RGPVNOTES.IN

Subject Name: **Distributed System**

Subject Code: **IT-6005**

Semester: **6th**

UNIT 3

Distributed Objects and Remote Invocation

In a distributed object system, an object's data should be accessible only via its methods;

- Objects can be accessed via object references.
- an interface provides a definition of the signatures of a set of methods without specifying their implementation;
- An action is initiated by an object invoking a method in another object.
- The state of an object consists of the values of its instance variables.
- Remote invocations are method invocations for objects in different processes (in the same machine or not).
- Remote objects are objects that can receive remote invocation.
- heart of the distributed object model:
  - remote object reference;
  - remote interface: specifies which methods can be invoked remotely;

Communication between distributed objects

The communication in distributed object is done by various middleware language like RMI (remote invocation method), CORBA (common object request broker). Invoking a method on a remote object is known as RMI or remote invocation, and is the object oriented programming analog of an RPC. Distributed object communication realizes communication between distributed objects. The widely used approach on how to implement the communication channel is realized by using stub and skeletons shown in fig 3.1.

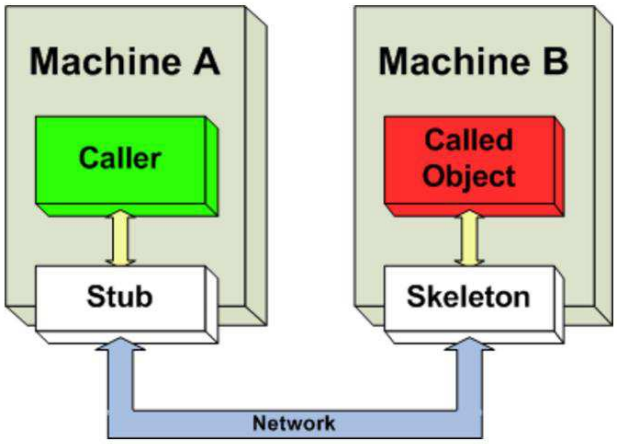


Fig 3.1 Communication between distributed objects

Stubs and Skeletons:

Stubs: RMI uses a standard mechanism (employed in RPC systems) for communicating with remote objects, stubs and skeletons. A stub for a remote object acts as a client's local representative or proxy for the remote object. The caller invokes a method on the local stub which is responsible for carrying out the method call on the remote object. In RMI, a stub for a remote object implements the same set of remote interfaces that a remote object implements.

When a stub's method is invoked, it does the following:

- initiates a connection with the remote JVM containing the remote object,
- marshals (writes and transmits) the parameters to the remote JVM,
- waits for the result of the method invocation,
- unmarshals (reads) the return value or exception returned, and
- returns the value to the caller.

The stub hides the serialization of parameters and the network-level communication in order to present a simple invocation mechanism to the caller.

Skeleton: In the remote JVM, each remote object may have a corresponding skeleton (in Java 2 platform-only environments, skeletons are not required). The skeleton is responsible for dispatching the call to the actual remote object implementation. When a skeleton receives an incoming method invocation it does the following:

- unmarshals (reads) the parameters for the remote method,
- Invokes the method on the actual remote object implementation, and marshals (writes and transmits) the result (return value or exception) to the caller.

Programming models for distributed programs/applications: applications composed of cooperating programs running in several different processes. Such programs need to invoke operations in other processes.

- **RPC** – client programs call procedures in server programs, running in separate and remote computers (e.g., Unix RPC).
    - Extended from procedure call
- **RMI** – extensions of object-oriented programming models to allow a local method (of a local object) to make a remote invocation of objects in a remote process (e.g., Java RMI).
    - Objected oriented version of RPC
- **EBP** (event-based programming) model – allows objects anywhere to receive notification of events that occur at other objects in which they have registered interest (e.g., Jini EBP).

RPC (Remote Procedure Call):

- RPC allows clients to call procedures in servers running on remote hosts. Parameters and results are packed in messages that are passed between the client and the server
- RPC offers access transparency. Clients call local procedures and remote procedures in the same way.
- Message passing is invisible to the programmer. Message passing is hidden in two library procedures- the client stub and the server stub (shown in fig 3.2).
- Example RPC systems: Sun RPC, DCE RPC, XML-RPC

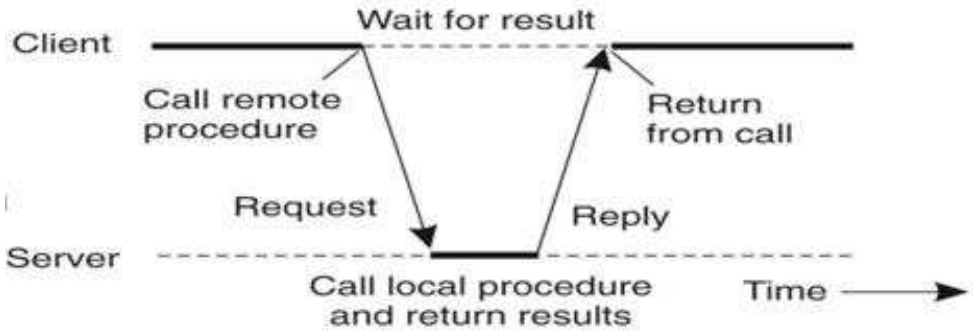


Fig 3.2 RPC between client and Server

RPC Steps:

1. The client process calls the client stub, which resides within the client's address space.
2. The client stub packs the parameters into a message. This is called marshaling. The client stub then executes a system call (e.g., sendto) to send the message.
3. The kernel sends the message to the remote server machine.
4. The server stub receives the message from the kernel.
5. The server stub unmarshals the parameters.
6. The server stub calls the desired procedure.
7. The server process executes the procedure and returns the result to the server stub.
8. The server stub marshals the results into a message and passes the message to the kernel.
9. The kernel sends the message to the client machine.
10. The client stub receives the message from the kernel.

11. The client stub unmarshals the results and passes them to the caller. (shown in fig 3.3)

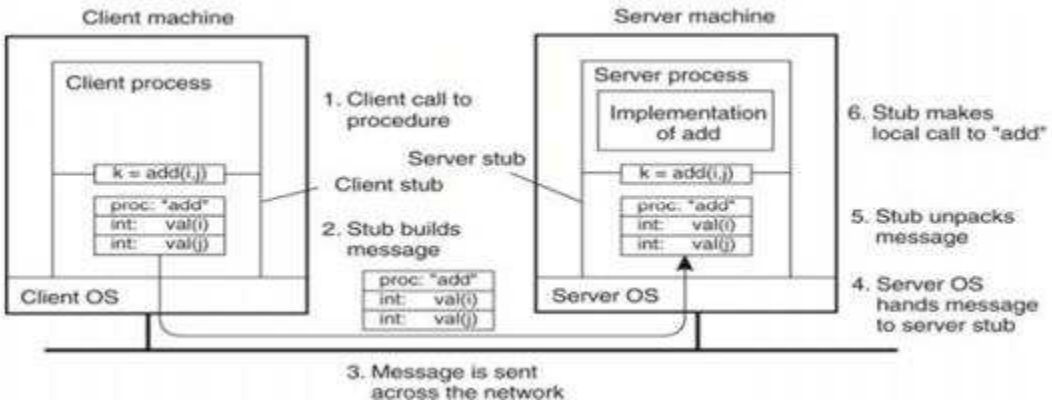


Fig 3.3 the steps involved in a RPC

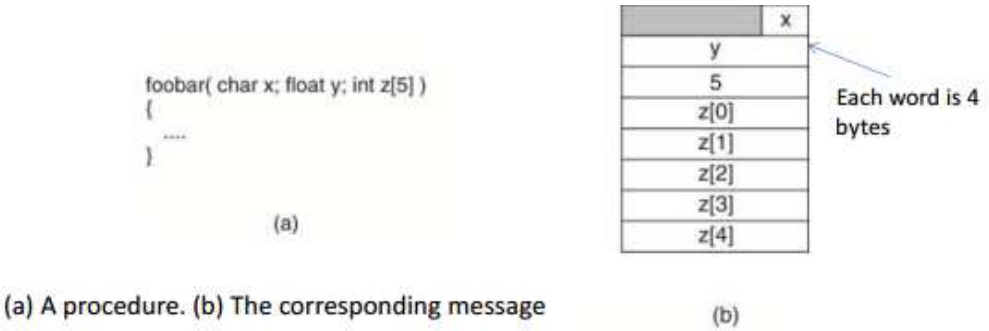
Marshaling and Unmarshaling:

- Marshaling is the process taken by client stub/server stub to pack parameters/return values into a message
  - A message contains a sequence of bytes, which can be represented by an array of unsinged chars in C
- Marshalling includes converting the representation of the parameters into a standard format, and copying each parameter into the message
  - Computers differ in host byte order and in representation of various data types → need convert parameters from the local format to a standard format
  - External Data Representation (XDR) (RFC 4506) is a standard data representation format used by many systems (e.g., NFS, ONC RPC)
- During unmarshaling, parameters in the message should be converted from the standard format to the local format

RPC Protocol:

Client and server must follow the same RPC protocol, which specifies

- The format of the messages exchanged
- The representation of various data types (e.g., integers in two's complement, characters in ASCII, floats in IEEE standard #754, with everything stored in big endian)
- The transport protocol used for message exchange (e.g., TPC, UDP, HTTP) shown in fig 3.4.

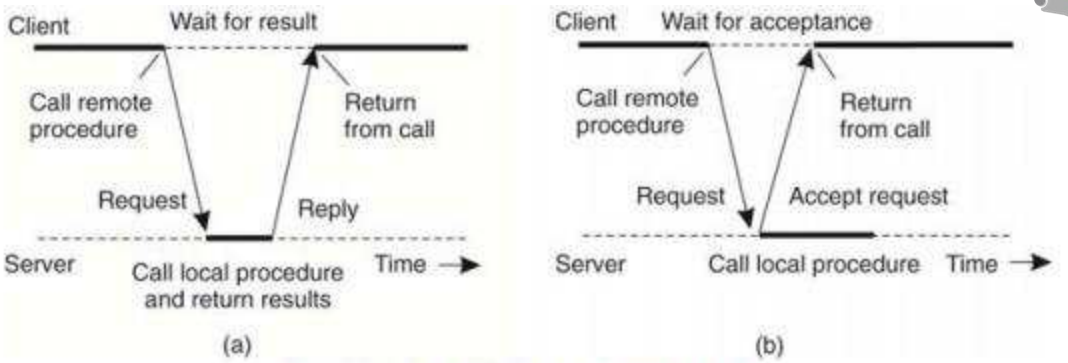


(a) A procedure. (b) The corresponding message

Fig 3.4 RPC protocols

Conventional and Asynchronous RPC:

- Conventional RPC is synchronous: client blocks until a reply is returned, shown in fig 3.5(a).
- Asynchronous RPC
  - RPC returns as soon as the server acknowledges acceptance of the request message
  - Useful when there is no need to wait for a reply (e.g., there is no result to return). shown in fig 3.5(b)



(a) Conventional RPC. (b) Asynchronous RPC.

Fig 3.5 conventional and Asynchronous RPC

Implementing RPC:

RPC is generally implemented over a request-reply protocol that supports two-way exchange of messages in client-server interactions

The protocol is based on 3 communication primitives

- doOperation: used by a client to invoke a remote operation. doOperation sends a request message to the remote server and returns the reply message
- getRequest: used by a server to acquire request messages
- sendReply: used by a server to send the reply message to the client after it has invoked the operation specified in the request message. Shown in fig 3.6.

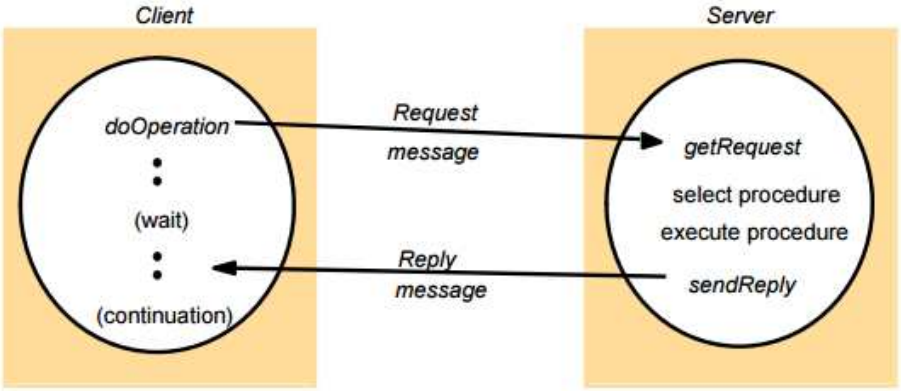


Fig 3.6 RPC message passing

Events and Notifications:

The idea behind the use of events is that one object can react to a change occurring in another object. The actions done by the user are seen as events that cause state changes in objects. The objects are notified whenever the state changes. Local event model can be extended to distributed event-based systems by using the publish-subscribe paradigm.

- In publish-subscribe paradigm
  - An object that has event publishes.
  - Those that have interest subscribe.
- Objects that represent events are called notifications.
- Distributed event-based systems have two main characteristics:
  - Heterogeneous – Event-based systems can be used to connect heterogeneous components in the Internet.
  - Asynchronous – Notification are sent asynchronously by event-generating objects to those subscribers.

A dealing room system could be modeled by processes with two different tasks (Figure 3.7):

An information provider process continuously receives new trading information from a single external source and applies to the appropriate stock objects.

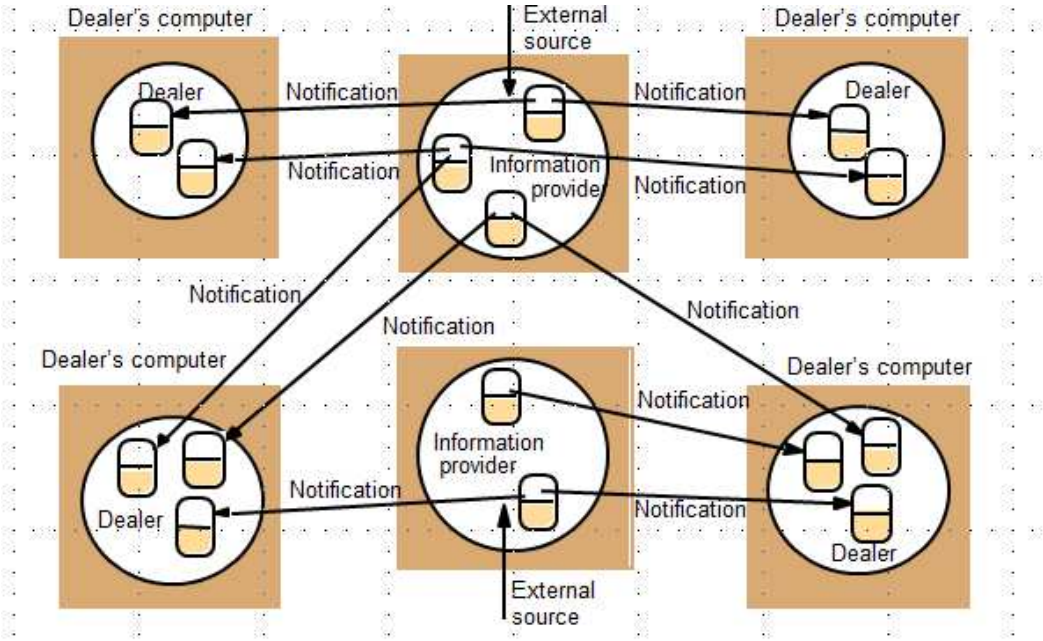


Fig 3.7 Event and notifications

A dealer process creates an object to represent each named stock that the user asks to have displayed.

An event source can generate events of one more different type. Each event has attributes that specify information about that event.

The architecture of distributed event notification specifies the roles of participants as in Fig.3.8: It is designed in a way that publishers work independently from subscribers. Event service maintains a database of published events and of subscribers' interests.

The roles of the participants are:

- Object of Interest – This is an object experiences change of state, as a result of its operations being invoked.
- Event – An event occurs at an object of interest as the result of the completion of a method invocation.
- Notification – A notification is an object that contains information about an event.
- Subscriber – A subscriber is an object that has subscribed to some type of events in another object.
- Observer objects – The main purpose of an observer is to separate an object of interest from its subscribers.
- Publisher – This is an object that declares that it will generate notifications of particular types of event.

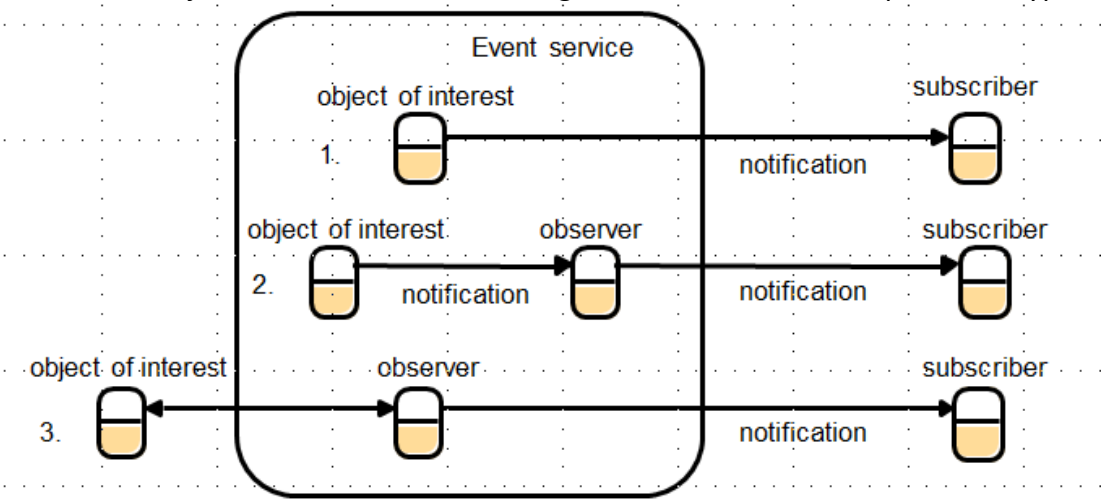


Fig 3.8 shows three cases:

- An object of interest inside the event service sends notification directly to the subscribers.

- An object of interest inside the event service sends notification via the observer to the subscribers.
- The observer queries the object of interest outside the event service and sends notifications to the subscribers.

A variety of delivery semantics can be employed:

- IP multicast protocol – information delivery on the latest state of a player in an Internet game
- Reliable multicast protocol – information provider / dealer
- Totally ordered multicast - Computer Supported Cooperative Working (CSCW) environment
- Real-time – nuclear power station / hospital patient monitor

Roles for observers – the task of processing notifications can be divided among observers:

- Forwarding – Observers simply forward notifications to subscribers.
- Filtering of notifications – Observers address notifications to those subscribers who find these notifications are useful.
- Patterns of events – Subscribers can specify patterns of events of interest.
- Notification mailboxes – A subscriber can set up a notification mailbox which receives the notification on behalf of the subscriber

Java RMI Case Study:

For case study go through the below link.

http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138781.html

SECURITY:

- There is a pervasive need for measures to guarantee the privacy, integrity, and availability of resources in distributed systems.
- Security attacks take various forms: Eavesdropping, masquerading, tampering, and denial of service.
- Designers of secure distributed systems must cope with the exposed interfaces and insecure network in an environment where attackers are likely to have knowledge of the algorithms used to deploy computing resources.
- Cryptography provides the basis for the authentication of messages as well as their secrecy and integrity.

Overview of Security Techniques:

Digital cryptography provides the basis for most computer security mechanisms, but it is important to note that computer security and cryptography are distinct subjects.

- Cryptography is an art of encoding information in a format that only intended recipient can access.
- Cryptography can be used to provide a proof of authenticity of information in a manner analogous to the use of signature in conventional transactions.

We will focus more on security of distributed systems and applications rather than algorithms

Cryptography:

Cryptography: encryption and decryption, Encryption is the process of encoding a message in such a way as to hide its contents. Modern cryptography includes several secure algorithms for encrypting and decrypting messages. They are based on keys.

A cryptography key is a parameter used in an encryption algorithm in such a way that the encryption cannot be reversed without knowledge of the key.

Classes of Cryptography Algorithms:

- There are two main classes:
  - Shared Secret Keys:
    - The sender and recipient share knowledge of the key and it must not be revealed to anyone.
  - Public/Private Key Pair:
    - The sender of a message uses a recipient's public key to encrypt the message.

- The recipient uses a corresponding private key to decrypt the message.

- Uses of Cryptography:
  - Secrecy and integrity (to stop eavesdropping and tampering) also use redundant information (checksums) for maintaining integrity.
  - Authentication
  - Digital Signatures

Algorithms to perform encryption and decryption

1. RSA (Rivest, Adi Shamir and Leonard Adleman):

RSA was first described in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman of the Massachusetts Institute of Technology. Public-key cryptography, also known as asymmetric cryptography, uses two different but mathematically linked keys, one public and one private. The public key can be shared with everyone, whereas the private key must be kept secret. In RSA cryptography, both the public and the private keys can encrypt a message; the opposite key from the one used to encrypt a message is used to decrypt it. This attribute is one reason why RSA has become the most widely used asymmetric algorithm: It provides a method of assuring the confidentiality, integrity, authenticity and non-reputability of electronic communications and data storage.

Algorithm:

RSA encrypts messages through the following algorithm, which is divided into 3 steps:

1) Key Generation

a) Choose two distinct prime numbers p and q.

b) Find n such that n = pq.
n will be used as the modulus for both the public and private keys.

c) Find the Φ
Φ =(p-1)(q-1).

d) Choose an e such that 1 < e < Φ, and such that e and Φ share no divisors other than 1 (e and Φ are relatively prime).
e is kept as the public key exponent.

e) Determine d (using Extended Euclidean algo table method) which satisfies the congruence relation
de ≡ 1 (mod φ(n)). Or ax +by= gcd(e, Φ)

step	a/x	b/y	d/f	k
1	1	0	Φ	-
2	0	1	e	d_1/d_2
3	$x_3= x_1-(x_2*k_2)$	$y_3=y_1-(y_2*k_2)$	$d_3=d_1-(d_2*k_2)$	d_2/d_3
4	------	-----	----	----

If (f==1) then stop the calculation
Now the corresponding row, the value of y is the original value of private key(d).

2) Encryption

a) Person A transmits his/her public key (modulus n and exponent e) to Person B, keeping his/her private key secret.

b) When Person B wishes to send the message "M" to Person A, he first converts M to an integer such that 0 < m < n by using agreed upon reversible protocol known as a padding scheme.

c) Person B computes, with Person A's public key information, the cipher text c corresponding to
$C \equiv M^e \pmod n$.

d) Person B now sends message "M" in cipher text, or c, to Person A.

3) Decryption

a) Person A recovers M from C by using his/her private key exponent, d, by the computation

$M \equiv C^d \pmod{n}$.

Example:

1. Choose p = 3 and q = 11
2. Compute n = p * q = 3 * 11 = 33
3. Compute ϕ = (p - 1) * (q - 1) = 2 * 10 = 20
4. Choose e such that 1 < e < ϕ and e and ϕ are coprime. Let e = 7
5. Compute a value for d such that (d * e) % ϕ = 1. One solution is d = 3 [(3 * 7) % 20 = 1] which is come with the help of extended Euclidean algo table method:

Step	x	y	F or d	k
1	1	0	20	-
2	0	1	7	2
3	1	-2	6	1
4	-1	3	1	6

Stop here because f==1

Now the value of d(private key is)= 3

6. Public key is (e, n) => (7, 33)
7. Private key is (d, n) => (3, 33)
8. The encryption of m = 2 is c = 27 % 33 = 29
9. The decryption of c = 29 is m = 293 % 33 = 2

2. DES (Data Encryption Standard):

The Data Encryption Standard (DES) is an outdated symmetric-key method of data encryption.

DES works by using the same key to encrypt and decrypt a message, so both the sender and the receiver must know and use the same private key. Once the go-to, symmetric-key algorithm for the encryption of electronic data, DES has been superseded by the more secure Advanced Encryption Standard (AES) algorithm.

Block diagram of DES: shown in fig 3.9.

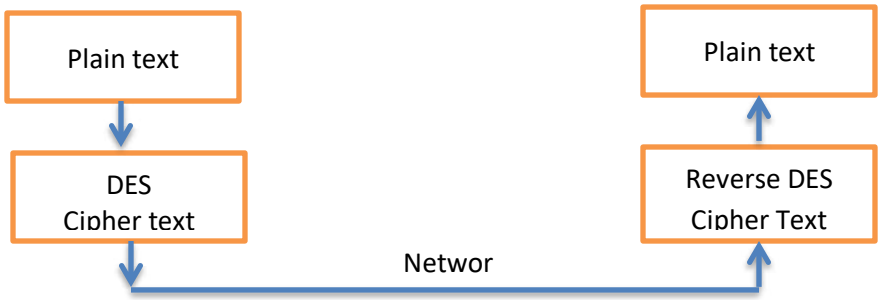


Fig 3.9 DES block diagram

Internal process of encryption from plain text to cipher text:

There are four processes (shown in fig 3.10):

1. Initial Permutation
2. 16 Feistel round
3. Left right swapping
4. Final permutation:

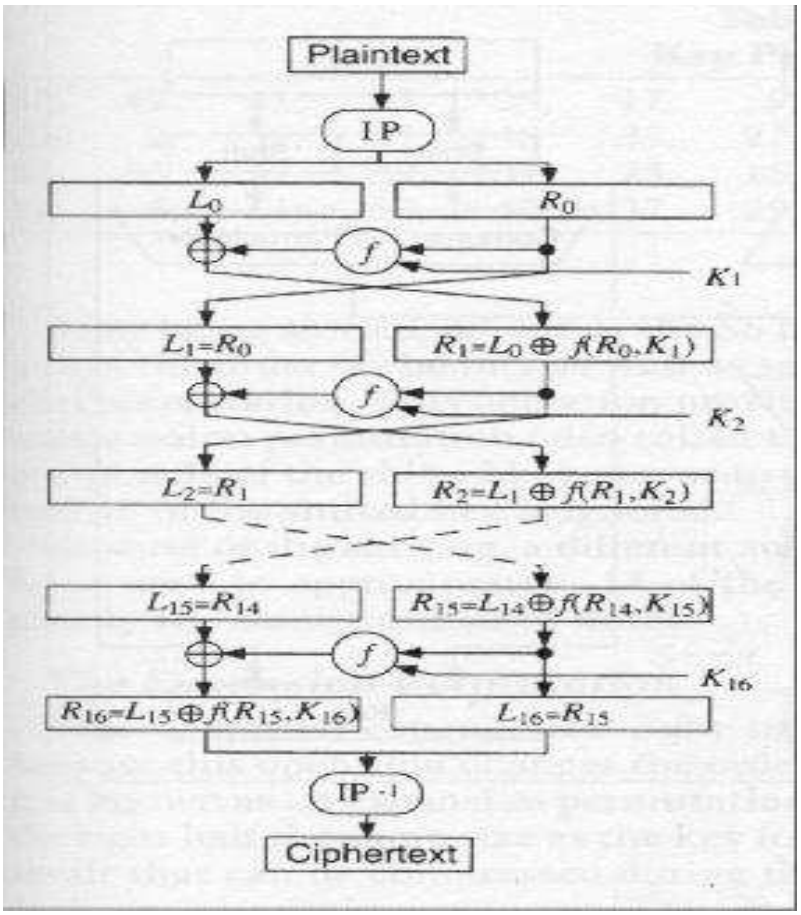


Fig 3.10 Internal process of encryption

- **Initial and final Permutation:** The initial and final permutations are straight Permutation boxes (P-boxes) that are inverses of each other. They have no cryptography significance in DES. The initial and final permutations are shown as follows – shown in fig 3.11.

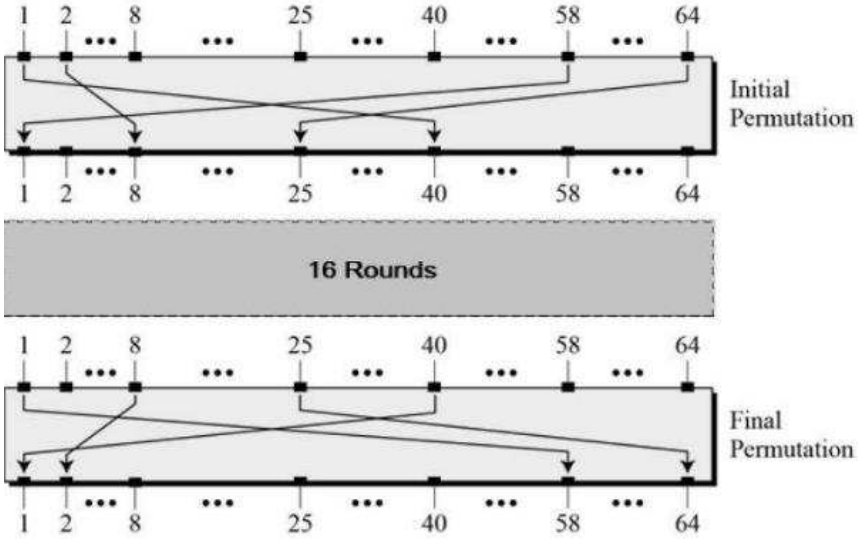


Fig 3.11 initial and final permutations

- **16 Feistel round:** The heart of this cipher is the DES function; f. The DES function applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output. Shown in fig 3.12.

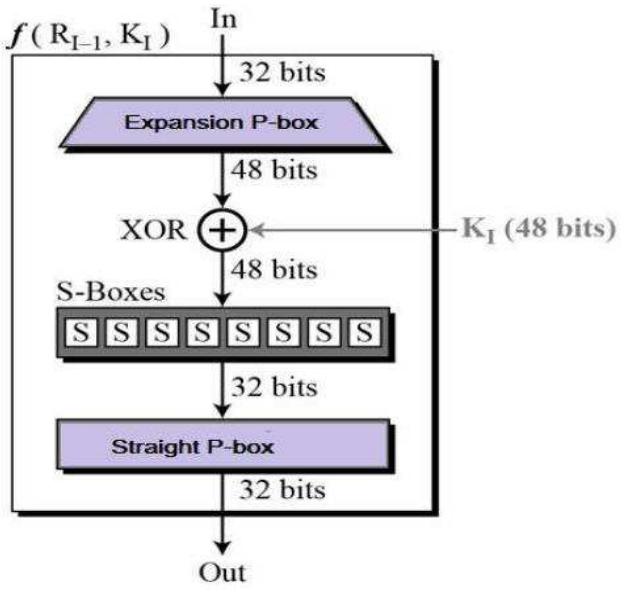


Fig 3.12 16-Feistel round

Expansion Permutation Box –Since right input is 32-bit and round key is a 48-bit, we first need to expand right input to 48 bits. Permutation logic is graphically depicted in the following illustration – shown in fig 3.13

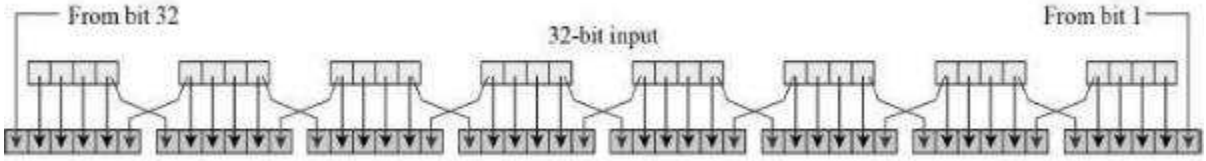


Fig 3.13 bit wise representation

XOR (Whitener) –After the expansion permutation, DES does XOR operation on the expanded right section and the round key. The round key is used only in this operation.

Substitution Boxes –The S-boxes carry out the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output. Refer the following illustration – shown in fig 3.14

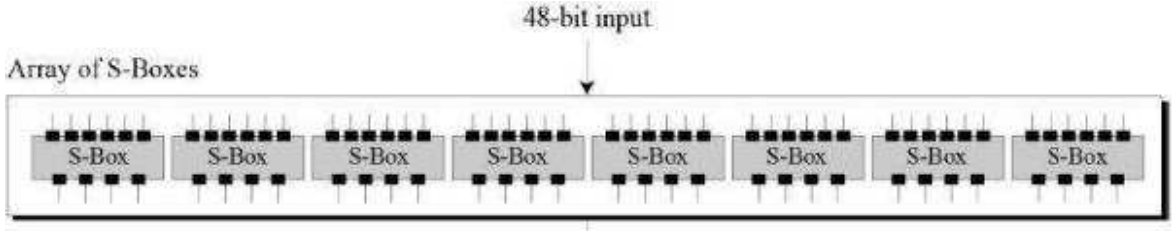


Fig 3.14 S-Boxes

The S-box rule is illustrated below – shown in fig 3.15

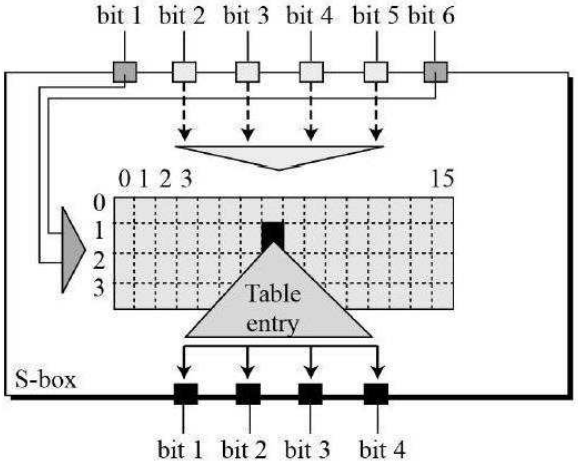


Fig 3.15 convert 48 bit into 32 bit

There are a total of eight S-box tables. The output of all eight s-boxes is then combined in to 32 bit section.

DES Analysis

The DES satisfies both the desired properties of block cipher. These two properties make cipher very strong.

- Avalanche effect – A small change in plaintext results in the very grate change in the cipher text.
- Completeness –Each bit of cipher text depends on many bits of plaintext.

During the last few years, cryptanalysis has found some weaknesses in DES when key selected are weak keys. These keys shall be avoided.

Distributed File Systems

A distributed file system is a client/server-based application that allows clients to access and process data stored on the server as if it were on their own computer. When a user accesses a file on the server, the server sends the user a copy of the file, which is cached on the user's computer while the data is being processed and is then returned to the server.

Ideally, a distributed file system organizes file and directory services of individual servers into a global directory in such a way that remote data access is not location-specific but is identical from any client. All files are accessible to all users of the global file system and organization is hierarchical and directory-based.

Since more than one client may access the same data simultaneously, the server must have a mechanism in place (such as maintaining information about the times of access) to organize updates so that the client always receives the most current version of data and that data conflicts do not arise. Distributed file systems typically use file or database replication (distributing copies of data on multiple servers) to protect against data access failures.

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
- A client module.

The relevant modules and their relationship is shown in Figure 3.16

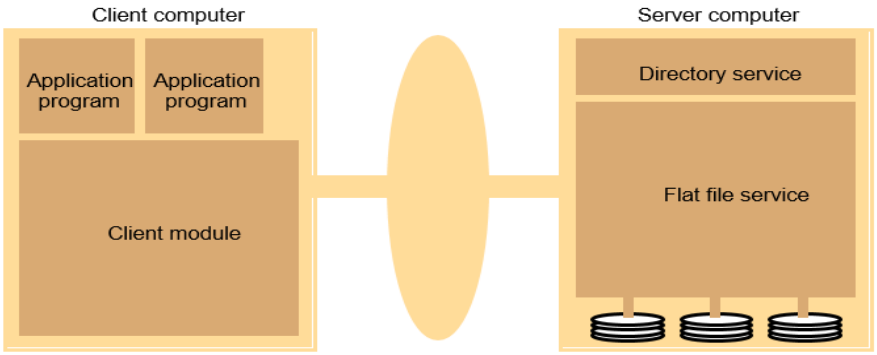


Fig 3.16 File service architecture

1. **A flat file service:** Concerned with the implementation of operations on the contents of file. Unique File Identifiers (UFIDs) are used to refer to files, all requests for flat file service operations. UFIDs are long sequences of bits chosen so that each file has a unique among all of the files in distributed system.
2. **A directory service**: Provides mapping between text names for the files & their UFIDs. Clients may obtain the UFID of a file by quoting its text name to directory service. Directory service supports functions needed generate directories, to add new files to directories.
3. **A client module:** It runs on each computer and provides integrated service (flat file and directory) as a single API to application programs. For example, in UNIX hosts, a client module emulates the full set of UNIX file operations.

Sun 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.

A DFS is a file system whose clients, servers and storage devices are dispersed among the machines of distributed system. A file system provides a set of file operations like read, write, open, close, delete etc. which forms the file services. The clients are provided with these file services. The basic features of DFS are multiplicity and autonomy of clients and servers. shown in fig 3.17.

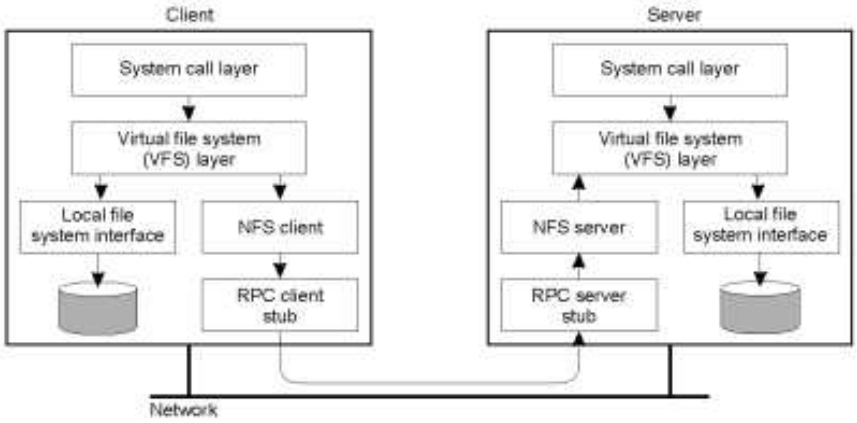


Fig 3.17 Architecture of NFS

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

- Naming
- Path Names (Mounting)
- Semantics

Naming

Naming is a mapping between logical and physical objects. For example, a user refers to a file by a textual name, but it is mapped to disk blocks. There are two notions regarding name mapping used in DFS.

- Location Transparency: The name of a file does not give any hint of file's physical storage location.
- Location Independence: The name of a file does not need to be changed when file's physical storage location changes.

A location independent naming scheme is basically a dynamic mapping. NFS does not support location independency.

There are three major naming schemes used in DFS. In the simplest approach, files are named by some combination of machine or host name and the path name. This naming scheme is neither location independent nor location transparent. This may be used in server side. Second approach is to attach or mount the remote directories to the local directories. This gives an appearance of a coherent directory. This scheme is used by NFS. Early NFS allowed only previously mounted remote directories. But with the advent of auto mount, remote directories are mounted on demand based on the table of mount points and file structure names. This has other advantages like the file-mount table size is much smaller and for each mounts point, we can specify many servers. The third approach of naming is to use name space which is identical to all machines. In practice, there are many special files that make this approach difficult to implement**.**

Mounting

The mount protocol is used to establish the initial logical connection between a server and a client. A mount operation includes the name of the remote directory to be mounted and the name of the server machine storing it. The server maintains an export list which specifies local file system that it exports for mounting along with the permitted machine names. UNIX uses /etc./exports for this purpose. Since, the list has a maximum length, NFS is limited in scalability.

Any directory within an exported file system can be mounted remotely on a machine. When the server receives a mount request, it returns a file handle to the client. File handle is basically a data-structure of length 32 bytes. It serves as the key for further access to files within the mounted system. shown in fig 3.18.

In UNIX term, the file handle consists of a file system identifier that is stored in super block and an inode number to identify the exact mounted directory within the exported file system. In NFS, one new field is added in inode that is called the generic number.

Mount can be is of three types -

- Soft mount: A time bound is there.
- Hard mount: No time bound.
- Automount: Mount operation done on demand.

Mount Protocol:

- Mount protocol establishes a local name for remote files
- Users access remote files using local names; OS takes care of the mapping

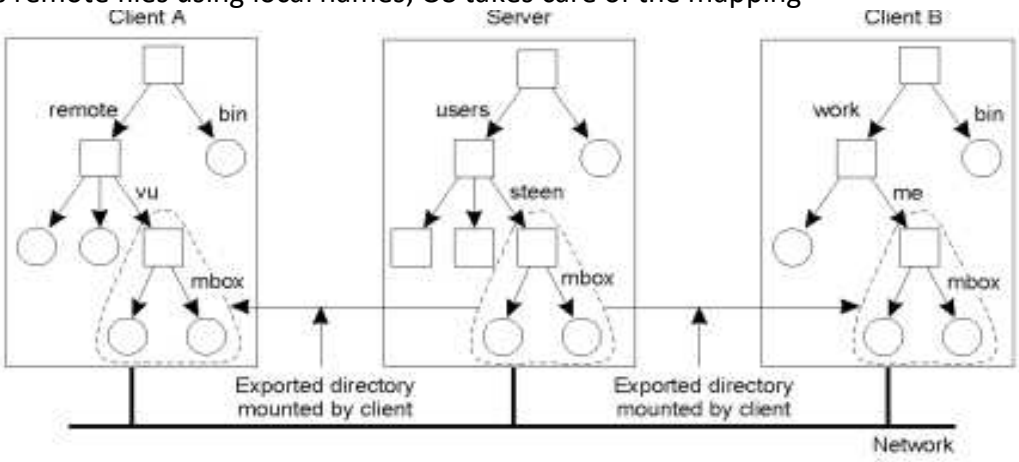


Fig 3.18 mounting

Crossing Mount Points

- Mounting nested directories from multiple servers
- NFS v3 does not support transitive exports (for security reasons)
- NFS v4 allows clients to detects crossing of mount points shown in fig 3.19

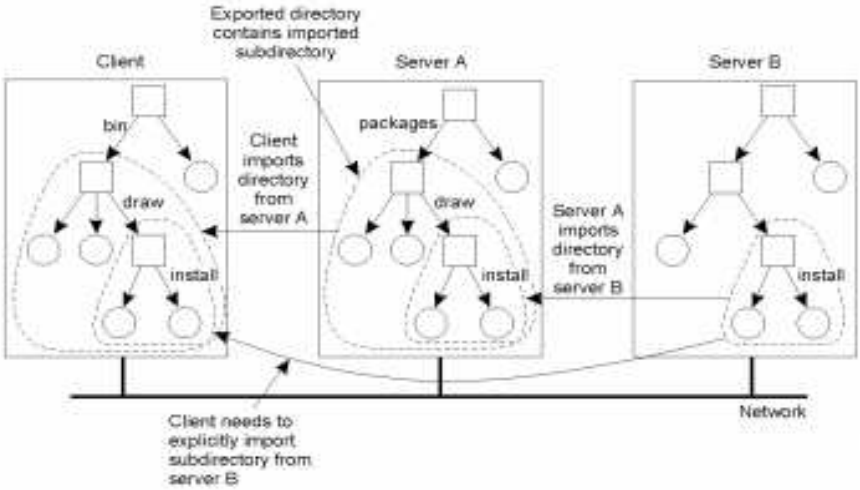


Fig 3.19 crossing mounting

Semantics of file sharing

- On a single processor, when a read follows a write, the value returned by the read is the value just written, shown in fig.(a)
- In a distributed system with caching, obsolete values may be returned. shown in fig.3.20

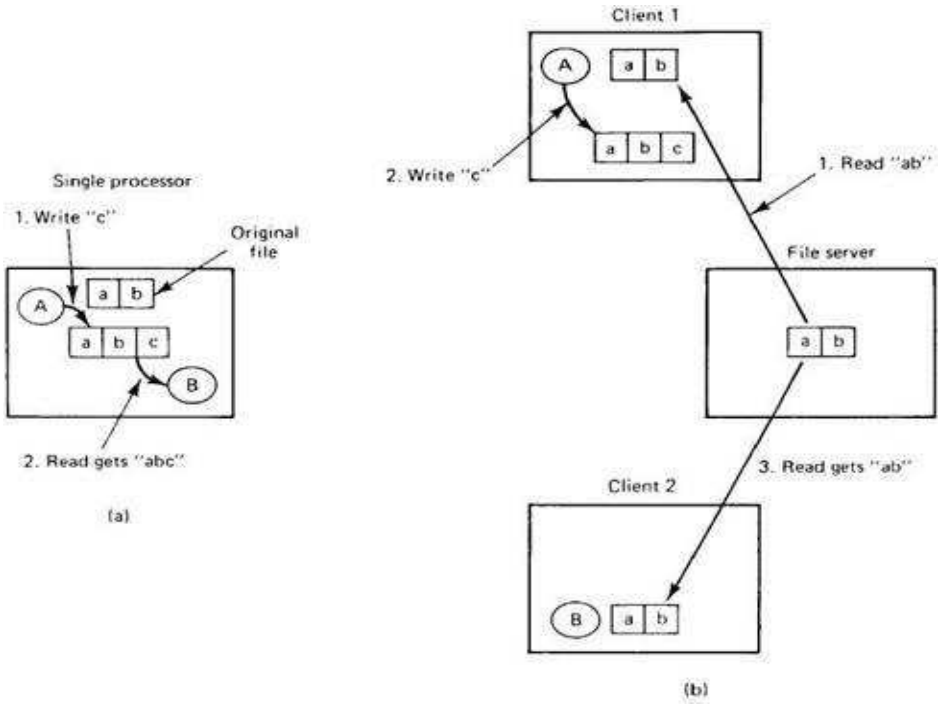


Fig 3.20 Semantics of file sharing

Client Caching: Delegation

- NFS V4 supports open delegation
- Server delegates local open and close requests to the NFS client
- Uses a callback mechanism to recall file delegation. shown in fig 3.21.

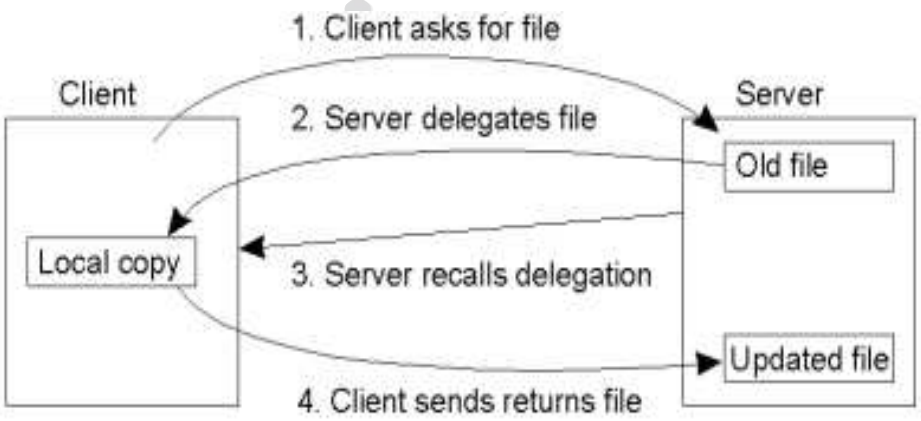


Fig 3.21 client cashing delegation

RPC Failures

Three situations for handling retransmissions: use a duplicate request cache

a) The request is still in progress

b) The reply has just been returned

c) The reply has been some time ago, but was lost.

Use a duplicate-request cache: transaction Ids on RPCs, results cached. shown in fig 3.22

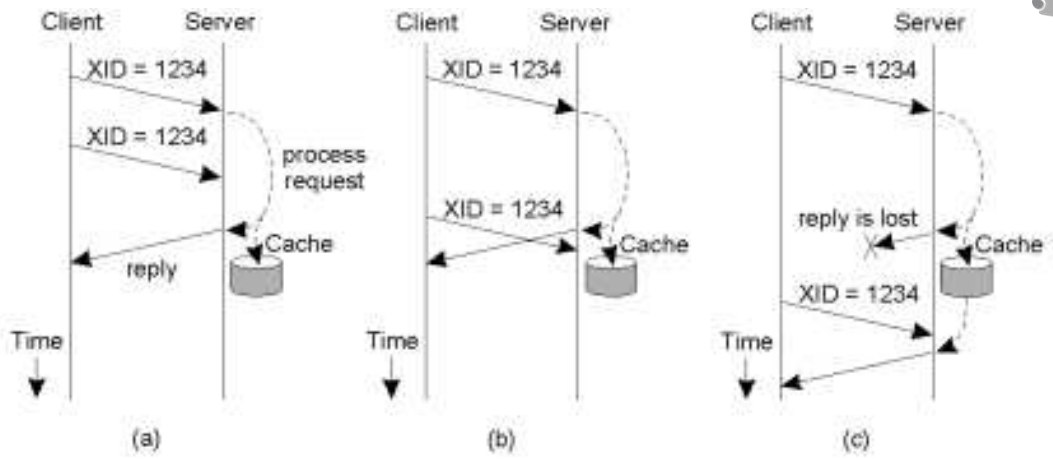


Fig 3.22 RPC failure

Andrews File System (AFS)

The Andrew File System (AFS) is a location-independent file system. AFS makes it easy for people to work together on the same files, no matter where the files are located. AFS users do not have to know which machine is storing a file. AFS is a distributed file system which makes it as easy to access files stored on a remote computer as files stored on the local disk. shown in fig 3.23

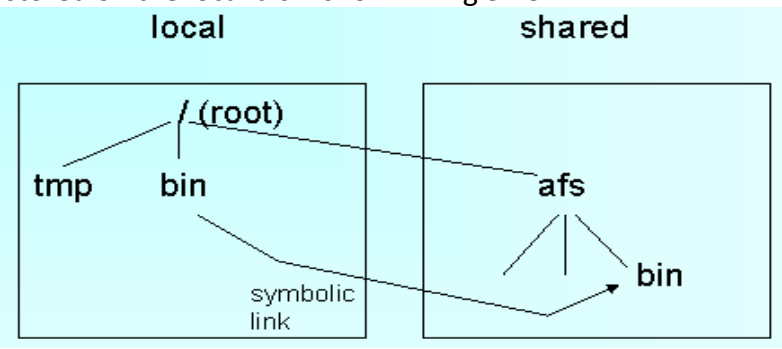


Fig 3.23 Andrews file system

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):

- 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

AFS distinguishes between client machines (workstations) and dedicated server machines. Caching files in the client side cache reduces computation at the server side, thus enhancing performance. However, the problem of sharing files arises. To solve this, all clients with copies of a file being modified by another client are not informed the moment the client makes changes. That client thus updates its copy, and the changes are reflected in the distributed file system only after the client closes the file. Various terms related to this concept in AFS are:

- Whole File Serving: The entire file is transferred in one go, limited only by the maximum size UDP/IP supports
- Whole File Caching: The entire file is cached in the local machine cache, reducing file-open latency, and frequent read/write requests to the server
- Write On Close: Writes are propagated to the server side copy only when the client closes the local copy of the file

RGPVNOTES.IN

We hope you find these notes useful.

You can get previous year question papers at https://qp.rgpvnotes.in .

If you have any queries or you want to submit your study notes please write us at rgpvnotes.in@gmail.com

LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in