How to keep data on a platform other than the main memory 🡪 that’s the question.

**FILE SYSTEMS**

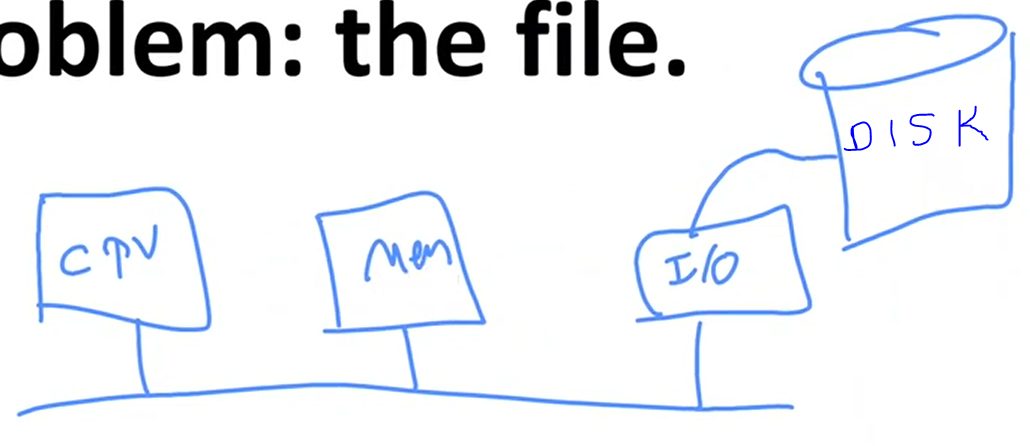
Essential requirements for long-term information storage:

1. It must be possible to store a very large amount of information.
2. Information must survive termination of process using it.
3. Multiple processes must be able to access information concurrently.

**A new abstraction solves the problem: the file.**

We need permanent storage. When I power off, all the data in memory is gone.

I need OS help to organize permanent data.



Disk is secondary storage. It doesn’t have to be disk, we use SSDs nowadays.

OS abstracted away the concept of the processor to create the abstraction of a process

It abstracted away the concept of physical memory to offer processes (virtual) address spaces

**A new abstraction solves the problem: the file.**

Together, the abstractions of processes (and threads), address spaces, and files are the most important concepts relating to OSs.

If you really understand these 3 concepts from beginning to end, you are well on your way to becoming an OSs expert.

Think of a disk as a linear sequence of fixed-size blocks and supporting 2 operations:

1. Read block k
2. Write block k

File is kinda linear address space, maybe a segment. It is divided into blocks. Each block could be 1-2 KB. It is like virtual memory. I can access it by block numbers.

You can have as many files as you like.

We are not talking about the speed anymore.

Environment is different, our algorithms will be different.

Our concerns are also data integrity and protection.

We have as many files as we like for a process as long as we have enough space. So there has to be some way of naming them.

Questions that quickly arise:

1. How do you find information?
2. How do you keep one user from reading another user’s data?
3. How do you know which blocks are free?

Design Decisions

* File names
  + Millions of files. Giving them numbers as segments is not meaningful.
  + I need to give them meaningful name so that ı can remember as end user.
* Hierarchical or non-hierarchical: structure
  + structure where I keep my files
  + You cannot keep all your files in single place, you have to structure them.
* Types of access control: Read-only or read-write
* Versioning
  + If I need previous version of the file after I update it, then version is important.
  + You can say save each of versions with different file name but why OS doesn’t help?
* Fault recovery
  + There is no software free of bugs. After the crash, you have to have a recovery method.
* File operations
  + Opening a file, closing a file, etc.
* Many more…

**Files and File Names**

A file is an abstraction mechanism.

It provides a way to store information on the disk and read it back later.

Depending on OS, each file is given a name.

File names:

* Case sensitivity
* Extensions – permitted, required, uninterpreted, length
  + Required and interpreted for windows
  + Permitted and uninterpreted for linux
* Character set
* Uniform or non-uniform file-naming scheme

Table

Description automatically generated

*Figure 4-1. Some typical file extensions.*

Extension names depend on OS.

For Linux it doesn’t matter what kind of extension you have, it has its own interpretation. Although it doesn’t require it, we as end users use them a lot.

File Systems

File systems: Part of the OS (for most OS, part of the kernel) dealing with files is known as the file system and is the subject of this chapter. File systems defines how we interact with the secondary storage.

* FAT-16: MSDoS
  + Very old.
  + Very popular even today bc it is free and simple.
  + Many embedded small systems use FAT-16 file system a lot.
  + You cannot have files larger than a few MB.
  + 4 GB is the largest disk space that you can support.
  + No recovery procedure.
  + Protection is not there.
* FAT-32: Windows 98-
  + Similar to 16 but can have larger files.
* NTFS: Windows NT
  + We use today.
  + Fast and reliable.
* Windows 8 ReFS (or Resilient File System)
  + More recent than NTFS.
  + Robust against crashes.
* exFAT for flash drives and large file systems
* ext4
  + Advanced linux file system.
  + Journaling file system 🡪 remembers what it did last time (added a new file, update a file, etc.)

**File Structures – How are the files structured?**

Old systems: Files were sequences of records, initially images of punch cards or printer lines

Unix and Windows: Randomly-addressable sequences of bytes

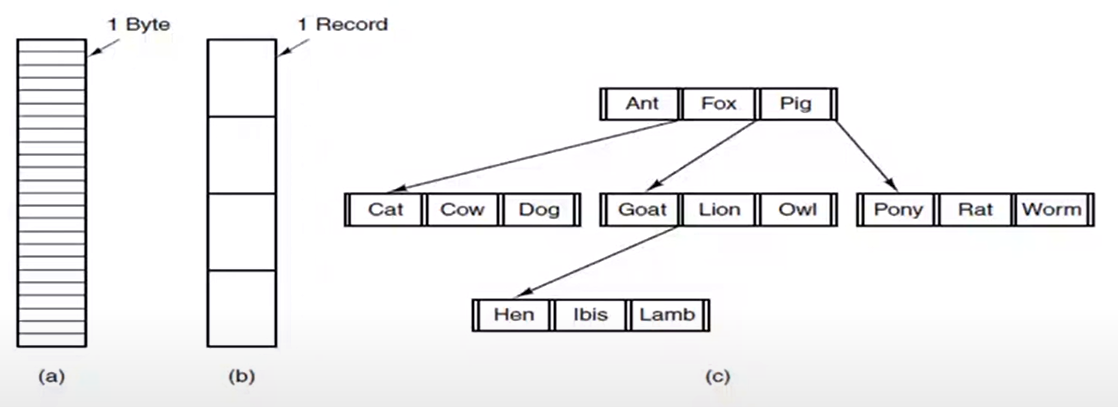
* I can access any byte that I like.
* Uninterpreted, sequences of bytes like the virtual memory.

Page-oriented file systems: Sequence of page-sized blocks

Having the OS regard files as nothing more than byte sequences provides the max amount of flexibility

Punchcards are actually file systems. Each punchcard can hold 80 columns of text.

If you have deck of cards, 100 cards, that means 8000 bytes I am storing physically somewhere.



*Figure 4-2. Three kinds of files.*

*(a) Byte sequence.*

*(b) Record sequence.*

*(c) Tree (B Tree)*

(a) is the one we like the most.

(b) is like each record is punchcard.

File Types – Depends on OS

Regular files are the ones that contain user information.

Directories are system files for maintaining the structure of the file system.

* Special file that contains other files in it.
* Structure of directory differs from file system to file system.

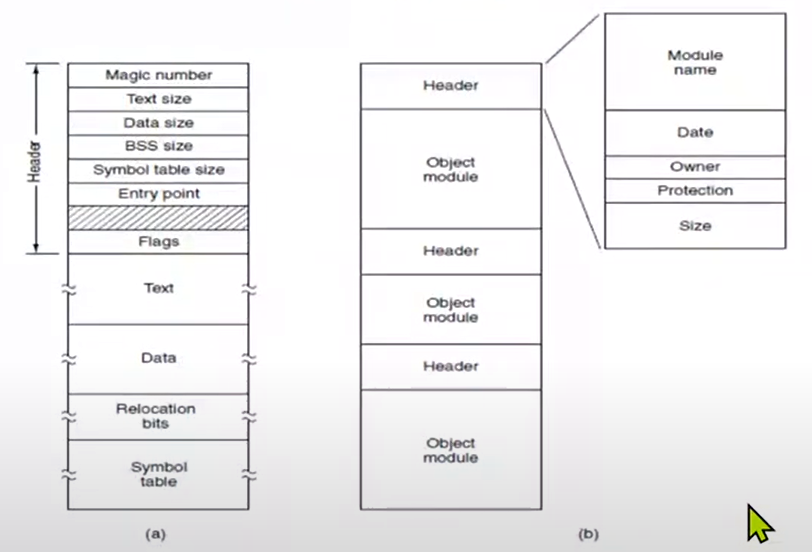
Character special files are related to input/output and used to model serial I/O devices, such as terminals, printers, and networks.

Block special files are used to model disks.

In this chapter we will be primarily interested in regular files and directories.

We use abstraction of files for data stored not only on secondary storage, maybe sometimes on memory too. Remember mmap files that OS provides. It looks like it is a file but actually I keep it on my memory. So I use this mechanism to communicate between 2 processes. Those are files too but we ignore them in this chapter.

With the file systems, most of the time we deal with how we store them physically on our secondary storage units. Disks are still widely used in server environments and OS has to deal with them. So we will talk about the mechanical structure of the hard disks and we are gonna discuss how the mechanical structure of the hard disk will affect our decisions on the file system design. After the file systems, we will talk about I/O.



*Figure 4-3. (a) An executable file. (b) An archive.*