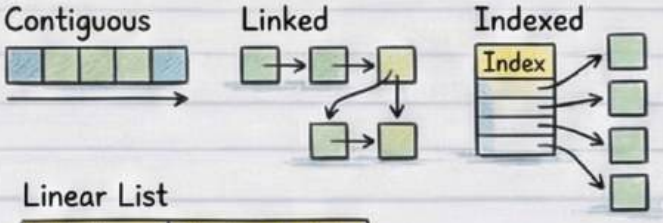
- Secondary-Storage Structure

- Disc Scheduling Algorithm

- Concept of File



- File Allocation Methods



- Directory Implementation

Linear List

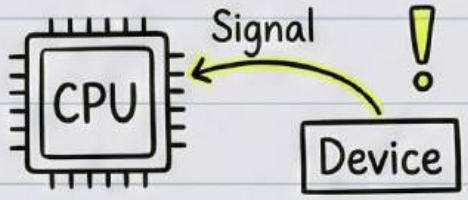
File Name	Pointer to Data
file1	ptr1
file2	ptr2
file3	ptr3

Module 6: I/O Hardware

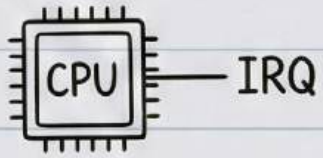
I/O Devices

Everything an OS does is **interrupt driven**.

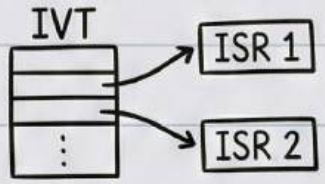
An **Interrupt** is a signal to the processor indicating an event needs attention.



The processor has an **interrupt-request (IRQ) line** to sense interrupts after each instruction.

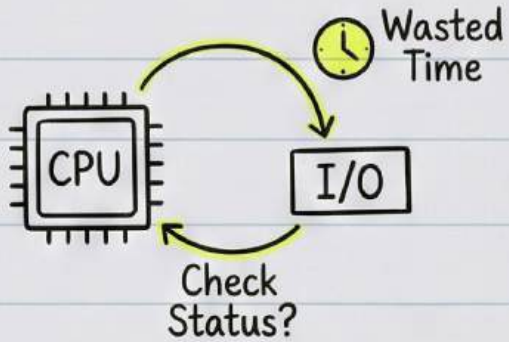


There is a program called **interrupt service routine (ISR)** for each interrupt. Addresses of all ISRs are in an **interrupt vector table (IVT)**.



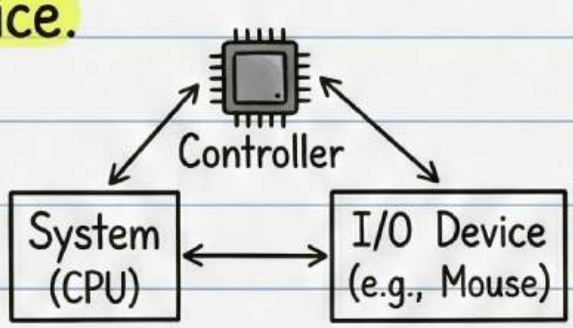
A **hardware interrupt** is from an external device (e.g., keyboard). **Software interrupts** are caused by exceptional conditions or special instructions.

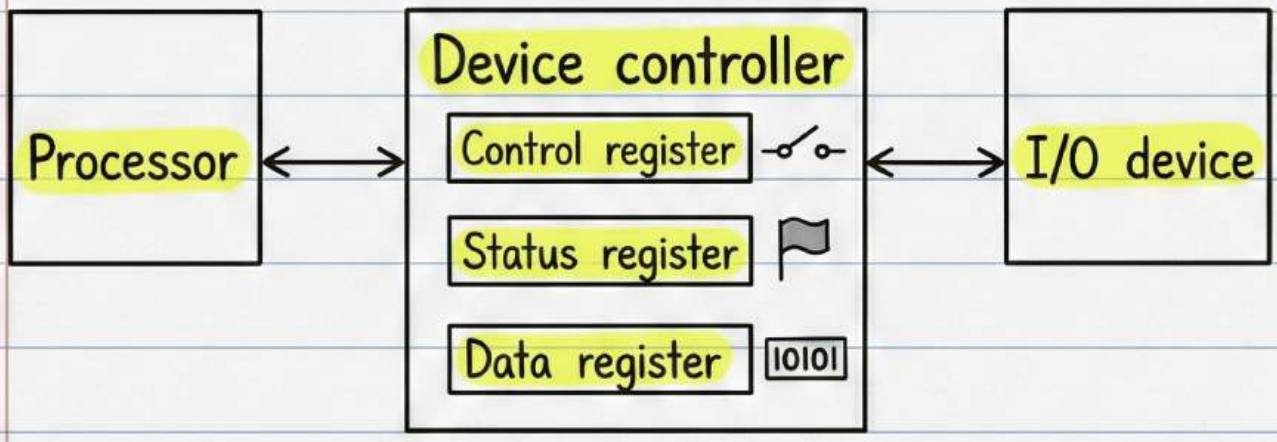
- In **programmed I/O technique**, processor time is **wasted** as it **continually interrogates** the I/O status.



Device Controllers

- A Device controller is an electronic device in the form of a chip or circuit that controls the communication between the system and the I/O device.





Direct Memory Access (DMA) Operation

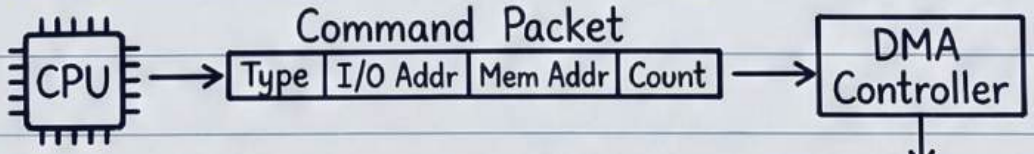
Why DMA?

- Used for transferring 'blocks of characters' without 'processor intervention'.

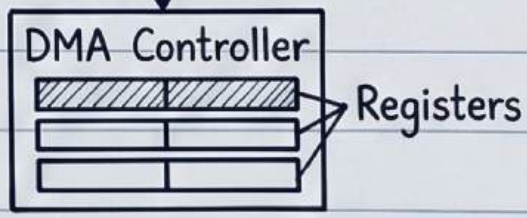
- Interrupt-driven I/O is 'not efficient' for large data due to a 'number of interrupts' (e.g., every 10ms for a character).

How it Works

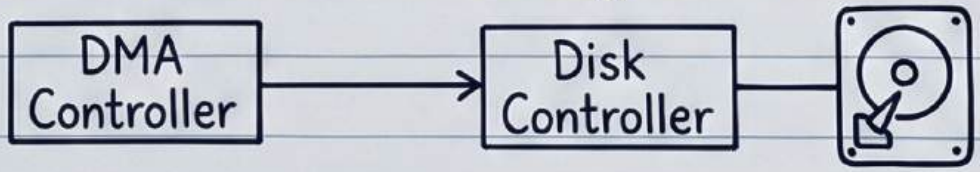
- Processor gives 'responsibility to do I/O' for a block to the 'DMA controller'.
- Processor sends to DMA controller:
  1. 'Type of request' (read/write).
  2. 'Address of the I/O device'.
  3. 'Start address of the memory' & 'total number of words'.



- DMA controller copies this to 'its registers'.

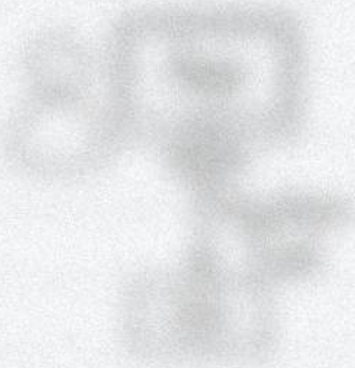


- Example (Disk Read): CPU sends info to DMA, which stores it in 'disk controller registers'.



Principles of Life: Cell Theory

All cells are organized at the cell level, but they may be specific tissues and a cell performs specific to the organ, does and tasks. The organ, consists the cells of the tissues that the can and the cells

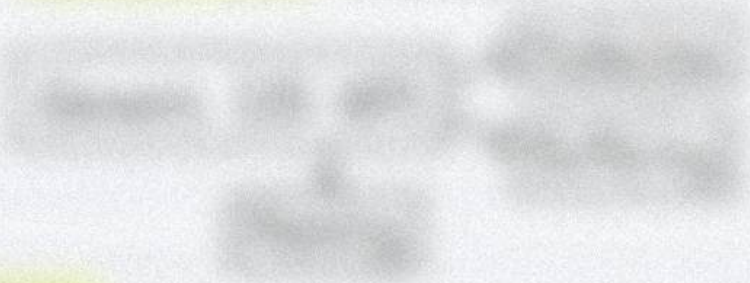


1. The Cell Theory

The provides the structure to the can It contains within the and other cell

2. Tissue Organization

The organ performs life functions that are in all tissue, such as tissue and providing a cell



3. Tissue Types

The the composition of specific tissue the organ tissue, tissue, tissue, has its own

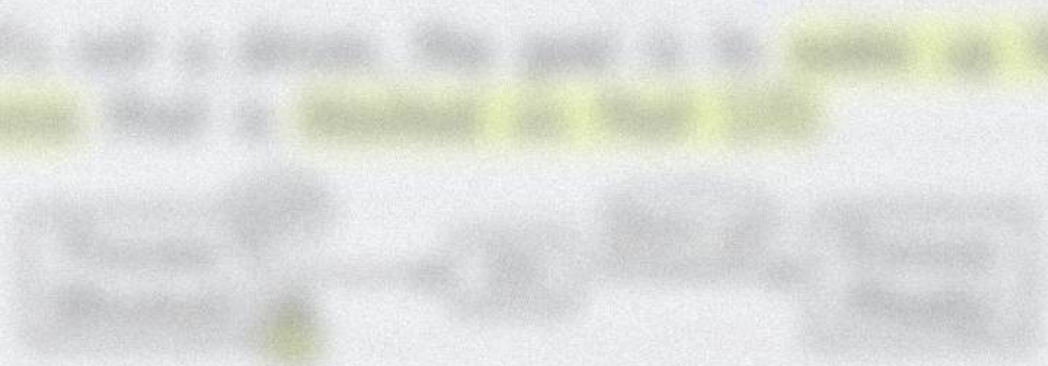
The tissue and the range and management of life

[illegible]

[illegible]

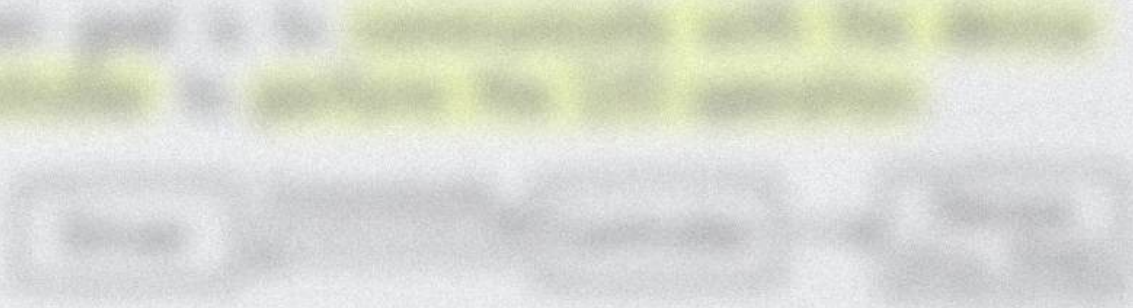
- [illegible]

- [illegible]

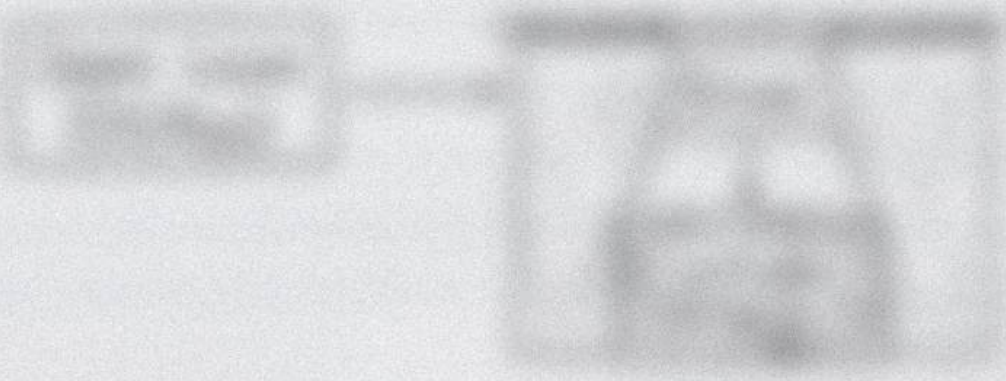


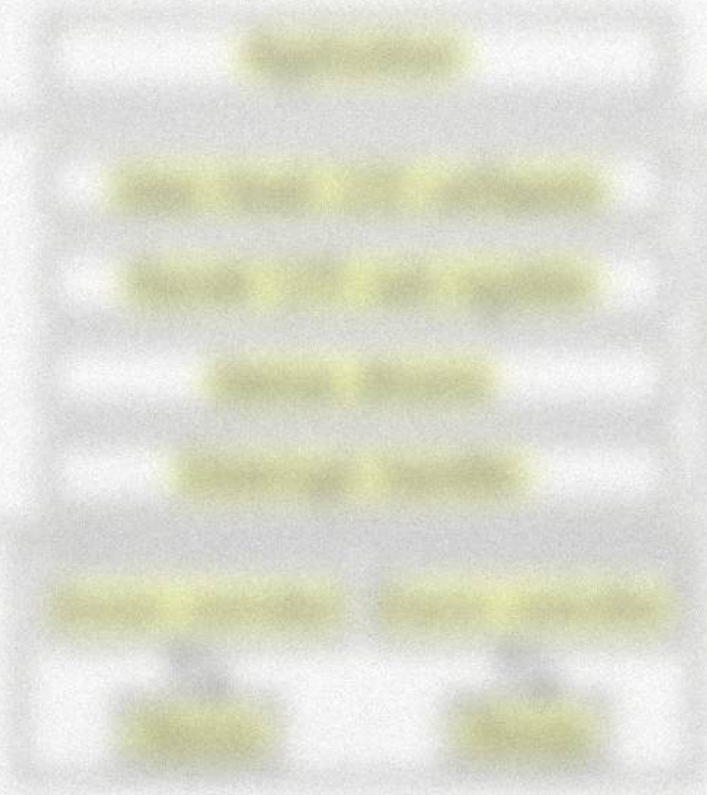
[illegible]

- [illegible]



- [illegible]





Device Drivers

- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]

Interrupt Handling

- [illegible] interrupt handling mechanism

- It is the [illegible] in the [illegible] memory

- After [illegible] the [illegible] generates an [illegible] to the device driver.

- The [illegible] [illegible] is executed to handle the specific interrupt

- The [illegible] [illegible] [illegible] is restored, [illegible] resumes.

Diet Variability Hypothesis

Hypothesis	Criteria	Predictions
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

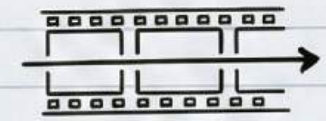
Concept of File & Access Methods

Concept of File

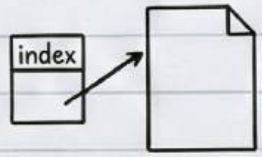
- A file is a sequence of 'blocks'. This is called 'record blocking'.
- Larger 'block size' = more records per block.

Access Methods

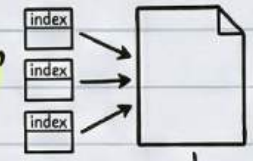
1. 'Sequential file access': Information is accessed in the 'stored order'.



2. 'Indexed sequential file access': Uses 'random access approach' with an 'index' (stores 'key field' & 'pointer' to main file).



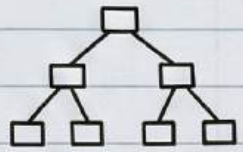
3. 'Indexed file access': Prepares 'multiple indexes' to search by attributes.



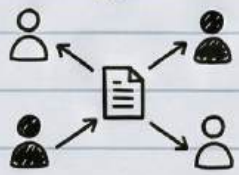
4. 'Direct file access': For 'random structure' of secondary storage like a disk.

Directory Structure

Logical structures: 'single level', 'two level', 'tree structure'.



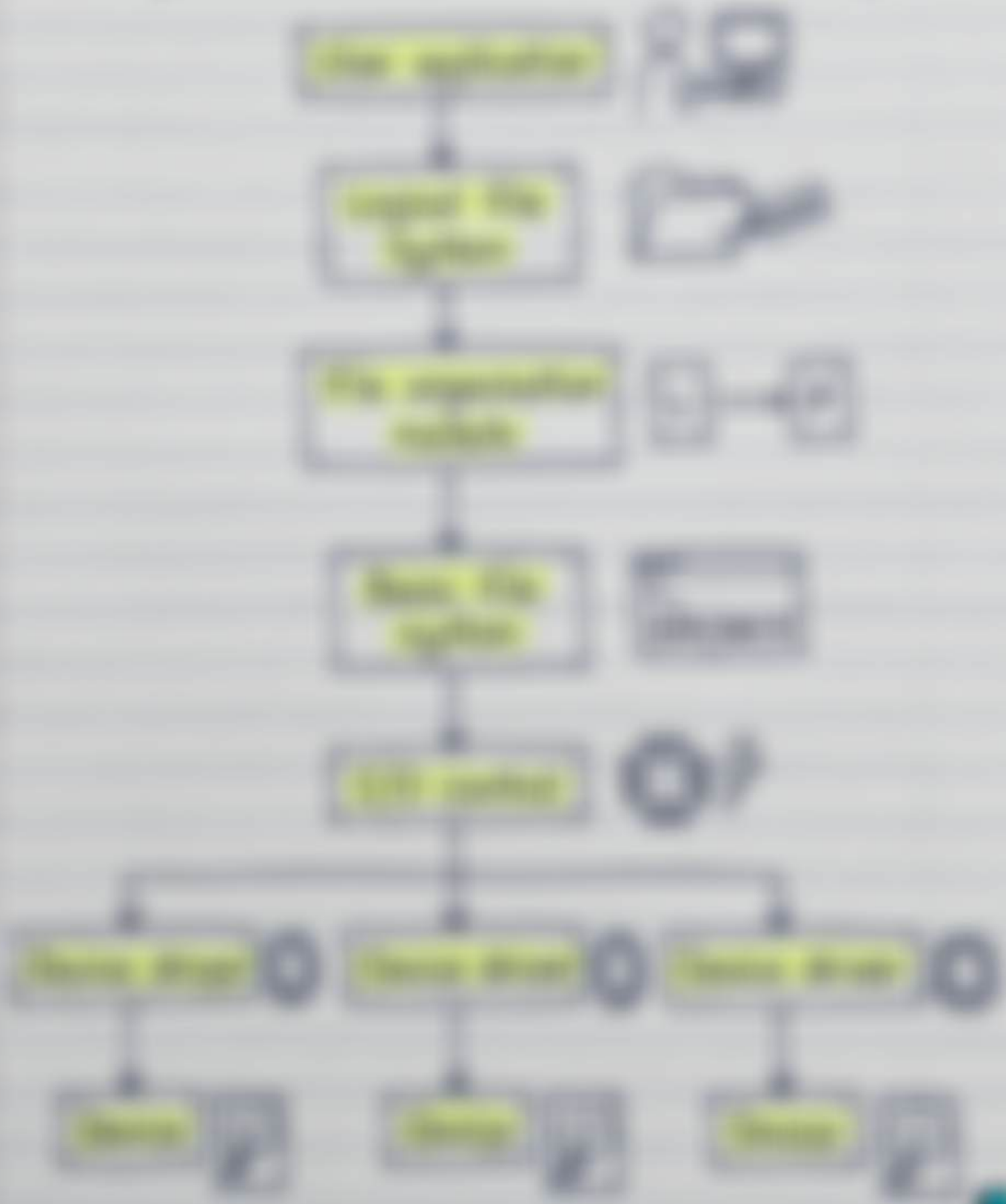
Paths: 'Absolute path' starts from 'root directory'.
'Relative path' starts from 'current/working directory'.
'Tree structure' does not allow sharing.
'Acyclic graph' allows sharing.



File System Structure

The file system is structured in several layers to accomplish operations.

Each layer uses the operation of its lower layer.



Logical File System & File Organization Module

Logical File System

- Concerned with the **logical structure** of the file.

- Manages **all information about file attributes**

- Maintains file structure via a **file control block** (FCB)

- Manages **directory structure** & provides **file location** to the **next module**.

File Organization Module

- Translates **logical blocks** to **physical blocks**

- Needs **location of the file** & its **allocation type** for translation.

- Disk space is allocated via **file allocation methods**

- Must know which **blocks are free**

- The **free space manager** tracks **unallocated blocks** & provides this info.

Python Input Output and Import statements

File Allocation Methods

Files are a logical concept that must be mapped to physical blocks on secondary storage (like a disk).

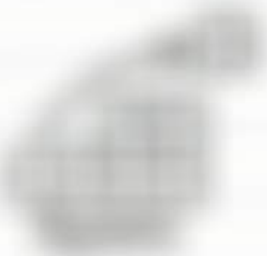
The file system is responsible for allocating the blocks to the file.

Disk vs. System Allocation

- **Disk allocation** allocates the minimum size if a file **is allowed** when it is created.

- **System allocation** allocates the blocks as per requirement.

- **System allocation is better** as there is **some wastage** in disk allocation.



Goals

- Ensure **quick access** to the contents of the file.
- Ensure **storage space is utilized efficiently**.

- There are **three methods** of file allocation.

Contiguous File Allocation

In this method, each file occupies a **contiguous set of blocks** on the disk. This means the blocks are located **next to each other**.

For each file, the directory entry stores the **start address** and the **length** (number of blocks).

File	Start	Length
File A	10	6

Advantages:

* Supports **sequential access.**
* **Excellent performance** due to minimal head movement.

Disadvantages:

* Suffering from **external fragmentation.**
* **Difficult to find contiguous space** for new files.
* File size must be **known in advance.**

Linked File Allocation

In this method, each file is a **linked list of disk blocks**. The blocks may be **scattered anywhere on the disk**.

The directory contains a pointer to the **first** and **last** blocks of the file.

Each block contains:

- The **data**.
- A **pointer to the next block**.

Advantages:

- **No external fragmentation**.
- **Easy to grow** a file.

Disadvantages:

- **Slow for random access** (must traverse the list)

- **Space overhead** for the pointers.
- **Reliability issue**: If a pointer is lost or damaged, the rest of the file is lost.

Indexed File Allocation

In this method, each file is allocated a special block called an **index block**. This block contains all the **block addresses** (pointers) of the file.

The directory contains the **address of the index block**.

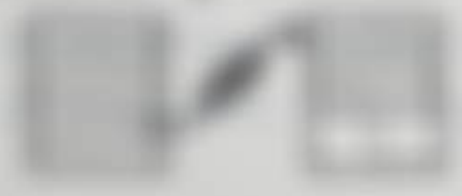
The i-th entry in the index block points to the i-th block of the file, allowing **direct access**.

Problem & Solution:

- Deciding the **size of the index block** is a problem.
- If the index block is too large, space is **wasted** for small files.

- If it's too small, it can't hold all pointers for a large file.

- Solution: An index block can be **linked** to another index block to support larger files.



Directory Implementation

A **directory** is used to **name and organize multiple files**.

- It **manages** a **list of files**, connecting each **name** with its **associated file**.

- The directory stores the **file name** and a **pointer** to the FCB (File Control Block).

- The **FCB** is the **handler**, storing **meta-information** and **pointers** to the **actual contents**.

- To find a file, the system searches for its **name** as a **key**, using the **FCB as a reference** to access the file's data.

Linear List Directory Implementation

The **simplest technique** to implement a directory is to have a **linear list**

However, an **unsorted linear list** containing directory entries may become **inefficient** when there are **large numbers of file names** as it needs to have a **scan of the entire directory**, thereby consuming a **significant lookup time**

In spite of the drawbacks of linear list implementation, it is used widely due to its **simplicity**

Requirements for Better Data Structures

- It should perform **efficient lookups** on the list.
- It should have **reasonable cost for insertions or deletions** in the list.

Therefore, the directory implementation can be better done with **B-tree**, **hash tables**, or **radix sorting** methods.

Full content is
available at
'topmate.io/saumy1905'
or you can navigate
to the specific section
of the subject on
'pyqfort.com'
to access it