MODULE-5

FILE SYSTEM MANAGEMENT

INTRODUCTION

The file system is the most visible aspect of an operating system. It provides the mechanism for online storage of and access to both data and programs of the operating system and all the users of the computer system.

The file consists of 2 parts:

- A collection of files- each storing related data.
- A directory structure- which organizes and provides information all the files in the system.

FILE CONCEPTS

- Computers can store information on various storage media, such as magnetic disks, magnetic tapes and optical disks.
- So that the 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.
- A file has a certain defined structure which depends on its type.
- A text file is a sequence of characters organized into lines.
- A source file is a sequence of subroutines and functions, each of which is 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 section that the loader can bring into memory and execute.

FILE ATTRIBUTES

A file is named, for the convenience of its human users, and is referred to by its name. A name is usually a string of characters. Some systems differentiate between uppercase and lowercase characters in names, whereas other systems do not.

When a file is named, it becomes independent of the process, the user, and even the system that created it. For instance, one user might create the file example.txt, and another user might edit that file by specifying its name. The file's owner might write the file to a floppy disk, send it in an e-mail, or copy it across a network, and it could still be called example.txt.

A file's attributes vary from one operating system to another but typically consist of these:

Name-The symbolic file name is the only information kept in human-readable form.

Identifier- This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.

Type-This information is needed for systems that support different types of files.

Location-This information is a pointer to a device and to the location of the file on that device.

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 determines who can do reading writing, executing, and so on.

Time, date, and user identification-This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

FILE OPERATIONS

A file is an abstract data type. To define a file properly, we need to consider the operations that can be performed on files. The operating system can provide 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.

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 file's location. next write is to take place. The write pointer must be updated whenever a write occurs. The system must keep a write pointer to the location in the file where the is stored.

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 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.

Repositioning within a file-The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value. Repositioning within a file need not 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.

Truncating a file-The user may want to erase the contents of a file but keep its attributes. 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 lets the file be reset to length zero and its file space released.

FILE IMPLEMENTION

On disk, the file system may contain information about how to boot an operating system stored there, the total number of blocks, the number and location of free blocks, the directory structure, and individual files.

A boot control block (per volume)- can contain information needed by the system to boot an operating system from that volume. If the disk does not contain an operating system, this block can be empty. It is typically the first block of a volume. In UFS, it is called the boot block; in NTFS, it is the partition boot sector.

A volume control block (per volume)- contains volume (or partition) details, such as the number of blocks in the partition, the size of the blocks a free-block count and free-block pointers, and a free-FCB count and FCB pointers. In UFS, this is called a superblock; in NTFS, it is stored in the master file table.

A directory structure (per file system) -is used to organize the files. In UFS, this includes file names and associated inode numbers. In NTFS, it is stored in the master file table.

A per-file FCB-contains many details about the file. It has a unique identifier number to allow association with a directory entry. In NTFS, this information is actually stored within the master file table, which uses a relational database structure, with a row per file.

The in-memory information- is used for both file-system management and performance ance impro improvement imp via caching. The data are loaded at mount time, updated during file-system operations, and discarded at dismount. Several types of structures may be included.

An in-memory mount table contains information about each mounted volume.

- An in-memory directory-structure cache holds the directory information of recently accessed directories. (For directories at which volumes are mounted, it can contain a pointer to the volume table.)
- The system-wide open-file table contains a copy of the FCB of each open file, as well as other information.
- The per-process open-file table contains a pointer to the appropriate entry in the system-wide open-file table, as well as other information.
- Buffers hold file-system blocks when they are being read from disk or written to disk.

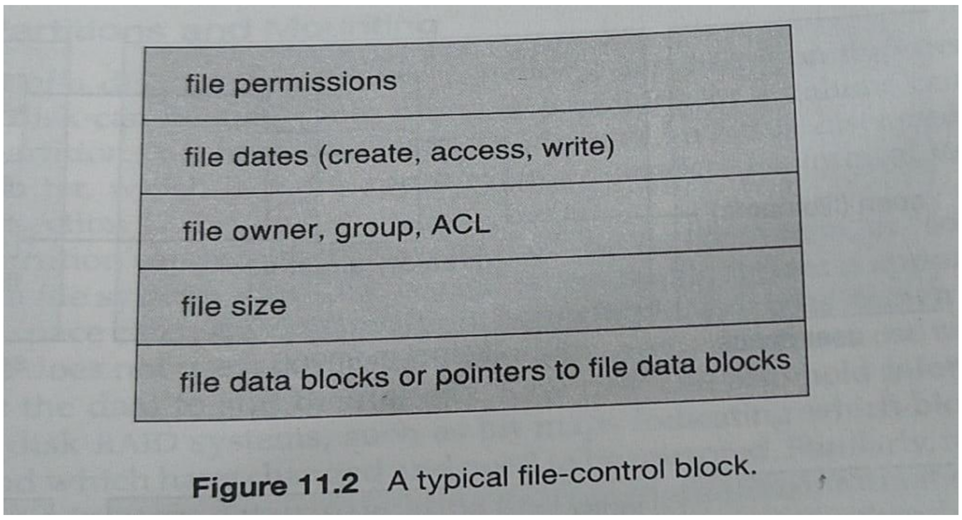

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

Figure 11.2 A typical file-control block.

FILE SYSTEM STRUCTURE

Disks provide the bulk of secondary storage on which a file system is maintained. They have 2 characteristics that make convenient medium for storing multiple files.

1. A disk can be rewritten in place, it is possible to read a block from the disk, modify the block, and write it back into the same place.
2. A disk can access directly any block of information it contains. Thus, it is simple to access any file either sequentially or randomly and switching from one file to another requires only moving the read-write heads and waiting for the disks to rotate.

- To improve I/O efficiency, I/O transfers between memory and disk are performed in units of blocks.
- Each block has one or more sectors.
- Sector size varies from 32bytes to 4096 bytes, the usual size is 512 bytes.

Layered structure of file system.

The file system is generally composed of many different levels. Each level in the design uses the features of lower levels to create new features for use by higher levels.

$$\begin{tikzcd}
\text{Application programs} \arrow[d] \\
\text{Logical file system} \arrow[d] \\
\text{File- organization module} \arrow[d] \\
\text{Basic file system} \arrow[d] \\
\text{I/O control} \arrow[d] \\
\text{Devices}
\end{tikzcd}$$

- ➢ The lowest level, the I/O control, consists of devices drivers and interrupt handlers to transfer information between the main memory and the disk system.
- ➢ The basic file system needs only to issue generic commands to the appropriate device drivers to read and write physical blocks on the disk.
    - Each physical block is identified by its numeric disk addresses
    - This layer also manages memory buffers and caches that hold various file system, directory and data blocks.
- ➢ The file-organization module knows bout files and their logical blocks, as well as physical blocks.
    - Each file’s logical blocks are numbered from 0 through N.
    - It includes the free space manager, which tracks unallocated blocks and provides these blocks to the file-organization module when requested.
- ➢ The logical file system manages the metadata information.
    - It includes all of the file system structure except the actual data.
    - It manages the directory structure to provide the file organization module with the information given a symbolic file name.

<u>VIRTUAL FILE SYSTEM</u>

Most operating system allow multiple types of file systems to be integrated into a directory structure. Users can access files that are contained within multiple file systems on local disk or even on file systems available across the network.

Data structures and procedures are used to isolate the basic system-call functionality from the implementation details. Thus, the file-system implementation consists of three major layers.

- The first layer is the file-system interface, based on the open(), read(), write(), and close() calls and on file descriptors.
- The second layer is called the virtual file system (VFS) layer.
- The third layer is local file system.

The VPS layer serves two important functions

1. It separates file-system-generic operations from their implementation by defining a clean VFS interface. Several implementations for the VIS interface may coexist on the same machine, allowing transparent access to different types of file systems mounted locally.

2. It provides a mechanism for uniquely representing a file throughout a network. The VFS is based on a file-representation structure, called a vnode, that contains a numerical designator for a network-wide unique file.

The four main object types defined by the Linux VFS are:

* The inode object, which represents an individual file

* The file object, which represents an open file

* The superblock object, which represents an entire file system

* The dentry object, which represents an individual directory entry

For each of these four object types, the VFS defines a set of operations that must be implemented. Every object of one of these types contains a pointer to a function table.

*int open (. . .) ----- open a file

*ssize_t read (. . .)----- read from a file

*ssize_t write(. . .) -------write to a file.

Int mmap(. . .) ------ memory map a file.

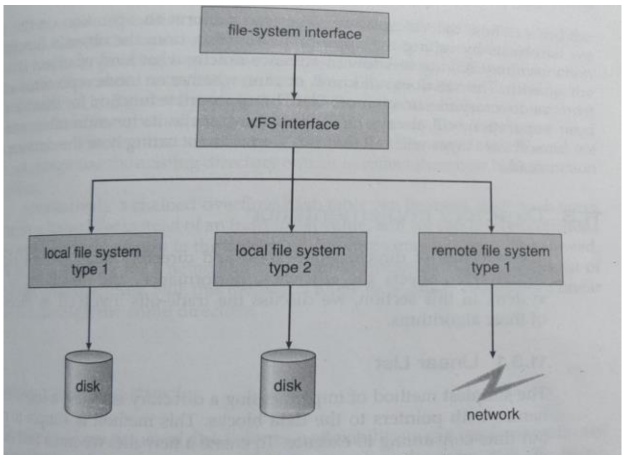


Figure 11.4 Schematic view of a virtual file system.

ALLOCATION METHODS IN FILE SYSTEM

The direct-access nature of disks allows us flexibility in the implementation of files. In almost every case, many files are stored on the same disk. The main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly.

Types:

Contiguous Allocation

Linked Allocation

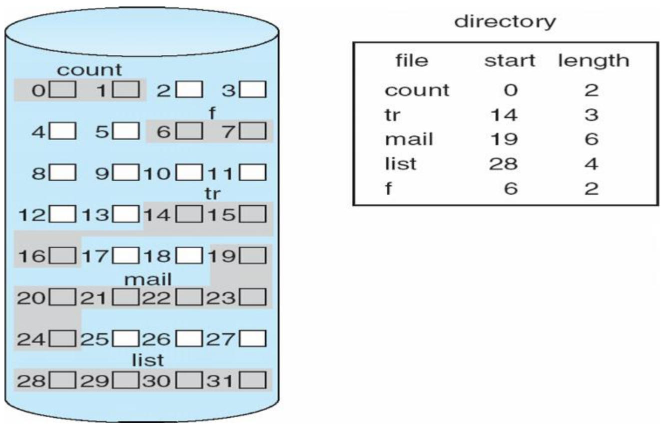
Indexed Allocation

Contiguous Allocation:

It requires that each file occupy a set of contiguous blocks on the disk. Disk addresses defines a linear ordering on the disk.

Assume that only one job is accessing the disk, accessing block b+1 after block b normally requires no head movement. When head movement is needed, the head need only move from one track to the next. Thus, the number of disks seeks required for accessing contiguously allocated files is minimal.

It is defined by the disk address and length 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.



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

Drawbacks:

- One difficulty is finding space for a new file.
- Problem faced in dynamic storage allocation.

As files are allocated and deleted, the free disk space is broken into little pieces. External fragmentation exists whenever free space is broken into chunks. It becomes a problem when the largest contiguous chunk is insufficient for a request, storage is fragmented into a number of holes, none of which is large enough to store the data.

Linked Allocation:

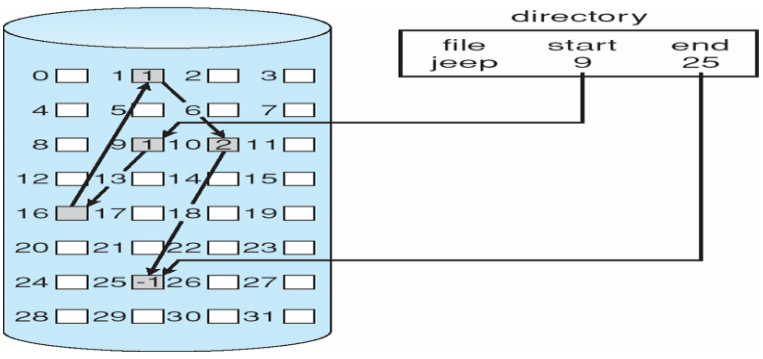
It solves all the problems of contiguous allocation. With linked allocation each file is linked list of disk blocks, the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of a file.

Example:

A file of 5 blocks might start at block9 and continue at block 16, then block 1, then block 10 and finally block 25.

Each block contains a pointer to the next block. These pointers are not made available to the user. Thus, if each block is 512 bytes in size and a disk address requires 4 bytes, then the user sees blocks of 508 bytes.

To create a new file, we simply create a new entry in the directory. With linked allocation, each directory entry has a pointer to the first disk block of the file. This pointer is initialized to nil to signify an empty file.



Drawback:

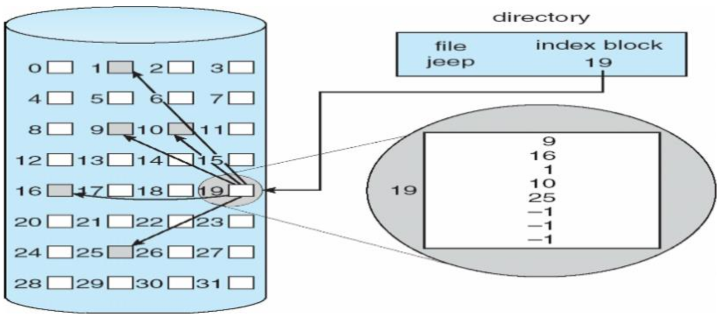
- It can be effectively used for sequential access files.
- To find the ith block of file, we must start at the beginning of that file and follow the pointer until we get to the ith block.
- More space required by pointer, if a pointer requires 4 bytes out of 512byte block, then 0.78% of the disk is being used by the pointer, rather than for information.

Indexed Allocation:

Linked allocation solves the external fragmentation and size-declaration problems of contiguous allocation. However, in the absence of a FAT, linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered with the blocks all over the disk and must be retrieved in order. Indexed allocation solves this problem by bringing all the pointers in order.

Each file has its own index blocks, 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.

When the file is created, all pointers in the index block are set to nil. When the ith block is first written, a block is obtained from the free- space manager, and its address is put in the ith index block entry.



FREE SPACE MANAGEMENT IN FILE SYSTEM.

Since disk space is limited, we need to reuse the space from deleted files for new files, if possible. (Write-once optical disks only allow one write to any given sector, and thus such reuse is not physically possible.) 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. To create a file, we search the free-space list for the required amount of space and allocate that space to the new file. This space is then removed from the free-space list. When a file is deleted, its disk space is added to the free-space list.

Types

- Bit Vector
- Linked List
- Grouping
- Counting
- Space maps

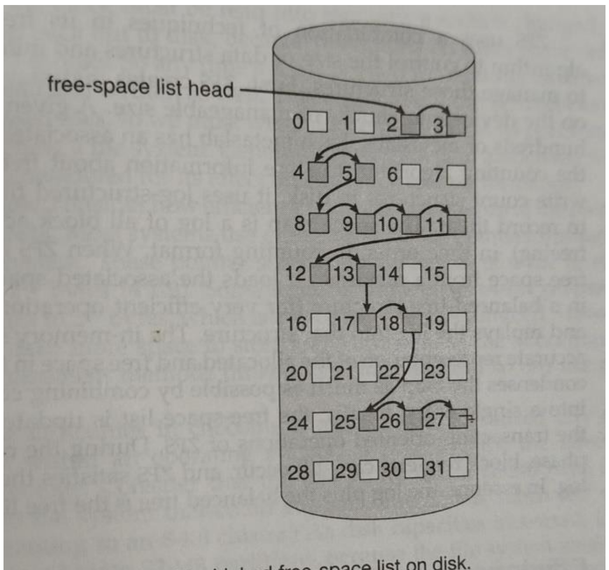


Figure 11.10 Linked free-space list on disk.

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
001111001111110001100000011100000

- The main advantage of this approach is its relative simplicity and is efficiency in finding the first free block or in consecutive free blocks on the disk.
- Indeed, many computers supply bit-manipulation instructions that can be used effectively for that purpose.
- One technique for finding the first free block on a system that uses a bit-vector to allocate dink space is to sequentially check each word in the bit map to see whether the value is not 0, since a 0-valued word contains only 0 bits and represents a of allocated blocks.
- 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 hit

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 neeed free disk block, and so on.
Example in which blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 were free and the rest of the blocks were allocated.

In this situation, we would keep a pointer 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.
This scheme is not efficient; to traverse the list, we must read each block, which requires substantial 1/0 time.

Grouping

- A modification of the free-list approach stores the addresses of in 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 it free blocks, and so on.
- The addresses of a large number of free blocks can now be found quickly.

Counting

Another approach takes advantage of the fact that, generally, several contiguous blocks may be allocated or freed simultaneously, particularly when space is allocated with the contiguous-allocation algorithm or through clustering. Thus, rather than keeping a list of in free disk addresses, we can 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.
- Although each entry requires more space than would a simple disk address, the overall list is shorter, as long as the count is generally greater than 1.
- These entries can be stored in a B-tree, rather than a linked list, for efficient lookup, insertion, and deletion.

Space Maps

The file system was designed to encompass huge numbers of files, directories, and even file systems.
The resulting data structures could have been large and inefficient if they had not been designed and implemented properly.
Example, that if the free-space list is implemented as a bit map, bit maps must be modified both when blocks are allocated and when they are freed. Freeing 1 GB of data on a 1-TB disk could cause thousands of blocks of bit maps to be updated, because those data blocks could be scattered over the entire disk.

Case study:
ABC Tech, a mid-sized software company, is facing several challenges in managing its UNIX-based file system. Employees save files in unorganized locations, leading to difficulty in retrieval and duplication of files. Additionally, unauthorized users sometimes access or modify critical project files, causing security concerns. The company also lacks a proper backup system, resulting in data loss due to accidental deletions. As an IT administrator, how would you address these issues using UNIX file system management techniques?

SOLUTION:
To effectively manage the UNIX file system and resolve these issues, the following strategies can be implemented:

1. File Organization:

Establish a structured directory hierarchy (e.g., /projects/project_A/, /projects/common_resources/).

Enforce naming conventions and use symbolic links (ln -s) to avoid file duplication.

2. Access Control & Security:

Set proper permissions using chmod, e.g., chmod 640 file.txt to restrict access.

Assign ownership with chown, e.g., chown alice:developers file.txt.

Use ACLs (setfacl -m u:bob:rwx file.txt) for granular access control.

3. Preventing Accidental Deletions:

Protect critical files with chattr +i important_file.txt.

Use a "Trash" directory instead of direct deletion (mv file.txt ~/Trash/).

4. Backup & Recovery:

Implement automated incremental backups using rsync:

```
rsync -av --delete /projects/ /backup/
```

Schedule backups via crontab (crontab -e and add 0 2 * * * rsync -av /projects/ /backup/).

Restore deleted files using delete if needed.
By applying these UNIX file system management techniques, ABC Tech can enhance file organization, security, and data recovery, ensuring a more efficient and reliable system.

QUESTION BANK

MODULE-1

1. Define operating system. Explain briefly with diagram the components of computer system.
2. What is an operating system? Explain briefly about computer system organization
3. List and explain the services provided by OS for the user and efficient operation of system.
4. What are system calls? Briefly point out its types.
5. Explain with a neat diagram of layered and microkernel structures of an operating system.
6. Describe storage management in computer system and explain each type in detail.
7. Explain dual mode operation in operating system with a neat block diagram.
8. What are virtual machines? Explain in detail about the implementation, benefits and one example of virtual machines.

MODULE-2

1. Illustrate with a neat sketch, the process states and process control block.
2. Explain the concept of threads and thread scheduling.
3. Explain in detail Dinning Philosopher's problem with algorithm.
4. Explain in detail reader's- writer's problem.
5. Explain the concept of synchronization using producer- consumer problem.
6. Write short note on:

a. Inter-process communication
b. Scheduling criteria
c. Message passing system.
d. Critical section problem.

7. Explain any 2-scheduling algorithm with example.

8. Consider the following set of processes with given length of CPU Burst. Draw the Gantt chart for FCFS and Priority scheduling. Find the average waiting time and average turnaround time for each scheduling algorithm.

Process	**P1**	**P2**	**P3**	**P4**	**P5**
Burst Time(ms)	**10**	**1**	**2**	**1**	**5**
Priority	**4**	**1**	**3**	**5**	**2**

Note: Consider least value as highest priority.

9. Consider the following set of processes with given length of CPU Burst. Draw the Gantt chart for FCFS, SFJ and Priority scheduling. Find the average waiting time and average turnaround time for each scheduling algorithm.

Process	**P1**	**P2**	**P3**	**P4**	**P5**
Burst Time(ms)	**8**	**2**	**2**	**3**	**5**
Priority	**4**	**1**	**3**	**5**	**2**

Note: Consider least value as highest priority.

MODULE-3

1. What is a deadlock? With a neat diagram, explain resource allocation graph.
2. Illustrate and explain the concept of deadlock prevention with example.
3. Explain any 2 methods for handling deadlock with necessary example.
4. Write short note on:
   a. Resource-Request algorithm
   b. Deadlock recovery.
5. Can you explain how to detect deadlock for single instance and multiple instance resource type.
6. Explain Banker's algorithm with example.
7. Illustrate and explain the concept of deadlock avoidance with example.

MODULE-4

1. Explain contiguous memory allocation along with different types of fragmentation.
2. Define paging. Explain the structure of a page table with a neat diagram.
3. Explain basic hardware structure of memory management with necessary diagrams.

4. Explain the different techniques for structuring page table with diagram.
5. Explain the concept of page replacement with diagram.
6. What is segmentation? Explain the basic methods of segmentation.

MODULE-5

1. Explain the concept of file, its attributes and functions.
2. Illustrate and explain with diagram the layered structure of file system.
3. Explain the systematic view of virtual file system with diagram.
4. Can you describe the different types of allocation methods in file system?
5. Explain the concept of free space management in file system.
6. Case study:
   ABC Tech, a mid-sized software company, is facing several challenges in managing its UNIX-based file system. Employees save files in unorganized locations, leading to difficulty in retrieval and duplication of files. Additionally, unauthorized users sometimes access or modify critical project files, causing security concerns. The company also lacks a proper backup system, resulting in data loss due to accidental deletions. As an IT administrator, how would you address these issues using UNIX file system management techniques?