Education is a weapon whose effects depend on who holds it in his hands and at whom it is aimed.

- Joseph Stalin

MESCENotes.in

An SFI MESCE intiative

www.mescenotes.in

/SfiMESCE

Module 4

Distributed File System

DISTRIBUTED FILE SYSTEM

Ques 1) What is the distributed file system?

Ans: Distributed File System (DFS)
A file is a logical storage of data and programs belonging to both the operating systems and users. File is an abstraction of the physical properties of conventional storage devices like magnetic disks, magnetic tapes, and optical disks.

A file system consists of:
1) A collection of files
2) Directory structure and
3) A set of partitions

A file system is responsible for the organisation, storage, retrieval, naming, sharing, and protection of files. File systems provide directory services, which convert a file name (possibly a hierarchical one) into an internal identifier (e.g., inode, FAT index). They contain a representation of the file data itself and methods for accessing it (read/write).

The file system is responsible for controlling access to the data and for performing low-level operations such as buffering frequently-used data and issuing disk I/O requests.

In a distributed file system, files are distributed and are physically available at different remote sites of the distributed system. A distributed file system stores files in one or more computers called servers. They are accessible by other computers called clients. The main function of a distributed file system is to coordinate the communication between conventional operating systems and operating systems.

Ques 2) What are the goals of distributed file system?

Ans: Goal of Distributed File System
Goals in designing a distributed file system are to present certain degrees of transparency to the user and the system. Some common goals are as follows:

1) **Access Transparency:** Clients are unaware that files are distributed and can access them in the same way as local files are accessed.
2) **Location Transparency:** A consistent name space exists encompassing local as well as remote files. The name of a file does not give it location.
3) **Concurrency Transparency:** All clients have the same view of the state of the file system. This means that if one process is modifying a file, any other processes on the same system or remote systems that are accessing the files will see the modifications in a coherent manner.
4) **Failure Transparency:** The client and client programs should operate correctly after a server failure.
5) **Heterogeneity:** File service should be provided across different hardware and operating system platforms.
6) **Scalability:** The file system should work well in small environments (1 machine, a dozen machines) and also scale gracefully to huge ones (hundreds through tens of thousands of systems).
7) **Replication Transparency:** To support scalability, one may wish to replicate files across multiple servers. Clients should be unaware of this.
8) **Migration Transparency:** Files should be able to move around without the client's knowledge.
9) **Support Fine-Grained Distribution of Data:** To optimise performance, one may wish to locate individual objects near the processes that use them.
10) **Tolerance for Network Partitioning:** The entire network or certain segments of it may be unavailable to a client during certain periods (e.g., disconnected operation of a laptop). The file system should be tolerant of this.

Ques 3) What are the main features of distributed file system?

Or

What are the requirements of distributed file system?

Ans: Features of Distributed File Systems/ Requirements of Distributed File System
A good distributed file system should have the following features:

1) **Transparency:** Transparency refers to hiding details from a user. The following types of transparency are desirable:

i) **Structure Transparency:** Multiple file servers are used to provide better performance, scalability, and reliability. The multiplicity of file servers should be transparent to the client of a distributed file system. Clients should not know the number or locations of file servers or the storage devices instead it should look like a conventional file system offered by a centralized, time sharing operating system.

ii) **Access Transparency:** Local and remote files should be accessible in the same way. The file system should automatically locate an accessed file and transport it to the client's site.

iii) **Naming Transparency:** The name of the file should not reveal the location of the file. The name of the file must not be changed while moving from one node to another.

iv) **Replication Transparency:** The existence of multiple copies and their locations should be hidden from the clients where files are replicated on multiple nodes.

2) **User Mobility:** The user should not be forced to work on a specific node but should have the flexibility to work on different nodes at different times. This can be achieved by automatically bringing the users environment to the node where the user logs in.

3) **Performance:** Performance is measured as the average amount of time needed to satisfy client requests, which includes CPU time plus the time for accessing secondary storage along with network access time. Explicit file placement decisions should not be needed to increase the performance of a distributed file system.

4) **Scalability:** A good DFS should cope with an increase of nodes and not cause any disruption of service. Scalability also includes the system to withstand high service load, accommodate growth of users and integration of resources.

5) **High Availability:** A distributed file system should continue to function even in partial failures such as a link failure, a node failure, or a storage device crash. Replicating files at multiple servers can help achieve availability.

6) **High Reliability:** Probability of loss of stored data should be minimized. System should automatically generate backup copies of critical files in event of loss.

7) **Data Integrity:** Concurrent access requests from multiple users who are competing to access the file must be properly synchronized by the use of some form of concurrency control mechanism. Atomic transactions can also be provided to users by a file system for data integrity.

8) **Security:** A distributed file system must secure data so that its users are confident of their privacy. File system should implement mechanisms to protect data that is stored within.

9) **Heterogeneity:** Distributed file system should allow various types of workstations to participate in sharing files via distributed file system. Integration of a new type of workstation or storage media should be designed by a DFS.

Ques 4) What are the different types of distributed file systems? List them.

Ans: Types of Distributed File System

There are various DFS. Some of them are listed below:

1) Sun NFS
2) Coda
3) Plan 9
4) XFS
5) SFS etc.

Ques 5) Discuss about the SUN NFS file system. Also discuss about its architecture.

Or

What are the feature of SUN NFS file system?

Ans: SUN NFS File System

NFS was developed by SUN microsystems for use its Unix based work stations. So NFS is an extension to Unix file systems, i.e. it is transparent to Unix applications. But NFS has an open specification of file service. This has been implemented by different system like windows.

As on today, Sun's NFS is a widespread distributed file system in use and is popular. Some significant **features** of this distributed file system are:

1) Any machine can be a client and/or a server.
2) NFS is made to support diskless workstations.
3) Even if client and server have different hardware on different operating system platforms. NFS supports them.
4) NFS provides access transparency. There is no migration transparency in NFS
5) Performance rate is high (caching at the client is used)
6) Remote accessing is made as simple as local access through caching and read–ahead.
7) Remote files are accessed through normal system calls.

8) NFS is stateless. UDP is used as a transport. If a server fails, the client retries.

9) The two protocols used by the NFS client–server communication over remote procedure calls are:
   i) **Mounting Protocol:** It is used for access requests to an exported directory and
   ii) **Directory and File Access Protocol:** It is used for accessing the files and directories.

NFS Architecture

The NFS has the option of having any of the two model types:

1) **Remote Access Model:** In the remote model, the file system lies down on the server machine. The client interacts with the server via interface. It sends a request to the server regarding various file operations on a particular file. These options are provided in the interface only.

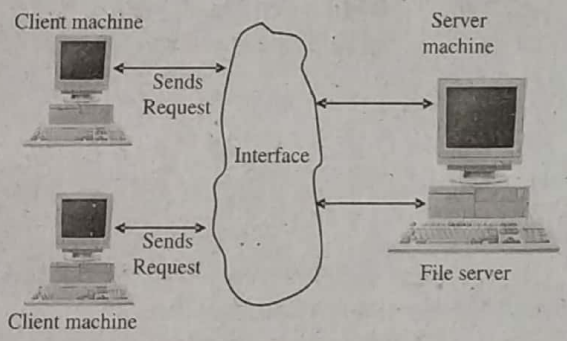

Figure 4.1: Remote Access Model

Depending on the clients request, sever performs the operations on the file. The file remains on the server only. The implementation details of various operations on the file lies with the server only. Since clients perform various operations on the remote machines (server) file, this model is called as **remote access model**.

2) **Upload/Download Model:** In the upload/download model, the client process may request for a certain file lying on the server machine. On the request, the server sends the file to client.

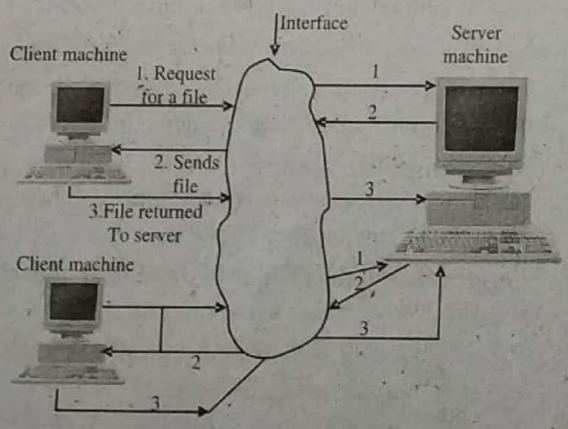

Figure 4.2: Upload/Download Model

The Client copies the file to its local disk and performs the required operations on the file. After its use, the file is returned back to the server machine. The implementation details of the file are with the client machine. The server has two versions of files available with it, the old file which was requested by the client and the new file which is being modified by the client.

Since the file is downloaded by the client before carrying out any operations on it and uploaded back to server after carrying out various operations, this type of model is called as **upload/download model**.

Though NFS has been implemented for many operating systems, UNIX based version is the most popular one.

Ques 6) Write short not on CODA File System

Ans: Constant Data Availability (CODA)

The predecessor of many distributed file systems that can be used for mobile operation is the Andrew File System (AFS).

AFS was designed to support the entire CMU community, which implied that approximately 10,000 workstations would need to have access to the system. To meet this requirement, AFS nodes are partitioned into two groups. One group consists of a relatively small number of dedicated **vice** file servers, which are centrally administered. The other group consists of a very much larger collection of **virtue** workstations that give users and processes access to the file system, as shown in **figure 4.3**.

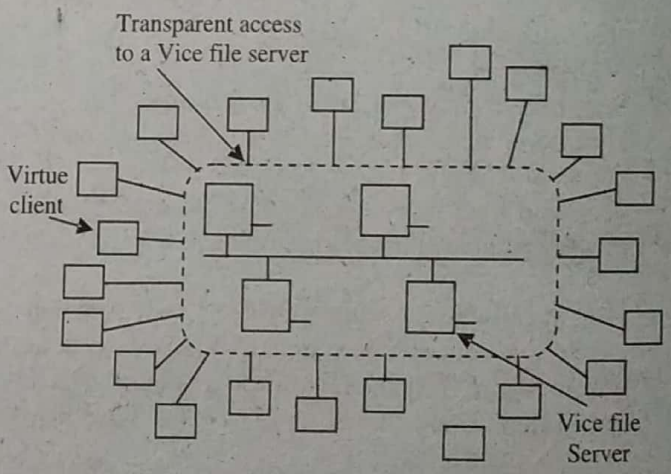

Figure 4.3: Overall Organization of AFS

CODA is the successor of AFS and offers two different types of replication – server replication and caching on clients. Disconnected clients work only on the cache, i.e., applications use only cached replicated files. **Figure 4.4** shows the cache between an application and the server.

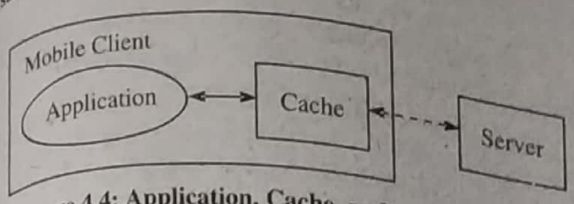

Figure 4.4: Application, Cache, and Server in Coda

CODA is a transparent extension of the client's cache manager. This very general architecture is valid for most of today's mobile systems that utilise a cache.

To provide all the necessary files for disconnected work, CODA offers extensive mechanisms for pre-fetching of files while still connected, called **hoarding**. If the client is connected to the server with a strong connection (**figure 4.5**), hoarding transparently pre-fetches files currently used.

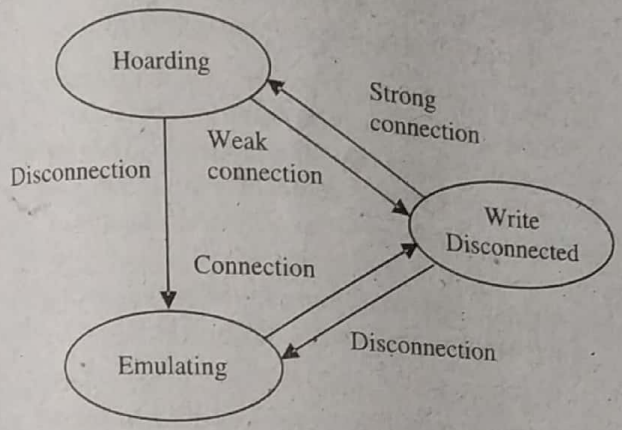

Figure 4.5: States of a Client in Coda

This automatic data collection is necessary for it is impossible for a standard user to know all the files currently used. While standard programs and application data may be familiar to a user, he/she typically does not know anything about the numerous small system files needed in addition (e.g., profiles, shared libraries, drivers, fonts).

A user can predetermine a list of files, which CODA should explicitly pre-fetch. Additionally, a user can assign priorities to certain programs. CODA now decides on the current cache content using the list and a Least-Recently-Used (LRU) strategy.

As soon as the client is disconnected, applications work on the replicates (**figure 4.5**, **emulating**). CODA follows an optimistic approach and allows read and write access to all files. The system keeps a record of changed files, but does not maintain a history of changes for each file. The cache always has only one replicate (possibly changed). After re-connection, CODA compares the replicates with the files on the server. If CODA notices that two different users have changed a file, re-integration of this file fails and CODA saves the changed file as a copy on the server to allow for manual re-integration.

Ques 7) What do you understand by plan 9 file system?

Ans: Plan 9 File System

During the 1980's era, there were strong centralized times sharing systems. In this era, they were replaced by powerful network workstations. But replacement led to a problem of transparency and so distributed system like plan was designed.

In plan 9 there are few centralized servers and many client machine. Plan 9 was developed by the same group of people who developed UNIX in the BELL labs.

The plan 9 was a file based distributed system. **Figure 4.6** shows the organization of plan 9. It consists of several servers offering services to clients. There is no distinction between servers and clients. The servers can act as a client to other server.

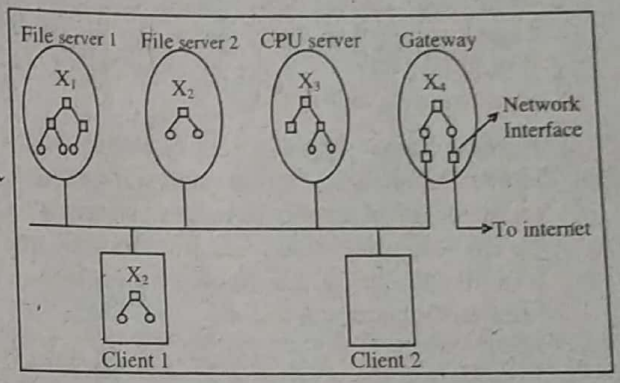

Figure 4.6: Organization of Plan 9

The client communicates with the server by locally mounting the servers name space into its local name space.

Plan 9 can be discussed with respect to following **points**:

1) **Communication:** In plan 9, the communication takes place at the level of network interfaces. The approach similar to UNIX is being used where the network interfaces are represented by file systems.

   9P is the protocol used for communication across the network. 9P runs on reliable transport protocol and offers services for file handling like opening and closing a file.

   The TCP connection is represented by a subdirectory of files. Various files are used for carrying out various functionalities.

   Some of the commonly used files are:

   i) **Data:** It is used to read and write the data.

   ii) **Listen:** The incoming connections set up requests are accepted by listen.

iii) **Local:** It is used to provide information on the callers side of the connection.

iv) **Ctl:** Ctl is used to write control commands. Such commands are protocol specific.

v) **Remote:** It provides information on the other side of the connection.

2) **Processes:** Plan 9 servers implement a hierarchical name space. The plan 9 file server is a stand-alone system. This system runs on a dedicated machine. Logically, it is a 3 layered server:

i) **Lowest Layer:** The client sees the file system actually stored on the device which is formed by the lower most layers. The lowest layer consists of Write Once and Read Many (WORM) devices. It provides bulk storage space.

ii) **Middle Layer:** A large caching system is provided for the WORM devices by the middle layer. The middle layer has a collection of large number of magnetic disks. The file access procedure reads from the WORM device and saves the data on the disk.

iii) **Highest Layer:** There are many buffers in the highest layer. These buffers act as in–memory cache for the magnetic disks. In this layer also, the file read procedure, reads from the disk and save the modifications to in–memory cache. Later on they are sent to disk.

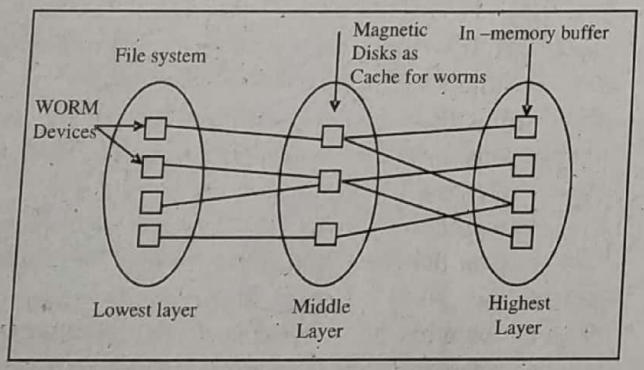

Figure 4.7: Plan 9 File Server

Other processes in plan 9 involve:

i) Client exporting its own files to remote server.
ii) Client sends messages to the external network.
iii) Client mounts its file system on to a gateway.
iv) Client communicates with devices such as mouse or screen etc.

3) **Naming**: Plan 9 locally mounts remote name spaces. This creates a private name space for each of the process in plan 9.

There is a union directory where multiple name spaces can be mounted on the same mount. The created file is labelled by two integers:

i) **Path:** It has unique file number. This number makes the identification of a file clear with respect to the server or device.

ii) **Version Number:** Every modification done to the file increments the version number.

All files are globally identified by four numbers. Two numbers are used to identify the server and two numbers are used to identify the files.

4) **Synchronization:** Plan 9 implements the same semantics as that of UNIX file sharing. When the client opens, the file, it is sent form the server. The client has the copy of the file in its local cache. The same copy of the file may also be needed and loaded by various other clients. All the clients are modifying the same file. The client which closes the file first and intimates the changes or modifications to the server actually changes the file. The changes done by other clients are not countable.

5) **Caching and Replication:** In plan 9 there is minimal support for caching and replication.

6) **Security:** Secure channels are maintained and the user authenticated in the same way as the secure channels in CODA file systems. This is done by sending secret keys. If a client wants to communicate with the server, it sends its identity along with a challenging message to the server. The challenging message is in the encrypted form. The clients also sends an encryption key.

The server accepts the challenge, decrypts the message with the secret key that is shared by the client and the server and thus proves its identity.

Now the server sends a challenging message along with its identity and the secret key. The client should decrypt the message and prove its identity. Once the authentication is proved, the server sends a session key which is used for further communication.

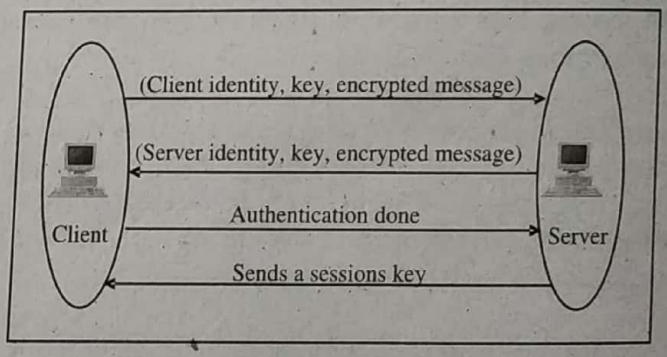

Figure 4.8: Plan 9 Mutual Authentication

Ques 8) Discuss about the file service architecture.

Ans: File Service Architecture

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:

1) **Flat File Service:** It is concerned with the implementation of operations on the contents of file. Unique File Identifiers (UFIDs) are used to refer to files in all requests for flat file service operations.

The main flat file server operations are:

i) **Read (FileId, i, n):** Reads a sequence of upto n items from a file starting at item i.

ii) **Write (FileId, i, Data):** Write a sequence of Data to a file, starting at item i.

iii) **Create():** Creates a new file of length 0 and delivers a UFID for it.

iv) **Delete(FileId):** Removes the file from the file store.

v) **GetAttributes(FileId):** Returns the file attributes for the file.

vi) **SetAttributes(FileId, Attr):** Sets the file attributes.

2) **Directory Service:** It provides mapping between text names for the files and their UFIDs. Clients may obtain the UFID of a file by quoting its text name to directory service. Directory service supports functions to add new files to directories.

Directory Service operations include:

i) **Lookup(Dir, Name):** Locates the text name in the directory and returns the relevant UFID. If Name is not in the directory, throws an exception.

ii) **AddName(Dir, Name, File):** 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.

iii) **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.

iv) **GetNames(Dir, Pattern):** Returns all the text names in the directory that match the regular expression Pattern.

3) **Client Module:** It runs on each computer and provides integrated service (flat file and directory) as a single API to application programs. It holds information about the network locations of flat-file and directory server processes.

The relevant client modules and their relationship is shown in **figure 4.9**:

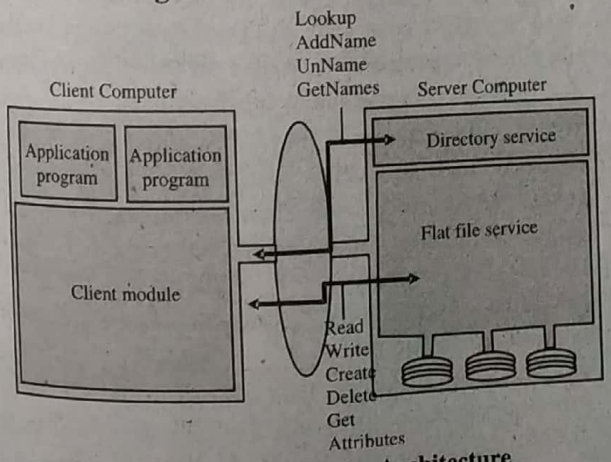

Figure 4.9: File Service Architecture

Ques 9) What do you understand by network file system?

Ans: Network File System

Network File System (NFS) is a distributed file system (DFS) developed by Sun Microsystems. This allows directory structures to be spread over the net- worked computing systems.

The Network File System (NFS) is a client/server application that lets a computer user view and optionally store and update files on a remote computer as though they were on the user's own computer. The NFS protocol is one of several distributed file system standards for network-attached storage (NAS).

NFS allows the user or system administrator to mount (designate as accessible) all or a portion of a file system on a server. The portion of the file system that is mounted can be accessed by clients with whatever privileges are assigned to each file (read-only or read-write). NFS uses Remote Procedure Calls (RPC) to route requests between clients and servers.

NFS follows the directory structure almost same as that in non-NFS system but there are some differences between them with respect to:

1) Naming
2) Path Names
3) Semantics

The NFS is built using a **client–server approach**. Servers store files and respond to requests from remote applications that need access those files. However, the applications are not requested to contact the server directly. Instead, file system calls from the application are redirected to a proxy of the remote service that executes on the client's machine. This proxy is simply called the **NFS client**, and it takes care of the interaction with the remote server. This interaction is performed using SUN's remote procedure call service. The design decision of using RPC to access the server, along with the strategic option of making the server's interface public, was one of the reasons that made NFS so popular. Since the interfaces and the format of the associated messages were known, it was possible for different companies to develop NFS components for many different architectures and operating systems.

The use of the client–server approach is made transparent to the client application through the addition of a virtual file system (VFS) layer to the Unix kernel. The layer introduces an additional level of indirection in the file system calls. Instead of calling a specific file system primitive, the application calls the VFS interface that, in turn, calls the NFS client primitives. The VFS allows the kernel to support several file systems simultaneously, including the NFS and the native file system.

The NFS server uses **stateless approach**, i.e., servers does not keep any state about open files. Also, servers keep no information about the number and state of their clients. This means that each request must be self–contained, i.e., the server must be able process the request just looking at its parameters, without any knowledge from past requests. Since the server remembers nothing, it does not lose any relevant information when it crashes.

Thus, a stateless approach has some advantages from the fault tolerance point of view: a server may crash, recover, pick a new request and continue as if nothing happened. On the other hand, since the server does not keep state, it is unable to check if a file is being accessed by one or more clients. This, as we will see, makes the task of preserving the Unix semantics of file sharing impossible. Due to this reason, NFS does not support the open primitive (open semantics require the system to "remember" that the file has been opened). Instead, a lookup primitive, that provided a handle for a file name is provided.

A partial list of the NFS server interface is presented in **Table 4.1**.

Table 4.1: NFS Interface (Partial)

Name (Parameters)	Returns
lookup (dirth, name)	(fh, attr)
create (dirfh, name, attr)	(newfh, attr)
remove (dirfh, name)	(status)
read (fh, offset, count)	(attr, data)
write (fh, offset, count, data)	(attr)
mkdir (dirfh, name, attr)	(newfh, attr)
readdir (dirfh, cookie, count)	(direntries)

Ques 10) What are the advantages and disadvantages of network file system (NFS)?

Ans: Advantages of NFS

Advantages of NFS are as follows:

1) Stateless server and client
2) Server can be rebooted and user on client might be unaware of the reboot
3) Client/server distinction occurs at the application/ user level not the system level
4) Highly flexible, so we need to be disciplined in administration/configuration.
5) NFS allows centralized management.
6) NFS allows granularity of access permissions.

Disadvantages of NFS

Disadvantages of NFS are as follows:

) Network slower than local disk
) Network or server may fail even when client OK
) Complexity, security issues.

Ques 11) What is andrew file system? Draw the architecture of andrew file system.

Ans: Andrew File System

Andrew File System (AFS) is a distributed network file system developed by Carnegie Mellon University. Enterprises use an AFS to facilitate stored server file access between AFS client machines located in different areas. AFS supports reliable servers for all network clients accessing transparent and homogeneous namespace file locations.

AFS is a distributed file system, with scalability as a major goal. Its efficiency can be attributed to the following practical assumptions (as also seen in UNIX file system):

1) Files are small (i.e. entire file can be cached)
2) Frequency of reads much more than those of writes
3) Sequential access common
4) Files are not shared (i.e. read and written by only one user)
5) Shared files are usually not written
6) Disk space is plentiful

An AFS may be accessed from a distributed environment or location independent platform. A user accesses an AFS from a computer running any type of OS with Kerberos authentication and single namespace features. Users share files and applications after logging into machines that interact within the Distributed Computing Infrastructure (DCI).

In distributed networks, an AFS relies on local cache to enhance performance and reduce workload. For example, a server responds to a workstation request and stores the data in the workstation's local cache. When the workstation requests the same data, the local cache fulfills the request.

AFS networks employ server and client components, as follows:

1) A client may be any type of machine that generates requests for AFS server files stored on a network.
2) After a server responds and sends a requested file, the file is stored in the client machine's local cache and presented to the client machine user.
3) When a user accesses the AFS, the client uses a callback mechanism to send all changes to the server. Frequently used files are stored for quick access in the client machine's local cache.

AFS equips users with multiple access control permissions, as follows:

1) **Look Up (1):** Users may access and list AFS directory and subdirectory content and review a directory's access control list (ACL).

2) **Insert (i):** Users may add new subdirectories or files.

3) **Delete (d):** Users may remove directory files.

4) **Administer (a):** Users may modify the home directory's ACL.

5) **Read (r):** Users may view file directory or subcategory contents, as AFS supports the standard Unix Owner Read permission control set.

6) **Write (w):** Users may modify or write files, as AFS supports the Unix Owner Write permission control set.

7) **Look (k):** Processors may use the directory to execute programs requiring Flock files.

The implementation of AFS is carried out by two software components. These software components exist as UNIX processes. The names of the components are 'Venus' and 'Vice':

1) **Vice** is a server software component lying on every server. It runs a user level Unix process.

2) **Venus** is a user level process that runs on every client.

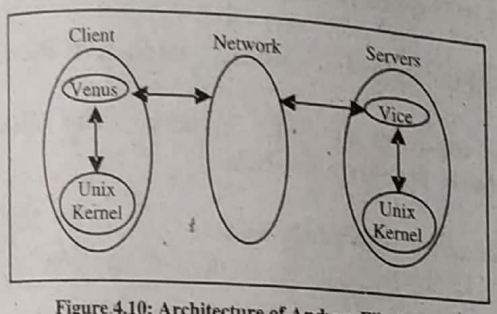

Figure 4.10: Architecture of Andrew File System

AFS was designed to support atleast 10,000 workstations.

Ques 12)What do you understand by name services?
Or
What is name and address?

Ans: Name Services

In a distributed system, names are used to refer to a wide variety of resources (such as computers, services, remote objects and files, etc.). Names facilitate communication and resource sharing.

Processes cannot share particular resources managed by a computer system unless they can name them consistently. Processes cannot share particular resources managed by a computer system unless they can name them consistently.

Name service stores a collection of one or more naming contexts. Naming contexts is the sets of bindings between textual names and attributes for objects (such as users, computers, services and remote objects).

Names and Addresses

Any process that requires access to a specific resource must possess a name for it. **Examples** of human-readable names are as follows:

1) **File Names:** /etc/passwd
2) **URLs:** http://www.cdk4.net
3) **Internet Domain Names:** www.cdk4.net

Name and address are described below:

1) **Name:** Name is an identifier permanently associated with an object, independent of its location within the distributed system.

   Names are also sometimes needed to refer to entities in distributed systems that are beyond the scope of any single service. The major examples of these entities are users (with proper names and email addresses), computers (with hostnames such as www.cdk5.net) and services themselves (such as file service or printer service).

2) **Address:** An address is an identifier associated with the current location of the object. **For example,** email addresses usually have to change when they move between organizations or ISPs; they are not enough in themselves to refer to a specific individual over time.

Ques 13)Discuss about URI and URL?

Ans: URI

A URI (Uniform Resource Identifier) is a sequence of characters that identifies a logical or physical resource.

In a distributed application, with multiple services, clients and other components needing to come together coherently, the task of identification is so important that it is formalized into a concept known as a Uniform Resource Identifier (URI).

For computing resources, a general scheme for expressing URIs has been defined and standardized. Such URIs has the following format:

<scheme name> : <hierarchical part? [? <query>] [# <fragment>]

Where,

1) The **scheme name** describes the mechanism, system or category of the URI. It tells you what to do with or how to with or how to interpret the rest of the identifier string.
2) The **hierarchical part** holds the main identifying information; it is so named because we frequently identify objects by gradual narrowing of scope.
3) The optional **query** in a URI represents additional information that is not necessarily part of the path, and

4) Finally the **fragment** represents an identifier for a specific part or section of the resource.

There are two types of URIs:

1) **Uniform Resource Identifiers (URLs):** This type of URI begins by stating which protocol should be used to locate and access the physical or logical resource on a network. If the resource is a web page, **for example**, the URI will begin with the protocol HTTP. If the resource is a file, the URI will begin with the protocol FTP or if the resource is an email address, the URI will begin with the protocol mailto. It is important to remember that URLs are not persistent. This means that if the resource's location changes, the URL also needs to change to point to the resource's new location.

2) **Uniform Resource Names (URNs):** This type of URI does not state which protocol should be used to locate and access the resource; it simply labels the resource with a persistent, location-independent unique identifier. A URN will identify the resource throughout its lifecycle and will never change. Each URN has three components: the label "urn", a colon and a character string that serves as a unique identifier.

Ques 14) What is domain name system? What is the format of DNS?

Ans: Domain Name System (DNS)

DNS is a naming service which has a scalable and distributed database of IP addresses. In DNS system, there are **Name Servers** (computers placed on various locations all over the world), which store a part of information in DNS database. Like a tree, DNS provides the hierarchical naming scheme. This is a host-based scheme with domain and distributed database concepts. This process associates the host name with the host computer IP address because it is difficult to remember numeric addresses (IP address).

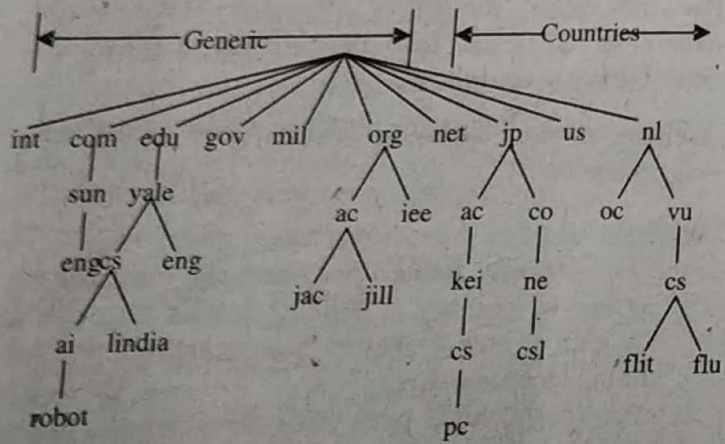

Figure 4.11: A Portion of the Internet Domain Name Space

There are hundreds of top-level domains which categorise the Internet and every domain has many hosts. These domains are divided into various sub-domains, which are further divided into various other levels and so on. **Figure 4.11** shows the tree structure of domains. In this figure, leaves domains have no sub-domain but may include single host (node) or may represent even an organisation having thousands of other hosts (nodes).

Format of Domain Names

Figure 4.12 shows a basic format of the domain name.

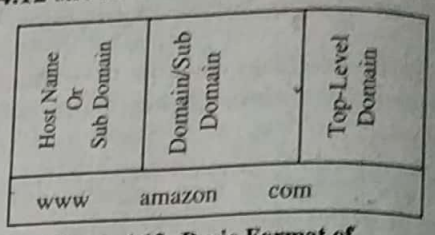

Figure 4.12: Basic Format of Domain Name

The exact layout may vary because sometimes an address may have some more parts. Domain is divided into many subdomains. Sometimes a domain name has the second and top-level domain such as 'www.tppl.co.in' where '.co' is the second level domain and '.in' is the top-level domain

Ques 15) What are the elements of domain names?

Or

What are the rules for naming second-level domain?

Or

What is top-level domain? What are the different categories of top-level domain?

Ans: Elements of Domain Names

Figure 4.13 shows elements of domain name system.

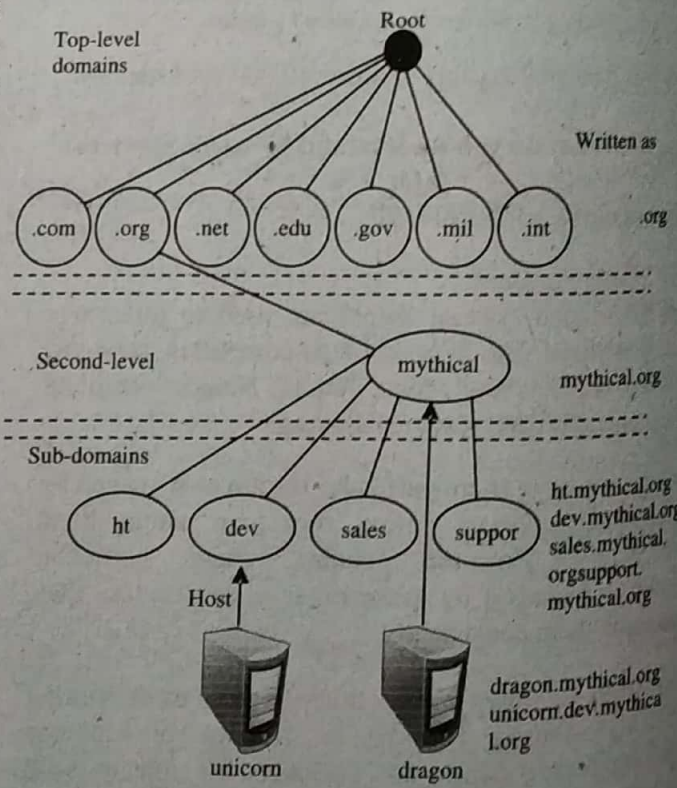

Figure 4.13: Elements of Domain Name System

Following are the fundamental elements of a domain name:

1) **Subdomain:** WWW is the subdomain of all domains. Some organisations like 'dell' use the 'del' as subdomain (e.g., **del.icio.us**) for marketing purpose. It also identifies some geographical region, e.g., 'sverige' is a subdomain which corresponds to Sweden.

2) **Second-Level Domain:** It is the main part of a web address, which reflects the title of the organisation. Various rules are applied on the second-level domain.

Rules for Naming Second-Level Domain

i) Only alphanumeric (means both letters and numbers) are used, except hyphen. Even hyphen cannot be used at the beginning and end of the domain name.
ii) Maximum character length used for second-level domain is 63.
iii) It starts only with a number or with a letter.
iv) No special character is used to define the second-level domain.

Registering Second-Level Domain

There are various authorised organisations available, which are responsible for registration of the second-level domain.

The top-level domains are also registered (by registrar) and controlled by the same organisation. For registration, organisations charge some fees. The registrars (who book the domain) always maintain the records to identify the owner of second-level domain and have the following contacts:

i) Administrative contact,
ii) Billing contact, and
iii) Technical contact.

Registrar maintains the IP address of two DNS servers because it is necessary for hosting the second-level domain. This address is used to know more about the domain. The top-level domain also maintains this information. DNS passes the request (for a specific host name) to the top-level domain and this information is further passed on to the authoritative server for the second-level domain, which further, replies to the request. The authoritative servers can be located anywhere. They contain the information of a host in a specific domain.

Before applying for the second-level domain, first the user has to apply for the top-level domain. Internet Corporation for Assigned Names and Numbers (ICANN) is an organisation which authorises the agencies for registering the names in Generic Top-Level Domains (gTLD). For completing the form of domain registration, one has to select a name (not registered by anyone else) and then identify a technical contact person, an administrative contact person and minimum two hosts (name servers).

3) **Top-Level Domain:** In the hierarchical domain name system, top-level domains are present at the top and installed in the root zone of the name space. Top-level domains are written at the end of a domain, i.e., they are the last part of the fully qualified domain name such as **.com** in www.google.com. It is not case sensitive, i.e., it can also be written as '.COM'.

Most of the top-level domains are managed by the organisation named as ICANN. ICANN handles the Internet Assigned Numbers Authority (IANA) and is accountable for DNS root zone maintenance.

Categories of Top-Level Domain

Users have the following domain choice while selecting the top-level domain:

i) **.edu:** It is Sponsored Top Level Domain (sTLD). This domain name is proposed for education institutions.

ii) **.com:** This extension is used for commercial sites. It is the gTLD.

iii) **.mil:** It is also an sTLD for the U.S Department of Defence and its subsidiary or affiliated organisations. '.mil' is derived from the word 'military'. This is the oldest top-level domain which was created in January 1985. Except the U.S.A. all other countries use this as the second-level domain for military.

iv) **.gov:** It is also an sTLD in the DNS. '.gov' is derived from the word government, so it is used by the government entities only. The '.gov' domains are managed by the General Services Administration (GSA). So, an authorised letter should be submitted to the GSA, for registration of this domain. GSA provides some guidelines for second level domain naming, e.g., for a state, user needs the full state name or its abbreviation.

v) **.net:** This domain name is derived from the word 'network' and originally created for the companies who deal with network technologies. **For example,** Internet Service Provider (ISP) '.net' is the top-level domain (created in January 1985), which is not included in the RFC 920. Registration of this domain is handled through accredited registrars.

vi) **.org:** It comes from the word 'organisation' and was created in January 1985. Generally, it is used by the non-profit or non-commercial organisations. MITRE Corporation is the first organisation which is registered (in January 1985) with .org domain (mitre.org). Registration of this domain is handled through accredited registrars worldwide.

vii) **.int:** It is also a sTLD and derived from the word 'international'.

According to current IANA policy, the '.int' sTLD is reserved for international treaty-based organisations, United Nations (UN) agencies and organisations or entities having 'Observer' status at the UN. This domain is firstly used by NATO, which had been assigned the .nato TLD.

Ques 16)Discuss about name resolution.

Ans: Name resolution

Whenever a symbolic name is used to identify a resource, the name must be translated to the corresponding physical address in order to locate the resource. The process of the translation is called name resolution, or more simply, **name lookup**.

To perform **name resolution**, a database (also called a directory or a registry) must exist containing the mapping between symbolic names and physical names. If the namespace of a naming scheme is of a limited size, then it is possible to perform name resolution manually. In the case of the DNS or XNS, a manual process is out of the question; instead, a network service has to be provided to support online name resolution.

For the DNS, the name lookup service is provided by machines that are called DNS servers. A central authority maintains the name database and sees to it that the database is distributed throughout the internet to the DNS servers. When a domain name is specified–whether entered into a browser or coded in a program being executed–the name is submitted to the nearest DNS server for resolution. If the nearest server does not have the mapping, that server forwards the request to another DNS server. The propagation of the request continues until the name is resolved, at which time the mapping is sent back to the process that originated the request.

Ques 17)What do you understand by directory service?

Ans: Directory Services

Directory services are network services that identify every resource such as email address, peripheral devices and computers on the network, and make these resources accessible to users and applications.

A directory service is a customizable information store that functions as a single point from which users can locate resources and services distributed throughout the network. This customizable information store also gives administrators a single point for managing its objects and their attributes. Although this information store appears as a single point to the users of the network, it is actually most often stored in a distributed form.

A directory service is an extension of a naming service that allows one to lookup objects based on names or based on attributes. Attributes have attribute identifiers and a set of attribute values.

A service that stores collections of bindings between names and attributes and that looks up entries that match attribute-based specifications is called a directory service. **Examples** are Microsoft's Active Directory Services, X.500 and its cousin Lightweight Directory Access Protocol (LDAP), Univers and Profile.

Directory services are sometimes called **yellow pages services**, and conventional name services are correspondingly called **white pages services**, in an analogy with the traditional types of telephone directory. Directory services are also sometimes known as **attribute-based name services**.

A directory service returns the sets of attributes of any objects found to match some specified attributes. So, for example, the request 'TelephoneNumber = 020 555 9980' might return {'Name = Deepak Singh', TelephoneNumber = 020 555 9980', 'emailAddress = deepaksingh@gmail.com', ...}. The client may specify that only a subset of the attributes is of interest – for example, just the email addresses of matching objects. X.500 and some other directory services also allow objects to be looked up by conventional hierarchic textual names.

Ques 18)Write short note on X.500 Directory Service.

Ans: X.500 Directory Service

X.500 is a standard for directory services developed by the International Telecommunications Union (ITU), the most recent version of which was published in 1993. It uses a distributed approach to implement a global directory service.

Systems based on X.500 include the Lightweight Directory Application Protocol (LDAP), Novell's Novell Directory Services (eDirectory), and Microsoft's Active Directory (Active Directory Domain Services). Directory information for an organisation is held in a database called a directory system agent (DSA). A single DSA may hold information for more than one

organisation, or conversely, directory information for a single organisation can reside in multiple DSAs. All DSAs within an X.500 directory service are interconnected in a virtual hierarchical data structure called the **Directory Information Tree (DIT)**, and can exchange data with each other using the Directory System Protocol (DSP). Each DSA can hold all or part of the global directory, and data can be replicated in two or more DSAs, reducing access time, and ensuring the availability of information in the event of a single DSA failure. The distribution of information among DSAs is transparent to users.

Directory information is stored as entries, each of which refers to an object of a specific class. The defined object classes include country, organisation, organisational unit and person. The object at the top of the directory information tree is called the **root object**, and contains all the other objects in the tree. The country, organisation and organisational unit objects are all container objects, and can contain other objects. The person object is a leaf object, and as such cannot contain other objects. The hierarchical structure of the directory information tree is illustrated in **figure 4.14**:

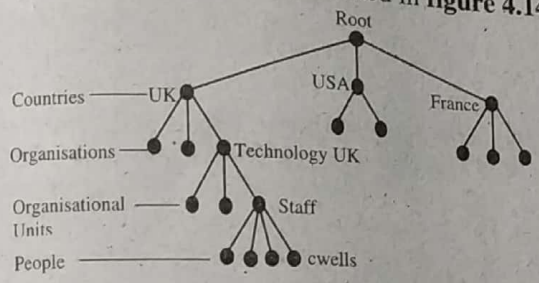

Figure 4.14: The Hierarchical Structure of the Directory Information Tree

The information contained in an entry consists of various object attributes, the number and nature of which will depend on the object class to which the entry belongs. **For example**, the object class person has attributes like common name, telephone number, and e–mail address. The entry shown in **table 4.2** would appear in the DIT as the node 'cwells' (the name of an entry must be unique within the container object in which it appears.)

Table 4.2: Typical Attribute–Value Pairs

Attribute	Value
Object class	Person
Common Name	cwells
Surname	Wells
Postal Address	TechnologyUK, North Prospect, Plymouth PL2 2QA
Telephone Number	01752 123456
Mail	cwells@technologyuk.net

A user can access and modify the information in the directory information tree using a Directory User Agent (DUA), which will be a client application of some kind, usually offering a graphical user interface. Access to the information is restricted to authorised users, and information about which users have access to which object (and at what level) is held in the directory information tree itself.

Searches can be carried out within the directory, or within a sub–tree of the directory, for object having specific attributes and attribute values. **For example**, the directory could be searched for all persons having a particular common name. Searches can usually be made on the basis of an exact match or an approximate match (using wildcards). Entries in the DIT are uniquely identified by their full distinguished name, consisting of the object's common name concatenated to the name of the container object in which it resides, and that of any other container objects in the hierarchical path between the object itself and the root object. The full distinguished name of the object 'cwells' in the directory information tree illustrated above would therefore appear as follows:

cwells.Staff.TechnologyUK.UK

/SfiMESCE

SFI MESCE

Unit Committee

www.mescenotes.in