Jisy Raju
Assistant Professor, CE Cherthala

Module 4

Distributed file system: File service architecture – Network file system- Andrew file system- Name Service

4.1 Distributed file Systems: Introduction

A file system is a subsystem of the operating system that performs file management activities such as organization, storing, retrieval, naming, sharing, and protection of files. A distributed file system(DFS) is a method of storing and accessing files based in a client/server architecture. In a distributed file system, one or more central servers store files that can be accessed, with proper authorization rights, by any number of remote clients in the network.

Files contain both data and attributes. The data consist of a sequence of data items , accessible by operations to read and write any portion of the sequence. The attributes are held as a single record containing information such as the length of the file, timestamps, file type, owner's identity and access control lists. The shaded attributes in the Fig. are managed by the file system and are not normally update by user programs.

File attribute record structure

File length
Creation timestamp
Read timestamp
Write timestamp
Attribute timestamp
Reference count
Owner
File type
Access control list

The term metadata is often used to refer to all of the extra information stored by a file system that is needed for the management of files. It includes file attributes, directories and all the other persistent information used by the file system.

File system modules

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

4.1.1 Distributed file system requirements

-Transparency
-Concurrent file updates
-File replication
- Security
- heterogeneity
- Fault tolerance
- Consistency
- Efficiency

Transparency: as the concealment from the user and the application programmer of the separation of component in a DS.

- Access Transparency: Client programs should be unaware of the distribution of files. A single set of operations is provided for access to local and remote files.
- Location transparency: enable files to be accessed without knowledge of location
- Mobility transparency: allow the movement of file without affecting the operation of user
- Performance T: Client programs should continue to perform satisfactorily while the load on the service varies within a specified range.
- Scaling transparency: The service can be expanded by incremental growth to deal with a wide range of loads and network sizes.

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 : In a file service that supports replication, a file may be represented by several copies of its contents at different locations. It enhances fault tolerance by enabling clients to locate another server that holds a copy of the file when one has failed.

Heterogeneity : The service interfaces should be defined so that client and server software can be implemented for different operating systems and computers.

Fault Tolerance: Service continue to operate in face of failure.

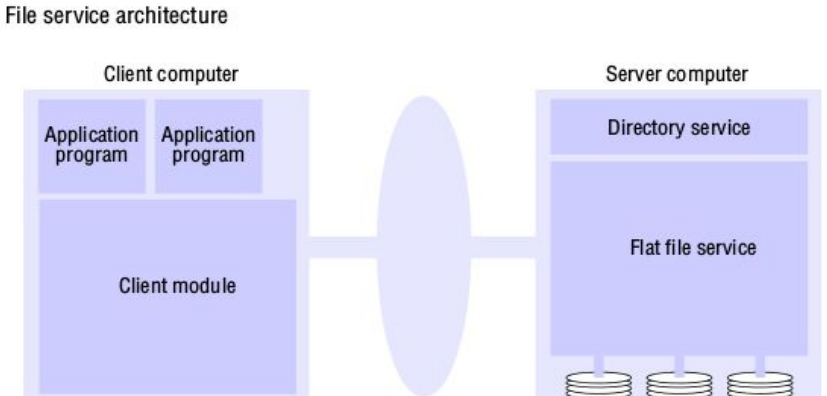
Security: In distributed file systems, there is a need to authenticate client requests so that access control at the server is based on correct user identities and to protect the contents of request and reply messages with digital signatures and (optionally) encryption of secret data.

Efficiency: Provide good level of performance

Consistency: If any changes made to one file, that changes must do in other replicated copies.

4.2 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 – a flat file service, a directory service and a client module.



Flat file service • The flat file service is concerned with implementing operations on the contents of files. Unique file identifiers (UFIDs) are used to refer to files in all requests for flat file service operations. 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.

In comparison with the UNIX interface, our flat file service has no open and close operations – files can be accessed immediately by quoting the appropriate UFID. The interface to our flat file service differs from the UNIX file system interface mainly for reasons of fault tolerance:

Repeatable operations: With the exception of Create, the operations are idempotent, Repeated execution of Create produces a different new file for each call.

Stateless servers: The interface is suitable for implementation by stateless servers. Stateless servers can be restarted after a failure and resume operation without any need for clients or the server to restore any state.

Figure 12.6 Flat file service operations

Read(FileId, i, n) → *Data* — throws *BadPosition*	If $1 \le i \le Length(File)$: Reads a sequence of up to *n* items from a file starting at item *i* and returns it in *Data*.
Write(FileId, i, Data) — throws *BadPosition*	If $1 \le i \le Length(File)+1$: Writes a sequence of *Data* to a file, starting at item *i*, extending the file if necessary.
Create() → *FileId*	Creates a new file of length 0 and delivers a UFID for it.
Delete(FileId)	Removes the file from the file store.
GetAttributes(FileId) → *Attr*	Returns the file attributes for the file.
SetAttributes(FileId, Attr)	Sets the file attributes (only those attributes that are not shaded in Figure 12.3).

Directory service • The directory service provides a mapping between text names for files and their UFIDs. Clients may obtain the UFID of a file by quoting its text name to the directory service.

Figure 12.7 Directory service operations

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

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.

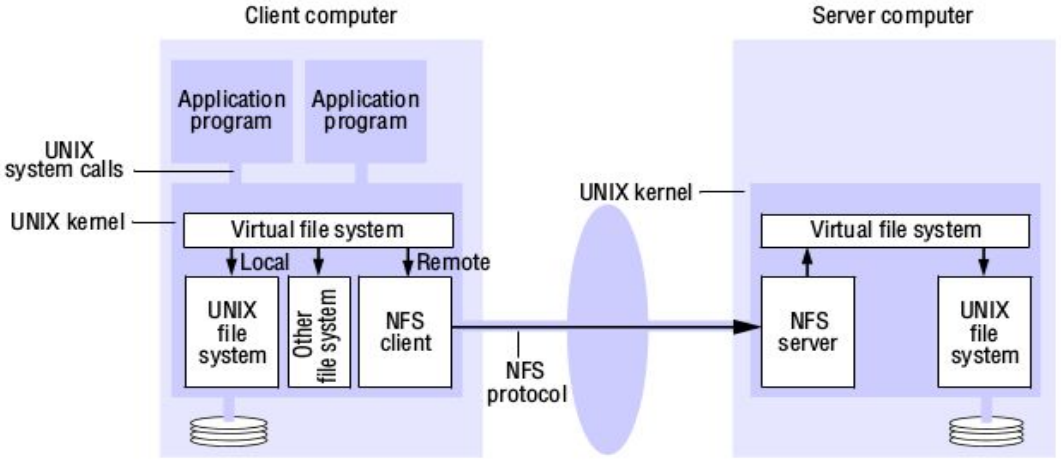
Access control:

- An access check is made whenever a file name is converted to a UFID
- A user identity is submitted with every client request, and access checks are performed by the server for every file operation.

Hierarchic file system • A hierarchic file system such as the one that UNIX provides consists of a number of directories arranged in a tree structure. Each directory holds the names of the files and other directories that are accessible from it.

File groups • 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.

4.3 Sun Network File System: Sun NFS



The NFS module resides in the kernel on each computer. 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 holding the relevant file system. The NFS client and server modules communicate using remote procedure calls. 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 : 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. The file identifiers used in NFS are called file handles.

File handle:	Filesystem identifier	i-node number of file	i-node generation number

- The filesystem identifier field is a unique number that is allocated to each filesystem when it is created.
- The i-node number is needed to locate the file in file sysytem and also used to store its attributeand i-node numbers are reused after a file is removed.
- The i-node generation number is needed to increment each time i-node numbers are reused after a file is removed.

The virtual file system layer has one VFS structure for each mounted file system and one v-node per open file. The v-node contains an indicator to show whether a file is local or remote.

Client Integration: The NFS client module cooperates with the virtual file system in each client machine. It operates in a similar manner to the conventional UNIX file system, transferring blocks of files to and from the server and caching the blocks in the local memory whenever possible. If the file is local, the v-node contains a reference to the index of the local file . If the file is remote, it contains the
file handle of the remote file.
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, to see whether the user is permitted to access the file in the manner requested.

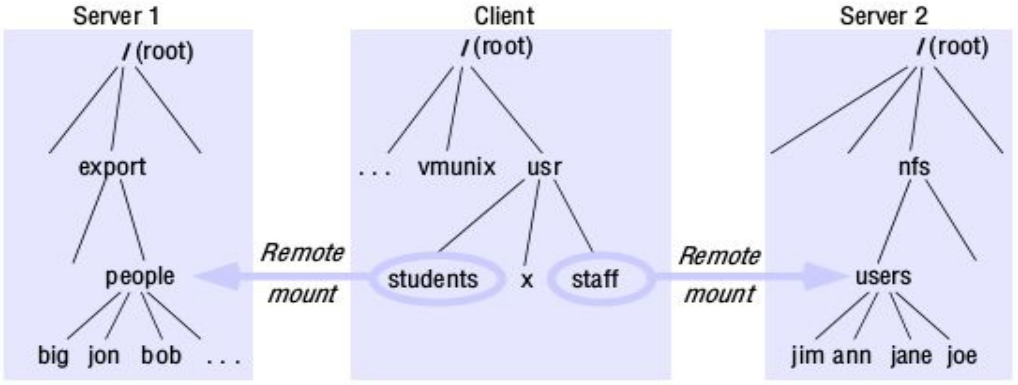
NFS server interface: The file and directory operations are integrated in a single service; the creation and insertion of file names in directories is performed by a single create operation, which takes the text name of the new file and the file handle for the target directory as arguments. The other NFS operations on directories are,

Figure 12.9 NFS server operations (NFS version 3 protocol, simplified)

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 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 file attributes of file *fh*. (Similar to the UNIX *stat* system call.)
setattr(fh, attr) → attr	Sets the attributes (mode, user ID, group ID, size, access time and modify time of a file). Setting the size to 0 truncates the file.
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 attributes of the file after the write has taken place.
rename(dirfh, name, todirfh, toname) → status	Changes the name of file *name* in directory *dirfh* to *toname* in directory *todirfh*.

Mount services: Mount is to make a group of files in a file system structure accessible to a user or user group.

Mount operation: mount(remotehost, remotedirectory, localdirectory)



Client with two remotely mounted file stores. The nodes people and users in filesystems at Server 1 and Server 2 are mounted over nodes students 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/students/jon and /usr/staff/ann.

Remote filesystems may be hard-mounted or soft-mounted in a client computer. When a user-level process accesses a file in a filesystem 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. Thus in the case of a server failure, user-level processes are suspended until the server restarts and then they continue just as though there had been no failure. But if the relevant filesystem is soft-mounted, the NFS client module returns a failure indication to user-level processes after a small number of retries.

Pathname translation : UNIX file systems translate multi-part file pathnames to i-node references in a step-by-step process. In NFS, pathnames cannot be translated at a server,

because the name may cross a ‘mount point’ at the client – directories holding different parts of a multi-part name may reside in filesystems at different servers. So pathnames are parsed, and their translation is performed in an iterative manner by the client. Each part of a name that refers to a emote-mounted directory is translated to a file handle using a separate lookup request to the remote server.

Automounter : The automounter was added to the UNIX implementation of NFS in order to mount a remote directory dynamically whenever an ‘empty’ mount point is referenced by a client.

Caching in both the client and the server computer are indispensable features of NFS implementations in order to achieve adequate performance.

Server caching: NFS servers use the cache at the server machine just as it is used for other file
accesses. The use of the server’s cache to hold recently read disk blocks does not raise any consistency problems; but when a server performs write operations, extra measures are needed to ensure that clients can be confident that the results of the write operations are persistent, even when server crashes occur. The write operation offers two options for this :

1. 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.

2. 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. The client can be sure that the data is persistently stored only when a reply to a commit operation for the relevant file has been received. Standard NFS clients use this mode of operation, issuing a commit whenever a file that was open for writing is closed.

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. 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:

-Tc is the time when the cache entry was last validated.

-Tm is the time when the block was last modified at the server.

Securing NFS with Kerberos •
The security
of NFS implementations has been strengthened by the use of the Kerberos scheme to
authenticate clients. In the original standard implementation of NFS, the user’s identity is included in
each request in the form of an unencrypted numeric identifier. NFS does not take any further steps to check the
authenticity of the identifier supplied. This implies a high degree of trust in the integrity of the client computer and its software by NFS, whereas the aim of Kerberos and other authentication-based security systems is to reduce to a minimum the range of components in which trust is assumed. Essentially, when NFS is used in a ‘Kerberized’ environment it should accept requests.

4.4 Andrew file system

An Andrew file system (AFS) is a location-independent file system that uses a local cache to reduce the workload and increase the performance of a distributed computing environment. Characteristics are,

Files are small (i.e. entire file can be cached)

- Frequency of reads much more than those of writes
- Sequential access common
- Files are not shared (i.e. read and written by only one user)
- Shared files are usually not written
- Disk space is plentiful

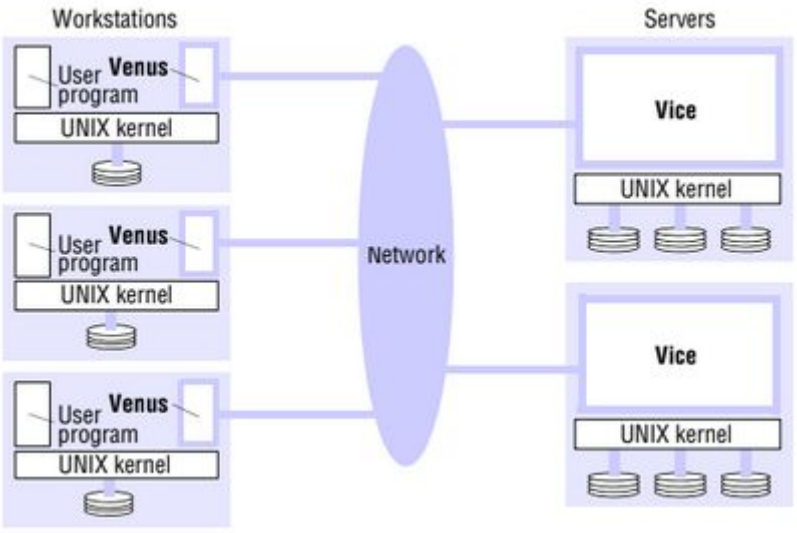
4.4.1 Implementation

The key software components in AFS are:

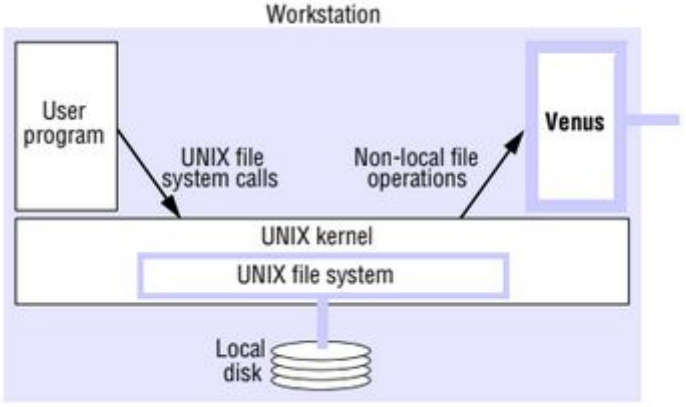
- Vice: The server side process that resides on top of the unix kernel, providing shared file services to each client
- Venus: The client side cache manager which acts as an interface between the application program and the Vice

The files available to user processes running on workstations are either local or shared. Local files are handled as normal UNIX files. They are stored on a workstation's disk and are available only to local user processes. Shared files are stored on servers, and copies of them are cached on the local disks of workstations.

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.

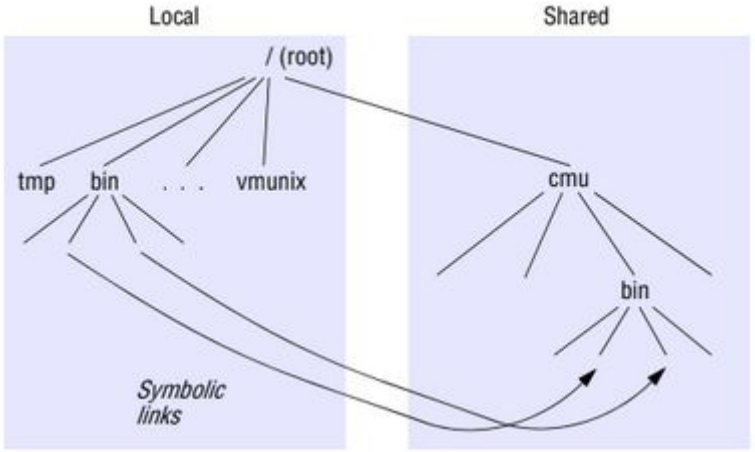


Step1: A first request for data to a server from a workstation is satisfied by the server and placed in a local cache.

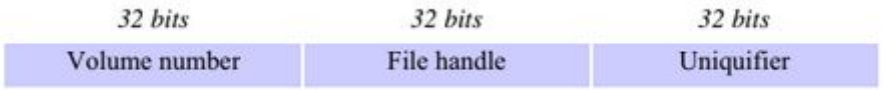


Step 2. A second request for the same data is satisfied from the local cache.

It is a conventional UNIX directory hierarchy, with a specific subtree (called cmu) containing all of the shared files. This splitting of the file name space into local and shared files leads to some loss of location transparency,



Files are grouped into volumes for ease of location and movement. Volumes are generally smaller than the UNIX filesystems. A flat file service is implemented by the Vice servers, Each file and directory in the shared file space is identified by a unique, 96-bit file identifier (fid) similar to a UFID. The Venus processes translate the pathnames issued by clients to fids.



In AFS, the server keeps track of which files are opened by which clients. AFS provides **location independence** (the physical storage location of the file can be changed, without having to change the path of the file, etc.) as well as **location transparency** (the file name does not hint at its physical storage location).

Stateful servers in AFS allow the server to inform all clients with open files about any updates made to that file by another client, through what is known as a **callback**. Callbacks to all clients with a copy of that file is ensured as a **callback promise** is issued by the server to a client when it requests for a copy of a file.

When Vice supplies a copy of a file to a Venus process it also provides a callback promise – a token issued by the Vice server that is guaranteeing that it will notify the Venus process when any other client modifies the file. Callback promises are stored with the cached files on the workstation disks and have two states:

- valid
- cancelled.

When a server performs a request to update a file it notifies all of the Venus processes to which it has issued callback promises by sending a callback to each – a callback is a remote procedure call from a server to a Venus process. When the Venus process receives a callback, it sets the callback promise token for the relevant file to cancelled.

Whenever Venus handles an open on behalf of a client, it checks the cache. If the required file is found in the cache, then its token is checked. If its value is cancelled, then a fresh copy of the file must be fetched from the Vice server, but if the token is valid, then the cached copy can be opened and used without reference to Vice.

When a workstation is restarted after a failure or a shutdown, Venus aims to retain as many as possible of the cached files on the local disk, but it cannot assume that the callback promise tokens are correct, since some callbacks may have been missed. Before the first use of each cached file or directory after a restart, Venus therefore generates a cache validation request containing the file modification timestamp to the server that is the custodian of the file. If the timestamp is current, the server responds with valid and the token is reinstated. If the timestamp shows that the file is out of

date, then the server responds with cancelled and the token is set to cancelled.

The basic file operations can be described more completely as:

User process	UNIX kernel	Venus	Net	Vice
open(FileName, mode)	If *FileName* refers to a file in shared file space, pass the request to Venus.	Check list of files in local cache. If not present or there is no valid *callback promise*, send a request for the file to the Vice server that is custodian of the volume containing the file.	→	Transfer a copy of the file and a *callback promise* to the workstation. Log the callback promise.
	Open the local file and return the file descriptor to the application.	Place the copy of the file in the local file system, enter its local name in the local cache list and return the local name to UNIX.	←	
read(FileDescriptor, Buffer, length)	Perform a normal UNIX read operation on the local copy.			
write(FileDescriptor, Buffer, length)	Perform a normal UNIX write operation on the local copy.			
close(FileDescriptor)	Close the local copy and notify Venus that the file has been closed.	If the local copy has been changed, send a copy to the Vice server that is the custodian of the file.	→	Replace the file contents and send a *callback* to all other clients holding *callback promises* on the file.

4.5 NAME

In a distributed system names are used to refer to a wide variety of resources such as computers, services, remote objects, and files as well as users. Names are used for identification as well as for describing attributes.

Names = strings used to identify objects (files, computers, people, processes, objects)

- **Textual names:** Human-readable names are file names such as /etc/passwd, URLs such as http://www.cdk5.net/ and Internet domain names such as www.cdk5.net.
- **Numeric addresses:** , e.g. 193.206.186.100 (IP host address)
- **Object identifiers**: object's address: a value that identifies the location of the object rather than the object itself.

For many purposes, names are preferable to identifiers, because the binding of the named resource to a physical location is deferred and can be changed and also they are more meaningful to users.

Currently, different name systems are used for each type of resource:

- file-pathname

- process-process id
- Port-port number

Uniform Resource Identifiers (URI) offer a general solution for any type of resource. There two main classes:
-URL: Uniform Resource Locator
• typed by the protocol field (http, ftp, nfs, etc.)
• part of the name is service-specific
• resources cannot be moved between domains

-URN: Uniform Resource Name
• requires a universal resource name lookup service

Format: urn: <nameSpace>:<name-within namespace>

- Examples:

a) urn:ISBN:021-61918-0

b) urn:dcs.qmul.ac.uk :TR2007-5

a)send a request to nearest ISBN-lookup service - it would return whatever attributes of a book are required by the requester

b)send a request to the urn lookup service at dcs.qmul.ac.uk
- it would return a url for the relevant document

4.5 Name services

Name System (or Service) an Internet service that translates textual names and attributes for objects.

Examples of Name Services

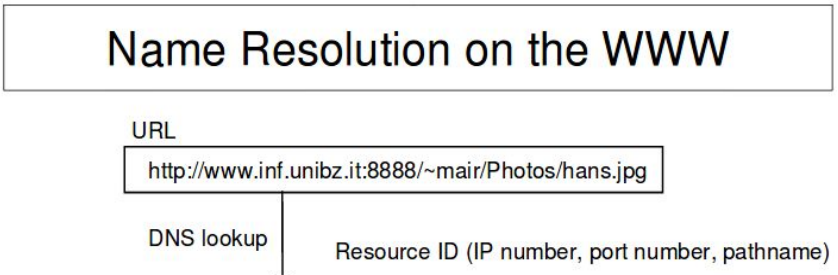
- File system: maps file name to file
- RMI registry:binds remote objects to symbolic names
- DNS: maps domain names to IP addresses
- X.500/LDAP directory service: maps person's name to email address, phone number

Every time you use a domain name, therefore, a DNS service must translate the name into the corresponding IP address. For example, the domain name www.example.com might translate to 198.105.232.4.

Name management is separated from other services largely because of the openness of distributed systems, which brings the following motivations:

- Unification: It is often convenient for resources managed by different services to use the same naming scheme. URIs are a good example of this.

- Integration: It is not always possible to predict the scope of sharing in a distributed system. It may become necessary to share and therefore name resources that were created in different administrative domains.



Three Design issues,

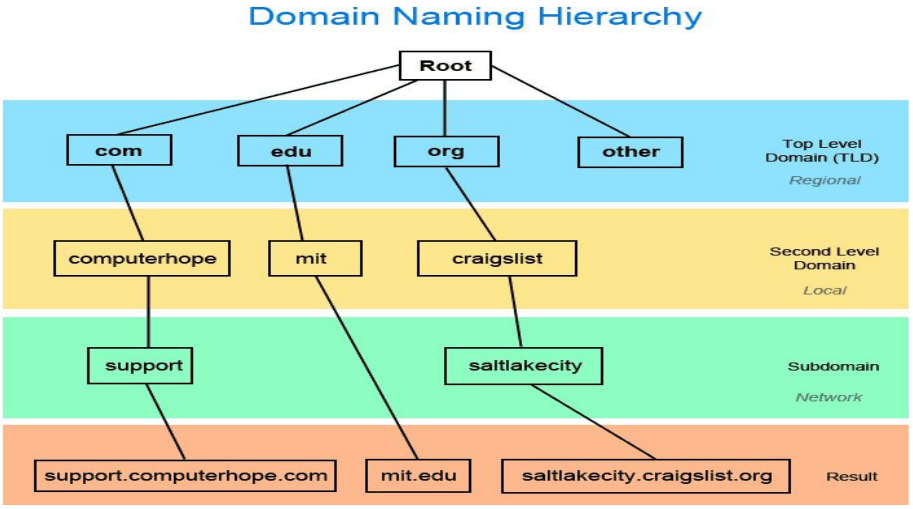
- Name spaces
- Name Resolution
- The domain name system

4.5.1 NAMESPACE AND DOMAIN NAME SYSTEM

DNS is the name service provided by the Internet for TCP/IP networks. DNS is broken up into domains, a logical organization of computers that exist in a larger network. Names may have an internal structure that represents their position,

- Hierarchy Namespace
- Flat name space:Single global context and naming authority for all names. So difficult to manage

Naming domains • A naming domain is a name space for which there exists a single overall
administrative authority responsible for assigning names within it.eg.- .net, .comThe domains exist at different levels and connect in a hierarchy that resembles the root structure of a tree. Each domain extends from the node above it, beginning at the top with the root-level domain. Under the root-level domain are the top-level domains, under those are the second-level domains, and on down into subdomains. DNS namespace identifies the structure of the domains that combine to form a complete domain name. For example, in the domain name sub.secondary.com, "com" is the top-level domain, "secondary" identifies the secondary domain name (commonly a site hosted by an organization and/or business), and "sub" identifies a subdomain within the larger network. This entire DNS domain structure is called the DNS namespace. The name assigned to a domain or computer relates to its position in the namespace.



Aliases • An alias is a name defined to denote the same information as another name.
eg.- http://espn.go.com/ and http://www.espn.com

Combining and customizing name spaces • The 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.

Merging:how to merge the entire UNIX file systems of two (or more) computers called red and blue. Each computer has its own root, with overlapping file

names. For example, /etc/passwd refers to one file on red and a different file on blue. The obvious way to merge the file systems is to replace each computer's root with a 'super root' and mount each computer's file system in this super root, say as /red and /blue. Users and programs can then refer to /red/etc/passwd and /blue/etc/passwd.

4.5.2 Domain name system(DNS)

Domain names

• The DNS is designed for use in multiple implementations, each of which may have its own name space. In practice, however, only one is in widespread use, and that is the one used for naming across the Internet. 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 original top-level organizational domains (also called generic domains) in use across the Internet were:

- com - Commercial organizations
- edu- Universities and other educational institutions
- gov- governmental agencies
- mil- military organizations
- net- Major network support centres
- org- Organizations
- int- International organizations

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 and for looking up electronic mail hosts, as follows

- Host name resolution:when a web browser is given a URL containing the domain name www.dcs.qmul.ac.uk, it makes a DNS enquiry and obtains the corresponding IP address.
- Mail host location: Electronic mail software uses the DNS to resolve domain names into the IP addresses of mail hosts. For example, when the address tom@dcs.rnx.ac.uk is to be resolved, the DNS is queried with the address dcs.rnx.ac.uk and the type designation 'mail'. It returns a list of domain names of hosts that can accept mail for dcs.rnx.ac.uk

Some other types of query that are implemented in some installations but are less frequently used than those just given are:

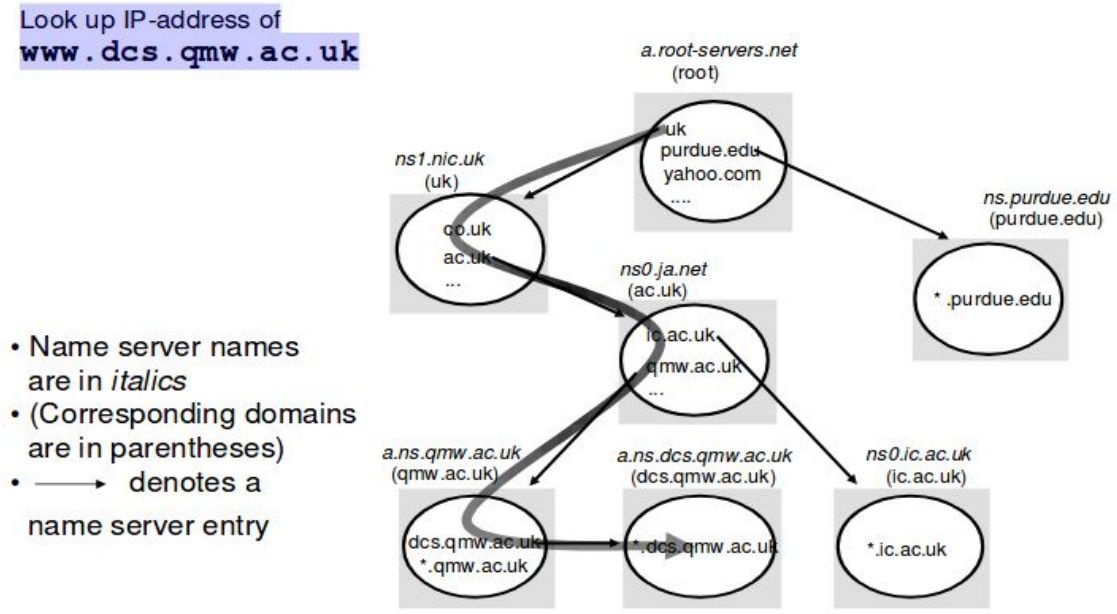
- Reverse resolution: Some software requires a domain name to be returned given an IP address.
- Host information: The DNS can store the machine architecture type and operating system with the domain names of hosts.

DNS name servers: All host names and addresses in one large master file stored on one central host.
Every domain name, which is a part of the DNS system, has several DNS settings, also known as DNS records. In order for these DNS records to be kept in order, the DNS zone

was created. Every zone must have at least two name servers

- exactly one master (= primary) server: contains the only writable copy of the "zone file"
- one or more secondary (= slave) servers: copies its zone file from the master

Example: DNS Servers:
Look up IP-address of www.dcs.qmw.ac.uk



DNS server caching the lookup result for a limited time, known as its Time To Live (**TTL**), ranging from a few minutes to a few days. People managing a DNS server can configure its TTL, so TTL values will vary across the Internet.

DNS resource records:

DNS holds resource records (RR).

RR format: (name, class,value, type,ttl)

Record type	*Meaning*	*Main contents*
A	A computer address (IPv4)	IPv4 number
AAAA	A computer address (IPv6)	IPv6 number
NS	An authoritative name server	Domain name for server
CNAME	The canonical name for an alias	Domain name for alias
SOA	Marks the start of data for a zone	Parameters governing the zone
PTR	Domain name pointer (reverse lookups)	Domain name
HINFO	Host information	Machine architecture and operating system
MX	Mail exchange	List of <*preference, host*> pairs
TXT	Text string	Arbitrary text

BIND *or Berkeley Internet Name Domain*, is most widely used Open source software that implements DNS protocols for internet, which provides us ability to implement IP to domain name conversion & vice-versa .

4.5.3 Name resolution (Name resolver)

DNS name resolution is nothing but resolving host names, such as www.nixcraft.com, to their corresponding IP addresses. DNS works as the “phone book” for the Internet by translating hostname into IP address or vise versa.

Name servers and navigation

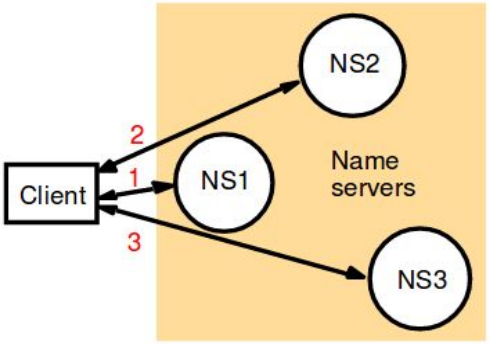
DNS, that stores a very large database and is used by a large population will not store all of its naming information on a single server computer. The process of locating naming data from more than one name server in order to resolve a name is called navigation.

- iterative navigation
- Multicast navigation
- server control navigation

-Recursive navigation
-Nonrecursive navigation

Iterative navigation:

To resolve a name, a client presents the name to the local name server, which attempts to resolve it. 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. DNS supports the model known as iterative navigation. Resolution continues until name resolved or name found to be unbound.



Example: If you enter www.example.com in the browser, the operating system's resolver will send this query for the record to the DNS server NS1. On receiving the query, it will look through its tables(cache) to find the IP address for the domain www.example.com. But if it does not have the entry then NS1 will reply back to client with a referral to another servers. Then operating system resolver, will send the query to NS2. And it continues the process until it resolved.

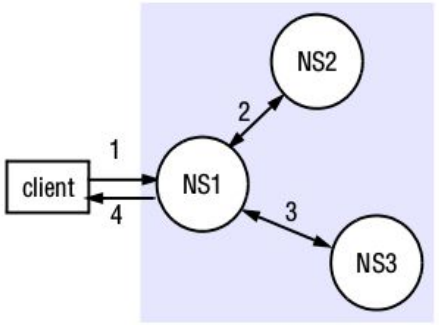
Multicast navigation:

In multicast navigation, a 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.

Non-recursive server-controlled navigation

Under non-recursive server-controlled navigation, any name server may be chosen by the client. This server communicates by multicast or iteratively with its peers

in the style described above, as though it were a client.

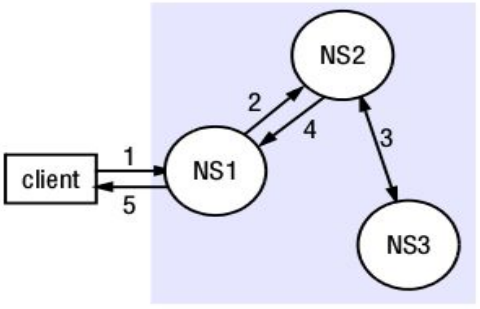


Non-recursive
server-controlled

Example: If you enter www.example.com in the browser, the operating system's resolver will send this query for the record to the DNS server NS1. On receiving the query, it will look through its tables(cache) to find the IP address for the domain www.example.com. But if it does not have the entry then NS1 contacts peers if it cannot resolve name itself by multicast or iteratively by direct contact. Answer for the query will send back to client by NS1.

Recursive server-controlled navigation:

Under recursive server-controlled navigation, the client once more contacts a single server. If this server does not store the name, the server contacts a peer storing a (larger) prefix of the name, which in turn attempts to resolve it. This procedure continues recursively until the name is resolved.



Recursive
server-controlled

Example: If you enter www.example.com in the browser, the operating system's resolver will send this query for the record to the DNS server NS1. On receiving the query, it will look through its tables(cache) to find the IP address for the domain www.example.com. But if it does not have the entry then NS1 contacts NS2. If NS2 does not have the entry then send the query to NS3. This procedure continues recursively until the name is resolved. Answer for the query will send back to client by NS1.

Caching • In DNS and other name services, client name resolution software and servers maintain a cache of the results of previous name resolutions. When a client requests a name lookup, the name resolution software consults its cache.

4.5.4 Directory services

A directory service is the collection of software and processes that store information (name,attribute). An example of a directory service is the Domain Name System (DNS), which is provided by DNS servers. A DNS server stores the mappings of computer host

names and other forms of domain name to IP addresses. A DNS client sends questions to a DNS server about these mappings. Thus, all of the computing resources (hosts) become clients of the DNS server. The mapping of host names enables users of the computing resources to locate computers on a network, using host names rather than complex numerical IP addresses.

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, eg. X.500, LDAP, MS Active Directory Services. However, any organization that plans to base its applications on web services will find it more convenient to use a directory service to make these services available to clients. This is the purpose of the Universal Description, Discovery and Integration (UDDI). UDDI provides both white pages and yellow pages services (a white pages service by name or a yellow pages service by attribute(IP)).

Discovery service: a special case of a directory service for services provided by devices in a spontaneous networking environment

- automatically updated as the network configuration changes
- discovers services required by a client (who may be mobile) within the current scope, for example, to find the most suitable printing service for image files after arriving at a hotel

4.5.5 Case study: The Global Name Service

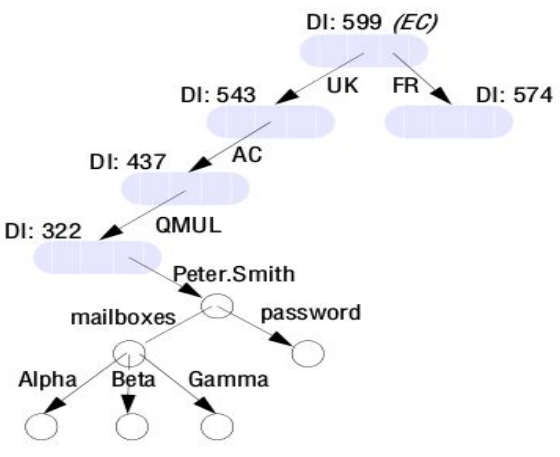
Designed and implemented by Lampson and colleagues at the DEC Systems Research Center (1986). Mainly used to merge two more name servers.

- Also Provide facilities for resource location, email addressing and authentication

The GNS manages a naming database that is composed of a tree of directories holding names and values. Directories are named by multi-part pathnames referred to a root, or relative to a working directory, much like file names in a UNIX file system. Each directory is also assigned an integer, which serves as a unique *directory identifier* (DI) and EC is directory. A directory contains a list of names and references. The values stored at the leaves of the directory tree are organized into *value trees*, so that the attributes associated with names can be structured values.

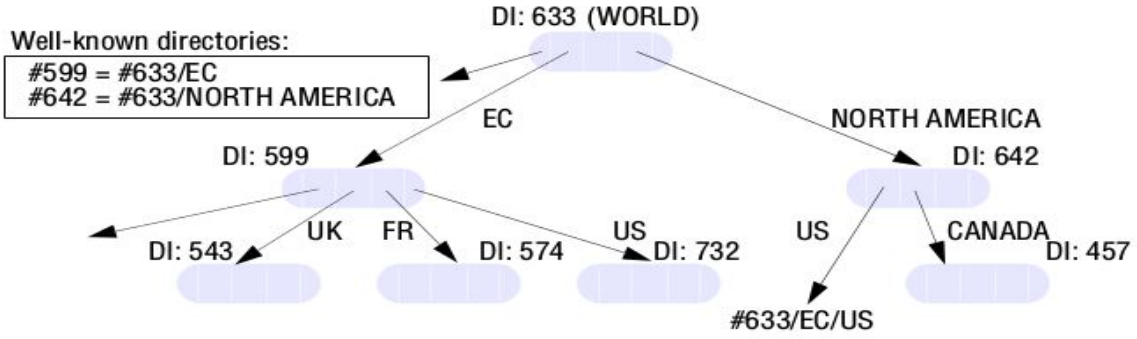
Names in the GNS have two parts: <*directory name*, *value name*>. The first part identifies a directory; the second refers to a value tree, or some portion of a value tree.

Mechanism -> add a new root node and make the exiting root node its children



Eg.The attributes of a user Peter.Smith in the directory QMUL would be stored in the value tree named <EC/UK/AC/QMUL, Peter.Smith>. The value tree includes a password, which can be
referenced as <EC/UK/AC/QMUL, Peter.Smith/password>, and several mail addresses, each of which would be listed in the value tree as a single node with the name <EC/UK/AC/QMUL, Peter.Smith/mailboxes>.

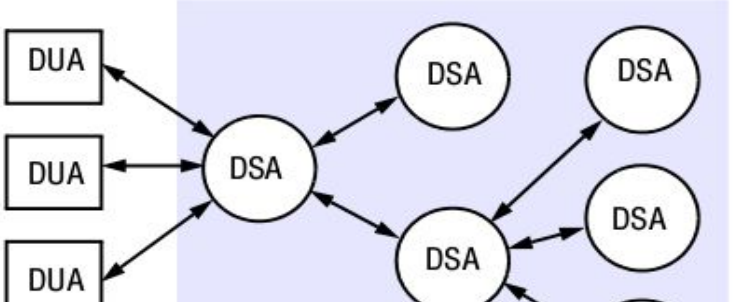
At the level of clients and administrators, growth is accommodated through extension of the
directory tree in the usual manner. But we may wish to integrate the naming trees of two previously separate GNS services.For example, how could we integrate the database rooted at the EC directory shown in above Figure with another database for NORTH AMERICA.Below Figure shows a new root, WORLD, introduced above the existing roots of the two trees to be merged. Here only Problem is Existing names need to be changed.



For example, </UK/AC/QMUL, Peter.Smith> is a name used by clients before integration. The root it refers to is EC, not WORLD. EC and NORTH AMERICA are working roots – initial contexts against which names beginning with the root '/' are to be looked up.The existence of unique directory identifiers can be used to solve this problem.
The working root for each program must be identified as part of its execution environment (much as is done for a program's working directory). When a client in the European Community uses a name of the form </UK/AC/QMUL, Peter.Smith>, its local user agent, which is aware of the working root, prefixes the directory identifier EC(#599), thus producing the name <#599/UK/AC/QMUL, Peter.Smith>.

4.5.6 Case study: The 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. Such a directory is sometimes called a global White Pages directory. The X.500 directory is organized under a common "root" directory in a "tree" hierarchy of: country,organization, organizational unit, and person.



In X.500, each local directory is called a Directory System Agent (DSA). A DSA can represent one organization or a group of organizations. The X.500 name tree is called the Directory Information Tree (DIT), and the
entire directory structure including the data associated with the nodes, is called the

Directory Information Base (DIB). The user interface program for access to one or more DSAs is a Directory User Agent (DUA). The University of Michigan is one of a number of universities that use X.500 as a way to route e-mail as well as to provide name lookup, using the Lightweight Directory Access Protocol (LDAP).