**UNIT III**

**3.1 INTRODUCTION TO DISTRIBUTED FILE SYSTEM (DFS)**

* Distributed file system support the sharing of information in the form of files and hardware resources in the form of persistent storage throughout an intranet.

* Distributed File systems allow sharing data over a long period in a secure and reliable way.

**3.1.1 Characteristics of file systems:**

* File systems are responsible for the organization, storage, retrieval, naming, sharing, and protection of files.

* Files contain both data and attributes.

* File systems are designed to store and manage large number *of* files, with facilities for creating, naming, and deleting files.

* The naming of files is supported by the use of directories. A directory provides a mapping from text names to internal file identifiers.

* File systems also take responsibility for the control of access to files, restricting access to files according to user's authorizations and the type of access requested.

* ***Meta data***refers to all of the extra information stored by a file system that is needed for the management of files.

* A layered module structure for the implementation of a non-distributed f i l e system is shown m Figure 2.10.

* Each layer depends only on the layers below it.

* The implementation of a distributed file service requires all of the components shown in Figure 3.1.

* With additional components to deal with client-server communication and with the distributed naming and location of files.

|  |
| --- |
| Directory module: relates file mines to file IDs |
| File module : relates file ID to particular files |
| Access Control Module: Checks permission for file requested |
| File access module : reads or writes file data |
| Block module : accesses and allocates disk blocked |
| Device module : disk I/O and buffering |

# Figure 3.1: File System Modules

**3.1.2 Distributed file system requirements**

 **Transparency**

* The file service is usually the most heavily loaded service in the intranet.

* The design of the file service should support many of the transparency requirements for distributed systems.

* The following forms of transparency are partially or wholly addressed by current file service

* + ***Access transparency****- Client programs should be unaware of distribution of the files*

* + ***Location transparency -*** *Client programs should see a uniform file naming space.*

* + ***Mobility transparency****- .either client programs nor system administration tables in client nodes need to be changed when files are moved.*
  + ***Performance transparency****- Client programs should continue to perform satisfactorily even when the load varies.*

* + ***Scaling transparency****- Service can be expanded by incremental growth*.
* **Concurrent file updates- Changes** to a 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.
* **Hardware and OS heterogeneity** - The service interfaces should be defined so that client and server software can be implemented for different OS and computers
* **Fault tolerance** - Service continue to operate in case of client and server failures.
* **Security-** authenticates client requests, protect the contents of request, and reply messages.
* **Efficiency-** Achieve comparable level of performance.

**3.2 FILE SERVICE ARCHITECTURE**

* + - The file service can be structured as three components: a flat file service, a directory service and a client module.

* + - The modules and their relationships are shown in Figure 2.11.

* + - The divisions of responsibilities between the modules are as follows:

* **File service**
  + - * The flat file service is concerned with implementing operations on the contents of files.

* + - * *Unique file identifiers* (UFID) 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 unique UFID 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.

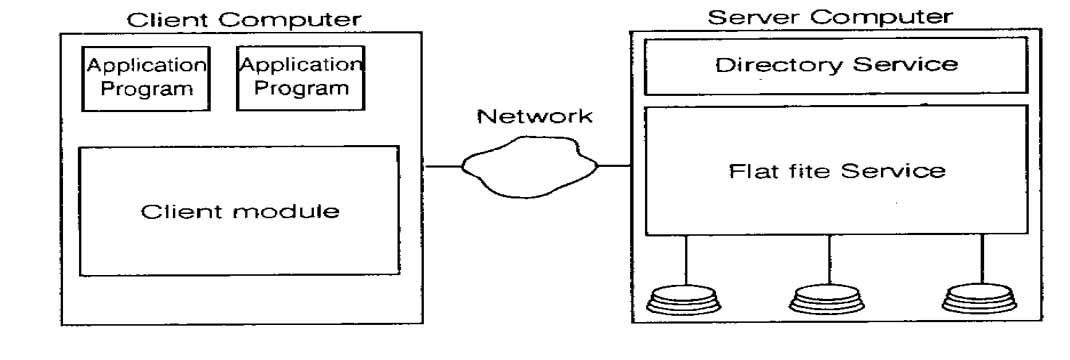
* + - * Clients may obtain the UFID of a file by quoting its text name to the directory service.

* + - * The directory service provides the functions needed to generate directories, to add new file names to directories and to obtain UFIDs from directories.
      * It is a client of the flat file service; i t s directory files are stored in the files of the flat file service.
* **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.

* + - * The client module also holds information about the network locations of the flat file server and directory server processes.

* + - * The client module can achieve satisfactory performance through the implementation of a cache of recently used file blocks at the client.



***F i g u re 3.2: File Service architecture***

* **Flat file service interface**

The definition of the interface to a flat file service is shown below. This is the RPC interface used by client modules.

*Read(Fileld, i, n) - Data - If 1 < I < Length (File): Reads a sequence of up-to n items from a file starting at item i and returns it in Data.*

*Write(Fileld, i, Data) - If 1< i< Length (File)+1; writes a sequence of data to a file, starting at item i, extending the file if necessary.*

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

*Delete(Fileld) - Removes the file from the file store.*

*GetAttributes(Fileld) - Attr - Returns the file attributes for the file.*

*SetAttributes(FileId, Attr) - sets the file attributes.*

* In comparison with the UNIX interface, out flat file service has no open and close operations - files can be accessed immediately by quoting the appropriate UFID.

* The read and write requests include a parameter specifying a starting point within the file for each transfer, whereas in UNIX, each read and write operation starts at the current position of the pointer and the pointer is advanced by the number of bytes transferred after each read or write.

* A seek operation is provided to reposition the read-write pointer.
* The interface to flat file service differs from UNIX file system interface mainly for reasons of fault tolerance:

* + - **Repeatable operations:** With the exception of create, the operations are idempotent, allowing the use of at-least-once RPC semantics, - clients may repeat calls to which they receive no reply.

* + - **Stateless servers:** Stateless servers can be restarted after a failure and resume operations without any need for clients or the server to restore any state.

* Directory service interface: The directory service operations are defined below:
  + ***Lookup (Dir,Name):*** *UFID.* Field - Locates the text name in the directory and returns the relevant.

* + ***Add Name(Dir,Name,File)*** - If name is not in the directory, adds (Name, File) to the directory and updates the files attribute record.If name is already in the directory ; throws an exception.

* + ***Unnamed (Dir,Name)***- If name is in the directory; the entry containing name is removed from the directory.If name is already in the directory ; throws an exception.

* + ***GetNames(Dir, Pattern****)*- Nameseq - Returns all the text names in the directory that match the regular expression pattern.

 **Hierarchic file system**

* A UNIX-like file-naming system can be implemented by the client module using the flat file and directory services.

* A tree structured network of directories is constructed with files at the leaves and directories at the other nodes of the tree.

* The root of the tree is a directory with a *well-known* UFID.
* The file attributes associated with files should include a type filed that distinguishes between ordinary files and directories.

 **File grouping**

* A file group is a collection of files located on a given server. A server may hold several file groups, and groups can be moved between servers, but a file cannot change the group to which it belongs.

* In a distributed file system that supports file groups the representation of UFID's includes a file group identifier component, enabling the client module in each client computer to take responsibility for dispatching requests to the server that holds the relevant file group.

* File group identifiers must be unique throughout a distributed system.

* Whenever a new file group is created, a unique identifier can be generated by concatenating the 32-bit IP address of the host with a 16-bit integer derived from the date, producing a unique 48-bit integer:

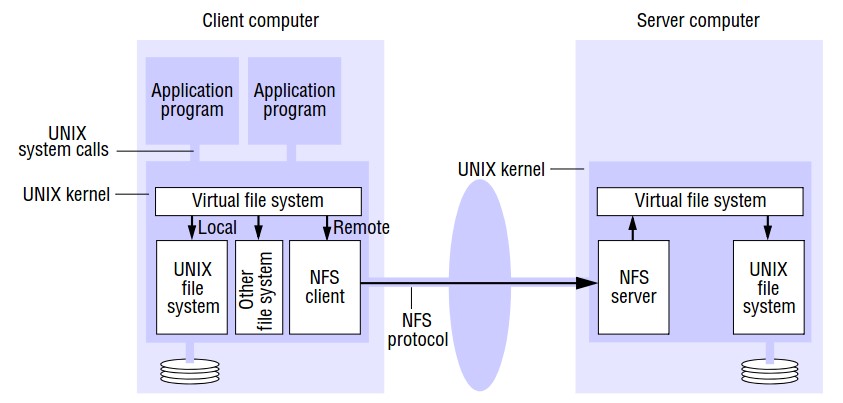
**IP Address: 32 bits**

**Date: 16 bits**

# 3.3 SUN NETWORK FILE SYSTEM

* It follows the abstract model discussed in the previous section. All implementation **of N***FS Support the NFS* protocol - a set of remote procedure calls that provide the means **for** client to perform operations on a remote file store.
* The NFS protocol is operating *system* independent but was originally developed for use in networks of

UNIX systems.



## Figure 3.3 NFS File system Architecture

* The NFS server module resides in the kernel on each computer that acts as an NFS server.

* Requests referring to files in a remote file system are translated by the client **module** to NFS protocol operations and then passed to the NFS server module at the computer holdingthe file system.

* The NFS client and server modules communicate using RPC. Sun’s RPC is configured to use either UDP or TCP.

* The RPC interface to the NFS server is open: any process can send requests to an NFS server; if the requests are valid and they include **valid** user credentials, they will be acted upon.

 **Virtual file system**

* It is possible to have multiple distributed file systems in a single architecture.

* In such case, the integration is achieved by a Virtual File System (VFS) module, which has been added to the UNIX kernel to distinguish between local and remote files and to translate between the UNIXindependent file identifiers used by NFS and the internal file identifiers normally used in UNIX and other file systems.

* In addition, VFS keeps track of the file systems that are currently available locally and remotely and it passes each request to the appropriate local system module.

* The file identifiers used in NFS are called file handles.

* It is derived from the following fields.

* + **File system identifier-** unique number allocated to each file system when it is created.

* + **I-node number** - number that serves to identify and locate the file within the file system.

* + **I-node generation number-** is needed because in the conventional UNIX file system, i-node numbers are reused after a file is removed. So a generation number is stored with each file and is incremented each time an i-node number is reused.

|  |  |  |
| --- | --- | --- |
| File System Identifier | I node Number | I node generation Number |

## Figure 3.4: File handle

* The virtual file system layer has one VFS structure for each mounted file system and one v- node per open file.
* A VFS structure relates a remote file system to the local directory on which it is mounted.

* The v-node contains an indicator to show whether a file is local or remote.
* If the file is local, v-node contains a reference to the index of the local file (i.e. i-node).

* If the file is remote, it contains the file handle of the remote file.

 **Client integration**

* The NFS client module plays the role described for the client module in all architectural models.

* But unlike our model client module, it emulates the semantics of the standard UNIX file system primitives and is integrated with the UNIX kernel.

* It is integrated with the kernel and not supplied as a library for loading into client processes so that:

 User programs can access files via UNIX system calls without recompilation or reloading.

* Single client module serves all of the user-level processes, with a shared cache of recently used blocks.

* The encryption key used to authenticate user IDs passed to the server can be retained in the kernel to prevent impersonation.

 **Access control and authentication**

* The NFS server is stateless and does not keep files open on behalf of its clients.

* So the server must check the user's identity against the file's access permission attributes on each request.

* *Kerberos* has been integrated with sun NFS to provide a stronger and more comprehensive solution to the

problems of user authentication and security

 **FS server interface**

A simplified representation of the RPC interface provided by the NFS server is shown below:

**lookup(dirfh, name) -» fh, attr** - Returns file handle and attributes for the file 'name ' in the directory 'dirfh'.

**create(dirfh, name, attr) -» newfh, attr** - creates a new file 'name' in the directory 'dirfh ' with attributes 'attr ' and returns the new file handle and attributes.

**remove(dirfh, name) -» status** - Removes file 'name ' from directory 'dirfh'

**getattr(fh) -» attr** - Returns the file attributes of file 'fh '.

**setattr(fh, attr) -»** sets the attributes

**read(fh, offset, count) -» attr, data** - Returns up to count bytes of data from a file starting at offset, Also, returns the latest attributes of the file.

**Write(fh, offset, count, data) -» attr** - writes 'count' bytes of data to a file starting at offset. Returns the latest attributes.

**Rename(dirfh, name, todirfh, toname) -» status** - changes the name of file 'name' in directory 'dirfh'to 'toname' in directory 'todirflx'.

**mkdir (dirfh, name, attr) —» newfh, attr** - creates a new directory 'name' with attributes 'attr ' and returns the new file handle and attributes.

**rmdir(dirfh, name) -» status** - Removes the empty directory 'name' from the parent directory 'dirfh'.

**statfs (fh ) -»fstats** - Returns file system information.

 **Mount Service**

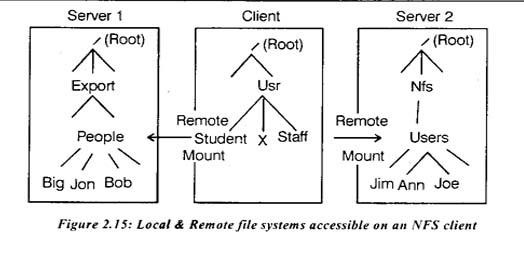
* The mounting of sub-trees of remote file systems by clients is supported by a separate mount service process that runs at user level on each NFS server computer.
* On each server, there is a file with a well- known name containing the names of local file systems that are available for remote mounting.

* An access list is associated with each file system name indicating which hosts are permitted to mount the file system.

* Client use a *mount* command to request mounting of a remote file system, specifying the remote host name, path name of a directory in the remote file system and the local name with which it is to be mounted.

* The mount command communicates with the mount service process on the remote host using a mount protocol.

* This is an RPC protocol and includes an operation that takes a directory pathname and returns the file handle of the specified directory if the client has access permission for the relevant file system.
* The location of the server and the file handle for the remote directory are passed on to the VFS layer and the NFS client.
* Figure 3.5 illustrates a client with two remotely mounted file stores. The nodes *'people'* and *'usr'* in file systems at server 1 and server 2 are mounted over nodes *'student’* and *staff* in client's local file store.
* The meaning of this is that programs running at client can access files at server 1 and server 2 by using pathnames such as *\usr\stuaent\Jon* and *usr\staff\Ann.*



***Figure 3.5: Local & Remote file systems accessible O. an .FS client***

* Remote file systems may be *hard-mounted* or *soft mounted* in a client computer.
* When a user-level process accesses a file in a file system that is hard-mounted, the process is suspended until the request can be completed and if the remote host is unavailable for any reason the NFS client module continues to retry the request until it is satisfied.

* But if the file system is soft mounted, the NFS client module returns a failure indication to user-level processes after a small number of retries.

 **Auto mounter**

* The automounter was added in order to mount a remote directory dynamically.

* It maintains a table of mount points (pathnames) with a reference to one or more NFS servers listed against each.

* It behaves like a local NFS server at the client machine.

* When the NFS client module attempts to resolve a pathname that includes one of these mount points, it passes a lookup( ) request to the local auto mounter which locates the required file system in its table and sends a *'probe '* requests to each server listed.
* The first server to respond is then mounted at the client using the normal mount service.

* The mounted file system is linked to the mount point using a symbolic link, file access then proceeds in the normal way without further reference to the auto mounter.

* **Server caching**

* + NFS servers use the cache at the server machine.

* + The use of servers’ cache does not raise any consistency problems; but when a server performs write operations, extra measures are needed to ensure that the clients can be confident that the results of write operations are persistent, even when server crashes occur.

* + In version 3 of NFS protocol, the write operation offers two options:

* + - Data in write operations received from clients is stored in the memory cache at the server and written to disk before a reply is sent to the client. This is called write-through caching. The client can be sure that the data is stored persistently as soon as the reply has been received.
    - Data in write operations is stored only in the memory cache. It will be written to disk when a *commit* operation is received for the relevant file.

* **Client caching**

* The NFS client module caches the results of read, write, getattr, lookup and readdir operations in order to reduce the number of requests transmitted to servers.

* Clients are responsible for polling the server to check the currency of the cached data that they hold.

* A timestamp-based method is used to validate cached blocks before they are used.

* Each data or metadata item in the cache is tagged with two timestamps namely TC, which is the time when the cache entry was last validated and Tm, which is the time when the block was last modified at the server.

* A cache entry is valid at time T if T-T is less than a freshness interval t, or if the value for Tm recorded at the client matches the value of Tm at the server.

Formally, the validity condition is:

(T - Tc < t) V (Tm\_client = Tm\_server )

* The selection of a value for t is a compromise between consistency and efficiency.

* Since NFS client cannot determine whether a file is being shared or not, the validation procedure must be used for all file accesses.

**3.4 INTRODUCTION TO 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, as well as to users.

* Users cannot communicate with one another via a distributed system unless they can name one another, e.g., e-mail address.

* Names are not the only useful means of identification, descriptive attributes are another.

* Sometimes clients do not know the name of the particular entity that they seek, but they do have some information that describes it.

**3.4.1 Names, addresses and other attributes**

* Any process that requires access to a specific resource must possess a name or an identifier for it.

* An object's address is a value that identifies the location of the object rather than the object itself.

* Addresses are efficient for accessing objects, but objects can sometimes be relocated, so addresses are inadequate as a means of identification.

* E.g., user's email address usually change when they move between organizations or ISPs; they are not enough to refer to a specific individual over time.

* A name is resolved when it is translated into data about the named resource or object, often in order to invoke an action upon it.

* The association between a name and an object is called ***binding****.*

* In general, names are bound to ***attributes***of the named objects.

* An attribute has the value of a property associated with an object.

* A key attribute of an entity that is usually relevant in a distributed system is its address.

Examples

* + - 1. DNS maps domain names to the attributes (its IP address, type of entry, etc . ) of a host

Computer.

* + - 1. X500 directory services c a n be used to map a person's name on to attributes including their e-

mail address and telephone number.

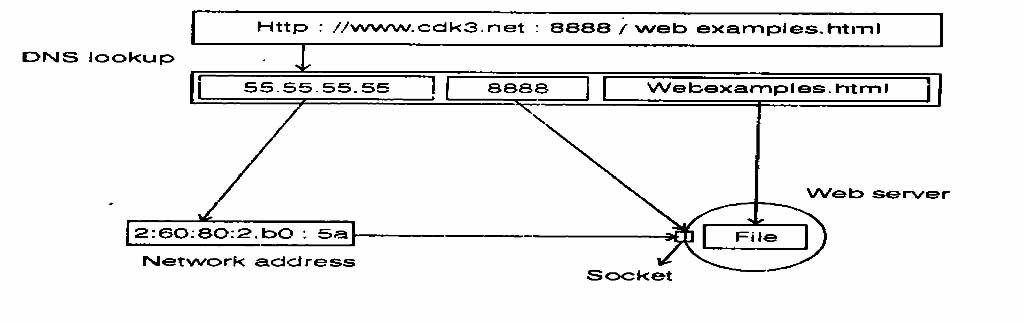
* Figure 2.16 shows the domain name portion of a URL resolved first via the DNS into an IP Address and then via ARP to an Ethernet address for the web server.

* The last part of the URL is resolved by the file system on the web server to locate the relevant file.

* Many of the names used in the distributed system are specific to some particular service.

* Names are also needed to refer to entities in a distributed system that is beyond the scope of any single service.

* The examples of these entities are users, computers and services themselves.



***Figure 3.6: Resolving VRL***

 **Uniform resource Identifiers (URI):**

Uniform Resource locators (URLs) are the principal means of identifying web resources. URLs are a particular type of URI.

* Because URLs are essentially addresses of web resources, they suffer from the disadvantage that if a resource is deleted or if it moves, say from one web s i t e to another, then there will be dangling links to the resource containing the old URL.
* If a user clicks on a dangling link, then the web server will either respond that the resource is not found or supplies a different resource that now occupies the same location.

* The other main type of URI is Uniform Resource Name (URN). URNs are intended to sole the dangling link problem and to provide richer model of finding resources on the web.

* The idea is for a web resource to have a URN" that persists, even though the resource may move.

The owner of a resource registers its name, along with i t s current URL. With a URN lookup service that will provide the URL when given the URN.

* The owner registers the new URL if the resource moves.

* Uniform Resource Characteristics (URC) is a subset of URNs.

* A URC is a description of a web resource consisting of attributes of the resource, such as *author* = *coulouris, keywords* = *network.*

* URCs are for describing web resources and for looking up web resources that match their attribute

Specification.

**3.5 NAME SERVICES AND THE DOMAIN NAME SYSTEM**

* A name service store collection of one or more naming contexts - sets of bindings between textual names and attributes for objects such as users, cars, computers, services and remote objects.

* The major operation that a name service supports is to resolve a name i.e., to look up attributes from a given name.

* The Global Name Service was developed with more ambitious goals including the following:

* To handle an essentially arbitrary number of names and to serve an arbitrary number of administrative organizations.
  + - * A long lifetime.
      * High availability.
      * Fault isolation.
      * Tolerance of mistrust.

**3.5.1 Name spaces**

* A name space is a collection of all valid names recognized by a particular service.

* Name spaces require a syntactic definition. Name may have an internal structure that represents their position in a hierarchic name space.

 The most important advantage of hierarchic name spaces is that each part of a name is resolved relative to a separate context, and the same name may be used with different meanings in difference contexts.

* In the file system each directory represents a context.

* Thus /net/pword is a hierarchic name with two components.
* The first *'net'* is resolved relative to the context '/' or root and the second part *'pword'* is relative to the context *'/net*'.

* The name /oldnet/pword can have a different meaning because its second component is resolved in different context.

* Similarly the same name /net/pword may resolve to different files in the contexts of two different computers.

* Hierarchic name spaces are potentially infinite, so they enable a system to grow indefinitely.

* Another potential advantage of a hierarchic name space is that different contexts can be managed by different people.

* The DNS (Domain name System) name space has a hierarchic structure: a domain name consists of one or more strings called *name components* or *labels,* separated by a delimiter V.

* There is no delimiter at the beginning or end of a domain name, the name components are non-null printable strings that does not contain V.

 DNS servers do not recognize relative names: all names are referred to the global root.

 **Aliases**

* An alias allows a convenient name to be substituted for a more complicated one.

The DNS allows aliases in which one domain name is defined to stand for another.

* The reason for having aliases is to provide for transparency.

* For example, aliases are generally used to specify the names of machines that run a web server or an FTP server.

* This has the advantage that clients can refer to the web server by a generic name that does not refer to a particular machine and if the web server is moved to another computer, then the alias has to be updated in DNS database.

 **Naming Domains**

* A naming domain is a name space for which there exists a single overall administrative authority for assigning names within it.
* This authority is in overall control of which names may be bound within the domain.

 **Combining and customizing name spaces:**

* DNS provides a global and homogeneous name space in which a given name refers to the same entity, no matter which process on which computer looks up the name.

* By contrast, some name services allow distinct name spaces - sometimes-heterogeneous name spaces to be embedded into them and some name services allow the name space to be customized to suit the needs of individual groups, users, or even processes.

**3.5.2. Name Resolution**

* Resolution is an iterative process whereby a name is repeatedly presented to naming contexts.

* A naming context either maps a given name onto a set of primitive attributes directly or it maps it onto a further naming context and a derived name to be presented to that context.

* Another example of the iterative nature of resolution is the use of aliases.

* Whenever a DNS server is asked to resolve an alias, the server first resolves the alias to another domain name

that must be further resolved to produce an IP address.

 **Name servers and navigation:**

* Any name service, such as DNS, that stores a very large database and is used by a large population will not store all of its naming information on a single server computer.
* Any heavily used name services should use replication to achieve high availability.

* The data belonging to a naming domain is usually stored by a local name server managed by the authority responsible for that domain.

* In some cases, a name server may store data for more than one domain, by partitioning the data.

* The partitioning of data implies that the local name server cannot answer all enquiries without the help of other name servers.

* The process of locating naming data from among more than one name server in order to resolve a name is called *navigation.*

* DNS supports the model known as *iterative navigation.*

* To resolve a name, a client presents the name to the local name server, which attempts to resolve i t .

* If the local name server has the name, it returns the result immediately. If it does not, it will suggest another server that will be able to help.

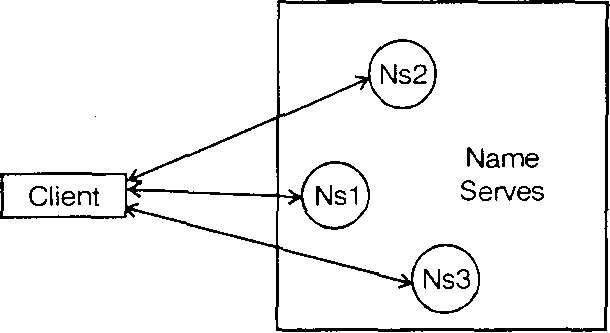
* Resolution proceeds at the new server, with further navigation as necessary until the name is located or it is discovered to be unbound.

* The DNS database is partitioned between servers in such a way as to allow many queries to be satisfied locally and others to be satisfied without needing to resolve each part of the name separately.

* In *multicast navigation,* client multicasts the name to be resolved and the required object type to the group of name servers.

* Only the server that holds the named attributes responds to the request.

* Another alternative to the iterative navigation model is one in which a name server coordinates the resolution of the name and passes the result back to the user agent.

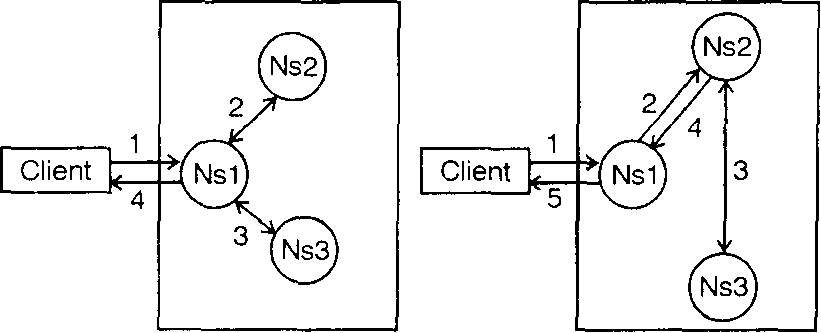


***Figure 3.7: Iterative .avigation***

* It included two types of navigation:
  + **Non-recursive server controlled navigation** –

Any name server may be chose by client. This server communicates by multicast or iteratively with its peers as though it is a client.

* + **Recursive server controlled navigation** - The client contacts a single server. If this server does not store the name, the server contacts a peer storing a prefix of the name, which in turn attempts to resolve it. This procedure continues recursively until the name is resolved.



Non - Recursive Recursive

Server- Controlled Server-Controlled

## Figure 3.8: Server controlled

 **Caching**

* In DNS and other name services, client name resolution software and servers maintain the cache of the results of previous name resolutions.

* When a client requests a name lookup, the name resolution software consults its cache.
* If it holds a recent result; it returns it to the client otherwise, it sets about finding it from a server.

* That server, in turn, may return data .ached from other servers.

**2.5.3 Domain name system (DNS)**

* DNS is primarily used for looking up host addresses and mail servers.

* In the following paragraphs, we concentrate on the topics DNS Name space, and the information stored in its nodes.

* The DNS is a name service design whose principal name database is used across the internet.
* It was devised to overcome the following shortcomings of the original Internet naming scheme.
* It did not scale to large number of computers.

* Local organizations wished to administer their own naming systems.

* A general name service was needed.

* In DNS, any type of object can be named and its architecture gives scope for a variety of implementations.
* Millions of names are bound by the Internet DNS and lookups are made against it from around the world. Any name can be resolved by any client.
* This is achieved by hierarchical partitioning of the name database, by replication of the naming data, and by caching.

* + **The D NS name Space**

* The DNS name space is hierarchically organized as a rooted tree. A label is a case insensitive string made up of alphanumeric characters.

* A label has a maximum length of 63 characters. The length of the complete path name is restricted to 255 characters.

* The string representation of a path name consists of listing i t s label, starting with the rightmost one, and seperating the labels by a dot (".").

* The root is represented by a dot.

* A subtree is called a domain; a path name to its root node is called a domain name.

* + **Domain names**
* The Internet DNS name space is partitioned both organizationally and according to geography.

* The names are written with the highest-level domain on the right.

* The top level organizational domains in use today across the Internet are
  + - com - Commercial Organizations.
    - edu - Universities and other educational institutions.
    - gov - US governmental agencies.
    - mil - US military organizations.
    - net - major network support centers.
    - org - organizations not mentioned above.
    - int - International organization.

In addition, every country has its own domains:

* us - united states
* uk - united kingdom
* fr - France

* **DNS queries**

The internet DNS is primarily used for simple host name resolution & for looking up electronic mail hosts, as follows:

* + - * + **Host name resolution** - to resolve host names into IP addresses. When a web browser is given a URL containing the domain name, it makes a DNS enquiry and obtains the corresponding IP address.
        + **Mail host location** - e-mail software uses the DNS to resolve domain names into the IP address of mail hosts. The DNS may return more than one domain name so that the mail software can try alternatives if the main mail host is unreachable for some reason.

* + - * + **Reverse resolution** - a domain name will be returned given an IP address.

* + - * + **Host information** - the DNS can store the machine architecture type and operating system against the domain names of hosts.

* + - * + **Well-known services** - A list of services run by a computer (e.g. telnet, FTP) and the protocol used to obtain them can be returned, given the computer's domain name.

* + - * + A **primary or master** server reads zone data directly from a master file locally. Secondary servers download zone data from a primary server.

* + - * + If a **secondary**' copy is out of date, the primary sends it the latest version.

* + - * + Each entry in a zone has a time-to-live value, when a non-authoritative server caches data from an authoritative server; it notes the time-to-live.

* + - * + It will only provide i t s cached data to clients for up to this time, when queried after the time period

has expired; it recontacts the authoritative server to check its data.

* **Navigation and query processing**

* + - A DNS client is called a *revolver.*

* + - It is implemented on library software.

* + - It accepts queries, formats them into messages, and communicates with one or more name servers in order to satisfy the queries.

* + - A simple request-reply protocol is used over UDP.

* + - The DNS architecture allows for recursive navigation as well as iterative navigation.

**3.6 DIRECTORY SERVICES**

* + - A service that stores collections of bindings between names and attributes and that look up entries that match attribute-based specifications is called a directory service.

* + - The directory service is used to keep track of the location of all resources in the system.

* + - These resources include machines, printers, servers, data, and much more, and they may be distributed geographically over the entire world.

* + - The directory service allows a process to ask for a resource and not have to be concerned about where it is.

* + - Unless the process cares. E.g., Microsoft's Active Directory services, X-500 Directory services are sometimes called *yellow pages* or *attribute-based name* services.

* + - A directory service returns attributes of any objects found to match some specified attributes.
    - A discovery service is a directory service that registers the services provided in a spontaneous networking environment.

* + - The requirement is for the set of clients and servers to change dynamically but to be integrated without user intervention.

* + - To meet these needs, a discovery service provides an interface for automatically registering and deregistering services, as well as an interface for clients to lookup the services that they require from those that are currently available.

* + - In discovery services, a context for discovery is sometimes called a *scope.*
* **Jini**

* + - * Recent developments in discovery services include the Jini discovery service.

* + - * Jini is a system that is designed to be used for spontaneous networking.

* + - * It is entirely Java-based. It assumes that JVMs run in all of the computers, allowing them to communicate with one another by means of RMI and to download code as necessary.

* + - * Jini provides facilities for service discovery, for transactions, for shared data spaces called Java spaces, and events.

* + - * The discovery-related components in a Jini system are lookup services, Jini services, and Jini clients.

* + - * The lookup service allows Jini services to register the services they offer, and Jini clients to request services that match their requirements.

* + - * A Jini service, such as a printing service, may be registered with one or more lookup services.

* + - * A Jim service provides, and the lookup services store, an object that provides the service as well as the attributes of the service.

* + - * Jini clients query lookup services to find Jini services that match their requirements; if a match is found, they download an object that provides the service from the lookup service.

* + - * When a Jini client or service starts up. It sends a request to the multicast address.

* + - Lookup services listen on a socket bound to the same address to receive such requests.
      * Any lookup service that receives the request and can respond to it replies with the unicast address on with it receives service requests, the requestor can then perform a remote invocation to lookup or register a service with it.

* + - * Lookup services also announce their existence through multicast address, so that the services and clients can learn about new lookup services.

* + - * When clients and services request a lookup service, they may specify any groups that they are particularly interested in, and only lookup services bound to those same group names will respond.

* + - * For example, in a company that is split into departments, a device that requires an *'admin '* printer can find one that belongs to the *'admin* ' group and a device that requires a *'finance'* printer will find one bound to the *'finance* ' group.

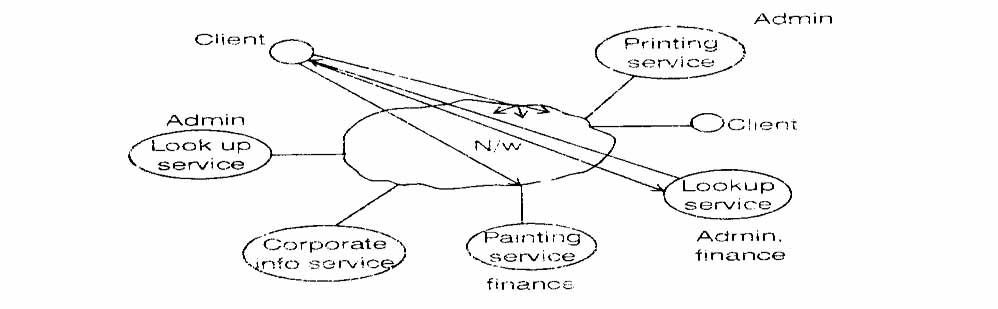
* + - * The client requires a lookup service in the finance group, so it multicasts a request bearing that group name. Only one lookup service is bound to the *'finance'* group and that service responds.

* + - * The response includes its address and the client communicates directly with it by RMI.

* + - * Only one printing service has registered itself under the finance group and an object to access that particular service I1-' returned.

* + - * The client then use.-, the printing service directly, using the returned object.

* + - * Figure 3.9 shows two printing services, one in the *'admin '* group and the other in the *'finance* ' group. There is also a corporate information service that is not bound to any particular group.

\*

## Figure 3.9: Service Discovery in Jini

3.7 **Case Study - Global Name Service**

A global naming service identifies (names) those enterprise-level networks around the world that are linked together via phone, satellite, or other communication systems. This world-wide collection of linked networks is known as the "Internet." In addition to naming networks, a global naming service also identifies individual machines and users within a given network.

FNS currently supports two global naming services:

* DNS. See ["FNS and DNS"](https://docs.oracle.com/cd/E19455-01/806-1387/6jam692di/index.html#af0intro-36277), below and ["Federating Under DNS"](https://docs.oracle.com/cd/E19455-01/806-1387/6jam692en/index.html).
* X.500/LDAP. See ["FNS and X.500"](https://docs.oracle.com/cd/E19455-01/806-1387/6jam692di/index.html#af0intro-27540), below and ["Federating Under X.500/LDAP"](https://docs.oracle.com/cd/E19455-01/806-1387/6jam692eo/index.html).

**Note -**

You can only federate a global naming service if your enterprise-level name service is NIS+ or NIS. If you are using a files-based name service for your enterprise, you cannot federate either DNS or X.500/LDAP.

[FNS and Global Naming Systems](https://docs.oracle.com/cd/E19455-01/806-1387/6jam692ej/index.html), for administration information regarding FNS and enterprise-level naming services.

## FNS and DNS

The Internet Domain Name System (DNS) is a hierarchical collection of name servers that provide the world Internet with host and domain name resolution. FNS uses DNS to name enterprise objects globally.

A domain name is the name DNS uses to identify an enterprise-level network (LAN or WAN). Networks using NIS+ permit creation of subdomains within the parent domain, and DNS can identify such subdomains.

Names can be constructed for any enterprise that is accessible on the Internet; consequently, names can also be constructed for objects exported by these enterprises. For more information about FNS and DNS, see ["Federating Under DNS"](https://docs.oracle.com/cd/E19455-01/806-1387/6jam692en/index.html).

## FNS and X.500

X.500 is a global directory service. Its components cooperate to manage information about objects in a worldwide scope. Such objects include countries, organizations, people, and machines. FNS federates X.500 to enable global access to enterprise name services. You can choose to use one of two APIs to access the X.500 global directory service:

* XDS/XOM API
* LDAP (Lightweight Directory Access Protocol) API.

# FNS and Applications

FNS supports:

* Solaris NFS file service (see ["FNS File Naming"](https://docs.oracle.com/cd/E19455-01/806-1387/6jam692dj/index.html#af0intro-37019), below).
* Printer naming (see ["FNS Printer Naming"](https://docs.oracle.com/cd/E19455-01/806-1387/6jam692dj/index.html#af0intro-38406)).
* Other applications (see ["FNS Application Support"](https://docs.oracle.com/cd/E19455-01/806-1387/6jam692dj/index.html#af0intro-33873)).

## FNS File Naming

FNS-based file naming integrates FNS naming into the Solaris file service. FNS-based file naming enables files to be named relative to users, hosts, sites, and organizations, using the FNS policies shared with other non-file applications.

FNS-based file naming gives clients a common view of the global and enterprise-wide file namespaces. Solaris applications that access the file system will, without modification, have access to the file namespaces supported by FNS.

## FNS Printer Naming

FNS-based printer naming provides the basic naming support for the unbundled Sun Microsystems Print Client (SSPC). FNS-based printer naming enables printers to be named relative to users, hosts, sites, and organizations, using the FNS policies shared with other non-printing-related applications.

FNS-based printer naming gives clients a common view of the global and enterprise-wide printer namespaces and allows centralized administration of the printer namespaces.

## FNS Application Support

Applications that are aware of FNS can expect the namespace to be arranged according to the FNS policies, and applications that bind names in the FNS namespace are expected to follow these policies.

Applications use FNS three ways:

* **Applications can be direct clients of the FNS interface and policies**. Application-level utilities such as the file system, the printing service, and the desktop tools (calendar manager, file manager) are examples of clients that use the FNS interface directly.
* **Applications can use FNS through existing interfaces**. A significant proportion of FNS use is through existing application programming interfaces. For example, consider a UNIX application that obtains a file name that it later supplies to the UNIX **open()** function. With FNS support for resolution of file names, the application need not be aware that the strings it deals with are composite names rather than the traditional local path names. Many applications can thereby support the use of composite names without modification.
* **Systems can export the FNS interface**. Naming systems, such as DNS and X.500, and naming systems embedded in other services, like the file system and printing service, are examples of naming systems that export the FNS interface.

# Administering FNS

FNS System administration varies according to the underlying naming service:

* **NIS+**. Under NIS+, FNS system administration tasks can only be performed by those with authorization to do so. The usual method of granting system administration privileges is to create an NIS+ group and assign that group the necessary privileges for that domain. Any member of the group can then perform system administration functions.
* **NIS**. Under NIS, FNS administration tasks must be performed by root on the NIS master server.
* **Files**. Under a files-based naming system, FNS administration tasks must be performed by someone with root access to the /var/fndirectory.

The ability of users to make changes to their own user sub-contexts varies according to the underlying naming service:

* **NIS+**. Under NIS+, a user's context (and associated sub-contexts) are owned by them. When logged in as an NIS+ principle, users who have the appropriate credentials and privileges can make changes to their own context using the fncreate, fnbind, fnunbind, and similar commands.
* **NIS**. Under NIS, users cannot make any changes to any FNS data. Only those with root access on the NIS master server can change FNS data.
* **Files**. Under a files-based naming system, users own their own contexts. Standard UNIX access controls apply to FNS files.