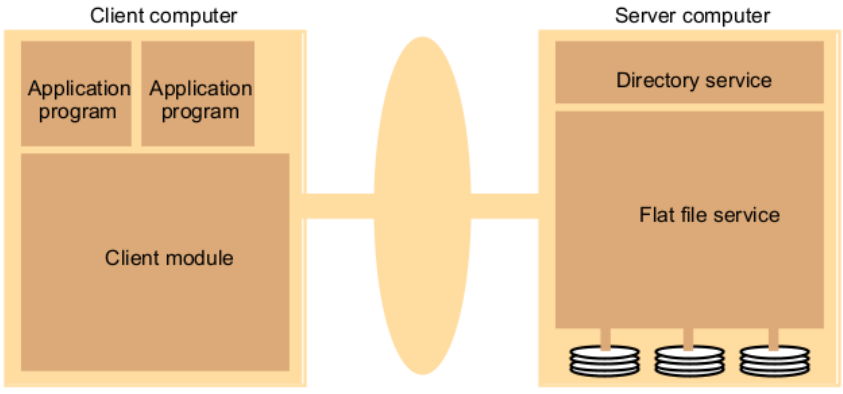
Distributed File System

1. Discuss techniques for achieving high-performance in distributed file systems.

2. Discuss the model architecture of distributed file system and its components.

Same as Q.3

3. With a neat diagram explain the components of file service architecture in brief w. r .t. following: FFS, DS, CM.

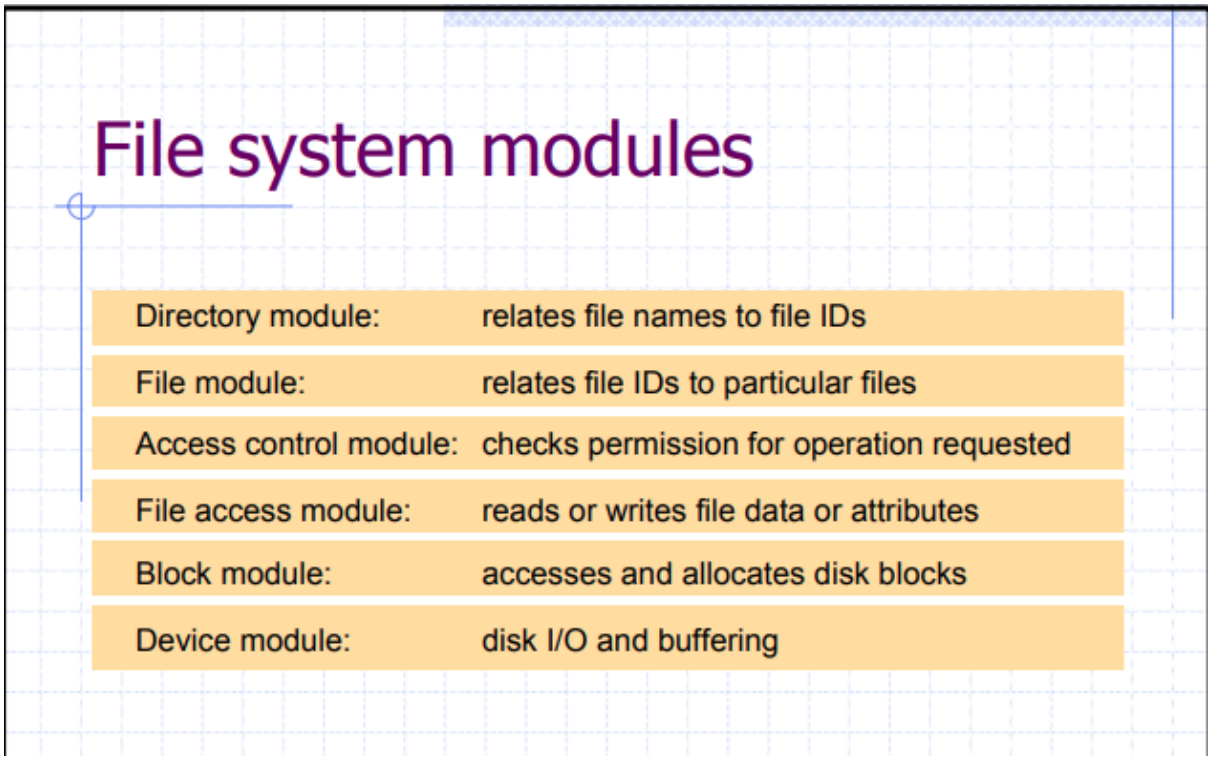


File service architecture is an architecture that offers a clear separation of the main concerns in providing access to files is obtained by structuring the file service as three components:

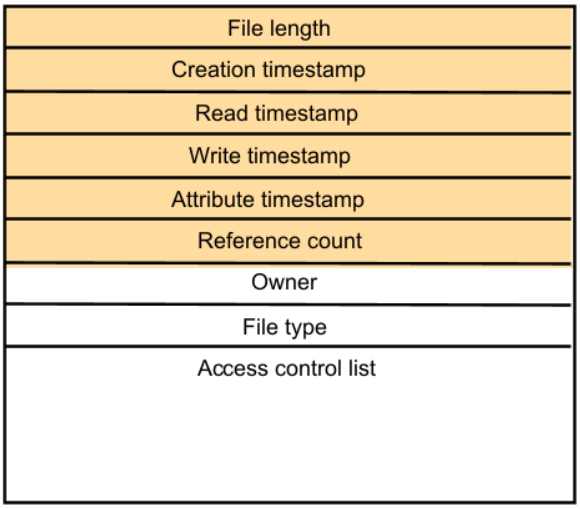
- **Flat File Service:**

    - The *Flat File Service* is concerned with the implementation of operations on the contents of the file.
    - Unique File Identifiers (UFIDs) are used to refer to files in all requests for flat-file service operations.
    - UFIDs are long sequences of bits chosen so that each file has a UFID that is unique among all of the files in a distributed system.
    - When the *flat file service* receives a request to create a file, it generates a new UFID for it and returns the UFID to the requester.
- **Directory Service:**
    - The *Directory Service* provides a mapping between text names for the files and their UFIDs.
    - Clients may obtain the UFID of a file by quoting its text name to the directory service.
    - The directory service provides the functions needed to generate directories, to add new files to directories, and obtain UFIDs from directories.
- **Client Module:**
    - A *Client Module* runs in each client computer, integrating and extending the operations of the *flat file service* and the *directory service* under a single application programming interface that is available to user-level programs in client computers.
    - It holds information about the network locations of flat-file and directory server processes.
    - It helps to achieve better performance through the implementation of a cache of recently used file blocks at the client.

4. List out file system modules.



5. Sketch the file attributes and record structure.



- File systems are responsible for the organization, storage, retrieval, naming, sharing, and protection of files.
- Files contain both data and attributes.

6. List out the transparencies in the file system.

- **Access Transparency:** Client programs should be unaware of the distribution of files. A single set of operations is provided for access to local and remote files. Programs written to operate on local files are able to access files without modification.
- **Location Transparency:** Client programs should see a uniform file namespace. Files or groups of files may be relocated without changing their pathnames, and user programs see the same namespace wherever they are executed.
- **Mobility Transparency:** Neither client programs nor system administration tables in client nodes need to be changed when files are moved. This allows file mobility. Files, or more commonly, sets or volumes of files may be moved, either by system administrators or automatically.
- **Performance Transparency:** Client programs should continue to perform satisfactorily while the load on the service varies within a specified range.
- **Scaling Transparency:** The service can be expanded by incremental growth to deal with a wide range of loads and network sizes.

7. List the directory service operation.

Lookup(Dir, Name) -> FileId -throws NotFound	Locates the text name in the directory and returns the relevant UFID. If *Name* is not in the directory, throws an exception.
AddName(Dir, Name, File) -throws NameDuplicate	If *Name* is not in the directory, adds(*Name,File*) to the directory and updates the file's attribute record. If *Name* is already in the directory: throws an exception.
UnName(Dir, Name)	If *Name* is in the directory, the entry containing *Name* is removed from the directory. If *Name* is not in the directory: throws an exception.
GetNames(Dir, Pattern) -> NameSeq	Returns all the text names in the directory that match the regular expression *Pattern*.

- The primary purpose of the directory service is to provide a service for translating text names to UFIDs.

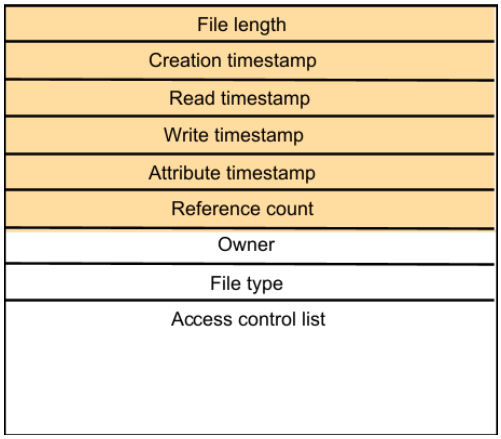
- For each operation, a UFID for the file containing the directory is required.
- The ***Lookup*** operation performs a single *Name → UFID* translation. It is a building block for use in other services or in the client module to perform more complex translations, such as hierarchic name interpretation found in UNIX.
- There are two operations for altering directories:
  - ***AddName:*** adds an entry to a directory and increments the reference count field in the file's attribute record.
  - ***UnName:*** removes an entry from a directory and decrements the reference count.
  - If this causes the reference count to reach zero, the file is removed.
- ***GetNames:*** is provided to enable clients to examine the contents of directories and to implement pattern-matching operations on file names such as those found in the UNIX shell.

8. Explain the name service in detail.

- In a distributed system, names are used to refer to a wide variety of resources such as:
  - Computers, services, remote objects, and files, as well as users
- Naming is a fundamental issue in DS design as it facilitates communication and resource sharing
- In a Distributed System, a Naming Service is a specific service whose aim is to provide consistent and uniform naming of resources, thus allowing other programs or services to localize them and obtain the required metadata for interacting with them.
- The role of names and name services:
  - Resources are accessed using *identifier* or *reference*
    - An identifier can be stored in variables and retrieved from tables quickly
    - The identifier includes or can be transformed to an address for an object. Eg: NFS file handle, Corba remote object reference.
    - A name is a human-readable value that can be resolved to an identifier or address. Eg: An Internet domain name, file pathname, process number.

- For many purposes, names are preferable to identifiers
  - because the binding of the named resource to a physical location is deferred and can be changed.
  - because they are more meaningful to users
- Resource names are resolved by name services to give identifiers and other useful attributes.

9. Describe the characteristics of the file system.



File attribute record structure

Directory module:	relates file names to file IDs
File module:	relates file IDs to particular files
Access control module:	checks permission for operation requested
File access module:	reads or writes file data or attributes
Block module:	accesses and allocates disk blocks
Device module:	disk I/O and buffering

Layered Module structure for the implementation of a non-distributed file system

- File systems are mainly responsible for the organization, storage, retrieval, naming, sharing, and protection of files.
- They provide a programming interface that characterizes the file abstraction about the details of storage allocation & layout.
- **Data and Attributes:**
  - **Data:** The data consists of a sequence of data items used for operations to read and write any portion of the sequence.
  - **Attributes:** It is a single record for holding information such as length of the file, timestamp, owner's identity, access control list.
- **Directory:** It provides a mapping from text names to internal file identifiers.
- **Metadata:** It contains information stored by the file system that is required for file management.

- Ex: File attributes, directories & all other persistent information

10. Discuss the distributed file system design requirements.

- **Transparency:** Some aspects of a Distributed system are hidden from the user.
  - **Access:** Client programs can be unaware of the distribution of files. The same set of operations is provided for access to remote as well as local files.
  - **Location:** The client program should see a uniform namespace.
  - **Mobility:** Client programs need not change their tables when files are moved to any other location.
  - **Performance:** The client program should continue to perform satisfactorily while the load on the service varies.
  - **Scaling:** The service can be expanded to deal with a wide range of load and network sizes.
- **Concurrent file updates:** Changes to the file by one client should not interfere with the operation of other clients simultaneously accessing or changing the same file.
- **File replication:** A file may be represented by several copies of its contents at different locations. It has the following benefits:
  - Load balancing to enhance the scalability of the service.
  - Enhances the fault tolerance.
- **Hardware and OS heterogeneity:** The service interfaces should be defined so that client and server software can be implemented for different operating systems and computers.
- **Fault tolerance:** To cope with transient communication failures, the design can be based on *at-most-once* invocation semantics.
- **Consistency:** Maintaining the consistency between multiple copies of files.
- **Security:** In distributed file systems, there is a need to authenticate client requests so that access control at the server is based on correct user identities and to protect the contents of the request and reply messages.