These two chapters deal with file system and its implementation, which resides on the secondary storage (commonly magnetic disk) and is designed to hold a large amount of data permanently.

File

- A file is an abstract data type defined and implemented by the operating system. It is a sequence of *logical records*. There are different **types** of files, and the type determines the **file structure**. For example a *text* file is a sequence of characters and a *source* file is a sequence of subroutines.
- A file commonly has following attributes: name, identifier, type, location, size, protection, time, date, and user identification. Operations on files include: create, write, read, reposition, delete, append, truncate, open(), close()
- The OS maintains a **system-wide open-file table** containing information about all open files in the system. Each entry in the open-file table corresponds to one open file, which might be shared/used by multiple processes. It contains information such as: file pointer, file-open count, file block location on the disk, access rights and protection. The OS also maintains s a **per-process open-file table**, and each entry points to a corresponding entry in the system-wide open-file table.
- A file can be accessed, **sequentially, directly** (randomly) or by **index**.
- **File-system mounting**: a file system must be mounted before it can be accessed, similar to the way that a file must be opened before it can be used. The OS is given the name of the device and the **mount point** – the location within the file structure where the file system is to be attached.

- **Directory Structure**
  - **Storage structure:** A disk is divided into *partitions*. Each partition can be thought of as a *virtual disk*; it has a directory that records information such as name, location, size and type for all files on the virtual disk.
  - **Directory:** this is used to organize a file system with potentially large number of files. Logically, this translates a file name into a directory entry.
  - **Directory operations:** search for a file, create a file, delete a file, list a directory, rename a file, and traverse the entire file system.
  - **Directory structures:** single-level, two-level, tree-structured, acyclic-graph, and general graph directories. There are several factors that need to be considered in the design of a directory structure, naming convenience, efficiency of file search, grouping capability, and whether files or/and sub-directories can be shared.

- **File System Design**
  - The user view: defining a file and attributes, operations allowed on the file, and directory structure for organizing the files.
  - Algorithm and data structure to map the logical file system onto the physical secondary-storage device.
  - A file system can have different structures and algorithms.

- **Layered File System**
  - File systems are often implemented in a layered structure. The lower levels deal with the physical properties of storage devices, while the upper levels deal with symbolic file names and logical properties of files. Intermediate levels map the logical file concepts into physical device properties.
  - [Device and I/O control]: transfer block between memory and disk. This deals with interrupt, device drivers (hardware specific instructions)
  - [The basic file system]: issue generic commands to the device driver to read/write physical blocks on the disk
  - [The file-organization module]: logical block to physical block translation and also manage free blocks on a disk
  - [The logical file system]: manage **metadata** (all information about files excluding the content of files), including file-system structure and directory structure. A file control block (FCB) contains information about the file.

- **File System Implementation**
  - Several **on-disk** and **in-memory** structures used to implement a file system.
  - On-disk structure**,** it may contain information about how to boot an operating system stored there (boot control block), the total number of blocks, number and location of free blocks (volume control block), directory structure, and individual file's FCB.
  - In-memory information used for both file-system management and performance improvement. The data are loaded at mount time, updated during file-system operations, and discarded at dismount time. These typically includes a **mount table** containing information about each mounted volume, a directory-structure cache holding the directory information of recently accessed directories, the system-wide open-file table contains a copy of the FCB of each open file, the per-process open-file table contains a pointer to the appropriate entry in the system-wide open-file tale.

- **Open()**
  - Each file must be opened before it can be used. The open() instruction creates an entry in per-process open-file table, which points to the corresponding entry in the system-wide open-file table. This enables any subsequent operations on the file (such as read and write) to easily access the file control block and data blocks of the file.

- **Directory Implementation**
  - **Linear list:** a linear list of file names with pointers to the data blocks. This is simple to program but time-consuming to execute (searching a file requires linear search, caching the most recently used directory information can be useful as directory information is frequently used)
  - **Hash Table:** A linear list stored the directory entries, but a hash is used which takes a value computed from the file name and return a pointer to the

file name in the linear list. This greatly reduces the directory search time, also makes it easier for insertion and deletion. Collisions are possible, in which two or more file names are hashed to the same location. We can further use a linked list (with the same hashed values for different file names) instead of a single value to resolve the collision.

- **Disk Space Allocation Methods (allocate disk blocks to files)**
  - **Objective**: effective utilization of the disk space and quick file access.
  - **Contiguous allocation**: each file occupies a set of contiguous blocks of the disk. This is simple, and supports both sequential and random access. It may require the declaration of the file size at the file creation. It suffers from problems similar to contiguous memory allocation as a dynamic storage allocation problem (first fit or best fit), external fragmentation, difficult to grow the file size.
  - **Linked allocation**: stores file by blocks with each block having pointer pointing to the next block in the file. The directory contains pointers to the first and last blocks. This solves size-declaration in contiguous allocation. But it only allows sequential access to the file and pointers in each block needs extra space.
  - A variation of the linked allocation is the use of **File Allocation Table** (FAT) (MS-DOS) stored at the beginning of a volume, in which one entry corresponds to a disk block and is indexed by block number. The directory entry contains the block number of the first block of the file. The FAT (or the table) entry indexed by that block number contains the block number (disk address) of the next block allocated to the file. If FAT is cached in the memory, random (direct) access time is improved
  - **Indexed allocation**: This solves the problem in the linked allocation by bringing all pointers into one location: the **index block**. Each file requires at least one index block with no external fragmentation. It supports direct access. The overhead is associated with index block. The size of index block is determined by the file size. For large files, multiple level index is needed.

- **Free-Space Management**
  - **Bit Vector**: simplicity and efficiency in finding the first (n consecutive) free block(s). This must be kept in memory, but may require a large memory space.
  - **Linked list**: each free block contains a pointer to the next free blocks. Traversing the linked list is inefficient; But we usually only need one (the first) free block