UNIT-3

INTER PROCESS COMMUNICATION AND SYNCRONIZATION

INTER PROCESS COMMUNICATION -

1. API for Internet Protocol - (API → Application program Interface)

Characteristics of interprocess communication - (IPC)

1. Synchronous and asynchronous communication -

In 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, the process (or thread) blocks until a message arrives.

In 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.

Non-blocking communication appears to be more efficient.

2. Message destination -

IPC can send messages to groups of destinations (either ports or processes). Also, Messages can be sent to (Internet address, local port) pairs. A local port is a message destination within a computer, specified as an integer.

Ports have advantages over processes, but they provide several alternative points of entry to a receiving process.

3. Reliability -

Reliable communication should have validity and integrity properties.

4. Ordering -

Some applications require that messages be delivered in the order in which they were transmitted by the sender.

Sockets -

IPC consists of transmitting a message between a socket in one process and a socket in another process.

Each socket is associated with a particular protocol - either UDP or TCP.
Each computer has a large number \(2^{16}\) of possible port numbers for use by local processes for receiving messages.

**JAVA API for Internet Addresses**

As the IP packets underlying UDP and TCP are units to internet addresses, Java provides a class, `InetAddress`, that represents Internet addresses. Users of this class refer to computers by DNS hostnames.

```java
InetAddress computer = InetAddress.getByName("www.google.com");
```

**UDP Datagram Communication**

A datagram sent by UDP is transmitted from a sending process to a receiving process without acknowledgment or retries. If a failure occurs, the message may not arrive.

**Issues related to datagram communication**

1. **Message Size** - The receiving process needs to specify an array of bytes of a particular size in which to receive a message. The underlying protocol allows packet lengths of up to \(2^{16}\) bytes which includes headers + message.
2. **Blocking** - Sockets normally provide non-blocking sends and blocking receives for datagram communication.
3. **Timeout** - It is not appropriate that a process that has issued a receive operation should wait indefinitely in situations where the potential sending process has crashed or the sent packet message has been lost. That's why timeouts context on sockets.
4. **Receive from any** - The receive method does not specify an origin from messages. Only internet address and local part of the thread is known.
Failure Model for UDP datagrams - Two failures -

1) Omisión failures - Messages may be dropped occasionally, either because of
    a checksum error or because no buffer space is available at the source or
destination.
2) Rerouting - Messages can sometimes be delivered out of order order.

Use of UDP -

1) Domain Naming Service is implemented over UDP.
2) Voice over IP (VoIP).
3) Do not suffer overhead (need to flow information, extra messages).

Java API for UDP datagrams -

Java API provides datagram communication by means of two classes
DatagramPacket and DatagramSocket.

DatagramPacket -

| array of bytes containing message | length of message | Internet Address | Port No |

get data method gives the message, get port and get address access the
port and Internet Address.

DatagramSocket -

send and receive methods are for transmitting datagrams between a pair
of sockets. These methods throw IOException.

setSoTimeout method allows the timeout to be set and throws
IOException.

receive method will block for the time specified before throwing
an InterruptedIOException.

connect method is used for connecting it to a particular remote port and
Internet address.

TCP stream communication -

The API to the TCP protocol provides the abstraction of a stream of
bytes to which data may be written and from which data may be read.

Characteristics of the network that are hidden by the stream abstraction -

1) Message Sizes - The application can choose how much data it writes
to a stream or reads from it. The underlying implementation of a TCP stream
decides how much data to collect before transmitting it as one or more IP packets.

2. Host Manager: TCP protocol uses an acknowledgement scheme.

3. Flow Control: TCP protocol attempts to match the speeds of the processes that read from and write to a stream.

4. Message Duplication and Ordering: Message identifiers are associated with each IP packet, which enforces the recipient to detect and reject duplicates, or to reorder messages that do not arrive in sequence.

5. Message Destination: A pair of communicating processes establish a connection before they can communicate over a stream.

> Issues related to stream communication -

1. Matching of data items: Two communicating processes need to agree as to the content of the data transmitted over a stream.

2. Blocking: The process that writes data to a stream may be blocked by the TCP flow control mechanism if the socket at the other end is queuing as much data as the protocol allows.

3. Threads: When a server accepts a connection, it generally creates a new thread in which to communicate with the new client.

> Failure Model for TCP stream -

To satisfy the integrity and reliability of communication, TCP streams use checksums to detect and reject corrupt packets and sequence numbers to detect and reject duplicate packets. For the sake of the validity, TCP streams use timeouts and autoretransmission to deal with lost packets.

When a connection is broken -

1. The processes using the connection cannot distinguish between network failure and failure of the process at the other end of the connection.

2. The communicating processes cannot tell whether the messages they sent recently have been received or not.

> Use of TCP - HTTP, FTP, Telnet, and SMTP.

info companion
Java API for TCP streaming

The Java interface to TCP streams is provided in the classes Socket and ServerSocket.

`ServerSocket` - to create a socket at a given port

accept method gets a connect request from the queue, or if the queue is empty, it blocks until one arrives. It gives access to streams for communicating with the client.

`Socket` -

gets InputStream and getOutputStream methods are for accessing the two streams associated with a socket.

`Socket` can throw an UnknownHostException or IOException.

2. External data representation and marshalling

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

Marshalling is the process of taking a collection of data items and assembling them into a form suitable for transmission on a network.

Unmarshalling is the process of disassembling them on arrival to produce an equivalent collection of data items at the destination.

Three alternative approaches to external data representation and marshalling are:

1. CORBA's common data representation (CDR) -

   It is concerned with an external representation for the structural and primitive types that can be passed as arguments and results of remote method invocation (RMI) in CORBA.

   **Primitive types** -
   - Supports both big-endian and little-endian
   - Transmitted in endian's ordering and the ordering specified
   - Receive translate if needed
   - 15 primitive types - short (16 bits), long (32 bits), unsigned short, unsigned long, float (32 bits), double (64 bits), char, boolean, octet (8 bit) and any
- Constructed types are -
  (1) **Sequence** - length (unpacked long) followed by elements in order
  (2) **String** - length (unpacked long) followed by characters in order
  (3) **Array** - array elements in order (no length specified because it is finite)
  (4) **Enumerated** - unpacked long (the values are specified by the order declared)
  (5) **Union** - type tag followed by the selected member

<table>
<thead>
<tr>
<th>Indic in sequence of bits</th>
<th>Length of thing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td></td>
</tr>
<tr>
<td>4-7</td>
<td>&quot;Smit&quot;</td>
</tr>
<tr>
<td>8-11</td>
<td>&quot;h_&quot;</td>
</tr>
<tr>
<td>12-15</td>
<td>6</td>
</tr>
<tr>
<td>16-19</td>
<td>&quot;Lond&quot;</td>
</tr>
<tr>
<td>20-23</td>
<td>&quot;on_&quot;</td>
</tr>
<tr>
<td>24-27</td>
<td>1934</td>
</tr>
</tbody>
</table>

The type of a data item is not given with the data representation in the message because it is assumed that the sender and receiver have common knowledge of the order and type of the data items in a message.

**Marshalling in CORBA** -

Marshalling operators can be generated automatically from the specification of the type of data items to be transmitted in a message or stored on a disk.

**Java object serialization** -

It is concerned with the flattening and external data representation of any single object or tree of objects that may need to be transmitted in a message or stored on a disk.

```java
public class Person implements Serializable {
    private String name;
    private String place;
    private int year;
    public Person (String aName, String aPlace, int aYear) {
        myCompanion
    }
```
name = a Name;
place = a Place;
year = a Year;

// followed by methods for accessing the instance variables.

In Java, the term serialization interface, which refers to the activity of flattening an object or a connected set of objects into a serial form that is suitable for storing on disk or transmitting a message. Deserialization consists of restoring the state of an object or a set of objects from their serialized form.

Indication of Java serialized form -

<table>
<thead>
<tr>
<th>Person</th>
<th>8-byte version number</th>
<th>h2</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| 3      | int year, java.lang.String name | java.lang.String place | class name, version no.,
|        | 1934 | "smith" | number types and names of
|        |      | 6 "london" | instance variables |

The true serialized form contains additional type markers; h2 and h1 are handles.

References are serialized as handles. To serialize an object, its class information is written out, followed by the types and names of its instance variables.

Serialization and deserialization of the arguments and results of remote invocations are generally carried out automatically by the middleware, without any participation by the application programmer.

The Use of Reflective -

The Java language supports reflection (the ability to enquire about the properties of a class, such as the names and types of its instance variables and methods. It also enables classes to be created from their names, and a constructor with given argument types to be created for a given class.

It helps in serialization and deserialization in a completely generic manner.
Extensible Markup Language (XML) -

It is a markup language defined by the World Wide Web Consortium (W3C) for general use on the Web. In general, the term markup language refers to a textual encoding that conveys both a text and details as to its structure or its namespace.

Tags relate to the structure of the text that is enclosed. XML data items are tagged with 'markup' strings.

XML is extensible in the sense that users can define their own tags.

The use of textual, rather than a binary representation, together with the use of tags makes the messages much larger, which causes them to acquire longer processing times and transmission times, as well as more space to store, but the ability to read XML can be useful when things go wrong.

XML elements and attributes -

```xml
<person id="123456789">
  <name>Smith</name>
  <place>London</place>
  <year>1934</year>
</person>
```

Elements consist of a portion of character data surrounded by matching start and end tags.

Attributes consist of name and values.

Binary data of XML can be represented in base64 notation.

Parsing and well-formed documents -

Every start tag has a matching end tag and all tags are correctly nested.

CDATA can be used where the section cannot be parsed or we can use &.

E.g. `<place> <![CDATA[King's Cross]]> </place>`

XML Prolog - Specifies the XML version, encoding and standalone status.

E.g. `<?xml version="1.0" encoding="UTF-8" standalone="yes"?>`

XML Namespaces -

It is a set of names for a collection of elements, types and attributes, that is referenced by a URL.
XML namespace

- `xmlns:pers = "http://www.cdk4.net/person"`
  
  prefix to refer to the elements

<XML schema>

It defines the elements and attributes that can appear in a document, how the elements are nested, and the order and number of elements. Whether an element is empty or can include text. For each element, it defines its type and default values. Eg - XML schema for the Person structure

```xml
<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
>
  <xsd:element name = "person" type="personType"/>
  <xsd:complexType name = "personType"/>
    <xsd:sequence>
      <xsd:element name = "name" type = "xsd:string"/>
      <xsd:element name = "place" type = "xsd:string"/>
      <xsd:element name = "year" type = "xsd:positiveInteger"/>
    </xsd:sequence>
  <xsd:attribute name = "id" type = "xsd:positiveInteger"/>
</xsd:complexType>
</xsd:schema>
```

Document type definitions (DTDs) define the structure of XML documents.

APIs for accessing XML - XML parsers and generators are available for most commonly used programming languages.

Remote object references - (Applies only to Java & CORBA, not XML)

It is an identifier for a remote object that is valid throughout a distributed system. It is passed in the invocation method message to specify which object is to be invoked. Remote object references must be unique.

<table>
<thead>
<tr>
<th>32 bits</th>
<th>32 bits</th>
<th>32 bits</th>
<th>32 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERNET ADDRESS</td>
<td>PORT NUMBER</td>
<td>TIME</td>
<td>OBJECT NUMBER</td>
</tr>
</tbody>
</table>

my companion
Group communication -

A multicast operation sends a single message from one process to each of the members of a group of processes, usually in such a way that the membership of the group is transparent to the sender.

Characteristics of multicast messages -

1. Fault tolerance based on replicated services.
2. Finding the discovery process in spontaneous networking.
3. Better performance through replicated data.
4. Propagation of event notifications.

IP multicast - an implementation of group communication.

IP multicast allows the sender to transmit a single IP packet to a set of computers that form a multicast group. The sender is unaware of the identities of recipients and of the size of the group.

The membership of multicast group is dynamic.

At the application programming level, IP multicast is available only via UDP. At the IP level, a computer belongs to a multicast group when one or more of its processes has sockets that belong to that group.

Multicast Routers - Multicast in the Internet which forward single datagrams to routers on other networks with members, where they are again multicast to local members.

Multicast address allocation - IP may be permanent or temporary assigned by Internet authority from the range 224.0.0.1 to 224.0.0.255.

Use time to live (TTL) to limit the number of hops.

Tools like sd (session directory) can help manage multicast addresses and find new ones.

Failure model for multicast datagrams - same as UDP datagrams that in transmission omission failure and annoying problem.

Java API to IP multicast - through class MulticastSocket, which is a sub class of DatagramSocket.

Multicast socket methods are joinGroup, leaveGroup and setTTL timeToLive.

*default is 1
Reliability and ordering of multicast:

- Effects of reliability and ordering of failure semantics of IP multicast can:
  1. Fault tolerance based on replicated servers:
     Ordering of the requests might be important, servers can be inconsistent with one another.
  2. Finding the discovery server in spontaneous networking — Not too problematic.
  3. Better performance through replicated data —
     non and out-of-order update could yield inconsistent data, sometimes this may be intolerable.
  4. Propagation of event notification — Not too problematic.

4. Client-Server Communication:

In synchronous request-reply communication, client waits for a reply whereas asynchronous request-reply communication, client doesn’t wait for a reply.

Request-reply protocol:

It is based on a trio of communication primitives: doOperation,

getReply and sendReply.

Client

\[
\text{doOperation} \\
\text{(wait)} \\
\text{(continuation)}
\]

Server

\[
\text{getRequest} \\
\text{select object} \\
\text{execute method} \\
\text{sendReply}
\]

REQUEST-REPLY COMMUNICATION

Operations of the request-reply protocol:

- public byte[] doOperation (RemoteObjectRef o, int methodId, byte[] arguments);
- public byte[] getReply();
- public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);
Request-Reply Message Structure:

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Int (0 = Request, 1 = Reply)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequestId</td>
<td>Int</td>
</tr>
<tr>
<td>ObjectReference</td>
<td>Remote ObjectReference</td>
</tr>
<tr>
<td>MethodId</td>
<td>Int or Method</td>
</tr>
<tr>
<td>Arguments</td>
<td>Array of bytes</td>
</tr>
</tbody>
</table>

Message identifiers - Cannot set two parts that is a RequestId and an identifier like port and Internet address.

When the value of the RequestId reaches the maximum value for an unsigned int (e.g., $2^{32} - 1$) it is reset to zero. The only restriction here is that the lifetime of a message identifier should be much less than the time taken to exhaust the value in the sequence of integers.

Failure model of the Request-Reply protocol:

If implemented over UDP datagrams, they suffer from omission failures and data loss. In addition, the protocol can suffer from failure of processes.

Timeout - Return immediately from operation with an indication to the client that the operation is failed. No getting a reply.

Duplicate request messages - Protocol is designed to recognize incoming messages with the same request identifier and to filter out duplicates.

Incorrect operation - It is an operation that can be performed repeatedly with the same effect as if it had been performed exactly once.

History - For services that require retransmission of replies without re-execution of operations, a history may be used. A client can make only one request at a time, therefore the history need only contain only the last reply message sent to each client.

RPC Exchange Protoco:

Three protocols, which produce differing behaviors in the presence of communication failures, are used to implement various types of RPC:

- Request (R) protocol
- Request-Reply (RR) protocol
- Request-Reply-Acknowledgment reply (RRR) protocol
<table>
<thead>
<tr>
<th>NAME</th>
<th>CLIENT</th>
<th>SERVER</th>
<th>CLIENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>Request</td>
<td>Reply</td>
<td></td>
</tr>
<tr>
<td>RR</td>
<td>Request</td>
<td>Reply</td>
<td></td>
</tr>
<tr>
<td>RRA</td>
<td>Request</td>
<td>Reply</td>
<td>Acknowledgement Reply</td>
</tr>
</tbody>
</table>

Use of TCP streams to implement the request-reply protocol:

1. Avoid implementing multi-packet protocols.
2. Allow arguments and results of any size to be transmitted.
3. Reliability issues

HTTP - an example of a request-reply protocol. (implemented over TCP)

Hypertext Transfer Protocol (HTTP) used by web browsers to make requests to web servers and to receive replies from them.

The protocol allows for content negotiation and form-based style authentication.

HTTP 1.1 uses persistent connections - connections that remain open over a series of request-reply exchanges between client and server until the connection is closed by the server or client at any time or by servers after timeout.

Requests and replies are marshalled into messages as ASCII text strings.

Resources implemented as data are supplied as MIME-like structures in arguments and entities. Multipurpose Internet Mail Extensions (MIME) is a standard for sending multimedia data containing, for eg. text, images, Sounds in email messages.

HTTP Methods:

- **GET** → request the resource whose URL is given as argument.
- **HEAD** → Return all the information about the data.
- **POST** → specify the URL of a resource that can deal with the data supplied with the request.
- **PUT** → request that the data supplied in the request is stored with the given URL as its identifier, either as a modification of an existing resource or as a new resource.
- **DELETE** → the server deletes the resource identified by the given URL.
- **OPTIONS** → the server supplies the client with a list of methods it allows to be applied to the given URL (eg. GET, HEAD, PUT) and its special requirements.
- **TRACE** → the server sends back the request message.
HTTP Request Message:
- method
- URL or pathname
- HTTP version
- headers
- message body

HTTP Reply Message:
- HTTP version
- status code
- reason
- headers
- message body

RPC (Remote Procedure Call) -

1. Implementing RPC Mechanism

   To achieve semantic transparency, implementation of RPC mechanism is based on the concept of stubs.

   A separate stub procedure is associated with both the client and server systems.

   To hide the underlying communication network, RPC communication package known as RPC Runtime is used on both the sides.

   The client, the client stub, RPC runtime, server stub and server on the client machine. The server, the server stub, and one instance of RPC runtime execute on the server machine.

<table>
<thead>
<tr>
<th>Client Machine</th>
<th>Server Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return Call</td>
<td>Call Execute → Return</td>
</tr>
<tr>
<td>Client Stub</td>
<td>Server stub</td>
</tr>
<tr>
<td>Unpack Pack</td>
<td>Unpack Pack</td>
</tr>
<tr>
<td>RPC Runtime</td>
<td>RPC Runtime</td>
</tr>
<tr>
<td>Receive → Wait</td>
<td>Receive Send</td>
</tr>
</tbody>
</table>

IMPLEMENTATION OF RPC MECHANISM
Client - It calls the local procedure called client stub for remote service

Client stub - It is responsible for two tasks:

1. On receipt of a call request from the client:
   - It packs a specification of the target procedure and the arguments into a message
   - Sends the local RPC runtime to send it to the server stub

2. On receipt of the result of procedure execution, it unpacks the result and
   passes it to the client

RPC Runtime - It handles transmission of messages across the network between
client and server machine. It is responsible for retransmission, acknowledgment,
routing and encryption

Server stub - It is responsible for two tasks:

1. On receipt of a call request message from the local RPC runtime, it unpacks
   it and makes a perfectly normal call to invoke the appropriate procedure in the
   server

2. On receipt of the result of procedure execution from the server, it unpacks the
   result into a message and then asks the local RPC runtime to send it to the client stub

Server - On receiving a call request from the server stub, the server executes the
appropriate procedure and returns the result of procedure execution to the server stub

2. Stub Generation -

   It can be generated in one of the following two ways:

1. Manually -
   The RPC implementor provides a set of translation functions from which
   a user can construct his own stubs. This method is simple to implement
   and can handle very complex parameter types.

2. Automatically -
   Interface Definition Language (IDL) is used here to define the
   interface between a client and the server.

   Interface definition - It is a list of procedure names supported by the
   interface together with the types of their arguments and results.
   It also plays a role in reducing data storage and controlling amount of data
   passed.
RPC system is independent of transport protocols and is not concerned on how a message is passed from one

user to another.

transformed over the internet network.

It has information about type definitions, enumerated types, and defined constants.

Export the interface - A server program that implements procedures in the

interface.

Import the interface - A client program that calls procedures from an

interface.

The interface definition is compiled by the IDL compiler.

IDL compiler generates:

1. Components that can be compiled with client and server programs,
   without making any changes to the existing compiler.
2. Client stub and server stub procedures.
3. The appropriate marshalling and unmarshalling operations.
4. A header file that supports the data types.

RPC Messages -

Two types of messages involved in the implementation of an RPC system are

1. Call Messages -
   Sent by the client to the server for requesting execution of a
   particular remote procedure.

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Client Identifier</th>
<th>Remote procedure identifier</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier</td>
<td>Type</td>
<td>Identifier</td>
<td>Program Number</td>
</tr>
</tbody>
</table>

RPC call message format:

- Identity test and O - call message for authentication
- Duplicate message, A - reply message and identification

2. Reply Messages -
   Sent by the server to the client for returning the result of remote
   procedure execution.

   Conditions for unsuccessful message sent by the server -
   1. The server finds that the call messages is not intelligible to it.
   2. Client is not authorized to use the service.
   3. Remote procedure identifier is missing.

companion
(iv) The remote procedure is not able to decode the supplied arguments.

(v) Occurrence of exception condition.

<table>
<thead>
<tr>
<th>Message Identifier</th>
<th>Message Type</th>
<th>Reply Status</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(Successful)</td>
<td></td>
</tr>
</tbody>
</table>

Successful reply message format

<table>
<thead>
<tr>
<th>Message Identifier</th>
<th>Message Type</th>
<th>Reply Status</th>
<th>Reason for failure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(Unsuccessful)</td>
<td></td>
</tr>
</tbody>
</table>

Unsuccessful reply message.

Synchronization -

1. Clock Synchronization -
   How computer clocks are implemented -

A computer clock usually consists of three components -

1. A quartz crystal that oscillates at a well-defined frequency.
2. A constant register is used to store a constant value that is decided based on the frequency of oscillation of the quartz crystal.
3. A counter register is used to keep track of the oscillations of the quartz crystal.

To make the computer clock function as an ordinary clock -

1. The value in the constant register is chosen so that 60 clock ticks occur in 6.
2. The computer clock is synchronized with real time.

Drifting of Clocks -

The difference in the oscillation period between two clocks might be extremely small, but the difference accumulated over many oscillations lead to an obvious computer clock drift from the real-time clock.

Clocks based on a quartz crystal, drift rate is approximately $10^{-6}$, giving a difference of 1 second every 1,000,000 seconds or 11.6 days.

A clock is said to be non-faulty if the following condition holds for it -

$$1 - \rho < \frac{dC}{dt} < 1 + \rho$$

where $C$ = time value of a clock

$\rho$ = maximum drift rate.

Clock drift time interval between two synchronizations

$$\Delta t \leq \frac{s}{2\rho}$$

where $s$ = time difference between two clocks.
Type of clock synchronization -

1. Synchronization of the computer clock with real-time (or internal) clocks.
2. Mutual (or internal) synchronization of the clocks of different nodes of the system.

Clock Synchronization Issues -

1. Clock synchronization requires each node to read the other nodes' clock values.
2. Time must never run backwards (in case of fast clock readjusted to actual time).

Clock Synchronization Algorithms -

1. Centralized Algorithms -

   One node has a real-time receiver. This node is usually called the time server node, and clock times of this node is regarded as correct and used as the reference time. The goal of the algorithm is to keep the clocks of all other nodes synchronized with the clock time of the time server node. Two types:

   (i) Passive time server centralized algorithm -

      In this method, each node periodically sends a message to the time server. When the time server receives the message, it quickly responds with a message.

      The clock is readjusted to \[ T + \frac{(T_T - T_0)}{2} \], where:
      
      \( T \rightarrow \) current time,
      \( T_0 \rightarrow \) when the client node sends the message,
      \( T_T \rightarrow \) when it receives the "time-to" message.

      Two cases -

      1. If no additional information is available then, \[ T + \frac{(T_T - T_0 - T)}{2} \], where
         
         \( T \rightarrow \) time taken by the time server to handle the interrupt and process a time request message.

      2. If additional information is not available then, several measurement of \( T_T - T_0 \) are made. Minimum value of \( T_T - T_0 \) is considered to be the most accurate one then

         \[ T + \frac{[\text{Min}(T_T - T_0)]}{2} \]

11. Active time server centralized algorithm -

      The time server periodically broadcasts its clock time. The other nodes receive the broadcast message and use the clock time in the messages for correcting their own clocks.
fault-tolerant average -
Time server chooses a subset of all clocks values having
≈ do not differ from one another by more than
a stipulated amount and then, the average is taken.

Nodes' clock is readjusted to the time \[ T + T_0 \] where \( T \) - current time.
\( T_0 \) – Prior knowledge of the approximate time \( T_0 \) required for the propagation of
the message from the server node to its own node.

Drawback - Not fault tolerant, requires broadcast facility.

To remove the above drawback, Berkeley algorithm can be used.

\[ \text{Berkeley algorithm -} \]

The time server periodically sends a message to all the computers
in the group. On receiving this message, each computer sends back its clock value
to the time server.

The time server has a priori knowledge of the approximate time required
for the propagation of a message from each node to its own node.
It then takes a fault-tolerant average of the clock values of all the
computers (including its own). The calculated average is the current time to
which all the clocks should be readjusted.

Servers readjusts its own and sends the amount by which each individual
computer's clock requires adjustment (positive or negative value).

Major Drawbacks of Centralized clock synchronization algorithms -
1. Single-point failure (time server node fails)
2. No scalability

2) Distributed Algorithms -
A simple method for clock synchronization may be equip each node
of the system with a real-time clock so that each node's clock can be
independently synchronized with real time

Theoretically, internal synchronization of clocks is not required in this approach.

Two types of distributed algorithms are -
1) Global averaging Distributed Algorithms -

In this approach, the clock process at each node broadcasts its local clock
time in front of a special "async" message when its local time equals \( T_0 + k \)
To – fixed time on the part agreed upon by all nodes, \( k \) – some integer
\( K \) – system parameter depends on factors like No. of nodes, maximum allowable
shift etc.
Broadcasting nodes waits for time $T$ during which it collects "async" messages by other nodes & record time of receipt according to its own clock. At the end of waiting time, it estimates the skew of its clock wrt other nodes on the basis of times at which it received "async" messages.

Calculate fault tolerant average of estimated skew $\mu$ & use it to correct its own local clock before start of next "async" interval.

Two algorithms used -

(i) Take the average of the estimated skew and use it as the correction for the local clock.

(ii) Each node limits the impact of faulty clocks by first discarding the $m$ highest and $n$ lowest estimated skew and then calculating the average of the remaining skew and then calculating the average and use it as the correction for the local clock.

(iii)localized Arranging Distributed Algorithms -

It attempts to overcome the drawbacks of the global averaging algorithms. Nodes are arranged in ring or grid. Periodically, each node exchanges its clock time to the average with its neighbors then set its clock time to the average of its own clock time & clock times of its neighbors.

Case Study: Distributed Time Synchronization (DTS) -

DTS is a component of DCE (Distributed Computing Environment) that is used to synchronize clocks of a network of computers running DCE.

DTS uses the usual client-server structure: DTS clients, daemon process a called DTS daemon, request the current time from some number of servers, receive responses, and then adapt their clocks to receive the new knowledge.

Components of DCE DTS are -

(1) DTS clients

(2) Time servers - three types -

- Hardware time server maintains the time synchronization of a given LAN
- Global time server, and casual time servers are used to synchronize time among interconnected LANs.
(3) DTS API provides an interface where application programs can access time information by the DTS.

- Computation of new clock value in DTS from obtained time intervals

\[ \text{Time interval} \rightarrow \text{Time} \]

- DTS event 1
- DTS event 2
- DTS event 3
- DTS event 4

- Highest interaction falling within the remaining interval

Midpoint of this interval is the new clock value

(2) Mutual Exclusion -

Mutual exclusion are introduced to prevent processes from executing concurrently with their associated critical sections.

An algorithm for implementing mutual exclusion must satisfy the requirement that is mutual exclusion and No starvation.

Three approaches for implementing mutual exclusion in distributed systems are -

(1) Centralized Approach -

One of the process in the system is elected as coordinator and coordinates the entry to the critical section.

<table>
<thead>
<tr>
<th>Status of request queue</th>
<th>P&lt;sub&gt;1&lt;/sub&gt; Status after 3</th>
<th>P&lt;sub&gt;2&lt;/sub&gt; Status after 4</th>
<th>P&lt;sub&gt;3&lt;/sub&gt; Status after 5</th>
<th>P&lt;sub&gt;4&lt;/sub&gt; Status after 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Status</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Advantages - Simple to implement & requires only three manages the critical section

Disadvantages - Single point failure (due to centralized coordinator)
2) Distributed Approach -

All processes that want to enter the same critical section cooperate with each other before reaching a decision on which process will enter the critical section next.

When a process wants to enter a critical section, it sends a request message to all other processes. Message contains process identifier, name of the critical section, and unique timestamp.

On receiving a request message -

1. Receive process checks waiting for reply if the process itself executing in the critical section and queues the request message.
2. If the receiving process waiting for its turn to enter the critical section, it compares the timestamps of itself and the received request message. If the timestamp of received request message is lower, the receiving process sends the reply and if the timestamp of receive process is lower, it defers sending a reply and queues the request message.
3. If the receiving process neither in the critical section nor is waiting for turn, then it immediately reply back a reply message.

A process enters the critical section as soon as it has received reply messages from all processes. After it finishes executing in the critical section, it sends reply messages to all processes in its queue & deletes them from its queue.

Status when processes $P_1$ and $P_2$ send request messages to other processes while process $P_3$ is already in the critical section.

<table>
<thead>
<tr>
<th>Process</th>
<th>Status</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>Still in critical section</td>
<td>Sends request to $P_2$ and $P_4$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>In critical section</td>
<td>Sends request to $P_3$ and $P_4$</td>
</tr>
<tr>
<td>$P_3$</td>
<td>In critical section</td>
<td>Sends request to $P_2$ and $P_4$</td>
</tr>
<tr>
<td>$P_4$</td>
<td>In critical section</td>
<td>Sends request to $P_2$ and $P_3$</td>
</tr>
</tbody>
</table>

When processes $P_1$ and $P_2$ send request messages to other processes while process $P_4$ is already in the critical section.

<table>
<thead>
<tr>
<th>Process</th>
<th>Status</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>Still in critical section</td>
<td>Sends request to $P_2$ and $P_4$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>In critical section</td>
<td>Sends request to $P_3$ and $P_4$</td>
</tr>
<tr>
<td>$P_3$</td>
<td>In critical section</td>
<td>Sends request to $P_2$ and $P_4$</td>
</tr>
<tr>
<td>$P_4$</td>
<td>In critical section</td>
<td>Sends request to $P_2$ and $P_3$</td>
</tr>
</tbody>
</table>
2(n-1) manages per critical section entry [n processes, n-1 acquire manages, n-1 reply manages]

Drawbacks - n points of failure, each process must know the identity of all the processes, waiting time may be large

(3) Token Passing Approach -

A single token is circulated among the processes in the system (organized in a ring structure - clockwise or anticlockwise). A token is a special type of message that entitles its holder to enter a critical section.

When a process receives a token, if it wants to enter the critical section, it keeps the token, enters the critical section and exits from the critical section and then passes the token along the ring (Note: one critical section at a time) or if it does not want to enter the critical section, it just passes the token along the ring.

Drawbacks - Process failure (logical ring breaks), lost token

(3) Election Algorithms -

They are meant for electing a coordinator process from among the currently running processes. It is based on following assumptions -

1. Each process in the system has a unique process number
2. Whenever an election is held, highest priority number process is elected
3. On recovery, failed processes can rejoin the set of active processes.

Two election algorithms are given as:

(1) The Bully Algorithm -

A process starts an election if it detects the coordinator is failed then:

- sends an election message to all processes with higher ids & wait for announcement (except the failed coordinator/failed)
- If no answer arrives in time, then the process becomes the coordinator and sends coordinator message (with its id) to all processes with lower ids.
- else, waits for a coordinator message or starts an election if timeout.
Receiving an election message -

- Sends an answer back and starts the election if it hasn’t started one.
- Also send election message to all higher-id processes (excluding the ‘failed coordinator’ because the coordinator might be up by now).

Receiving a coordinator message - set elected to the new coordinator.

If failed coordinator recovers then it simply sends a coordinator message to all other processes andปลKick the current coordinator into submission.

(2) Ring Algorithm -

- A process starts the election. If it detects the coordinator is failed then starts the election message and marks itself as participant by placing itself in an election message and sends the message to its neighbor.
- The election message returns back to the process then.

- Select highest priority processes as coordinator and inform others about the new coordinator along the ring.
- When coordinator process recovers from failure then it immediately creates on inquiry message and sends along the ring until it reaches the current coordinator which starts sending informing other processes of the new coordinator.

Bully Algorithm -

- Worst case → O(n²) messages
- Best case → n-2 messages

Ring Algorithm -

- Worst case → 2(n-1) messages
- Best case → 2(n-1) messages

Ring algorithm is more efficient and easier to implement than bully algorithm.