UNIT - V

File System Interface and Operations

- Introduction
- Access methods
- Directory Structure
- Protection
- File System Structure
- Allocation methods
- Free-space Management
- System Calls - **open, create, read, write, close, lseek, stat, ioctl**

UNIT - V
Introduction

- The file system is the most visible aspect of an operating system. It provides the mechanism for on-line storage of and access to both data and programs of the operating system.
- ***The file system consists of two distinct parts:***

1. A collection of files, each storing related data, and a directory structure which organizes
2. Provides information about all the files in the system.

UNIT - V
Introduction-File Concept

- A file is a named collection of related information that is recorded on secondary storage.
- In general, a file is a sequence of bits, bytes, lines, or records whose meaning is defined by the file's creator and user.
- Many different types of information may be stored in a file: source programs, object programs, executable programs, numeric data, text, payroll records, graphic images.
- A file has a certain defined *structure* according to its type.

UNIT - V
Introduction-File Concept

- A *source* file is a sequence of subroutines and functions, each of which is further organized as declarations followed by executable statements.
- An *object* file is a sequence of bytes organized into blocks understandable by the system's linker.
- An *executable* file is a series of code sections that the loader can bring into memory and execute.

UNIT - V
Introduction-File Attribute

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

- **Name :** The symbolic file name is the only information kept in human readable form.
- **Type :** This information is needed for those systems that support different types.
- **Location** : This information is a pointer to a device and to the location of the file on that device.

UNIT - V
Introduction-File Attribute

- **Size :** The current size of the file (in bytes, words or blocks), and possibly the maximum allowed size are included in this attribute.
- **Protection :** Access-control information controls who can do reading, writing, executing, and so on.
- **Time, date, and user identification** : This information may be kept for (1) creation, (2) last modification, and (3) last use. These data can be useful for protection, security, and usage monitoring.

UNIT - V
Introduction-File Operations

- To define a file properly, we need to consider the operations that can be performed on files.
- The operating system provides system calls to create, write, read, reposition, delete, and truncate files.
- **Creating a file :** Two steps are necessary to create a file. First, space in the file system must be found for the file. The directory entry records the name of the file and the location in the file system.

UNIT - V
Introduction-File Operations

- **Writing a file :** To write a file, we make a system call specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the location of the file. The system must keep a *write* pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.

UNIT - V
Introduction-File Operations

- **Reading a file** : To read from a file, we use a system call that specifies the name of the file and where (in memory) the next block of the file should be put. Again, the directory is searched for the associated directory entry, and the system needs to keep a ***read*** **pointer** to the location in the file where the next read is to take place. Once the read has taken place, the **read pointer** is updated. Since, in general, a file is either being read or written, most systems keep only one ***current-file-position*** **pointer**. Both the read and write operations use this same pointer, saving space and reducing the system complexity.

UNIT - V
Introduction-File Operations

- **Repositioning within a file :** The directory is searched for the appropriate entry, and the current-file-position is set to a given value. Repositioning within a file does not need to involve any actual I/O. This file operation is also known as a file *seek*.
- **Deleting a file :** To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space (so that it can be reused by other files) and erase the directory entry.

UNIT - V
Introduction-File Operations

- **Truncating a file :** There are occasions when the user wants the attributes of a file to remain the same, but wants to erase the contents of the file. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged (except for file length) but for the file to be reset to length zero.

UNIT - V
Introduction- File Types

- One major consideration in designing a file system/ and the entire operating system, is whether the operating system should recognize and support file types. If an operating system recognizes the type of *a* file, it can operate on the file in reasonable ways.

- A common technique for implementing file types is to include the type as part of the file name. The name is split into two parts - a name and an *extension,* usually separated by a period character.

UNIT - V
Introduction- File Types

File type	Usual extension	Function
Executable	exe, com, bin or none	ready-to-run machine-language program
Object	obj, o	compiled, machine language, not linked
Source code	c, p, pas, f77, asm, a	source code in various languages
Batch	bat, sh	commands to the command interpreter
Text	txt, doc	textual data, documents
Wordprocessor	wp, tex, rrf, etc	various word-processor formats
Library	lib, a	libraries of routines for programmers
Print or view	ps, dvi, gif	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

UNIT - V
Introduction - File Structure

- File types also may be used to indicate the internal structure of the file. Source and Object files have structures that match the expectations of the programs that read them.

- Further, certain files must conform to a required structure that is understood by the operating system.

UNIT - V
ACCESS METHODS

- There are several ways that the information in the file can be accessed. Some systems provide only one access method for files.
- On other systems are suppose different access methods. *There are **three ways to access a file** into a computer system:*
  1. Sequential-Access
  2. Direct Access
  3. Index sequential Method

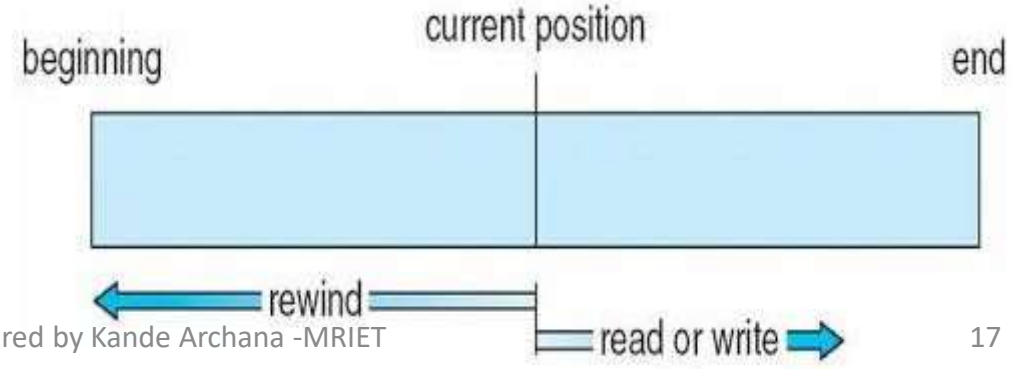
UNIT - V
ACCESS METHODS- Sequential Access

- It is the simplest access method. Information in the file is processed in order, one record after the other. This mode of access is by far the most common.
- Read and write make up the bulk of the operation on a file. A read operation -*read next*- read the next position of the file and automatically advance a file pointer, which keeps track I/O location.
- Similarly, for the write *write next* append to the end of the file and advance to the newly written on data.

UNIT - V
ACCESS METHODS- Sequential Access

Key points

- Data is accessed one record right after another record in an order.
- When we use read command, it move ahead pointer by one.
- When we use write command, it will allocate memory and move the pointer to the end of the file Such a method is reasonable for tape.



UNIT - V
ACCESS METHODS- Direct-Access

- Another method is *direct access method* also known as *relative access method*.
- A filed-length logical record that allows the program to read and write record rapidly in no particular order.
- The direct access is based on the disk model of a file since disk allows random access to any file block.
- For direct access, the file is viewed as a numbered sequence of block or record.

UNIT - V
ACCESS METHODS- Direct-Access

- Thus, we may read block 14 then block 59 and then we can write block 17.
- There is no restriction on the order of reading and writing for a direct access file.

Sequential access	Implementation for direct access
reset	*cp* := 0;
read next	*read cp;* *cp* := *cp*+1;
write next	*write cp;* *cp* := *cp*+1;

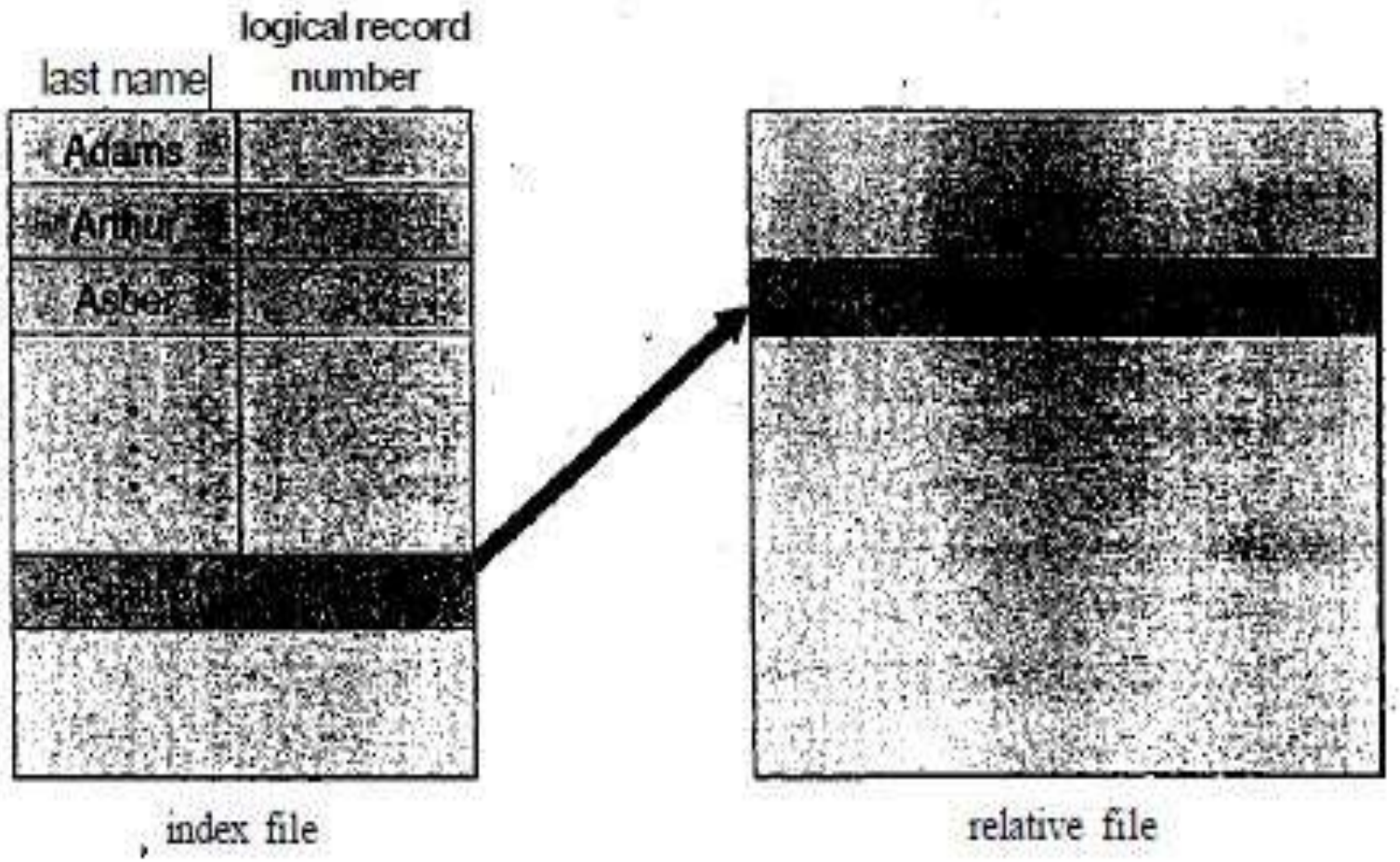
UNIT - V
ACCESS METHODS- Index sequential method

- These methods construct an index for the file. The index, like an index in the back of a book, contains the pointer to the various blocks.
- To find a record in the file, we first search the index and then by the help of pointer we access the file directly.

Key points

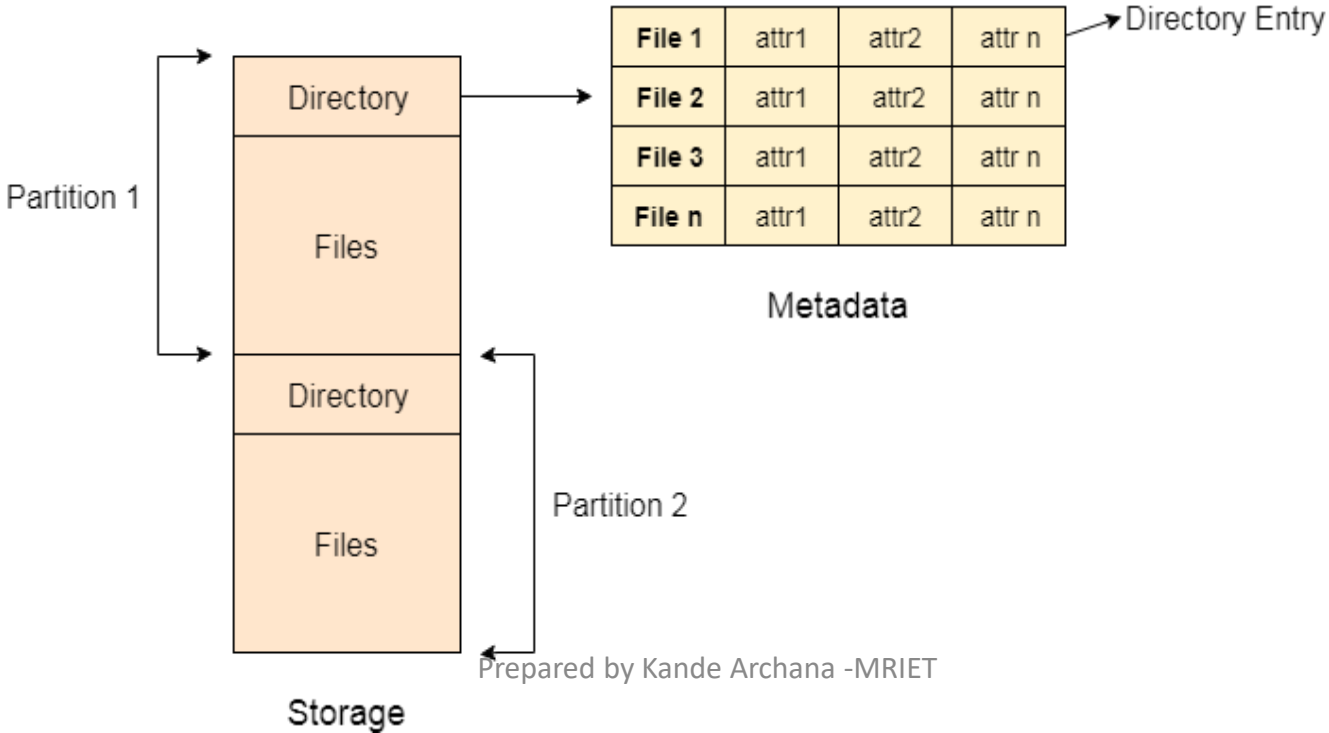
- It is built on top of Sequential access.
- It control the pointer by using index.

UNIT - V
ACCESS METHODS- Index sequential method



UNIT - V
DIRECTORY STRUCTURE

- Directory can be defined as the listing of the related files on the disk. The directory may store some or the entire file attributes.
- A directory entry is maintained for each file in the directory which stores all the information related to that file.



UNIT - V
DIRECTORY STRUCTURE

- A directory can be viewed as a file which contains the Meta data of the bunch of files. ***Every Directory supports a number of common operations on the file***:
  1. File Creation
  2. Search for the file
  3. File deletion
  4. Renaming the file
  5. Traversing Files
  6. Listing of files

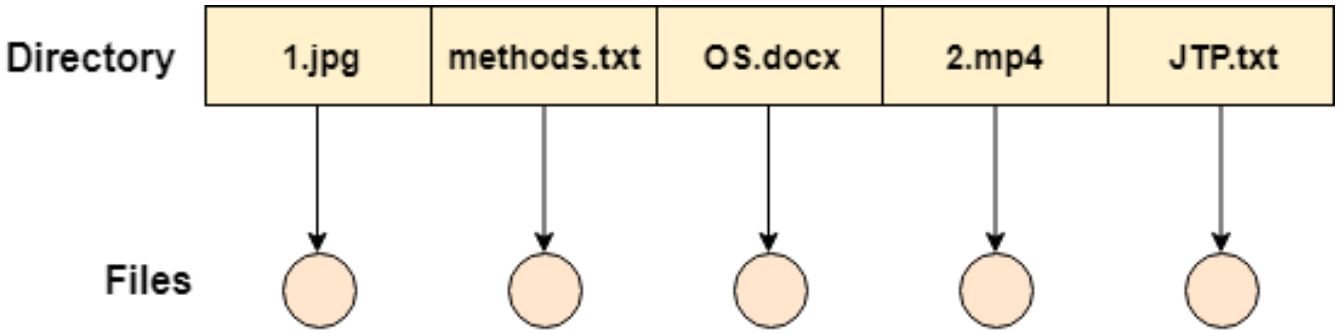
UNIT - V
DIRECTORY STRUCTURE

There are most common schemes on for defining the logical structure of a directory

1. Single Level Directory
2. Two Level Directory
3. Tree Structured Directory
4. Acyclic-Graph Structured Directories
5. General Graph Directory

UNIT - V
DIRECTORY STRUCTURE- Single Level Directory

- The simplest directory structure is the single-level directory. All files are contained in the same directory, which is easy to support and understand.
- A single-level directory has significant limitations, however, when the number of files increases or when there is more than one user.



Single Level Directory

UNIT - V
DIRECTORY STRUCTURE- Single Level Directory

Advantages

- Implementation is very simple.
- If the sizes of the files are very small then the searching becomes faster.
- File creation, searching, deletion is very simple .

Disadvantages

- We cannot have two files with the same name.
- The directory may be very big therefore searching for a file may take too much time.
- Protection cannot be implemented for multiple users.
- There are no ways to group same kind of files.

UNIT - V
DIRECTORY STRUCTURE- Two Level Directory

- In the two-level directory structure, each user has her own user file directory (UFD).
- Each UFD has a similar structure, but lists only the files of a single user.
- When a user job starts or a user logs in, the system's master file directory (MFD) is searched.
- The master file directory is indexed by user name or account number, and each entry points to the UFD for that user.
- When a user refers to a particular file, only his own UFD is searched.

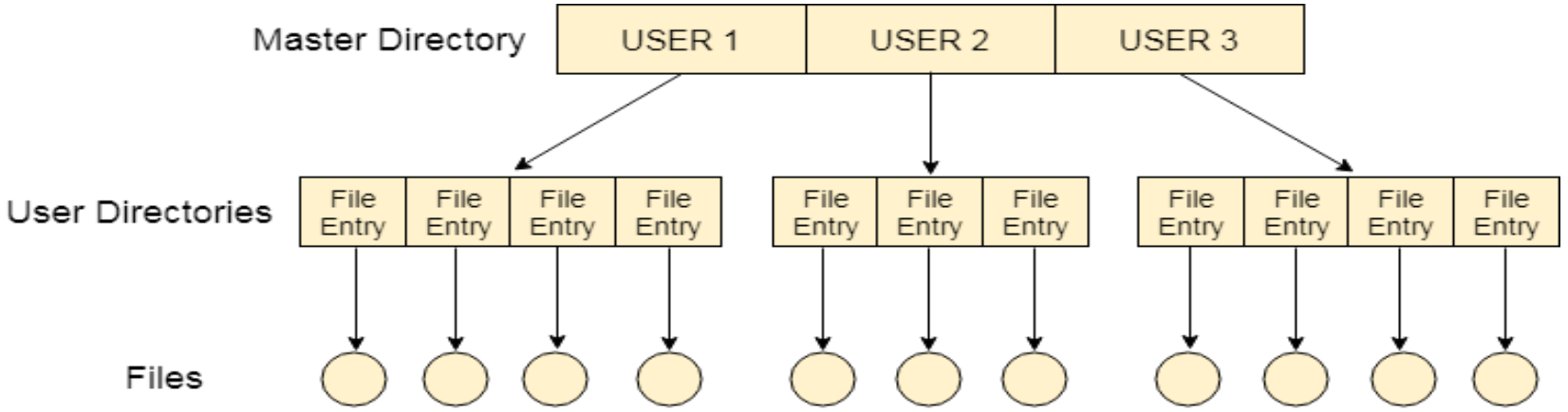
UNIT - V
DIRECTORY STRUCTURE- Two Level Directory

- In the two-level directory structure, each user has her own user file directory (UFD).
- Each UFD has a similar structure, but lists only the files of a single user.
- When a user job starts or a user logs in, the system's master file directory (MFD) is searched.
- The master file directory is indexed by user name or account number, and each entry points to the UFD for that user.
- When a user refers to a particular file, only his own UFD is searched.

UNIT - V

DIRECTORY STRUCTURE- Two Level Directory

- The user directories themselves must be created and deleted as necessary. The program creates a new user file directory and adds an entry for it to the master file directory.
- The execution of this program might be restricted to system administrators.



Two Level Directory

UNIT - V
DIRECTORY STRUCTURE- Two Level Directory

Characteristics of two level directory system

- Each files has a path name as ***/User-name/directory-name/***
- Different users can have the same file name.
- Searching becomes more efficient as only one user's list needs to be traversed.

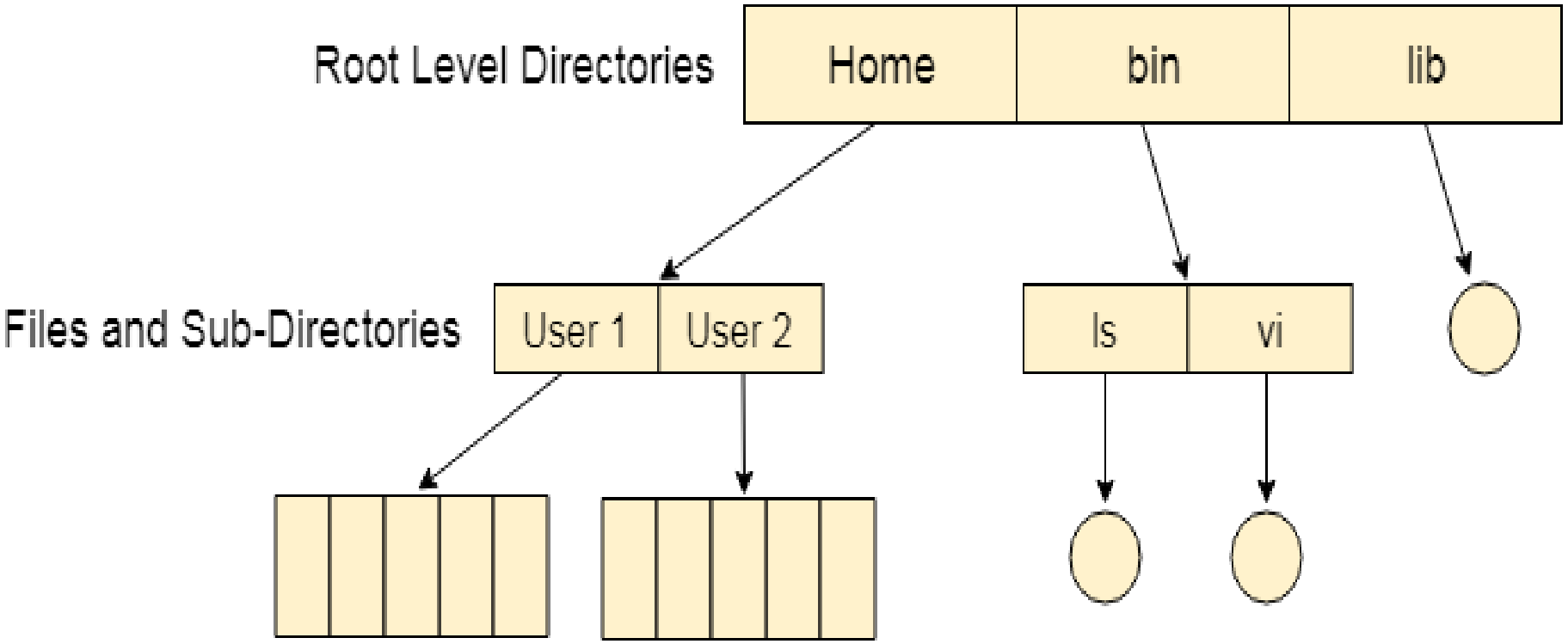
UNIT - V
DIRECTORY STRUCTURE-Tree Structured Directory

- A tree is the most common directory structure. The tree has a root directory. Every file in the system has a unique path name.
- A path name is the path from the root, through all the sub-directories, to a specified file.
- All directories have the same internal format. One bit in each directory entry defines the entry as a file(0) or as sub-directory (1). Special system calls are used to create and delete directories.

UNIT - V
DIRECTORY STRUCTURE-Tree Structured Directory

- The current directory should contain most of the files that are of current interest to the user.
- If a file is needed that is not in the current directory, then the user must either specify a path name or change the current directory to be the directory holding that file.
- To change the current directory to a different directory, a system call is provided that takes a directory name as a parameter and uses it to redefine the current directory.

UNIT - V
DIRECTORY STRUCTURE-Tree Structured Directory



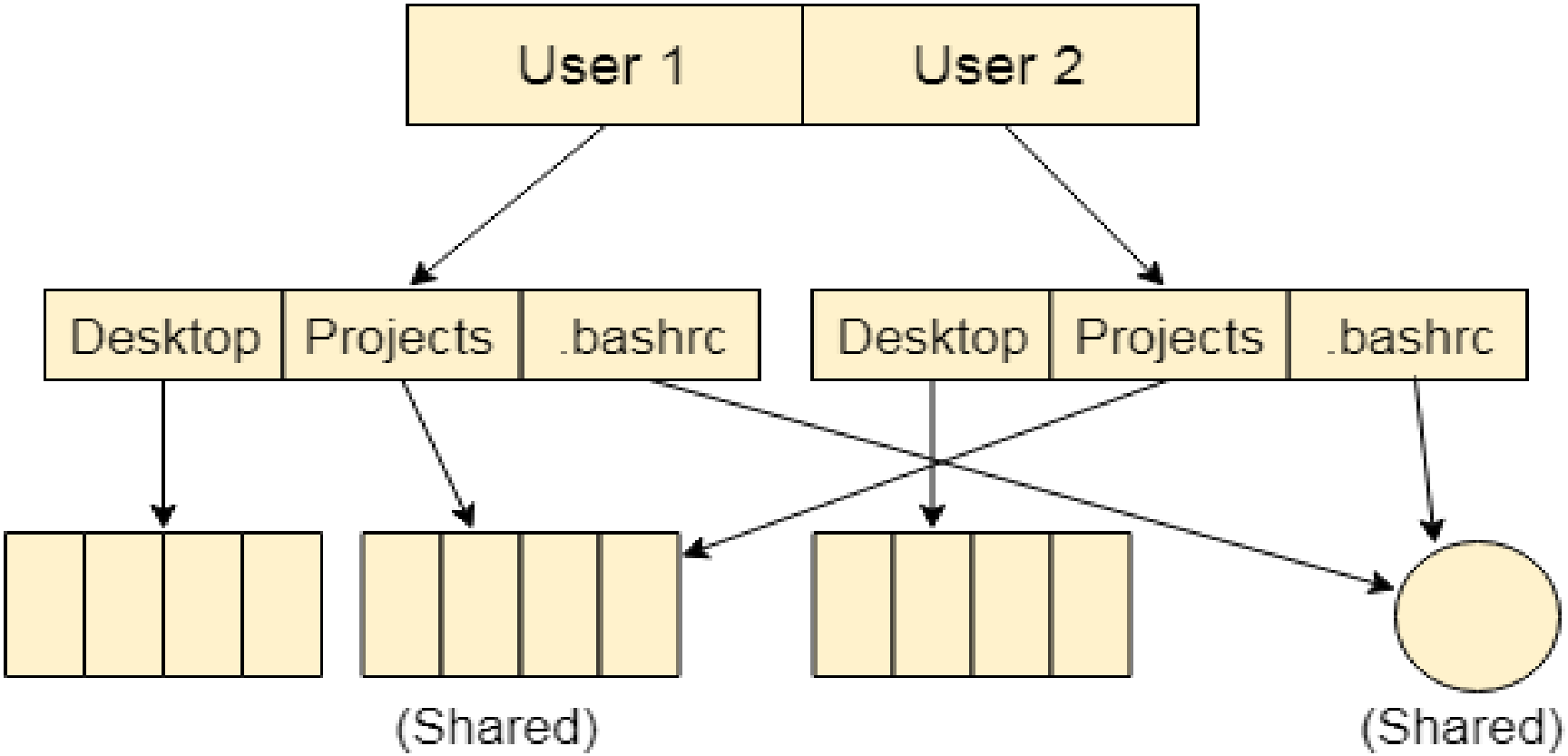
The Structured Directory System

UNIT - V
DIRECTORY STRUCTURE- Acyclic-Graph Structured Directories

- The tree structured directory system doesn't allow the same file to exist in multiple directories therefore sharing is major concern in tree structured directory system.
- We can provide sharing by making the directory an acyclic graph.
- In this system, two or more directory entry can point to the same file or sub directory.
- That file or sub directory is shared between the two directory entries.

UNIT - V

DIRECTORY STRUCTURE- Acyclic-Graph Structured Directories

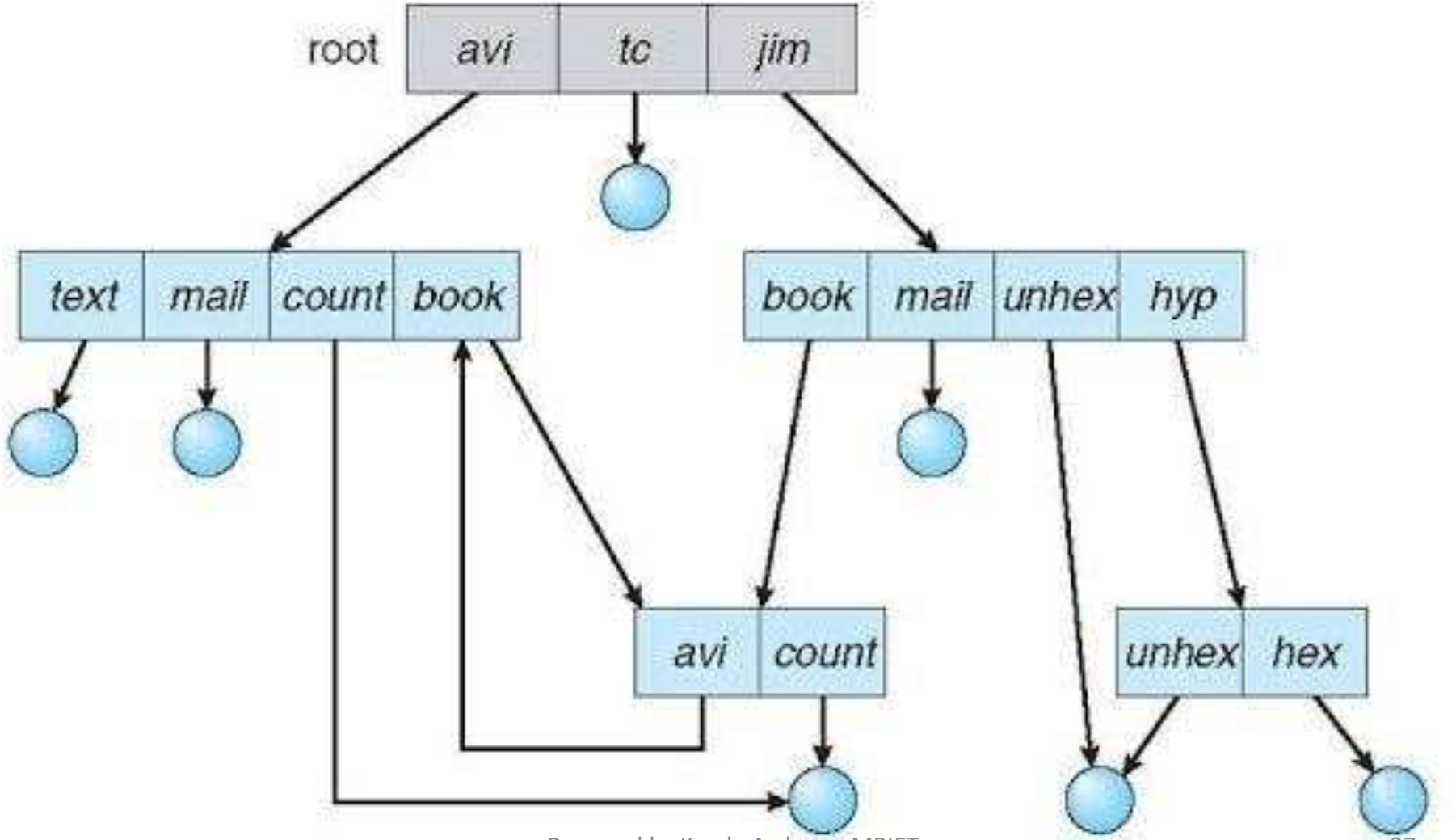


Acyclic-Graph Structured Directory System

UNIT - V
DIRECTORY STRUCTURE- General Graph Directory

- One serious problem with using an acyclic graph structure is ensuring that there are no cycles.
- If we start with a two-level directory and allow users to create subdirectories, a tree-structured directory results.
- It should be fairly easy to see that simply adding new files and subdirectories to an existing tree-structured directory preserves the tree-structured nature.

UNIT - V
DIRECTORY STRUCTURE- General Graph Directory



UNIT - V
PROTECTION

- When information is kept in a computer system, a major concern is its protection from both physical damage *(reliability)* and improper access *(protection).*
- **Reliability** is generally provided by duplicate copies of files. Many computers have systems programs that automatically (or through computer-operator intervention) copy disk files to tape at regular intervals (once per day or week or month) to maintain a copy should a file system be accidentally destroyed.

UNIT - V
PROTECTION

- **Protection** can be provided in many ways. For a small single-user system, we might provide protection by physically removing the floppy disks and locking them in a desk drawer or file cabinet. In a multiuser system, however, other mechanisms are needed.

UNIT - V
PROTECTION

Need of Protection:

- To prevent the access of unauthorized users and
- To ensure that each active programs or processes in the system uses resources only as the stated policy.
- To improve reliability by detecting latent errors.

UNIT - V
PROTECTION

Role of Protection

- The role of protection is to provide a mechanism that implement policies which defines the uses of resources in the computer system.
- Some policies are defined at the time of design of the system, some are designed by management of the system and some are defined by the users of the system to protect their own files and programs.
- Policy is different from mechanism. Mechanisms determine how something will be done and policies determine what will be done. Policies are changed over time and place to place. Separation of mechanism and policy is important for the flexibility of the system.

UNIT - V
PROTECTION - Types of Access

Protection mechanisms provide controlled access by limiting the types of file access that can be made. Access is permitted or denied depending on several factors, one of which is the type of access requested.

1. **Read :** Read from the file.
2. **Write :** Write or rewrite the file.
3. **Execute :** Load the file into memory and execute it.
4. **Append** : Write new information at the end of the file.
5. **Delete :** Delete the file and free its space for possible reuse.
6. **List** : List the name and attributes of the file

UNIT - V

PROTECTION - Access Lists and Groups

- When a user requests access to particular file the operating system checks access list associated with that file. If that user is listed for the requested access, the access is allowed. Otherwise, a protection violation occurs.

1. **Owner** : The user who created the file is the owner.
2. **Group** : A set of users who are sharing the file and need similar access is a group, or workgroup.
3. **Universe :** All other users in the system constitute the universe.

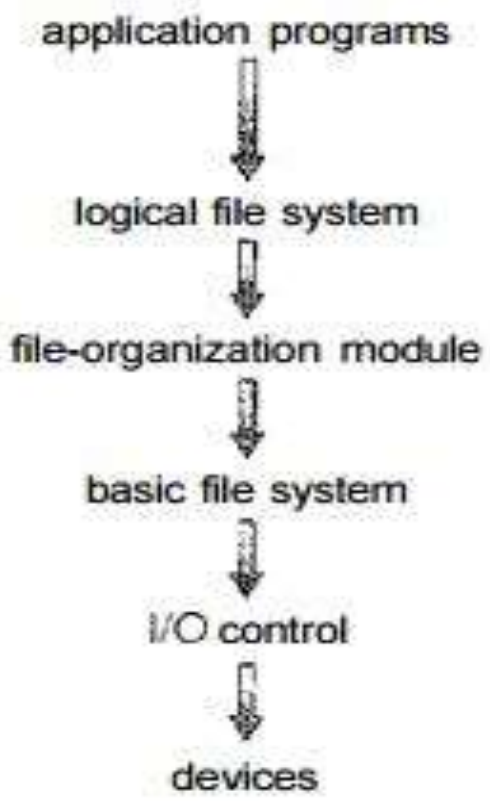
UNIT - V FILE SYSTEM STRUCTURE

- File System provide efficient access to the disk by allowing data to be stored, located and retrieved in a convenient way. A file System must be able to store the file, locate the file and retrieve the file.

- Most of the Operating Systems use layering approach for every task including file systems. Every layer of the file system is responsible for some activities.

UNIT - V FILE SYSTEM STRUCTURE

- The file system is divided in different layers, and also the functionality of each layer.



UNIT - V FILE SYSTEM STRUCTURE

- When an **application program** asks for a file, the first request is directed to the logical file system. The logical file system contains the Meta data of the file and directory structure.
- Generally, files are divided into various **logical blocks**. Files are to be stored in the hard disk and to be retrieved from the hard disk. Hard disk is divided into various tracks and sectors. Therefore, in order to store and retrieve the files

UNIT - V FILE SYSTEM STRUCTURE

- Once **File organization** module decided which physical block the application program needs, it passes this information to basic file system. The basic file system is responsible for issuing the commands to I/O control in order to fetch those blocks.
- **I/O controls contain** the codes by using which it can access hard disk. These codes are known as **device drivers**. I/O controls are also responsible for handling interrupts.

UNIT - V ALLOCATION METHODS

There are three file allocation methods.

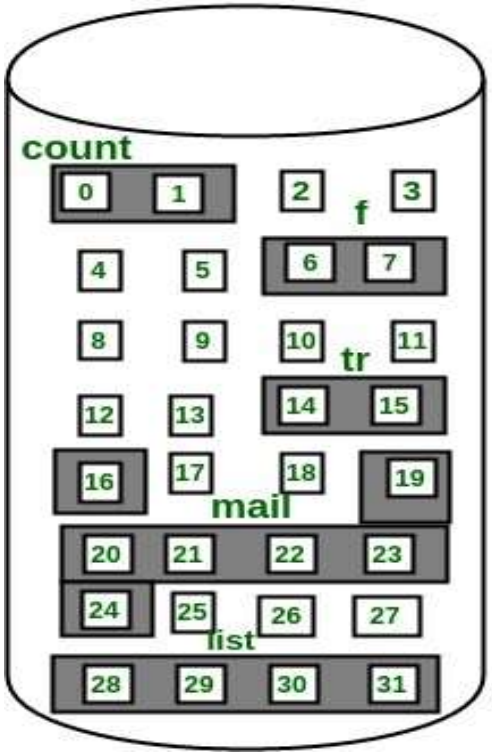
1. Contiguous Allocation
2. Linked Allocation
3. Indexed Allocation

UNIT - V
ALLOCATION METHODS-Contiguous Allocation

- In this scheme, each file occupies a contiguous set of blocks on the disk.
- For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: *b, b+1, b+2,……b+n-1.*
- This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.
- The directory entry for a file with contiguous allocation contains.
  - Address of starting block
  - Length of the allocated portion.

UNIT - V
ALLOCATION METHODS-Contiguous Allocation

- The *file 'mail'* in the following figure starts from the block 19 with length = 6 blocks. Therefore, it occupies *19, 20, 21, 22, 23, 24* blocks.



Directory

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

UNIT - V
ALLOCATION METHODS-Contiguous Allocation

Advantages:

- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

Disadvantages:

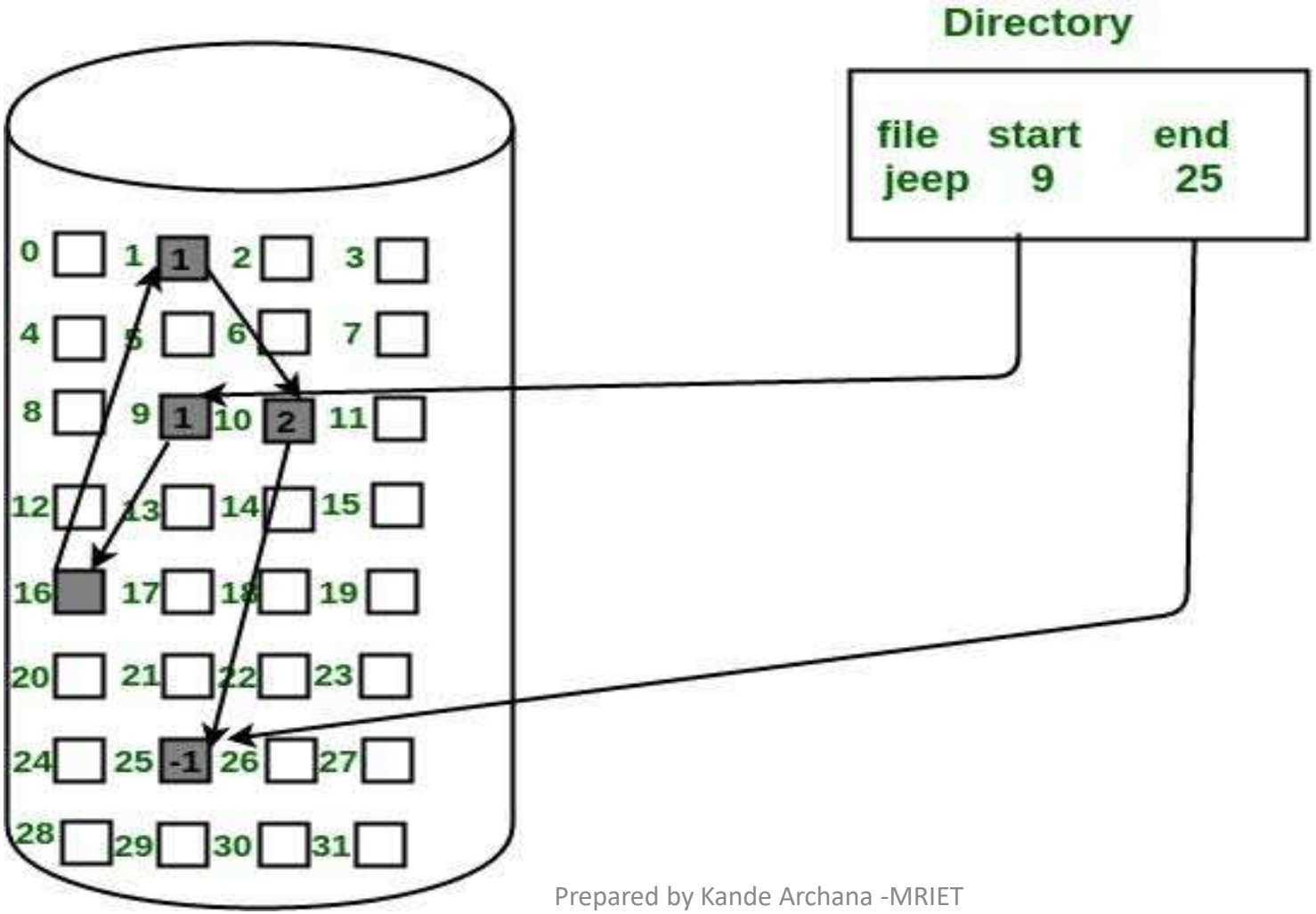
- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

UNIT - V
ALLOCATION METHODS- Linked List Allocation

- In this scheme, each file is a linked list of disk blocks which **need not be** contiguous. The disk blocks can be scattered anywhere on the disk.
- The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.
- The file ‘jeep’ in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.

UNIT - V

ALLOCATION METHODS- Linked List Allocation



UNIT - V
ALLOCATION METHODS- Linked List Allocation

Advantages:

- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

Disadvantages:

- Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
- It does not support random or direct access. We cannot directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead.

UNIT - V

ALLOCATION METHODS- Indexed Allocation

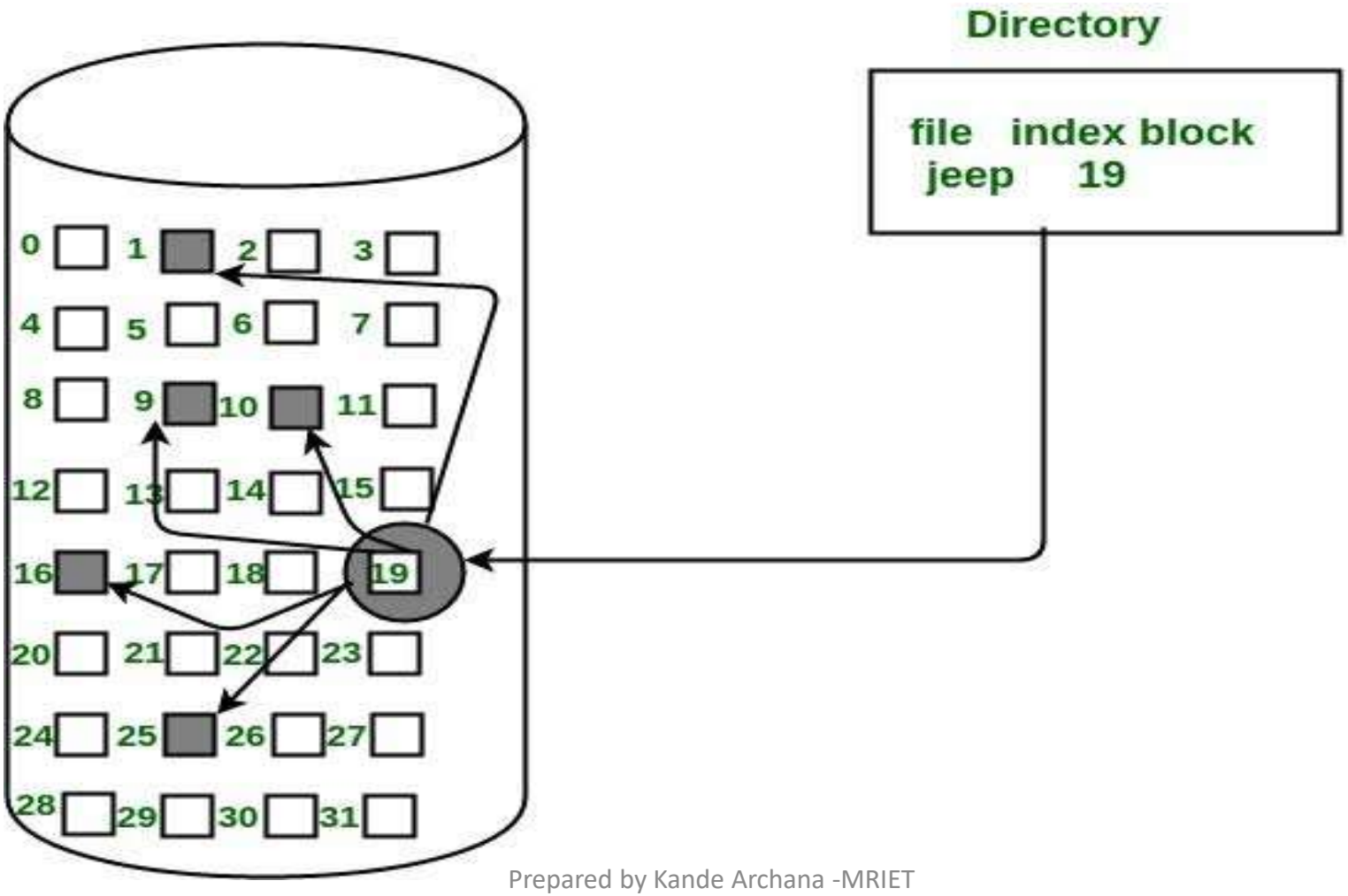
In this scheme, a special block known as the **Index block** contains the pointers to all the blocks occupied by a file.

Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block.

The directory entry contains the address of the index block as shown in the image:

UNIT - V

ALLOCATION METHODS- Indexed Allocation



UNIT - V
ALLOCATION METHODS- Indexed Allocation

Advantages:

- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

Disadvantages:

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

Free-space Management

- There is only a limited amount of disk space, it is necessary to reuse the space from deleted files for new files, if possible.
- To keep track of free disk space, the system maintains a ***free-space list***. The free-space list records all disk blocks that *are* ***free*** — those not allocated to some file or directory.
- To **create a file**, we search the **free-space list** for the required amount of space, and **allocate** that space to the **new file**.

Free Space Management - Bitmap or Bit vector

- Frequently, the **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.**
- **For 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**

00111100111111000110000011100000 ...

Free Space Management - Bitmap or Bit vector

Advantages

- The main advantage of this approach is that it is **relatively simple .**
- **Efficient** to find the first free block, or *n* free blocks on the disk.

Free Space Management - Linked-List

- Another approach 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.
- Example keep a pointer to block 2, as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on.

Free Space Management - Linked-List

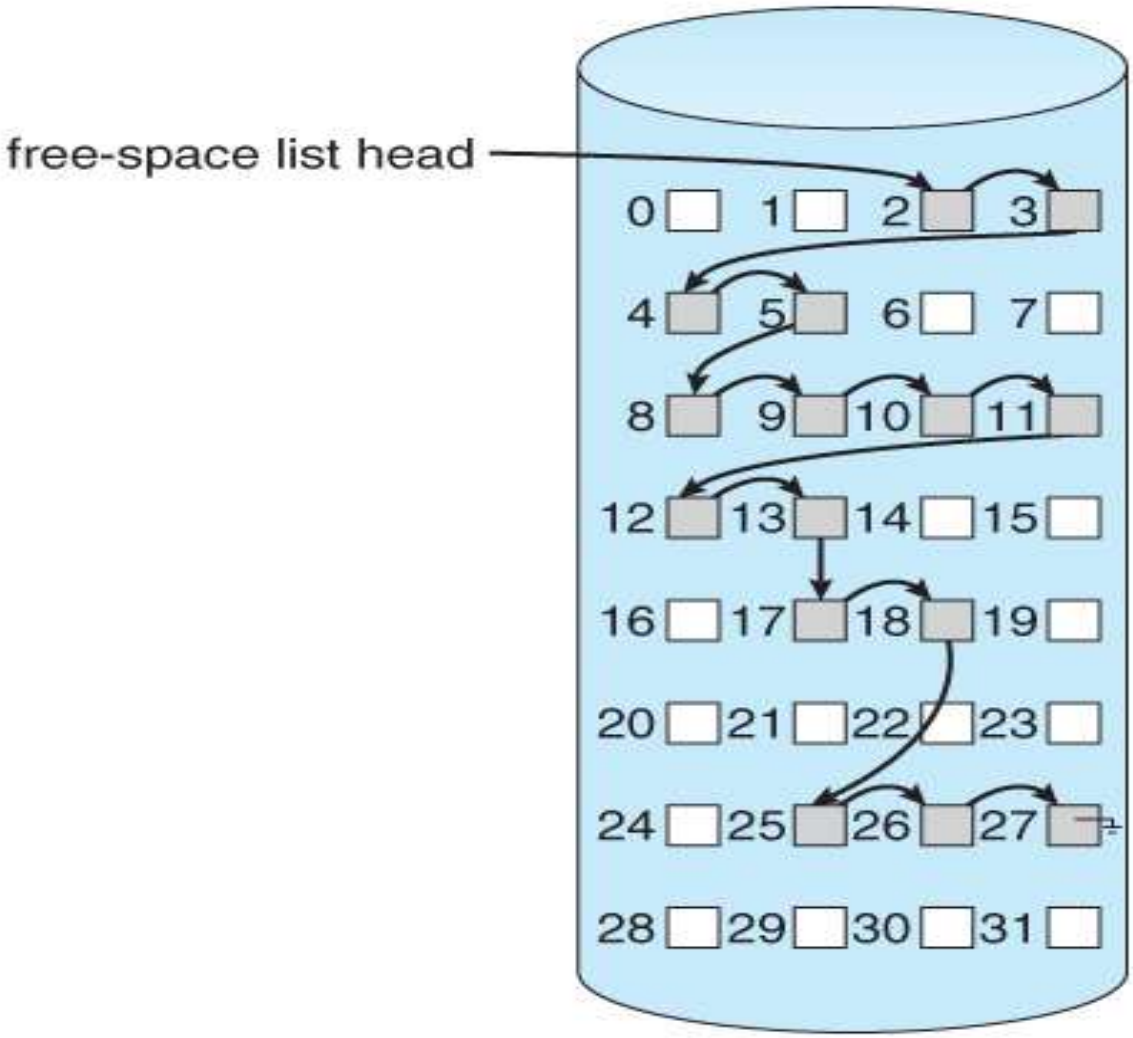


Fig: Linked free-space list on disk

Free Space Management -Grouping

- A modification of the free-list approach is 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, and so on.
- The importance of this implementation is that the addresses of a large number of free blocks can be found quickly, unlike in the standard linked-list approach.

- An **advantage** of this approach is that the addresses

Free Space Management -Grouping

- An **advantage** of this approach is that the addresses of a group of free disk blocks can be found easily.

UNIT - V
SYSTEM CALLS OF Create, Read, Write, Open ,Close, Lseek, Stat, Ioctl

- **File descriptor** : File descriptor is integer that uniquely identifies an open file of the process.
- **File Descriptor table**: File descriptor table is the collection of integer array indices that are file descriptors in which elements are pointers to file table entries.
- **File Table Entry:** File table entries is a structure In-memory surrogate for an open file, which is created when process request to opens file and these entries maintains file position.

UNIT - V
SYSTEM CALLS OF Create, Read, Write, Open ,Close, Lseek, Stat, Ioctl

- **Standard File Descriptors**: When any process starts, then that process file descriptors table's fd(file descriptor) 0, 1, 2 open automatically, (By default) each of these 3 fd references file table entry for a file named **/dev/tty.**
  - **/dev/tty**: In-memory surrogate for the terminal
  - **Terminal**: Combination keyboard/video screen

UNIT - V
SYSTEM CALLS - Create

Create: Used to Create a new empty file.

- **Syntax in C language:**

  int creat(char *filename, mode_t mode)

- **Parameter :**
  - **filename :** name of the file which you want to create
  - **mode :** indicates permissions of new file.
  - **Returns :**
    - return first unused file descriptor (generally 3 when first creat use in process beacuse 0, 1, 2 fd are reserved)
    - return -1 when error

UNIT - V
SYSTEM CALLS - Open

Open: Used to Open the file for reading, writing or both.

- **Syntax in C language**

  #include<sys/types.h>

  #includ<sys/stat.h>

  #include <fcntl.h>

  int open (const char* Path, int flags [, int mode]);

- **Parameters**
- **Path :** path to file which you want to use
  - use absolute path begin with “/”, when you are not work in same directory of file.
  - Use relative path which is only file name with extension, when you are work in same directory of file.
- **flags :** How you like to use
  - **O_RDONLY**: read only, **O_WRONLY**: write only, **O_RDWR**: read and write, **O_CREAT**: create file if it doesn’t exist, **O_EXCL**: prevent creation if it already exists

UNIT - V
SYSTEM CALLS - close

close: Tells the operating system you are done with a file descriptor and Close the file which pointed by fd.

- **Syntax in C language**

  #include <fcntl.h>

  int close(int fd);

- **Parameter**

  fd :file descriptor

- **Return**
  - **0** on success
  - **-1** on error

UNIT - V
SYSTEM CALLS - read

read: From the file indicated by the file descriptor fd, the read() function reads cnt bytes of input into the memory area indicated by buf. A successful read() updates the access time for the file.

- **Syntax in C language** size_t read (int fd, void* buf, size_t cnt);

- **Parameters**
  - **fd:** file descripter
  - **buf:** buffer to read data from
  - **cnt:** length of buffer

- **Returns: How many bytes were actually read**
  - return Number of bytes read on success
  - return 0 on reaching end of file
  - return -1 on error
  - return -1 on signal interrupt

UNIT - V
SYSTEM CALLS - write

write: Writes cnt bytes from buf to the file or socket associated with fd. cnt should not be greater than INT_MAX (defined in the limits.h header file). If cnt is zero, write() simply returns 0 without attempting any other action.

- **Syntax in C language:**
  - size_t write (int fd, void* buf, size_t cnt);

- **Parameters**
  - **fd:** file descripter
  - **buf:** buffer to write data to
  - **cnt:** length of buffer

- **Returns: How many bytes were actually written**
  - return Number of bytes written on success
  - return 0 on reaching end of file
  - return -1 on error
  - return -1 on signal interrupt

UNIT - V
SYSTEM CALLS - lseek

lseek (C System Call): lseek is a system call that is used to change the location of the read/write pointer of a file descriptor. The location can be set either in absolute or relative terms.

- **Function Definition**

  off_t lseek(int fildes, off_t offset, int whence);

- **FieldDescription**
  int fildes : The file descriptor of the pointer that is going to be moved
  off_t offset : The offset of the pointer (measured in bytes).
  int whence : The method in which the offset is to be interpreted (rela, absolute, etc.). Legal value r this variable are provided at the end.
  return value : Returns the offset of the pointer (in bytes) from the beginning of the file. If the return value is -1, then there was an error moving the pointer.

UNIT - V
SYSTEM CALLS - lseek

lseek (C System Call): lseek is a system call that is used to change the location of the read/write pointer of a file descriptor. The location can be set either in absolute or relative terms.

- **Function Definition**

  off_t lseek(int fildes, off_t offset, int whence);

- **FieldDescription**
  int fildes : The file descriptor of the pointer that is going to be moved
  off_t offset : The offset of the pointer (measured in bytes).
  int whence : The method in which the offset is to be interpreted (rela, absolute, etc.). Legal value r this variable are provided at the end.
  return value : Returns the offset of the pointer (in bytes) from the beginning of the file. If the return value is -1, then there was an error moving the pointer.

UNIT - V
SYSTEM CALLS -Stat

Stat : System calls provided by the linux kernel are exposed in the C programming language via glibc. When a system call is used, you are communicating to the OS and on return the OS communicates to you through the parameters that are returned to system call functions (return values).

Stat System Call

- Stat system call is a system call in Linux to check the status of a file such as to check when the file was accessed. The stat() system call actually returns file attributes. The file attributes of an inode are basically returned by Stat() function. An inode contains the metadata of the file. An inode contains: the type of the file, the size of the file, when the file was accessed (modified, deleted) that is time stamps, and the path of the file, the user ID and the group ID, links of the file, and physical address of file content.

UNIT - V
SYSTEM CALLS -Stat

Syntax of C Stat system call

- To use the stat system call in C programming language, you have to include the following header file:

  #include <sys/stat.h>

- Stat is used to get the status of a file. The syntax of C stat system call may not be same for every operating system. In Linux the syntax for stat system call is as follows:
- **int stat(const char *path, struct stat *buf)**
- The return type of the function in int, if the function is executed successfully, 0
- is returned if there are any errors, -1 will be returned.