Module 3

Interprocess Communication

INTERPROCESS COMMUNICATION

Ques 1) What is Inter-process communication?

Ans: Inter-Process Communication

Inter-process communication or Inter-Process Communication (IPC) refers specially to the mechanisms an operating system provides to allow the processes to manage shared data.

In a distributed system, the inter-process communication is an important issue to be handled. This is so, because in the distributed system, the processes or tasks need to communicate with each other. In a distributed computing usually there is no shared memory, so there is a need for passing mechanisms.

Figure 3.1 shows that process X sends a message to process Y. Process Y receives it.

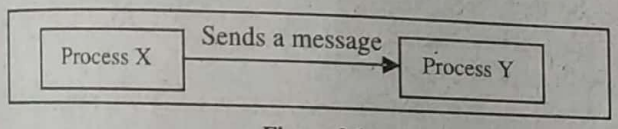

Figure 3.1

The send and receive of message may be synchronous, i.e., process X waits till process Y receives the message. The send and receive of message may be asynchronous, i.e., process X proceeds with its further computation after sending the message without waiting for process Y to receive it, i.e.,

```
Process X
    send (message, Y)
    Receive (Reply, Y)
Process Y
    Receive (message, X)
    send (Reply, X)
```

Ques 2) What are the characteristics of Interprocess communication?

Ans: Characteristics of Interprocess Communication

Message passing between a pair of processes can be supported by two message communication operations, **send** and **receive**, defined in terms of destinations and messages. To communicate, one process sends a message (a sequence of bytes) to a destination and another process at the destination receives the message. This activity involves the communication of data from the sending process to the receiving process and may involve the synchronization of the two processes.

A queue is associated with each message destination. Sending processes cause messages to be added to remote queues and receiving processes remove messages from local queues. Communication between the sending and receiving processes may be either synchronous or asynchronous. Some of its characteristics are as follows:

1) **Synchronous and Asynchronous Communication:** In the synchronous form of communication, the sending and receiving processes synchronize at every message. In this case, both **send** and **receive** are blocking operations. Whenever a **send** is issued the sending process (or thread) is blocked until the corresponding **receive** is issued. Whenever a **receive** is issued by a process (or thread), it blocks until a message arrives.

   In the asynchronous form of communication, the use of the **send** operation is non-blocking in that the sending process is allowed to proceed as soon as the message has been copied to a local buffer, and the transmission of the message proceeds in parallel with the sending process. The **receive** operation can have blocking and non-blocking variants. In the non-blocking variant, the receiving process proceeds with its program after issuing a **receive** operation, which provides a buffer to be filled in the background, but it must separately receive notification that its buffer has been filled, by polling or interrupt.

2) **Message Destinations:** In the Internet protocols, messages are sent to (Internet address, local port) pairs. A local port is a message destination within a computer, specified as an integer. A port has exactly one receiver but can have many senders. Processes may use multiple ports to receive messages. Any process that knows the number of a port can send a message to it. Servers generally publicize their port numbers for use by clients.

3) **Reliability:** Reliable communication in defined terms of validity and integrity. As far as the validity property is concerned, a point-to-point message service can be described as reliable if messages are guaranteed to be delivered despite a 'reasonable' number of packets being dropped or lost. In contrast, a point-to-point message service can be described as unreliable if messages are not guaranteed to be delivered in the face of even a single packet dropped or lost. For integrity, messages must arrive uncorrupted and without duplication.

4) **Ordering:** Some applications require that messages be delivered in sender order – that is, the order in which they were transmitted by the sender. The delivery of messages out of sender order is regarded as a failure by such applications.

Ques 3) What is socket? What is client and server-side sockets?

Ans: Socket

Sockets are a programming abstraction which is used to implement low-level IPC. **For example,** two processes exchange information by each process having a socket of its own. They can then read/write from/onto their sockets.

Sockets are created, and if in a client-server approach, then these sockets are prepared for sending and receiving messages. With this, we can see that sending data is just done by writing to one's socket, and receiving data is done by reading from the socket.

Client-Side Sockets

A Client-side socket is best understood as the endpoint of a conversation. **For example,** they are short-lived. In a web browser, a socket is created to send a request, receive the response, and then once that is done, the socket is destroyed (discarded).

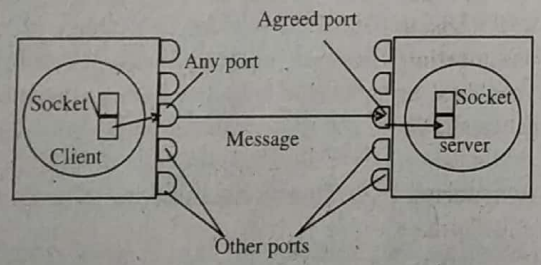

Figure 3.2: Sockets and Ports

In order to set up a client-side, we do as follows:

1) Create a socket for a given transport (TCP) and IP (IPv4).
2) Connect to a server on the port corresponding to the desired protocol.
3) When the connect returns without error, then the client-side set-up is complete.

Once the socket is up, the message-exchange stage can take place:

1) The client sends a request for a specific HTML page, through its socket.
2) Then, through the same socket, it waits for a response to come, and then processes it.

Once this is done, the socket is then discarded.

Server-Side Sockets

A server socket behaves more like a dispatcher. This is because they do not normally send or receive any data; typically, they simply listen for connections on the host and port that the socket server is bound to.

When a server socket gets one connection, it also gets a new socket in response to that event. It will then spawn a handler process, or thread, which is the one that actually uses the new socket to exchange messages with the client-connected socket.

After this, the server socket then simply goes back to listen for more connections.

To set up a server socket, we follow these steps:

1) Create the server socket for a given transport (TCP) and IP (IPv4).
2) Set the server socket options.
3) Bind the server socket to a host address (localhost) and a port (80).
4) Start listening to connections on the server socket and set the maximum number that could be left waiting.

The connection handling stage is a loop, in which:

1) The server socket first blocks waiting connections to accept connections.
2) When a connection is accepted, two pieces of information result: the new client-connected socket has been created and the host and the port of connecting client.
3) The server process then handles the connection, for example it could spawn a dispatcher passing it the socket to be used and start it.
4) The server process can then continue the loop.
5) The client-connected socket must be ordered to shutdown and close when the handler is done and returns to the parent.

Ques 4) Discuss about the UDP datagram communication?

Or

Discuss the issues relating to datagram communication?

Or

Describe the failure model of UDP datagram communication?

Ans: UDP Datagram Communication

The steps of UDP datagram communication are as follows:

1) Client finds an available port for UPD connection.

2) Client binds the port to local IP.
3) Server finds designated port, publicises it to clients, and binds it to local IP.
4) Server process issues a receive methods and gets the IP and port # of sender (client) alongwith the message.

Issues relating to Datagram Communication

1) **Message Size:** Set to 8KByte for most, general protocol support 216 bytes, possible truncation if receiver buffer is smaller than message size.
2) **Blocking:** Send is non-blocking and op returns if message gets pass the UDP and IP layers; receive is blocking (with discard if no socket is bound or no thread is waiting at destination port).
3) **Timeouts:** Reasonably large time interval set on receiver sockets to avoid indefinite blocking
4) **Receive from Any:** No specification of source (senders), typically many-to-one, but one-to-one is possible by a designated send-receive socket.

UDP Failure Model

A failure model for communication channels and defines reliable communication in terms of two properties – integrity validity. The integrity property requires that messages should not be corrupted or duplicated. The use of a checksum ensures that there is a negligible probability that any message received is corrupted. UDP datagrams suffer from the following failures:

1) Due to omission of send or receive (either checksum error or no buffer space at source or destination).
2) Due to out-of-order delivery.
3) UDP lacks built in checks, but failure can be modelled by implementing an ACK mechanism.

Ques 5) Describe the TCP stream communication.

Ans. TCP Stream Communication

TCP provides a communication channel between processes on each host system. The channel is reliable, full-duplex, and streaming. To achieve this functionality, the TCP drivers break up the session data stream into discrete segments, and attach a TCP header to each segment.

The API to the TCP protocol, which originates from BCD 4.x UNIX, provides the abstraction of a stream of bytes to which data may be written and from which data may be read.

The following characteristics of network are hidden by the stream abstraction:

1) **Message Sizes:** User application has option to set IP packet size, small or large.
2) **Lost Messages:** Sliding window protocol with ACKs and retransmission is used.
3) **Flow Control:** Blocking or throttling is used.
4) **Message Duplication and Ordering:** Seq #s with discard of dups and reordering.
5) **Message Destinations:** A connection is established first, using connection-accept methods for rendezvous, and no IP addresses in packets. Each connection socket is bidirectional – using two streams – output/write and input/read. A client closes a socket to sign off, and last stream of bytes are sent to receiver with 'broken-pipe' or empty-queue indicator.

Ques 6) What are the main issues of TCP stream communication? Also discuss its failure model.

Ans: Issues of TCP Stream Communication

1) **Matching of Data Items:** Both client/sender and server/receiver must agree on data types and order in the stream.
2) **Blocking:** Data is streamed and kept in server queue; empty server queue causes a block AND full server queue causes a blocking of sender.
3) **Threads:** Used by servers (in the background) to service clients, allowing asynchronous blocking (Systems without threads, e.g., Unix, use select).

Failure Model

1) **Integrity:** Uses checksums for detection/rejection of corrupt data and seq #s for rejecting duplicates.
2) **Validity:** Uses timeout with retransmission techniques (takes care of packet losses or drops).
3) **Pathological:** Excessive drops/timeouts signal broken sockets and TCP throws in the towel (no one knows if pending packets were exchanged) – unreliable.

Ques 7) Discuss about the external data representation and marshalling? Also discuss the approaches used for external data representation and marshalling.

Ans: External Data Representation and Marshalling

The information stored in running programs is represented as data structures – **for example,** by sets of interconnected objects – whereas the information in messages consists of sequences of bytes.

One of the following methods can be used to enable any two computers to exchange binary data values:

1) The values are converted to an agreed external format before transmission and converted to the local form on receipt; if the two computers are

known to be the same type, the conversion to external format can be omitted.

2) The values are transmitted in the sender's format, together with an indication of the format used, and the recipient converts the values if necessary.

An agreed standard for the representation of data structures and primitive values is called an **external data representation**.

Marshalling is the process of transforming the memory representation of an object to a data format suitable for storage or transmission, and it is typically used when data must be moved between different parts of a computer program or from one program to another.

Marshalling is similar to serialization and is used to communicate to remote objects with an object, in this case a serialized object. It simplifies complex communication, using composite objects in order to communicate instead of primitives. The inverse, of marshalling is called **unmarshalling**.

The purpose of a data marshalling mechanism is to represent the caller's arguments in a way that can be efficiently interpreted by a server program. Since the client and server will often be coded in different programming languages and perhaps running in different operating systems or on different hardware, marshalling also seeks to provide universal representations that can be understood by any platform.

In the most general cases, this mechanism deals with the possibility that the computer on which the client is running uses a different data representation than the computer on which the server is running.

Approaches to External Data Representation and Marshalling

Three approaches to external data representation and marshalling are:

1) **CORBA Common Data Representation (CDR):** CORBA CDR is the external data representation defined with CORBA 2.0. CDR can represent all of the data types that can be used as arguments and return values in remote invocations in CORBA.

i) **Primitive Types:** These consist of primitive types, which include:
   a) Short (16-bit),
   b) Long (32-bit),
   c) Unsigned short,
   d) Unsigned long,
   e) Float (32-bit),
   f) Double (64-bit),
   g) Char,
   h) Boolean (TRUE, FALSE),
   i) Octet (8-bit) etc.

ii) **Constructed Types: Table below** shows the CORBA CDR for constructed types:

Type	Representation
Sequence	Length (unsigned long) followed by elements in order.
String	Length (unsigned long) followed by characters in order (can also can have wide characters).
Array	Array elements in order (no length specified because it is fixed).
Struct	In the order of declaration of the components.
Enumerated	Unsigned long (the value are specified by the order declared)
Union	Type tag followed by the select member.

Figure 3.3 shows a message in COBRA CDR that contains the three fields of a struct whose respective types are string, string and unsigned long.

The figure shows the sequence of bytes with four bytes in each row. The representation of each string consists of an unsigned long representing its length followed by the characters in the string.

Index in sequence of bytes	← 4 bytes →	notes on representation
0-3	5	length of string
4-7	"Smit"	'Smith'
8-11	"h_"	
12-15	6	Length of string
16-19	"Lond"	'London'
20-23	"On"	
24-27	1934	Unsigned long

The flattened form represents a Person struct with value: {'Smith', London, 1934}

Figure 3.3: CORBA CDR Message

2) **Java Object Serialization:** Java's object serialization allows one to take any object that implements the Serializable interface and turn it into a sequence of bytes that can later be fully restored to regenerate the original object. This is even true across a network, which means that the serialization mechanism automatically compensates for differences in operating systems.

Java object serialization is a specification of encoding Java objects in an octet stream so that the object can be exchanged over a network. Therefore, serialization in Java object serialization has the same meaning as the marshaling. To use Java object serialization, classes to be serialized must implement the Serializable interface.

3) **Extensible Markup Language (XML):** XML is a markup language for documents containing structured information.

Ques 8) What is XML? Discuss about its elements, namespace and attributes.

Or

Describe the XML schema and DTD.

Ans: Extensible Markup Language (XML)

XML is a markup language for documents containing structured information. Structured information contains both content (words, pictures, etc.) and some indication of what role that content plays (**for example,** content in a section heading has a different meaning from content in a footnote, which means something different than content in a figure caption or content in a database table, etc.). Almost all documents have some structure.

The syntax of XML can be thought of at two distinct levels:

1) First, there is the general low-level syntax of XML that imposes its rules on all XML documents.
2) The other syntactic level is specified by either Document Type Definitions (DTDs) or XML schemas.

Elements and Attributes

Elements consist of three character-level components: the start tag, the end tag, and the content, if there is any. The names of elements must follow the production rules of XML names:

1) **Element Content:** The content is whatever lies between the start tag and the end tag.
2) **Empty Elements :** Empty elements have only one tag, not a start and end tag. You may be familiar with empty elements from HTML; e.g., the HTML <IMG>, <LI>, <HR>, and
 elements are empty, which is to say that they do not enclose any content (either character data or markup). Empty elements are represented with only one tag (in HTML there is no closing </IMG>, <LI>, <HR>, and
 tags). In XML, you can declare elements to the empty in the document's DTD.
3) **Root Element**: The document instance consists of the root element. Every other element must be contained within a root element.

An **attribute** is a property. Attributes are expressed as information within an element's start tag. XML attributes are composed structurally as a name=value, where name is the name of the attribute and value is the value of the attribute.

For example, an element that describes a book can be written as:

```
<book title="Effective web design" author="anny"
publisher="sybex"/>
```

Title, author and publisher are the attributes.

CDATA Sections

CDATA sections contain nothing but character data, no matter what their contents look like. They can contain the < and the & literal values. This means if you use them in a CDATA section, you don't need to escape them.

Syntax:

```
<![CDATA [content here]]
```

For example, let a typical example, using Java:

```
<?xml version = "1.0"
encoding-="UTF-8" standalone="yes" ?>
<fragment>
<![CDATA[
(while i <= 8)
sum + = i+ +;
)]]>
</fragment>
```

Using CDATA sections provides clear advantages, and it's a good idea to use them whenever there's even a threat of a character that might be construed as XML markup by a processor.

XML Namespaces

A namespace is an identifier used to distinguish between XML element names and attribute names which might be the same. Use of namespaces prevents problems which might arise when the same element or attribute name is used by different developers for different reasons.

The XML namespace is a special type of reserved XML attribute that anyone place in an XML tag. The reserved attribute is actually more like a prefix that you attach to any namespace one creates. This attribute prefix is "xmlns:", which stands for XML NameSpace. The colon is used to separate the prefix from namespace that anyone is creating.

XML Schemas

An **XML schema** is a description of a type of XML document, typically expressed in terms of constraints on the structure and content of documents of that type, above and beyond the basic syntactical constraints imposed by XML itself.

For example,

```
<? xml version = "1.0">
<TASKFORCE XMLS = "X – schema:
schemal.xml">
<EMPLOYEE> George Patton</EMPLOYEE>
<EMPLOYEE> Doulas
MacArthur</EMPLOYEE>
<DESCRIPTION> XML Programming
Taskforce</DESCRIPTION>
</TASKFORCE>
```

In this case, the schema for this XML document is **schema1.xml**.

Document Type Definitions (DTD)

A DTD describes the grammar expected of documents that use its vocabulary. The elements must follow the rules as specified by the DTD. The DTD provides the means to verify the document's conformance that is its validity. DTD needs to be declared in the prolog of an XML document.

For example,

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

Output

```
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

The DTD above is interpreted like this:

1) !DOCTYPE note defines that the root element of this document is note
2) !ELEMENT note defines that the note element must contain four elements: "to, from, heading, body"
3) !ELEMENT to defines the to element to be of type "#PCDATA"
4) !ELEMENT from defines the from element to be of type "#PCDATA"
5) !ELEMENT heading defines the heading element to be of type "#PCDATA"
6) !ELEMENT body defines the body element to be of type "#PCDATA".

Ques 9) What is group communication? Discuss its implementation.

Ans: Group Communication

Remote procedure calls assume the existence of two parties – a client and a server. This, as well as the socket-based communication we looked at earlier, is an example of point-to-point, or unicast, communication. Sometimes, however, we want one-to-many, or group, communication.

Groups are generally dynamic (**figure 3.4**). They may be created and destroyed. Processes may join or leave groups and processes may belong to multiple groups. An analogy to group communication is the concept of a mailing list. A sender sends a message to one party (the mailing list) and multiple users (members of the list) receive the message. Groups allow processes to deal with collections of processes as one abstraction. Ideally, a process should only send a message to a group and need not know or care who its member are.

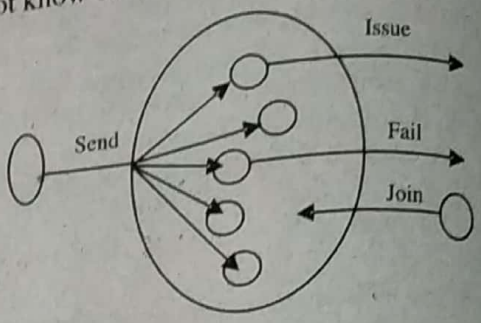

Figure 3.4: Group Dynamics

Implementing Group Communication

Group communication can be implemented in several ways. Hardware support for multicasting allows the software to request the hardware to join a multicast group. Messages sent to the multicast address will be received by all network cards listening on that group(s) (**figure 3.5**). If the hardware does not support multicasting, an alternative is to use hardware broadcast and software filters at the receivers.

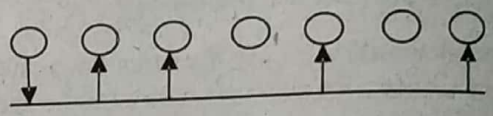

Figure 3.5: Group Multicast

Each message is tagged with a multicast address. The software processing the incoming messages extracts this address and compares it with its list of multicast addresses that it should accept. If it is not on the list, the message is simply dropped (**figure 3.6**). While this method generates overhead for machines that are not members of the group, it requires the sender to only send out a single message.

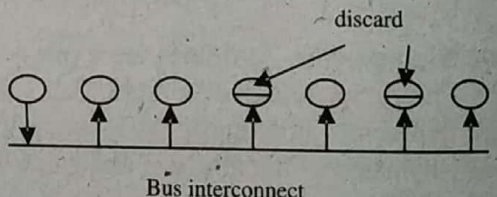

Figure 3.6: Simulating a Multicast via a Broadcast

Another implementation option is to simulate multicasting completely in software. In this case, a separate message will be sent to each receiver. This can be implemented in two ways. The sending process can know all the members of the group and send the same message to each group member (**figure 3.7**).

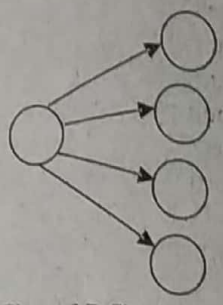

Figure 3.7: Simulating Group Communication with Multiple Unicasts

Alternatively, some process on some computer can be designated as a group coordinator – a central point for group membership information **(figure 3.8)**.

The sender will send one message to the group coordinator, which then iterates over each group member and sends the message to each member.

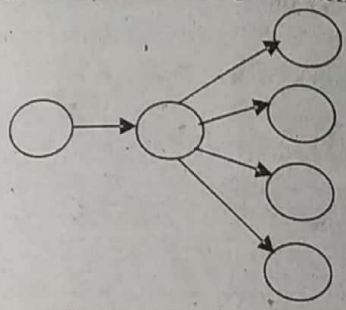

Figure 3.8: Simulating Group Communication with a Central Coordinator

As with unicast communication, group communication also requires a transport-level protocol. Even with hardware support, there must be a mechanism for directing data to the interested process(es).

Ques 10)What are the design issues of group communication?

Ans: Design Issues of Group Communication

A number of design alternatives for group communication are available. These will affect how the groups behave and send messages. Some design issues are as follows:

1) **Closed Group *versus* Open Group:** With closed groups, only the group members may send a message to the group. This is useful when multiple processes need to communicate with others in solving a problem, such as parallel processing applications.

   The alternative is open groups, where non-members can send a message to a group. An example of this type of group is an implementation of a replicated server (such as a redundant file system).

2) **Peer Groups *versus* Hierarchical Groups:** With peer groups, every member communicates with each other. The benefits are that this is a decentralised, symmetric system with no point of failure. However, decision-making may be complex since all decisions must be made collectively (a vote may have to be taken).

   The alternative is hierarchical groups, in which one member plays the role of a group coordinator. The coordinator makes decisions on who carries out requests. Decision-making is simplified since it is centralised. The downside is that this is a centralised, asymmetric system and therefore has a single point of failure.

3) **Centralised Group Membership *versus* Distributed Membership:** If control of group membership is centralised, we will have one group server that is responsible for getting all membership requests. It maintains a database of group members. This is easy to implement but suffers from the problem that centralised systems share – a single point of failure.

   The alternative mechanism is to manage group membership in a distributed way where all group members receive messages announcing new members (or the leaving of members).

   Several problems can arise in managing group membership. Suppose a group member crashes. It effectively leaves the group without sending any form of message informing others that it left the group. Other members must somehow discover that it is missing.

   Leaving and joining a group must be synchronous with message delivery. No messages should be received by a member after leaving a group. This is easier to achieve if a group coordinator/group server is used for message delivery and membership management.

   A final design issue is that if the processes, the computers they run on, and/or the network die so the group cannot function, how are things restarted?

Ques 11)What is multicast communication? What are the characteristics of multicast communication?

Ans: Multicast Communication

Multicast is group communication where data transmission is addressed to a group of destination computers simultaneously. Multicast can be one-to-many or many-to-many distribution.

Multicast is often employed in Internet Protocol (IP) applications of streaming media, such as IPTV and multipoint videoconferencing.

A multicast operation is an operation to send a single message from one process to a group of processes. This operation suits the service that is implemented as a number of different processes in different computers.

In the Inter-Process Communication (IPC), data is sent from a source process, the sender, to one destination process, the receiver. This form of IPC can be called unicasting, the sending the information to a single receiver, as opposed to multicasting, the sending of information to multiple receivers. Unicasting provides one-to-one IPC; multicasting supports one-to-many IPC (see **figure 3.9**).

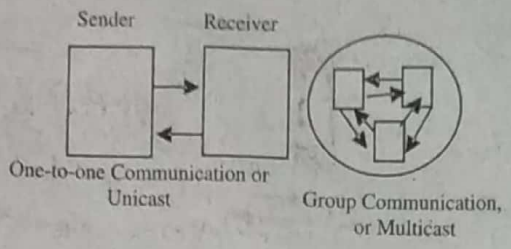

Figure 3.9: Unicast IPC and Multicast IPC

Whereas the majority of network services and network applications use unicasting for IPC, multicasting is useful for applications such as instant messages, groupware, online conferences, and interactive learning, and it can be used for applications such as a real-time online auction. Multicasting is also useful in the replication of services for fault tolerance.

In an application or network service that makes use of multicasting, a set of processes forms a groups, called a multicast group. Each process in a group can send and receive messages. A message send by any process in the group can be received by each participating process in the group. A process may also choose to leave a multicast group.

In an application such as online conferencing, a group of processes interoperate using multicasting to exchange audio, video, video, and/or text data.

Characteristics of Multicast Communication

Multicast messages provide a useful infrastructure for constructing distributed systems with the following **characteristics**:

1) **Fault Tolerance Based on Replicated Services:** Client requests are multicast to a group of servers. Even when some members fail, clients can still be served.
2) **Finding the Discovery Servers in Spontaneous Networking:** Multicast messages can be used by servers and clients to locate discovery services.
3) **Better Performance through Replicated Data:** Data are replicated to increase the performance of a service.
4) **Propagation of Event Notifications:** Multicast to a group may be used to notify processes when something happens. **For example,** a news system and the Jini system.

Ques 12)What is IP Multicast?

Ans: IP Multicast

IP multicast is a method of sending Internet Protocol (IP) datagrams to a group of interested receivers in a single transmission.

It is a form of point-to-multipoint communication employed for streaming media and other applications on the Internet and private networks. IP multicast is a technique for one-to-many communication over an IP network.

IP multicast built on top of the Internet Protocol allows the sender to transmit a single IP packet to a set of computers. A multicast group is specified by a class D Internet address – an address whose first 4-bits are 1110 in IPv4 (**figure 3.10**).

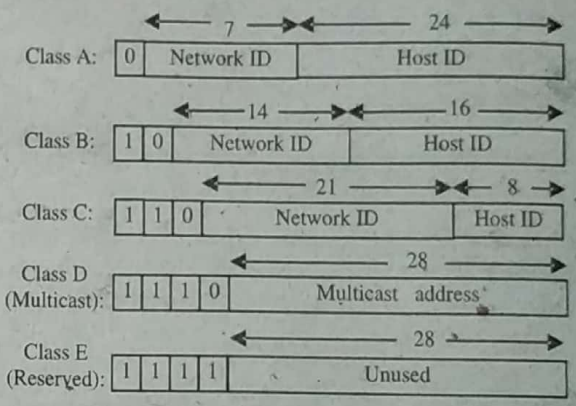

Figure 3.10: Internet Address Structure, Showing Field Sizes in Bits

The membership of multicast groups is dynamic. At the application programming level, IP multicast is available via UDP. At the IP level, a computer belongs to a multicast group when sockets belong to that group.

The following details are specific to IPv4:

1) **Multicast Routers:** IP packets can be multicast on a local network and on the Internet.
   i) Local multicasts use the multicast capability of the local network such as an Ethernet.
   ii) Multicast in the Internet make use of **multicast routers**. The distance of propagation is specified in the Time To Live (**TTL**).
2) **Multicast Address Allocation:** Class D addresses (range from 224.0.0.0 to 239.255.255.255) are reserved for multicast traffic and managed globally by the Internet Assigned Numbers Authority (IANA).

Address space are partitioned into a number of blocks, including:

i) Local Network Control Block (224.0.0.0 to 224.0.0.225), for multicast traffic within a given local network.

ii) Internet Control Block (224.0.1.0 to 224.0.1.225).

iii) Ad Hoc Control Block (224.0.2.0 to 224.0.255.0), for traffic that does not fit any other block.

iv) Administratively Scoped Block (239.0.0.0 to 239.255.255.255), which is used to implement a scoping mechanism for multicast traffic (to constrain propagation).

Multicast address can be temporary or permanent.

i) A temporary multicast group requires a free multicast address to avoid accidental participation in an existing group before use and cease after use.

ii) Permanent groups exist even when there are no members – their addresses are assigned by IANA and span the various blocks.

Datagrams multicast over IP multicast suffer from omission failures. This is an unreliable multicast.

Ques 13)Discuss the failure model of IP multicast.

Ans: Failure Model of IP Multicast

The following situations could occur:

1) **Omission Failure:** A recipient could drop a message because its buffer is full. A datagram sent from on multicast router to another might be lost.

2) **Process Failure:** If a multicast router fails, the group members beyond that router won't receive the message.

3) **Ordering Issue:** The packets could arrive in a different order.

Ques 14)What is remote procedure call?

Ans: Remote Procedure Call (RPC)

Remote Procedure Call (RPC) is a powerful technique for constructing distributed, client-server based applications. It is based on extending the conventional local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them.

When making a Remote Procedure Call (**figure 3.11**):

1) The calling environment is suspended, procedure parameters are transferred across the network to the environment where the procedure is to execute, and the procedure is executed there.

2) When the procedure finishes and produces its results, its results are transferred back to the calling environment, where execution resumes as if returning from a regular procedure call.

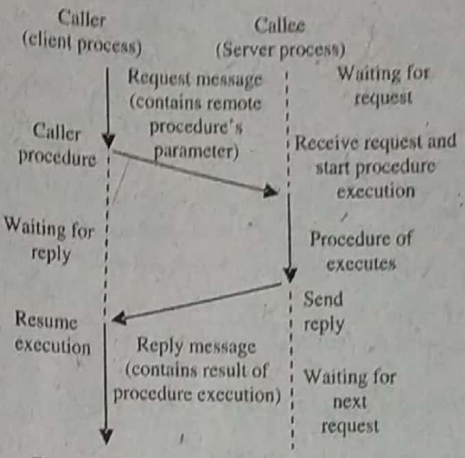

Figure 3.11: Remote Procedure Call Model

Ques 15)Write the working of RPC.

Or

Discuss about the implementation of RPC mechanism.

Ans: Working of RPC

The following steps take place during a RPC:

Step 1) A client invokes a client stub procedure, passing parameters in the usual way. The client stub resides within the client's own address space.

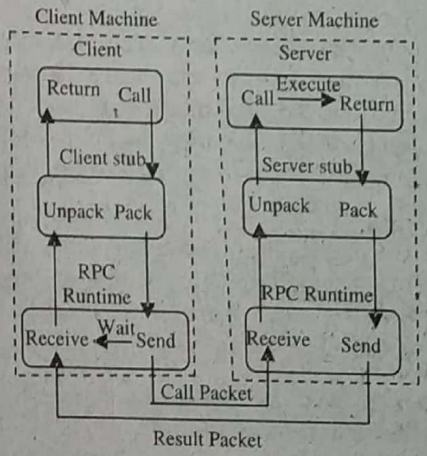

Figure 3.12: Implementation of RPC Mechanism

Step 2) The client stub marshalls (pack) the parameters into a message. Marshalling includes converting the representation of the parameters into a standard format, and copying each parameter into the message.

Step 3) The client stub passes the message to the transport layer, which sends it to the remote server machine.

Step 4) On the server, the transport layer passes the message to a server stub, which demarshalls(unpack) the parameters and calls the desired server routine using the regular procedure call mechanism.

Step 5) When the server procedure completes, it returns to the server stub (e.g., via a normal procedure call return), which marshalls the return values into a message. The server stub then hands the message to the transport layer.

Step 6) The transport layer sends the result message back to the client transport layer, which hands the message back to the client stub.

Step 7) The client stub demarshalls the return parameters and execution returns to the caller.

Ques 16)Explain the issues related to RPC.

Ans: Issues Related to RPC

Issues that must be addressed are as follows:

1) **RPC Runtime:** RPC run-time system, is a library of routines and a set of services that handle the network communications that underline the RPC mechanism. In the course of an RPC call, client-side and server-side run-time systems' code handle binding, establish communications over an appropriate protocol, pass call data between the client and server, and handle communications errors.

2) **Stub:** The function of the stub is to provide transparency to the programmer-written application code:
   i) **On the Client Side:** The stub handles the interface between the client's local procedure call and the run-time system, marshalling and unmarshalling data, invoking the RPC run-time protocol, and if requested, carrying out some of the binding steps.
   ii) **On the Server Side:** The stub provides a similar interface between the run-time system and the local manager procedures that are executed by the server.

3) **Binding:** How does the client know who to call, and where the service resides?
   The most flexible solution is to use dynamic binding and find the server at run time when the RPC is first made. The first time the client stub is invoked, it contacts a name server to determine the transport address at which the server resides

Binding consists of two parts:

i) **Naming:** Remote procedures are named through interfaces. An interface uniquely identifies a particular service, describing the types and numbers of its arguments. It is similar in purpose to a type definition in programming languages.

ii) **Locating:** Finding the transport address at which the server actually resides. Once we have the transport address of the service, we can send messages directly to the server.

A server having a service to offer exports an interface for it. Exporting an interface registers it with the system so that clients can use it.

A client must import an (exported) interface before communication can begin.

Ques 17)What are the advantages of RPC?

Ans: Advantages of RPC

1) RPC provides abstraction i.e. message-passing nature of network communication is hidden from the user.
2) RPC often omits many of the protocol layers to improve performance. Even a small performance improvement is important because a program may invoke RPCs often.
3) RPC enables the usage of the applications in the distributed environment, not only in the local environment.
4) With RPC code re-writing / re-developing effort is minimized.
5) Process-oriented and thread oriented models supported by RPC.

Ques 18)What do you understand by network virtualization?

Ans: Network Virtualization

Network virtualization is the process of combining hardware and software network resources and network functionality into a single, software-based administrative entity, a virtual network. Network virtualization involves platform virtualization, often combined with resource virtualization.

Network virtualization is a method of combining the available resources in a network by splitting up the available bandwidth into channels, each of which is independent from the others, and each of which can be assigned (or reassigned) to a particular server or device in real time. Each channel is independently secured. Every subscriber has shared access to all the resources on the network from a single computer.

Network virtualization is categorized as either **external virtualization**, combining many networks or parts of

networks into a virtual unit, or **internal virtualization**, providing network-like functionality to software containers on a single network server.

Network virtualization is intended to improve productivity, efficiency, and job satisfaction of the administrator by performing many of these tasks automatically, thereby disguising the true complexity of the network. Files, images, programs, and folders can be centrally managed from a single physical site. Storage media such as hard drives and tape drives can be easily added or reassigned. Storage space can be shared or reallocated among the servers.

Network virtualization is intended to optimize network speed, reliability, flexibility, scalability, and security. Network virtualization is said to be especially effective in networks that experience sudden, large, and unforeseen surges in usage.

Ques 19) What is overlay network?

Ans: Overlay Network

An overlay network is a telecommunications network that is built on top of another network and is supported by its infrastructure. An overlay network decouples network services from the underlying infrastructure by encapsulating one packet inside of another packet. After the encapsulated packet has been forwarded to the endpoint, it is de-encapsulated.

Most overlay networks run on top of the public Internet, which itself began as an overlay research network running over the infrastructure of the public switched telephone network (PSTN). Other examples of overlay network deployments include virtual private networks (VPNs), peer-to-peer (P2P) networks, content delivery networks (CDNs), voice over IP (VoIP) services such as Skype and non-native software-defined networks.

An overlay network is a layer of virtual network topology on top of the physical network, which directly interfaces to users. With the rapid advancement of Internet and computing technology, much more aggregate information and computing resources are available from clients or peers than from a limited number of centralized servers.

Typical overlay networks include multicast overlays, peer-to-peer overlays (**e.g.** Gnutella and Kazaa), parallel file downloading overlays (**e.g.** BitTorrent and eDonkey), routing overlays (**e.g.** skype for VoIP).

Ques 20) What are the advantages and disadvantages of overlay network?

Ans: Advantages of Overlay Network

1) Overlay networks allow both networking developers and application users to easily design and implement their own communication environment and protocols on top of the Internet, such as data routing and file sharing management.
2) Data routing in overlay networks can be very flexible, quickly detecting and avoiding network congestions by adaptively selecting paths based on different metrics, such as probed latency.
3) The end-nodes in overlay networks are highly connected to each other due to flexible routing. As long as the physical network connections exist, one end-node can always communicate to another end-node via overlay networks. Thus, scalability and robustness in overlay networks are two attractive features.
4) The high connectivity of increasingly more end-nodes to join overlay networks enables effective sharing of a huge amount of information and resources available in the Internet.

Disadvantages of Overlay Network

1) Overlay networks not only have no controls of physical networks, but also lack critical physical network information.
2) Because of the indirect or even mis-communications between overlay and underlay networks, in practice, inefficient usage network resources are quite often in many overlay applications, such as mismatch between overlay and underlay topology, inaccurate probing results among end-to-end nodes due to network dynamics, generating a large amount of redundant messages, and others.
3) Since the overlay networks are open to all kinds of Internet users, security and privacy issues can be quite serious.
4) Overlay networks are highly decentralized, thus they are likely to have weak ability for resource coordinations.
5) Fairness of resource sharing and collaborations among end-nodes in overlay networks are two critical issues that have not been well addressed.

Ques 21) Write a case study on skype.

Or

Using an example explain the overlay network.

Ans: Case Study – Skype/Example of Overlay Network Skype is a Voice-Over-IP (VOIP) P2P application with a historical connection to Kazaa and FastTrack. It uses a centralised server for finding the address of a friend. The client-to-client voice connection is direct and does not pass through any server. Instant messaging is also a centralised service that employs client presence, detection and location. A user registers its IP address with the central server when it comes online and contacts the central server to find IP address for friends. In this environment, chat between two users is P2P.

Skype is a proprietary application-layer protocol with a hierarchical overlay containing super-nodes, as illustrated in **figure 3.13**. A P2P Voice-Over-IP (VOIP) application may be PC-to-PC, PC-to-Phone, or Phone-to-PC. Like Fastrack, the supporting network contains selected nodes (supernodes) with high-speed links and high computational power as well as ordinary nodes and a login server that is a centrally managed service. Within this architecture, a Skype client authenticates the user with the login server , advertises its presence to other peers, determines the type of NAT and firewall it lies behind and discovers nodes that have public IP addresses. In order to connect to the Skype network, the host cache must contain a valid login entry in order to establish a TCP connection to a supernode; otherwise the login will fail.

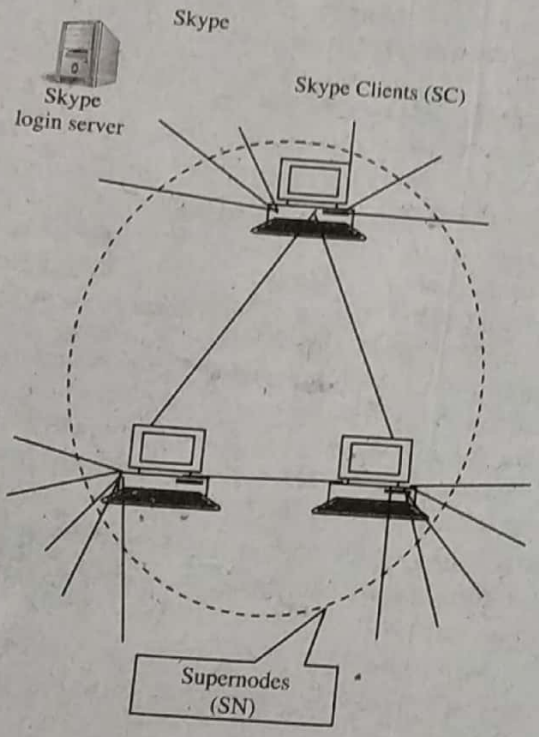

Figure 3.13: Skype Network

Step-by-Step Procedure for Making a Phone Call Using Skype

The process of making a call with Skype is outlined in the following steps, and illustrated in **figure 3.14**, where SC represents a Skype client and SN represents a supernode. Both Alice and Bob must have Skype accounts that are obtained by registering at Skype's website. A set of SNs are preloaded in the Skype client software.

Step 1: SC connects to SN using a list of bootstrap SNs.

Step 2: SC logs in to authenticate Alice.

Step 3: SC makes a call by contacting the SN with callee ID for Bob.

Step 4: SN contacts other SNs to find the IP address of Bob and Bob's IP address is returned to SC.

Step 5: SC directly contacts Bob over TCP.

Step 6: Alice starts Skype conversation with Bob.

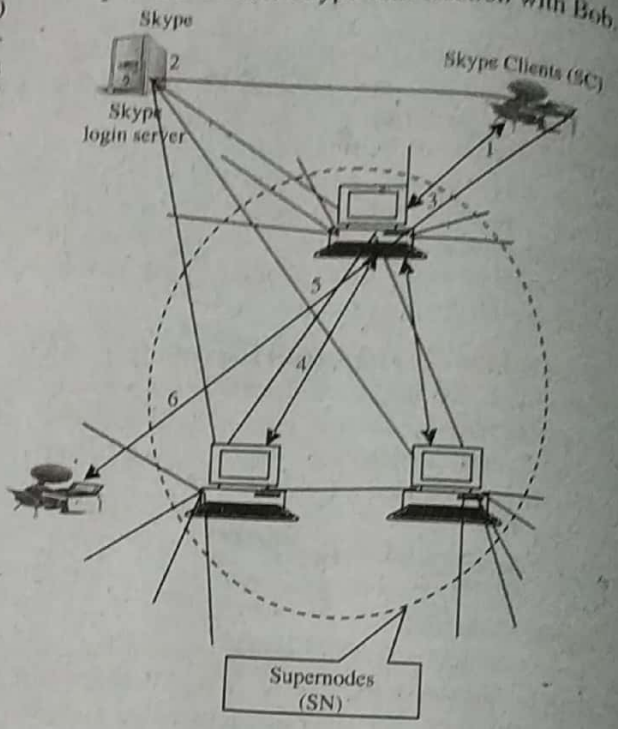

Figure 3.14: Alice Makes a Call to Bob Using Skype

Procedural Details Involved in a Skype Call

The details of the Skype calling are outlined as follows:

Step 1: Alice establishes a connection to one of the Super Nodes.

Step 2: Alice is authenticated with the login server.

Step 3: User Bob's IP address is looked up.

Step 4: A connection is made to callee Bob.

Step 5: The signal message is sent and then caller and callee talk.

The initial step in making a call with Skype involves contacting the super node as illustrated in **figure 3.15**. As indicated, the caller attempts a TCP connection Request/reply Message using the host cache on a HTTP port. The super nodes are responsible for such things as accepting connections, locating users and routing calls.

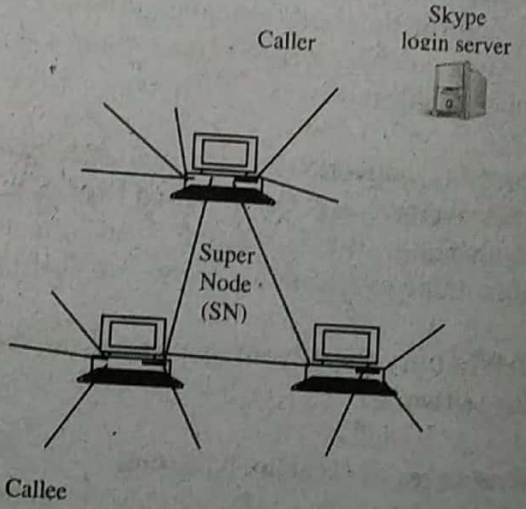

Figure 3.15: Alice Registers with Super Node

the user, Alice, then requests a login from the Skype login server, as shown in **figure 3.16**.

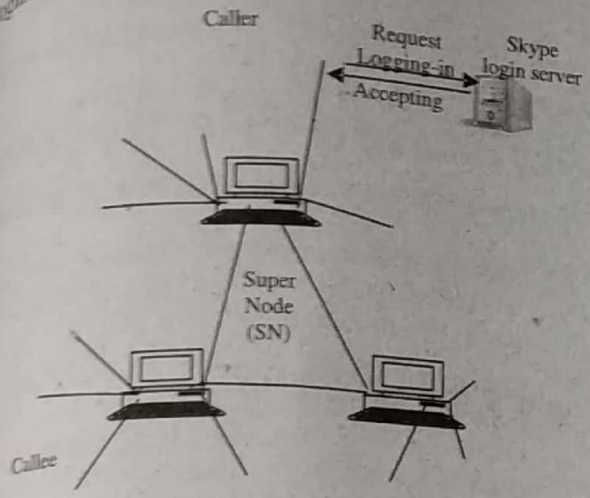

Figure 3.16: User Authentication

The caller requests authentication and provides and ID and password.

After a successful login, the caller, Alice, now needs to determine callee's (Bob's) IP address and port number using Bob's ID, as indicated in **figure 3.17**. This data is obtained through the Skype users directory, i.e., a tree, which is distributed on the Skype network.

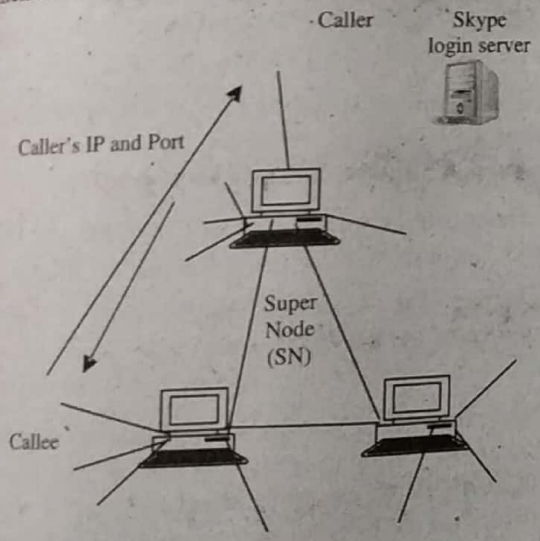

Figure 3.17: Acquiring IP Address

Hence, a supernode will contact other supernodes in order to determine callee's IP address, if necessary, and forward it to the caller.

Once the callee's IP address and port number have been obtained, caller requests a direct TCP connection to callee, as shown in **figure 3.18**.

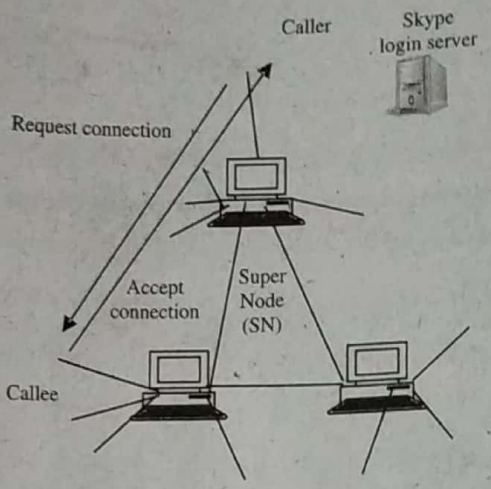

Figure 3.18: Caller/Callee Direct Connection

Finally, a signal message is sent and the skype call connection is complete, as indicated in **figure 3.19**. Now caller and callee can talk.

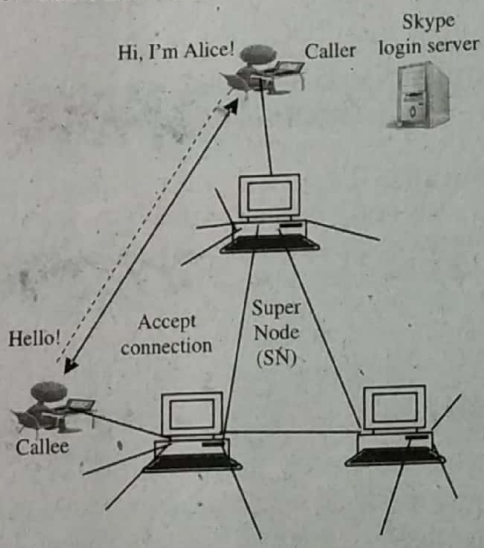

Figure 3.19: Completing the Connection