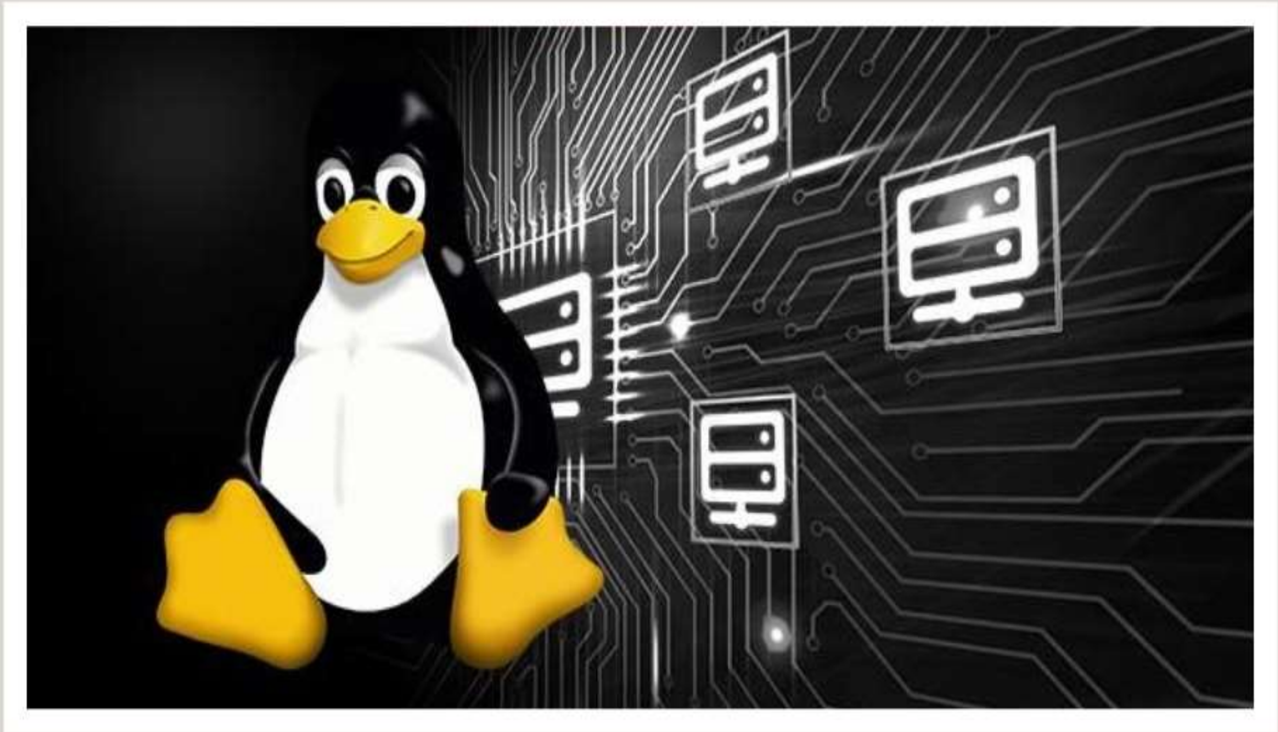


Operating systems

By
I Ravindra kumar, B.Tech, M.Tech,(Ph.D.)
Assistant professor,
Dept of CSE, VNR VJIET

File System

File Concept:

- Computers can store information on several different storage media
  - magnetic disks, magnetic tapes, and optical disks
- Computer system will be convenient to use,

  The operating system provides a uniform logical view of information storage.
- The operating system abstracts from the physical properties of its storage devices to define a logical storage unit (the file).
- Files are mapped, by the operating system, onto physical devices
- A file is a named collection of related information that is recorded on secondary storage

- **File Attributes**

  A file has certain other attributes, which vary from one operating system to another, Typically consist of these:

  - **Name:**
  - **Identifier:**
  - **Type:**
  - **Location:**.
  - **Size:**.
  - **Protection:**
  - **Time, date, and user identification:**

- **File Operations:** A file is an **abstract data type.**
  - **Creating a file:**
  - **Writing a file:**
  - **Reading a file:current-file-position pointer**
  - **Repositioning within a file:**
  - **Deleting a file:**
  - **Truncating a file:**
- The operating system keeps a small table containing information about all open files (the **open-file table).**
- several pieces of information are associated with an open file.
  - **File pointer:**
  - **File open count:**
  - **Disk location of the file:**
  - **Access rights**

File Types

file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

File Structure

- The operating system may
  - require that an executable file have a specific structure
  - it can determine where in memory to load the file
  - What is the location of the first instruction
- one of the disadvantages
  - multiple file structures:
- The resulting size of the operating system
  - cumbersome.
- If the operating system defines five different file structures,
  - it needs to contain the code to support these file structures.
- Severe problems may result from new applications that require information structured in ways not supported by the operating system.
- The Macintosh operating system also supports a minimal number of file structures.
- It expects files to contain two parts: a **resource fork and a data fork.**
- The resource fork contains information of interest to the user.

Internal File Structure

- Locating an offset within a file can be complicated
- All disk I/O is performed in units of one block (physical record), and all blocks are the same size.
  - Logical records may even vary in length.
- **Packing a number of logical** records into physical blocks is a common solution to this problem.
- For example, the UNIX operating system defines all files to be simply a stream of bytes.
  - Each byte is individually addressable by its offset from the beginning (or end) of the file. In this case, the logical record is 1 byte.
- The file system automatically packs and unpacks bytes into physical disk
- The logical record size, physical block size, and packing technique determine how many logical records are in each physical block.
- The packing can be done either
  - the user's application program or
  - by the operating system.
- The wasted bytes allocated to keep everything in units of blocks (instead of bytes) is *internal fragmentation.*
  - the larger the block size, the greater the internal fragmentation.

File-System Structure

- Two characteristics that make them a convenient medium for storing multiple files:
  - They can be rewritten in place;
  - They can access directly any given block of information on the disk.
- **A file system poses two quite different design problems.**
  - how the file system should look to the user
  - Creating algorithms and data structures to map the logical file system onto the physical secondary-storage devices.
- A file control block **(FCB) contains**
  - **information about the file,**
  - **ownership,**
  - permissions
  - location of the file contents
- UNIX uses the UNIX file system **(UFS) as a base.**
- **Windows NT supports disk** file-system formats of
  - **FAT, FAT32 and NTFS (or Windows NT File System),**
  - **as** well as CD-ROM, DVD, and floppy-disk file-system formats.

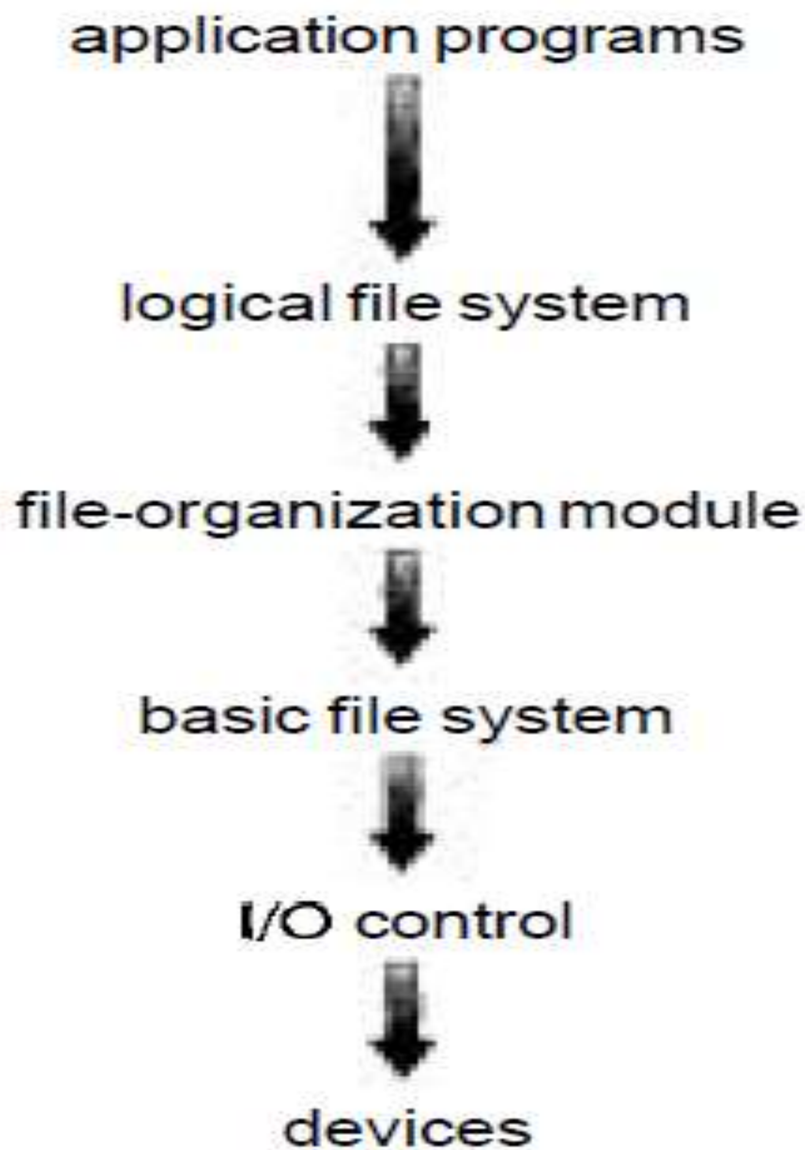


Figure 12.1 Layered file system.

- **Directory Implementation:**

  selection of directory-allocation and directory-management algorithms has a large effect on the

  – efficiency, performance, and reliability of the file system

- **Linear List**

  – To use a linear list of file names with pointers to the data blocks.

  – Requires a linear search to find a particular entry.

  – Simple to program but time-consuming to execute

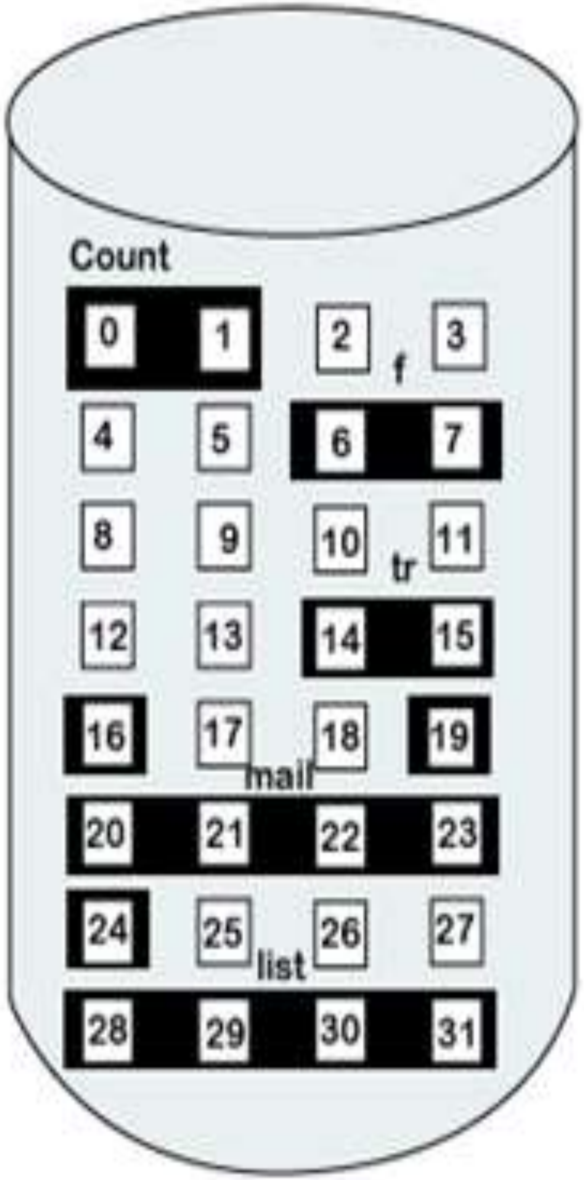
Hash Table

- 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
- Insertion and deletion are also fairly straightforward,
- 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

Allocation Methods:

– how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly.

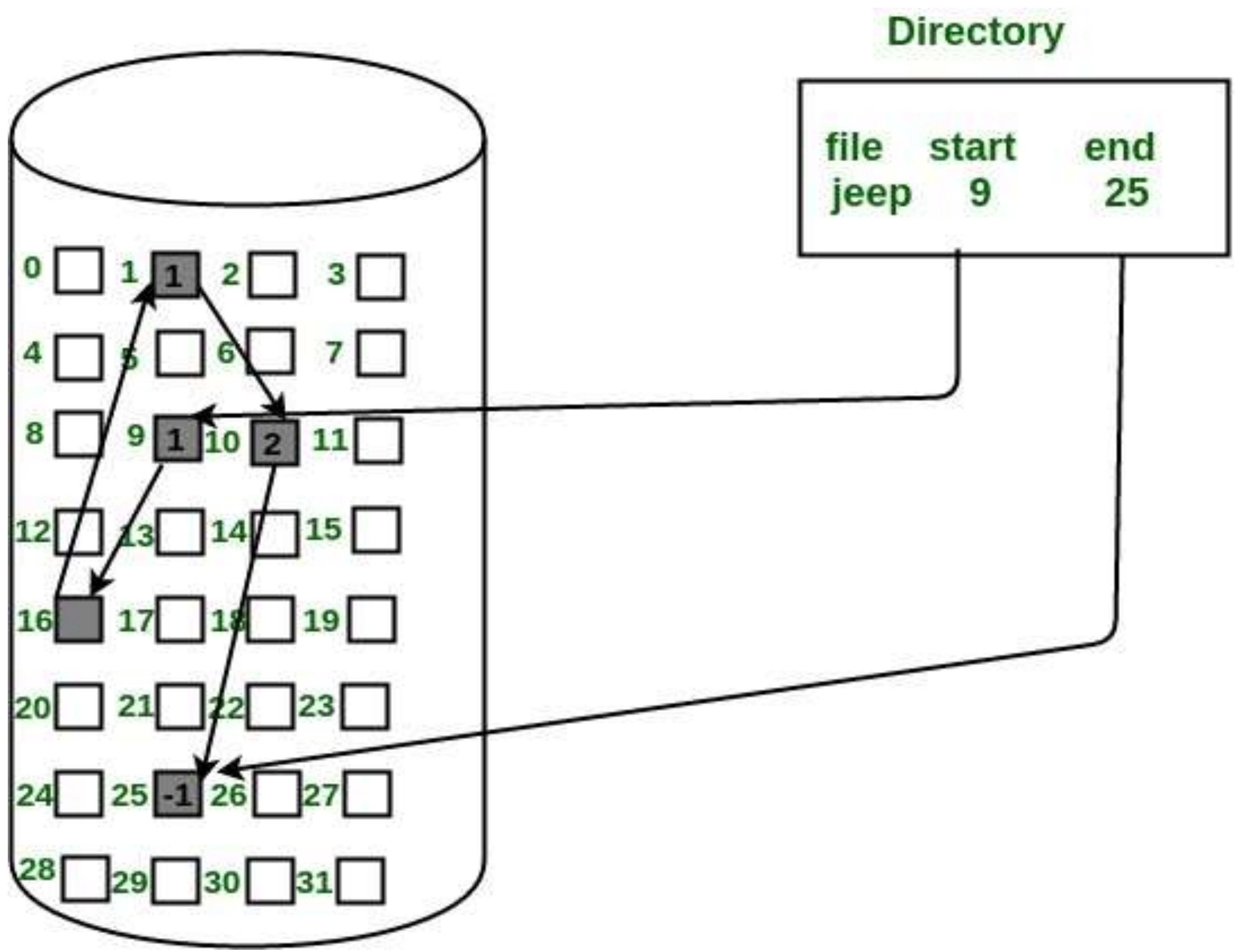
- **Contiguous Allocation:**
  – requires each file to occupy a set of contiguous blocks on the disk
  – Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block.
  – If the file is *n blocks long and starts at location* b, then it occupies blocks b, b + 1, b + 2, ..., b + *n - 1*.
  – The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file



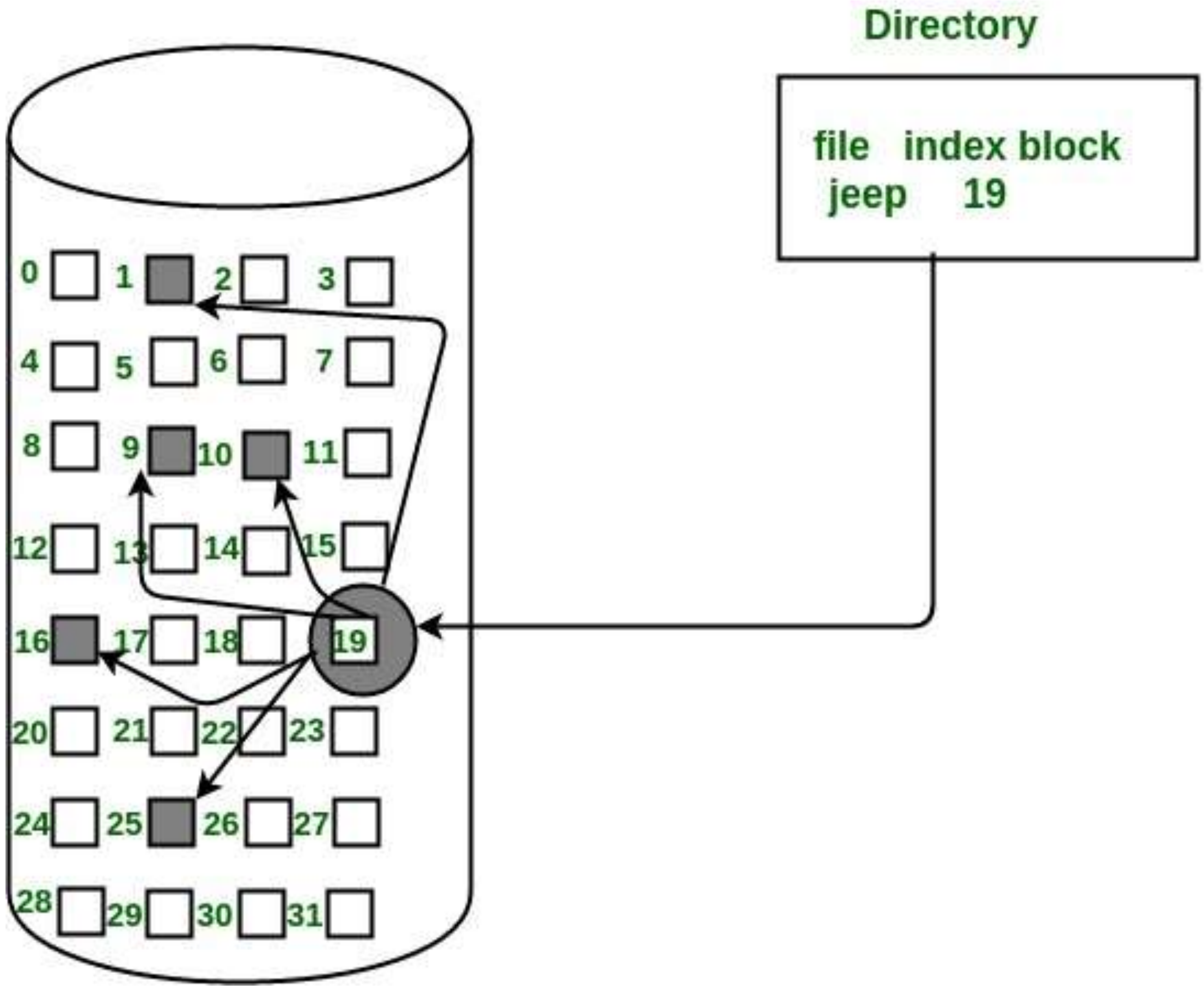
Directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

- **Linked Allocation:**
  - With linked allocation, each file is a linked list of disk blocks;
  - the disk blocks may be scattered anywhere on the disk.
  - The directory contains a pointer to the first and last blocks of the file.
  - An important variation on the linked allocation method is the use of a file allocation table **(FAT).**
    - **This simple but efficient method of disk-space allocation**
    - It is used by the MS-DOS and OS/2 operating systems.
  - A section of disk at the beginning of each partition is set aside to contain the table.


Directory
file start end
jeep 9 25
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
16 17 18 19
20 21 22 23
24 25 26 27
28 29 30 31
1
1
2
-1

- **Indexed Allocation**
  - the pointers to the blocks are scattered with the blocks themselves all over the disk and need to be retrieved
  - in order, **Indexed allocation solves this problem by bringing all the pointers** together into one location: the **index block.**
  - Each file has its own index block, which is an array of disk-block addresses.
  - The ith entry in the index block points to the ith block of the file.
  - The directory contains the address of the index block



Directory
file index block
jeep 19
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

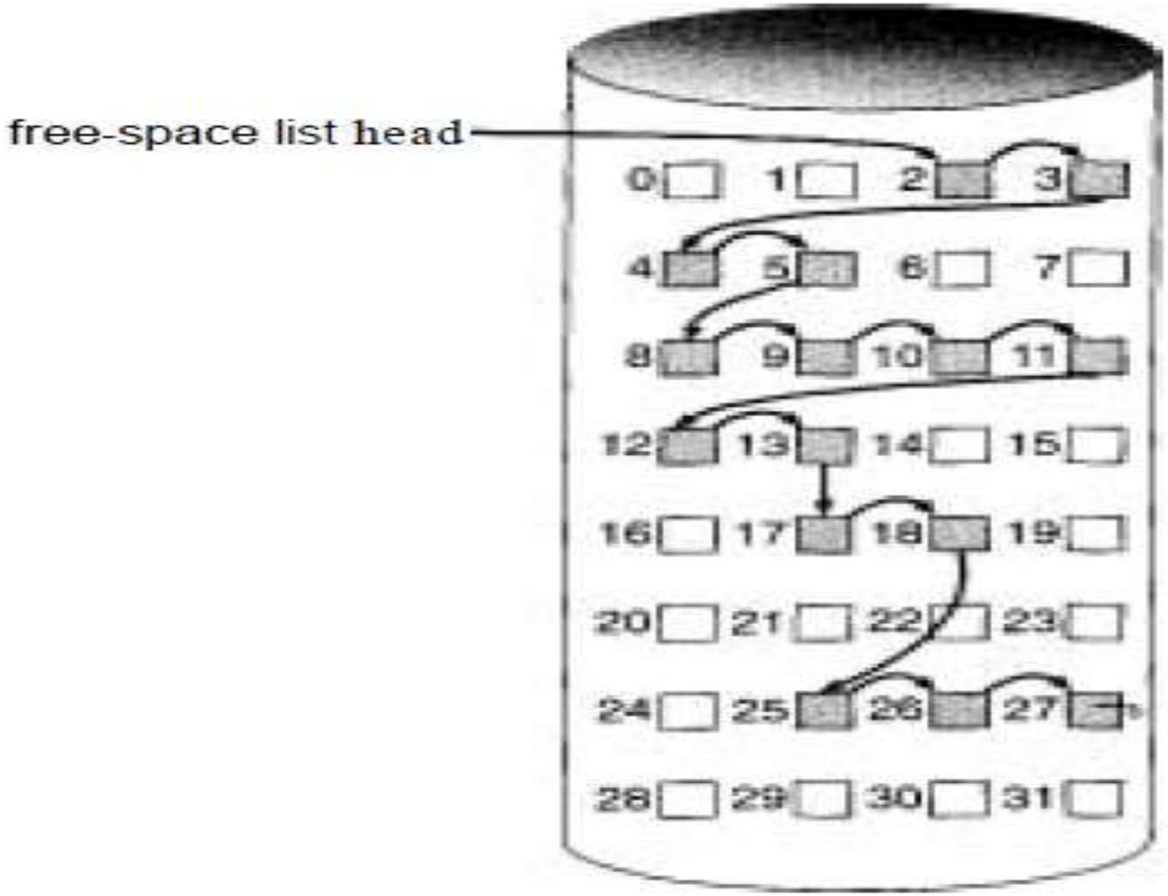
If the index block is too small, however, it will not be able to hold enough pointers for a large file, and a mechanism will have to be available to deal with this issue:

- **Linked scheme:**
- **Multilevel index:**
- **Combined scheme:**
  - say, 15 pointers of the index block in the file's inode.
  - The first 12 of these pointers point to **direct blocks;**
  - The next **3 pointers point to indirect blocks.**
    - **The first indirect block pointer is the address of a single indirect block.**
    - Then there is a **double indirect block pointer,**
    - The last pointer would contain the address of a **triple indirect block.**

- **Free-Space Management:**
  - 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.
- **Bit Vector:**
  - free-space list is implemented as a bit **map or 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.
  - its relatively simplicity and efficiency in finding the first free block, or *n consecutive free blocks on the disk.*
  - The first non-0 word is scanned for the first 1 bit, which is the location of the first free block.
  - The calculation of the block number is

  (number of bits per word) x (number of 0-value words) + offset of first 1 bit.

- **Linked List:**
  – keeping a pointer to the first free block in a special location on the disk and caching it in memory



- **Grouping:**
  - to store the addresses of n free blocks in the first free block.
  - The first n-1 of these blocks are actually free.
  - The last block contains the addresses of another n free blocks
- **Counting:**
  - Keep the address of the first free block and the number n of free contiguous blocks that follow the first block.
  - Each entry in the free-space list then consists of a disk address and a count.

Efficiency:

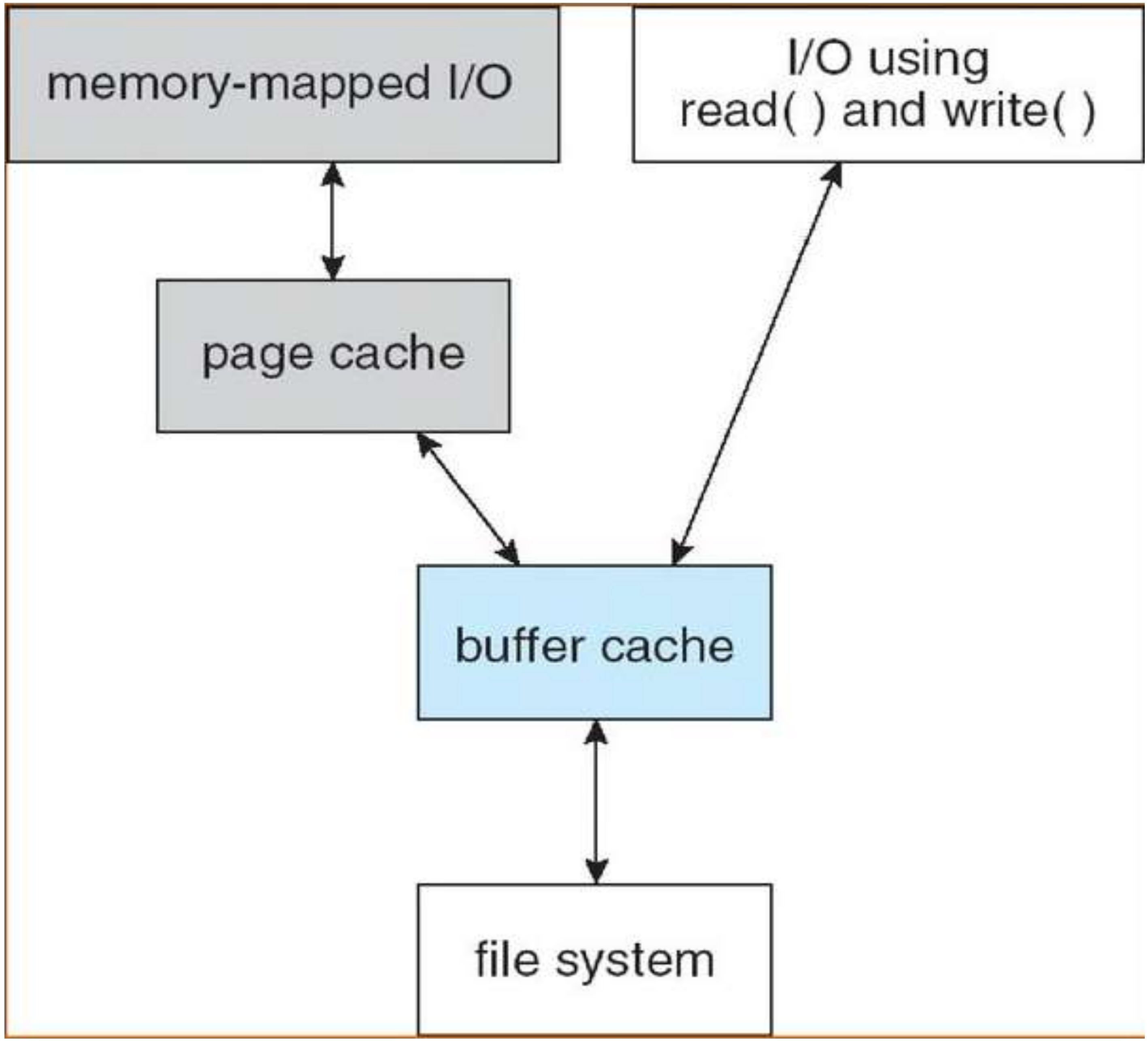
- For instance, UNIX inodes are preallocated on a partition.
- Even an "empty" disk has a percentage of its space lost to inodes.
- by preallocating the inodes and spreading them across the partition, we improve the file system's performance
- BSD UNIX varies the cluster size as a file grows.
- Large clusters are used where they can be filled, and small clusters are used for small files and the last cluster of a file.
- The types of data normally kept in a file's directory (or inode) entry also require consideration.
- Commonly, a "last write date" is recorded to supply information to the user and to determine whether the file needs to be backed up.
- Some systems also keep a "last access date," so that a user can determine when the file was last read.
- The result of keeping this information is that, whenever the file is read, a field in the directory structure must be written to.
- This change requires the block to be read into memory, a section changed, and the block written back out to disk, because operations on disks occur only in block (or cluster) chunks.

Performance:

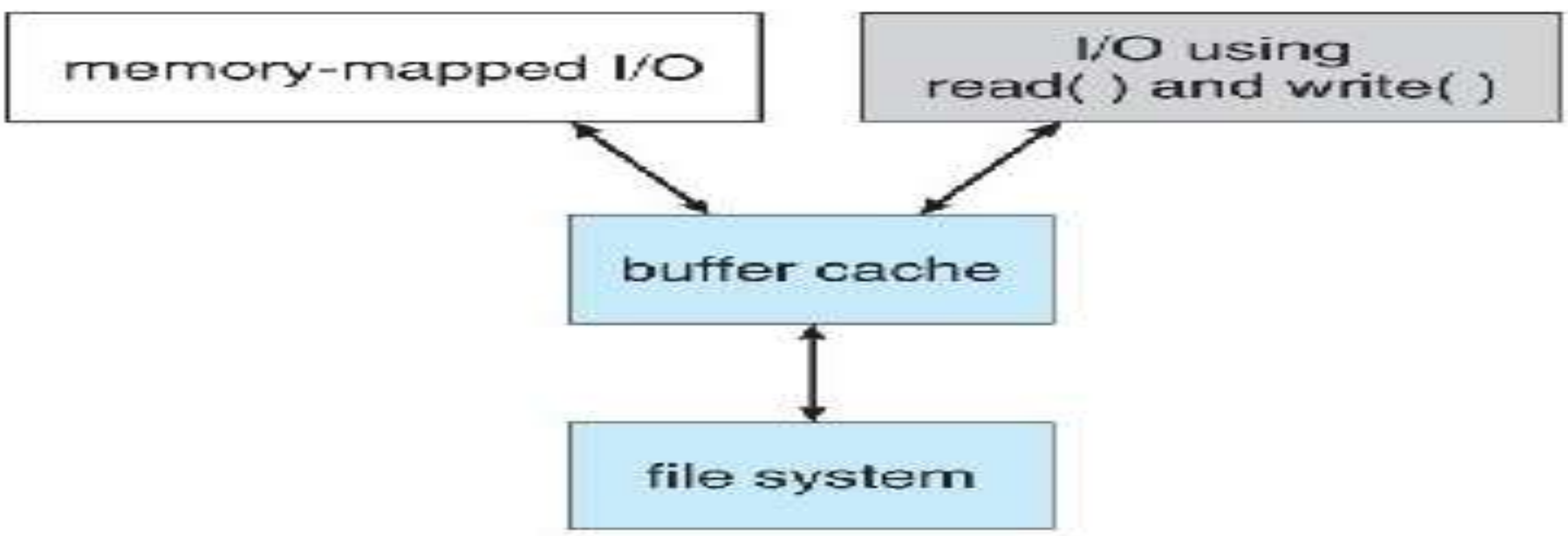
– disk controllers include local memory to form an on-board **cache that is sufficiently large to store entire** tracks at a time

– maintain a separate section of main memory for a **disk cache, where blocks are kept under the assumption that they will be used again** shortly

– Other systems cache file data using a **page cache**


memory-mapped I/O
I/O using
read( ) and write( )
page cache
buffer cache
file system

- Some versions of UNIX provide a **unified buffer cache**
- Consider the two alternatives of opening and accessing a file.
  - One approach is to use memory mapping
  - the second is to use the standard system calls read and write



- **Synchronous writes**
  - **occur in the order in which the disk subsystem** receives them, and the writes are not buffered.
  - Thus the calling routine must wait for the data to reach the disk drive before it can proceed.
- **Asynchronous writes**
  - are done the majority of the time
- **Free-behind**
  - removes a page from the buffer as soon as the next page is requested.
  - The previous pages are not likely to be used again and waste buffer space.
- **read-ahead,**
  - **a requested page and several subsequent pages** are read and cached