OS Chapter – 7

File Management

Prepared By : Sheetal J. Nagar,
Atmiya Institute of Technology & Science

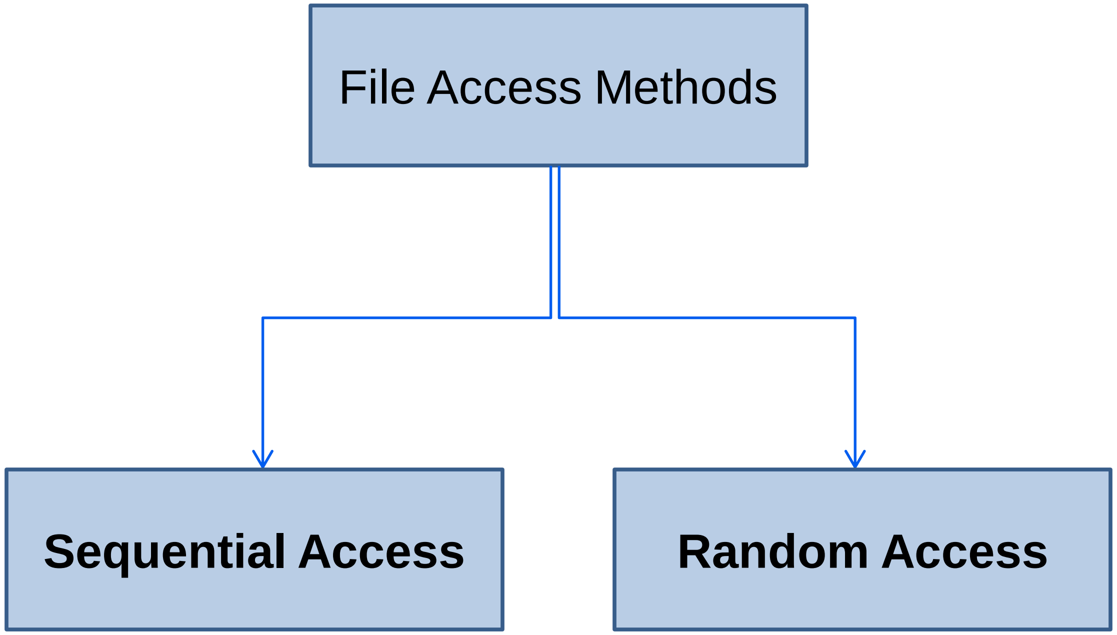
Topics Covered

1) File concept
2) Access methods
3) File types
4) File Attributes
5) File operation
6) Directory structure
7) File System structure
8) Allocation methods (contiguous, linked, indexed)
9) Free-space management (bit vector, linked list, grouping)
10) Directory implementation (linear list, hash table)

1.
File Concepts

- **“Files are logical units of information”** created by processes.
- **“A file is a unit of storing data on a secondary storage device such as a hard disk or other external media”**.
- Information stored in files must be persistent, that is, not be affected by process creation and termination.
- Files are managed by the operating system. How they are structured, named, accessed, used, protected, implemented, and managed are major topics in operating system design.
- As a whole, **“the part of the operating system dealing with files is known as the file system”**.

2.
File Access Methods


File Access Methods
Sequential Access
Random Access

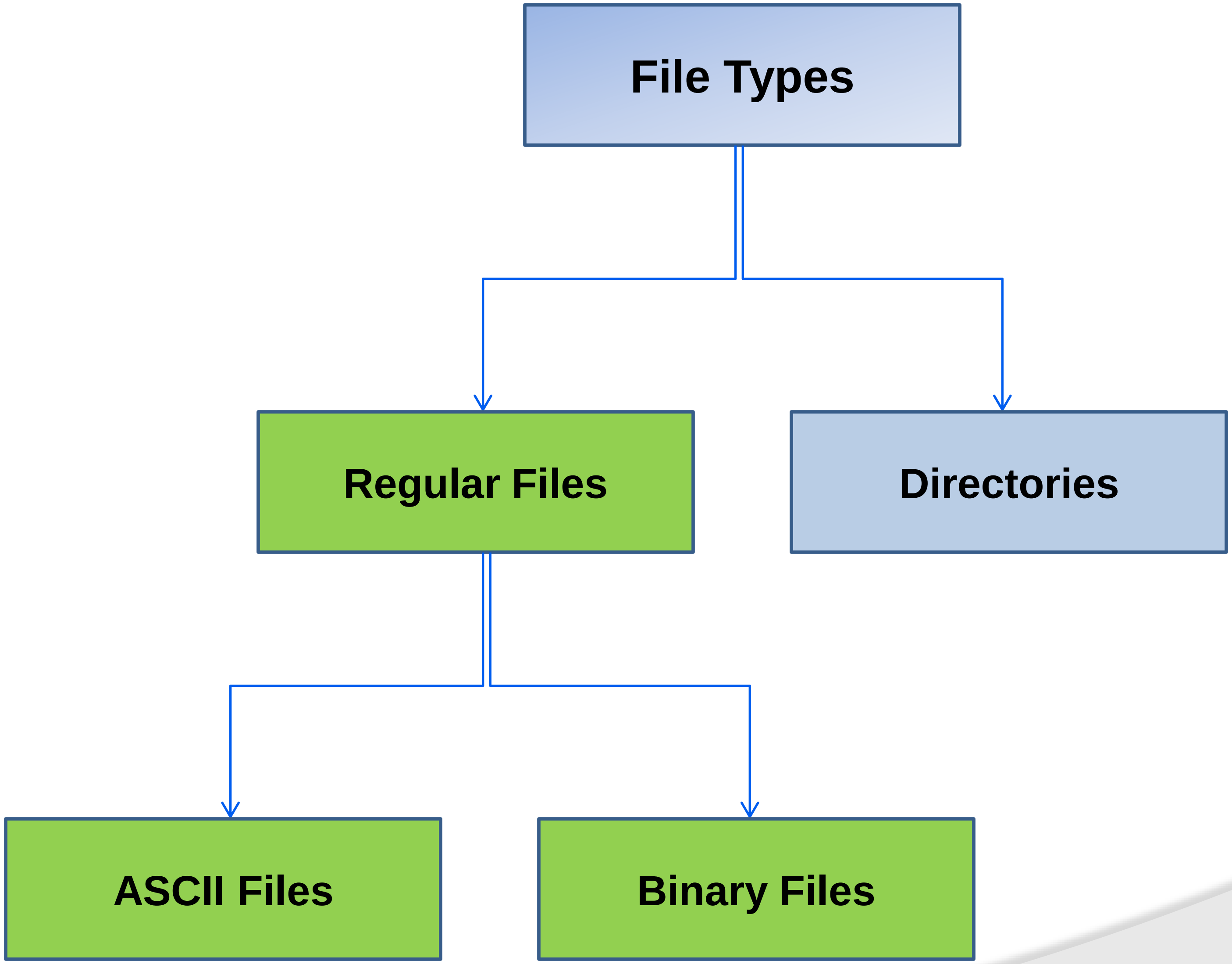
1) Sequential File Access

- In Sequential access, process could read all the bytes or records from a file in order, starting from the beginning, but could not skip around and read them out of order.
- Sequential files could be rewound.
- These files were convenient when the storage medium was magnetic tape or CD-ROM.

2) Random File Access

- Files whose bytes or records can be read **in any order** are called random access files.
- Random access files are essentials for many applications, for example, data base systems.
- If an airline customer calls up and wants to reserve a seat on a particular flight, the reservation program must be able to access the record for that flight without having to read the records for thousands of other flights.

3.
File Types


File Types
Regular Files
Directories
ASCII Files
Binary Files

1) Regular File

- Contains user information.
- Has no other predefined internal structure.
- Application programs are responsible for understanding the structure and content of any specific regular file.

2) Directories

- Directories are system files for maintaining the structure of the file system.
- To keep track of files, file systems normally have directories or folder.

3) ASCII files

- ASCII file consists of line of text.
- Advantage of ASCII files is :

  - they can be displayed & printed as it is

  - they can be edited with ordinary text editor.
- If number of programs use ASCII files for input and output, it is easy to connect the output of one program to the input of another.
- **C** / **C++** / **HTML** / **Perl** files are all examples of ASCII files.

4) Binary files

- Binary files contain formatted information that only certain applications or processors can understand.
- Binary files must run on the appropriate software or processor before humans can read them.
- Examples of binary files:
  - Executable files,
  - compiled programs,
  - spreadsheets,
  - compressed files,
  - graphic (image) files

4.
File Attributes

File Attributes

- Every file has a name and its data.
- In addition, all operating systems associate other information with each file.
- for example, the date and time the file was last modified and the file’s size. We will call these extra items the file’s **attributes.**
- This is also known as **metadata.**
- **The list of attributes varies considerably from** system to system.

Attribute	**Meaning**
Protection	Who can access the file and in what way.
Password	Password needed to access the file.
Creator	ID of the person who created the file.
Owner	Current owner.

- These attributes relates to the files attributes and tell who may access it and who may not.

Attribute	Meaning
Read only flag	0 for read/write, 1 for read only.
Hidden flag	0 for normal, 1 for do not display the listings.
System flag	0 for normal, 1 for system file.
Archive flag	0 for has been backed up, 1 for needs to be backed up.
Random access flag	0 for sequential access only, 1 for random access.
Temporary flag	0 for normal, 1 for delete file on process exit.
Lock flag	0 for unlocked, 1 for locked.

- Flags are bits or short fields that control or enable some specific property.

Attributes	**Meaning**
Record length	Number of bytes in a record.
Key position	Offset of the key within each record.
Key length	Number of bytes in key field.

- The record length, key position and key length are only present in files whose record can be looked up using the key. They provide the information required to find the keys.

Attributes	Meaning
Creation time	Date and time the file was created.
Time of last access	Date and time of file was last accessed.
Time of last change	Date and time of file was last changed.

- The various times keep track of when the file was created, most recently accessed and most recently modified.

Attribute	Meaning
Current size	Number of bytes in the file.
Maximum size	Number of bytes the file may grow to.

- The current size tells how big the file is at present. And some old mainframe operating system requires the maximum size to be specified.

5.

File Operations

File Operations

- File exists to store information and allow it to be retrieved later.
- Different system provides different operations to allow storage and retrieval.
- The most common system calls are shown below.

File Operations

1) Create
2) Delete
3) Open
4) Close
5) Read
6) Append
7) Seek

6.
Directory Structures

Directory Structures

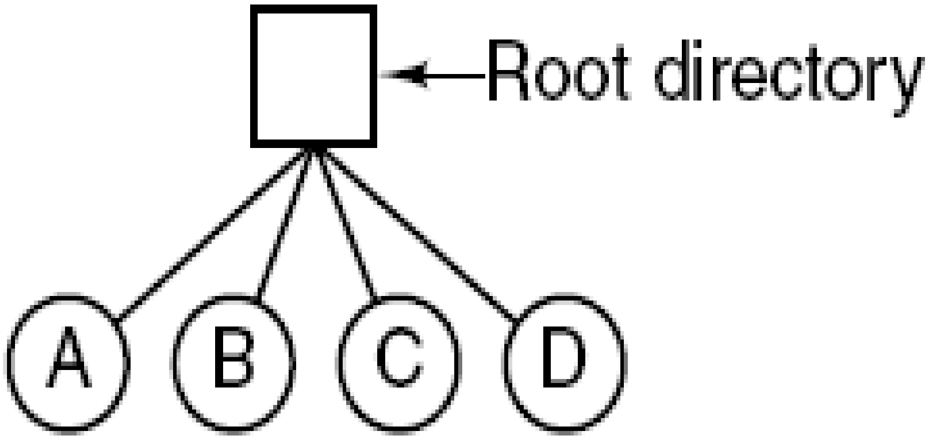
- To keep track of files, file systems normally have directories or folders.
- **Directories** are <u>system files</u> for maintaining the structure of the file system.

Single Level Directory system

- The simplest form of directory system is having one directory containing all the files.
- This is some time called as root directory.
- The advantages of this scheme are its simplicity and the ability to locate files quickly there is only one directory to look, after all.

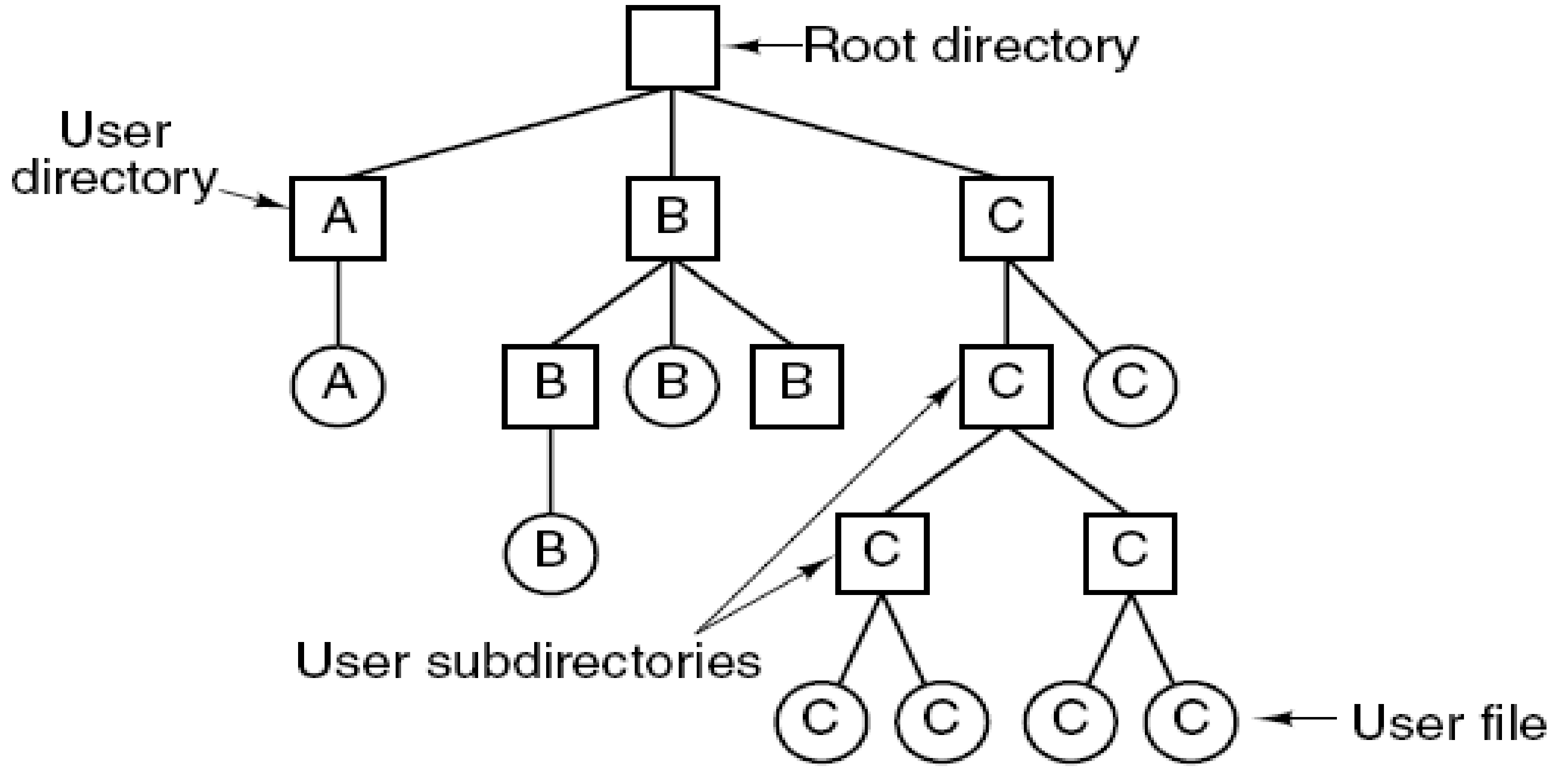
Single Level Directory system

- Devices such as digital cameras and some portable music players use this structure.



Hierarchical Directory system

- The ability for users to create any number of subdirectories provides a powerful structuring tool for user to organize their work.



Two methods are used to specify file names:

1) Absolute path name
2) Relative path name

1) Absolute Path

"Absolute Path name consisting of the path from the root directory to the file".

- As an example, the path

  / home / administrator / OS / Factorial.bash

- ***Absolute path names always start at the root* directory and are unique**.
- In UNIX the components of the path are separated by ***/***.
- In Windows the separator is \ .

2) Relative Path

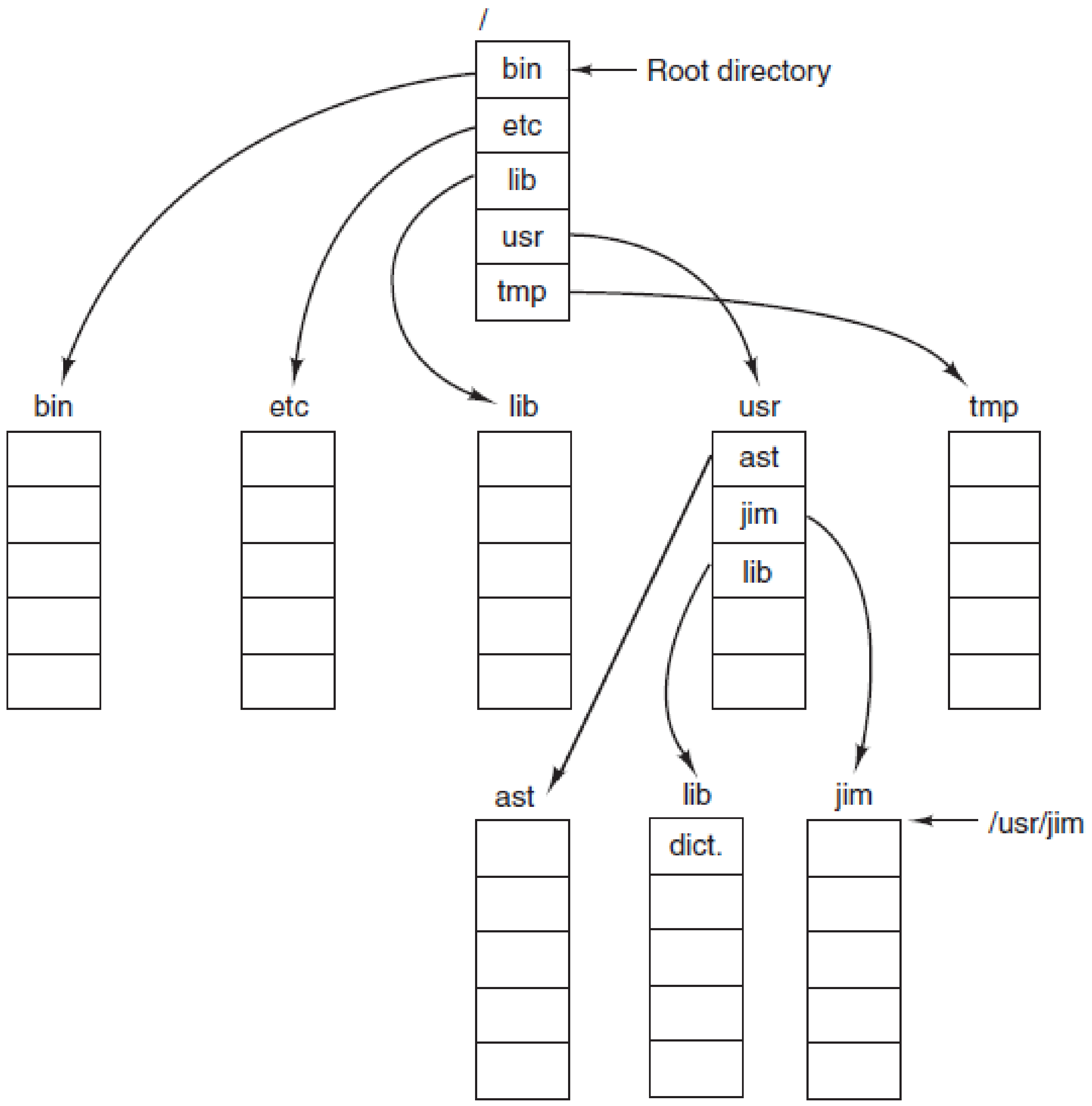
- “User can designate one directory as the **current working directory**, in which case, all path names (not beginning at the root directory) are taken relative to working directory.
- For example if the current working is ***/home/administrator***, then the file whose absolute path is ***/home/administrator/OS/Factorial.bash*** can be referenced simply as ***OS/Factorial.bash***

- Most operating systems that support a hierarchical directory system have two special entries in every directory,

  "**.**" Or "dot" refers to current directory

  "**..**" or "dotdot" refers to parent directory

Unix Directory Tree



Directory Operations

1. Create
2. Delete
3. Opendir
4. Closedir
5. Readdir
6. Rename
7. **Link**
8. **Unlink**

Link

- “Linking is a technique that allows a file to appear in **more than one directory**.”
- This system call specifies an **existing file and a path name**, and creates a link from the existing file to the name specified by the path.
- In this way, the same file may appear in multiple directories.

Types of Links

Two Types of Links :

1) Hard Link (or Physical Link)
2) Soft Link (or Symbolic Link)

1) Hard Link :

A link, which increments the counter in the file’s i-node (to keep track of the number of directory entries containing the file), is called a **hard link.**

Example of Hard Link :

- Consider that a file “my_file” has following absolute path:

  /home/usr/OS/my_file.

- Consider that the file is just created and no link is created for this file,

  So file’s i-node contains **count = 1**

- Now one hard link “**link1**” is created for “**my_file**” and placed in the home directory

  /home/link1

  thus, this file is now accessible from 2 paths, so **count = 2**

- Now if the file is removed from it’s original path, then just count is decremented. So **count = 1.**
- As count is not yet 0, the original file will not be removed ant it will be still accessible through the hard link “link1”

Soft Link :

- Shortcuts are known as Symbolic links or Soft Links.
- **Instead, of having** two names point to the same internal data structure representing a file, a name can be created that points to a tiny file naming another file.
- It just stores the target path.
- Symbolic links do not count and do not cause the counter for the target file to be incremented.
- Whenever a link is accessed the file in the target path is opened.

Example of Soft Link :

- Consider that a file “my_file” has following absolute path:

  /home/usr/OS/my_file.

- Now one soft link “**link2**” is created for “**my_file**” and placed in the home directory

  /home/link2

  thus, this file is now accessible from 2 paths.

- Now if the file is removed from it’s original path, then **original file itself is removed** from the disk.
- And **Link2** starts pointing to a file which doesn’t exist.
- Subsequent attempts to use the file “my_file” via a symbolic link “link2” will fail when the system is unable to locate the file.

Unlink

- For Hard Link:
  - A directory entry is removed. If the file being unlinked is **only present in one directory** (the normal case), it is removed from the file system.
  - If **it is present in multiple directories**, only the path name specified is removed. The others remain.

Unlink

- For Soft Link:
  - If the file is removed from it’s original path, then **original file itself is removed** from the disk.
  - So, Subsequent attempts to use the original file via a symbolic link will fail when the system is unable to locate the file

7.

File System Structure

(Boot Block , Super Block, etc..)

File-System Layout

- File systems are stored on disks.
- Most disks can be divided up into one or more partitions, with independent file systems on each partition.
- Sector 0 of the disk is called the **MBR** (Master Boot Record) and is used to boot the computer.
- After Master Boot Record there is a partition table.

Partition table:

- This table gives the starting and ending addresses of each partition.
- One of the partitions in the table is marked as “active”.

1) When the computer is booted, the BIOS (Basic Input Output system) reads and executes the MBR.
2) The first thing the MBR program does is
    1. **Locate the Active partition,**
    2. **Read in its Boot block, i.e. first block, and**
    3. **Execute it. The program in the boot block loads the operating system contained in that partition.**

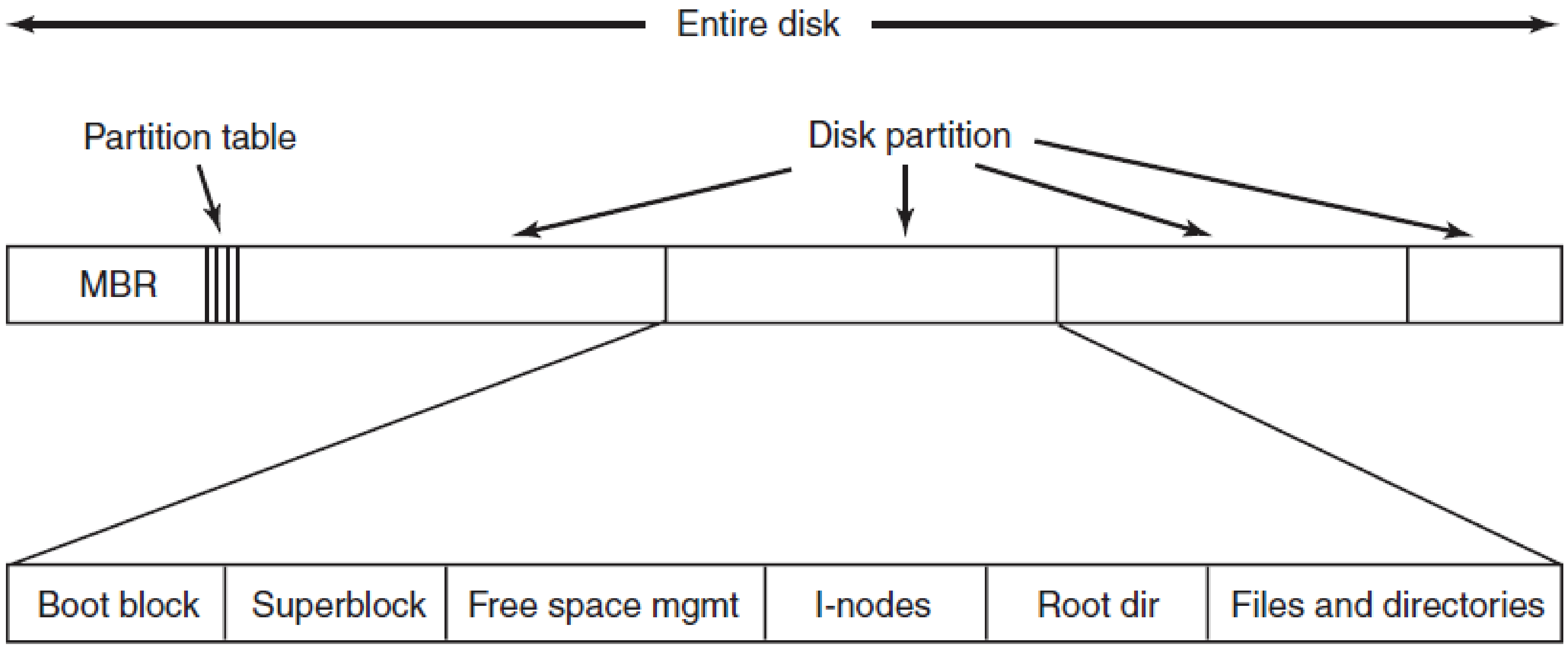


Figure: A possible file-system layout.

- For uniformity, every partition starts with a boot block, even if it does not contain a bootable operating system.
- Besides, it might contain one in the future.
- Often the file system will contain some of the items shown in figure.

- Next block is the **superblock**. **It contains all the key parameters** about the file system and is read into memory when the computer is booted or the file system is first touched.
- Typical information in the superblock includes
  - A magic number
    - to identify the **file-system type**,
    - the **number of blocks** in the file system,
    - and other key **administrative** information.

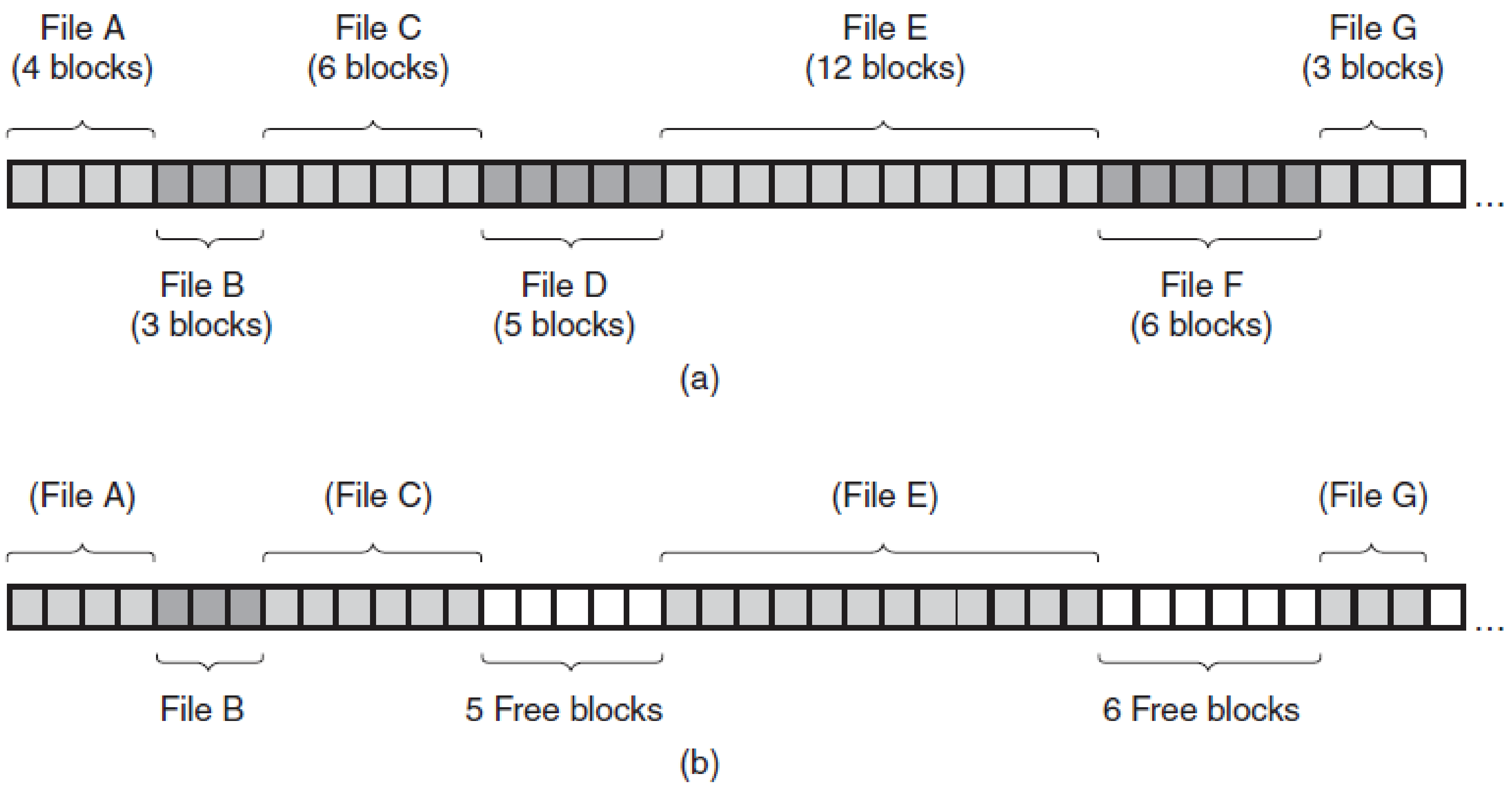
- Next block contains information about free blocks in the file system.
- for example in the form of a bitmap or a list of pointers.
- This might be followed by the i-nodes “an array of data structures, one per file, telling all about the file”.
- After that might come the root directory, which contains the top of the file-system tree.
- Finally, the remainder of the disk contains all the other directories and files.

8.
File Implementation

- Various methods to implement files are listed below,
  1) Contiguous Allocation
  2) Linked List Allocation
  3) Linked List Allocation Using A Table In Memory
  4) I-nodes

1) Contiguous Allocation

- The simplest allocation scheme is to store each file as a contiguous run of disk blocks.
- Thus on a disk with 1-KB blocks, a 50-KB file would be allocated 50 consecutive blocks.
- With 2-KB blocks, it would be allocated 25 consecutive blocks.



(a) Contiguous allocation of disk space for seven files.

(b) The state of the disk **after files *D and F have been removed.***

Advantages

1) Simple to implement because it is **easy to find** where a file is stored by knowing

    - The disk <u>address of the first block</u> and

    - The <u>number of blocks in the file</u>.

2) The **read** performance is excellent because the entire file can be read from the disk in a single operation. Only one seek is needed (to the first block). After that, no more seeks or rotational delays are needed

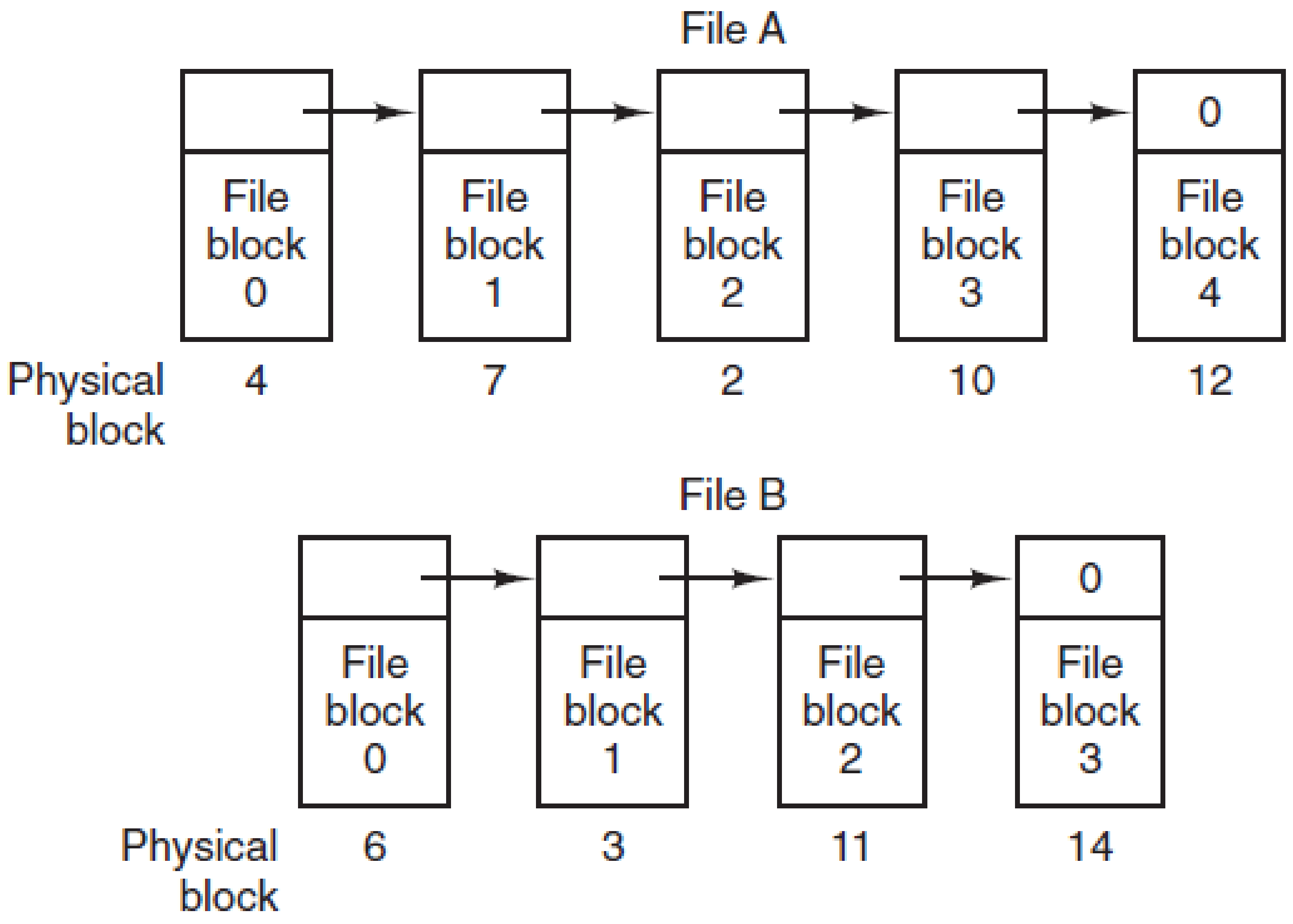
3) Thus contiguous allocation is simple to implement and has high performance.

Disadvantages

1) Over the course of time, the disk becomes **fragmented**.
2) Initially, fragmentation is not a problem, since each new file can be written at the end of the disk, following the previous one.
3) However, eventually the disk will fill up and it will become necessary to either **compact** the disk, is expensive, or to reuse the free space in the holes.
4) Reusing the space requires maintaining a list of hole.
5) However, when a new file is to be created, it is necessary to know its final size in order to choose a **hole of the correct size** to place it in.

2) Linked-List Allocation

- keep each file as a linked list of the disk blocks.
  1) The first word of each block is used as a pointer to the next one.
  2) The rest of the block is for data.
- Unlike contiguous allocation, every disk block can be used in this method.
- No space is lost to disk fragmentation.
- It is sufficient for a directory entry to store only disk address of the first block, rest can be found starting there.



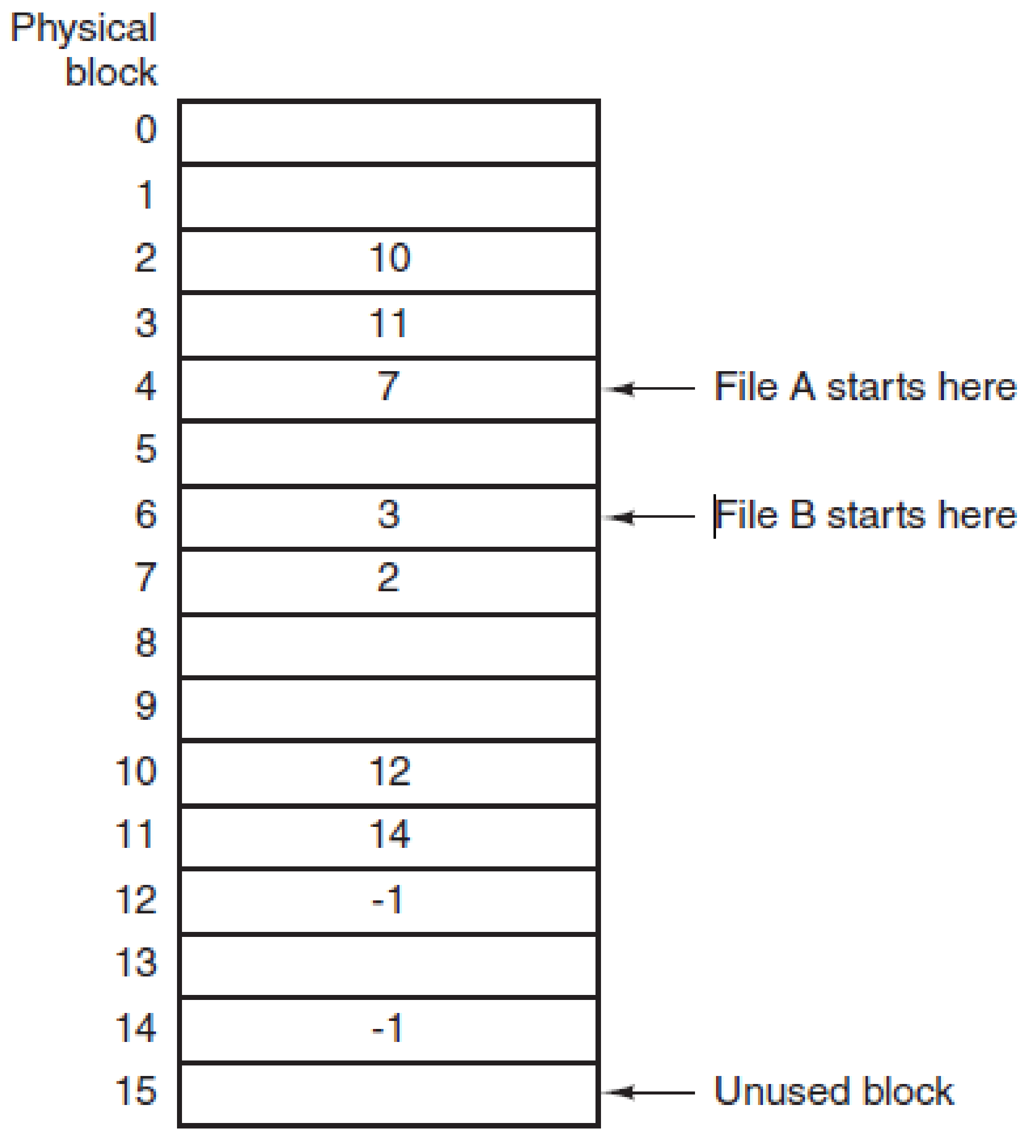
Storing a file as a linked list of disk blocks

Disadvantages

1) Although reading a file sequentially is straightforward, random access is **extremely slow**. To get to block n, the operating system has to start at the beginning and read the n-1 blocks prior to it, one at a time.
2) With the first few bytes of each block occupied to a pointer to the next block, reads of the full block size require acquiring and concatenating information from two disk blocks, **which generates extra overhead.**

3) Linked-List Allocation using a table in memory

- This implementation is also known as FAT (File Allocation Table).
- “Disadvantage of the linked list allocation can be eliminated by taking the **pointer word from each disk block** and putting it in a table in memory”.
- Ex. File A uses disk blocks 4, 7, 2, 10, and 12 in that order.
- File B uses disk blocks 6, 3, 11, and 14 in that order.
- Both chains are terminated with a special marker (eg. -1) that is not a valid block number. Such a table in main memory is called a FAT (File Allocation Table).



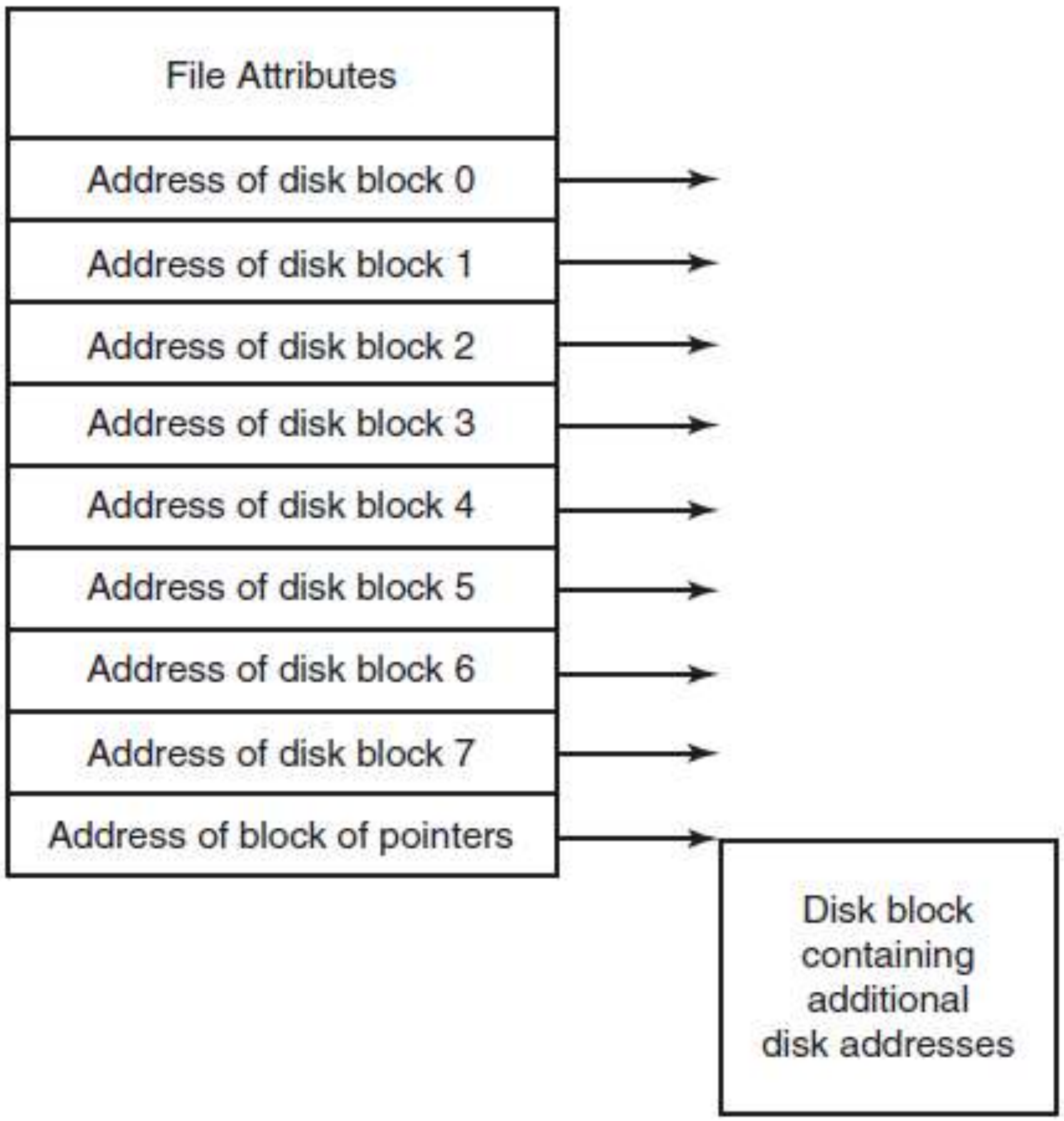
Linked-list allocation using a **FAT** in main memory.

Disadvantages

- The entire table must be in memory all the time to make it work.

4) i-nodes

- "To keep the track of <u>which blocks belong to which file</u>, a data structure can be associated with each file which is known as an i-node (index-node), which contains
  1) The attributes and
  2) disk addresses of the file's blocks."
- Given the i-node, it is then possible to find all the blocks of the file.



example i-node.

Advantages

1) The big advantage of this scheme over linked files using an in-memory table is that i- node need only be in memory when the corresponding file is **open**.
2) If each i-node occupies n bytes and a maximum of k files may be open at once, the total memory occupied by the array holding the i-nodes for the open files is only *kn* bytes. Only this much space needs to be reserved in advance.
3) This space is usually far smaller than the space occupied by the file table.

Disadvantages

1) One problem with i-nodes is that if each one has room for a fixed number of disk addresses, what happens when a file grows beyond this limit?

   Solution : One solution is to reserve the last disk address

   <u>not for a data block</u>, but instead for the <u>address of a block</u>

   containing more disk block addresses, as shown in Figure.

9.
Free Space Management

Free Space Management

- "To keep track of free disk space, the system maintains a free-space list".
- The free-space list records *all free* disk blocks—those not allocated to some file or directory
- Various methods for free space management :

1) Bit Vector
2) Linked List
3) Grouping
4) Counting (Not in syllabus)

1) Bit Vector

- **Each** block is represented by 1 bit.
- If the block is **free** → the bit is **1**;
- if the block is allocated → the bit is 0.
- Example: consider a disk where blocks

  2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25,26, and 27 are free

  and the rest of the blocks are allocated.
- The free-space bit map would be

  00**1111**00**111111**000**11**0000000**111**00000

Advantages

1) Simple to implement
2) efficient in finding the first free block or *n consecutive free blocks on the* disk.

2) Linked List

- Another approach to free-space management is to **link together** all the free disk blocks, **keeping a pointer to the first free block in a special location** on the disk and caching it in memory.
- This first block contains a pointer to the next free disk block, and so on.

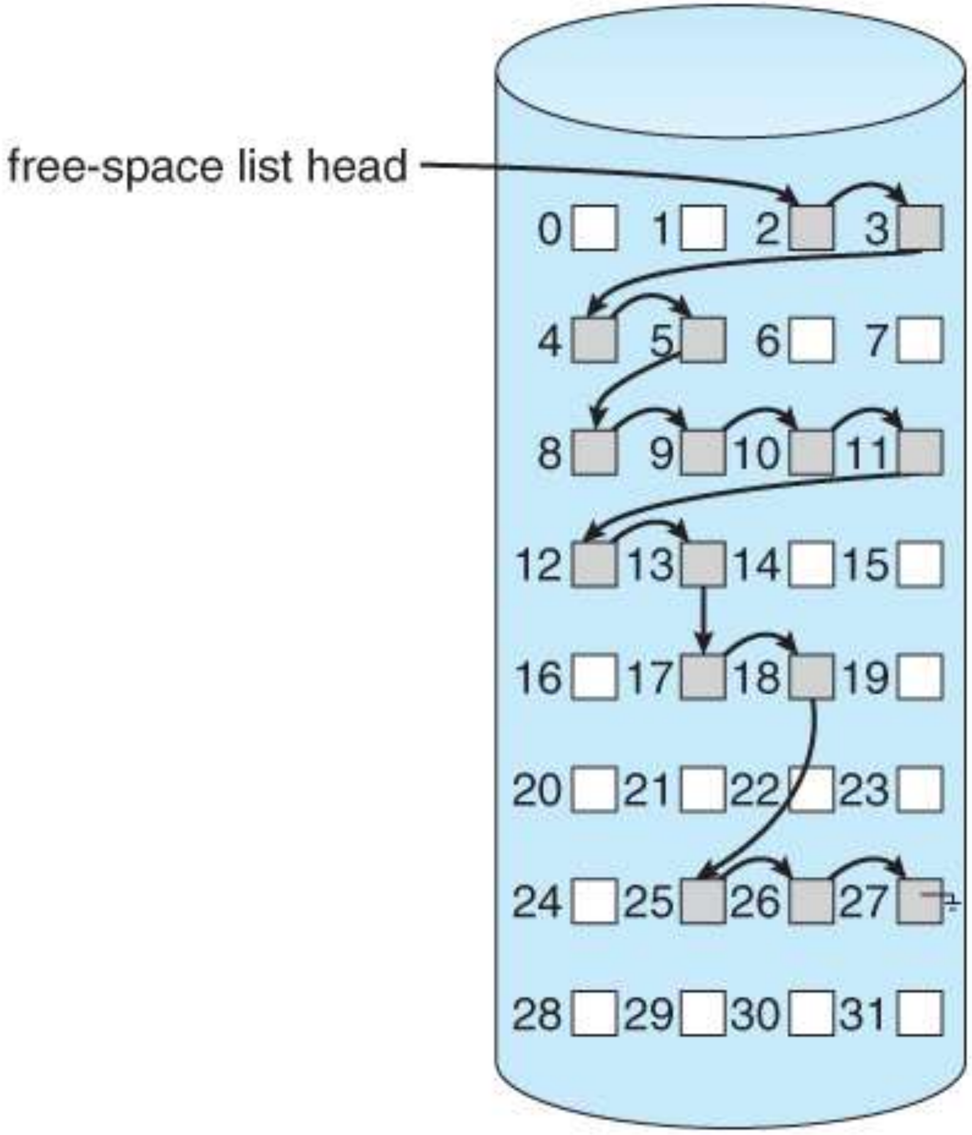


Figure : Linked free-space list on disk

Disadvantage

- Not efficient.
- To traverse the list, we must read each block, which requires substantial I/O time.
- Fortunately, traversing the free list is not a frequent action. Usually, the OS simply needs a free block so that it can allocate that block to a file, so the first block in the free list is used.
- The FAT method incorporates free-block accounting into the allocation data structure. No separate method is needed.

3) Grouping

- It stores :
  1) The addresses of n free blocks in the first free block.
  2) The first n - 1 of these blocks are actually free.
  3) The last block in the series contains the addresses of another n free blocks, and so on.
- The addresses of a large number of free blocks can now be found quickly.

10.
Directory Implementation

Directory Implementation

- When a file is opened, the operating system uses the path name supplied by the user to locate the directory entry on the disk.
- The directory entry provides the information needed to find the disk blocks.
- Depending on the system, this information may be the disk address of the entire file (with contiguous allocation), the number of the first block (both linked- list schemes), or the number of the i-node.

Directory Implementation

- **Two types of directory implementation**

1) **Linear list**
2) **Hash table**

1) Linear List

- The simplest method of implementing a directory
- A linear list of file names with pointers to the data blocks is used.
- To <u>create</u> a new file :
  1) First search the directory to be sure that no existing file has the same name.
  2) Then we add a new entry at the end of the directory.
- To <u>delete</u> a file :
  1) First search the directory for the file,
  2) Then release the space allocated to it.

- To <u>reuse</u> the directory entry:
  1) We can mark the entry as unused.
  2) or we can attach it to a list of free directory entries.
- **Disadvantage** of a linear list
  1) Finding a file requires a linear search. Directory information is used frequently, and users will notice if access to it is slow.

Solutions :

1) cache can be used to store the most recently used directory information. A cache hit avoids the need to constantly reread the information from disk.
2) A sorted list allows a binary search and decreases the average search time.
3) B-tree, can also be used.

2) Hash Table

- With this method, a linear list stores the directory entries, but a hash data structure is also used.
- The hash table takes a value computed from the file name and returns a **pointer to the file name** in the linear list.
- It can greatly decrease the directory search time.
- Insertion and deletion are also straightforward, although some provision must be made for **collisions – “**situations in which two file names hash to the same location.”

Disadvantages

- Hash tables are generally of fixed size.
- Hash functions are dependent on that size.
- For example, assume that we make a linear-probing hash table that holds 64 entries.
- The hash function converts file names into integers from 0 to 63, for instance, by using the remainder of a division by 64.

- If we later try to create a 65th file, we must enlarge the directory hash table—say, to 128 entries.
- As a result, we need a **new hash function** that must map file names to the range 0 to 127, and we must **reorganize** the existing directory entries to reflect their new hash-function values.
- Alternatively, a **chained-overflow** hash table can be used. Each hash entry can be a linked list instead of an individual value, and we can resolve collisions by adding the new entry to the linked list.

- Lookups may be somewhat slowed, because searching for a name might require stepping through a **linked list of colliding table entries**. Still, this method is likely to be much faster than a linear search through the entire directory.

Reference Books:

1. "Operating System Concepts" by Silberschatz, Peter B. Galvin and Greg Gagne
2. "Modern Operating Systems" by Andrew S Tanenbaum

THANK YOU