UNIT-3

Date ___/___/___

Page ______

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 set messages to group of destinations (either ports or processes). ~~A to~~ Messages are sent to (Internet address, local port) pairs. A local port is a message destination within a computer, specified as an integer.

Ports have advantage over processes ~~bus~~ as they provide several alternative points of entry to a receiving process

(3) Reliability -

Reliable communication should have validity and integrity property.

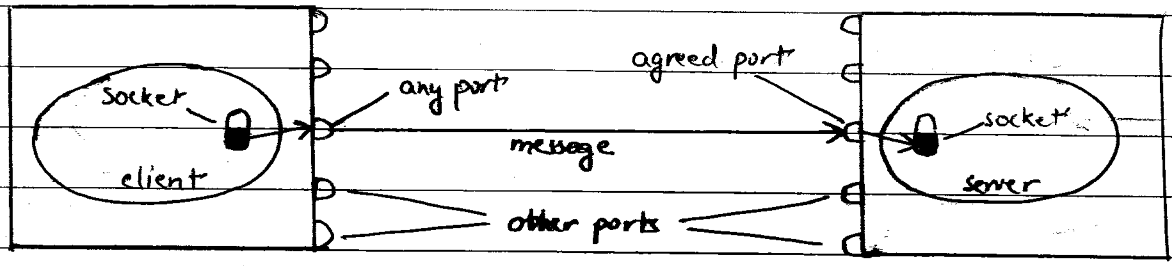
(4) Ordering -

Some application require that messages be delivered in sender order, that is, 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 Address -

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

InetAddress aComputer = 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 relating ~~do~~ 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 IP 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) Timeouts - It is not appropriate that a process that has used a receive operation should wait indefinitely in situations where the potential sending process has crashed or the expected message has been lost. That's why timeouts can be set on sockets.

(4) Receive from any - The receive method does not specify an origin from messages. Only internet address and local port of the sender is known.

Date ___/___/___
Page ______

→ Failure Model for UDP datagrams – Two failures –
(1) Omission 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) Ordering – Messages can sometimes be delivered out of sender order.
→ Use of UDP –
(1) Domain Naming service is implemented over UDP.
(2) Voice Over IP (VoIP).
(3) Do not suffer from overheads (need to store 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.

getData method gives the message, getPort and getAddress access the port and Internet ~~Access~~ Address.
DatagramSocket –
send and receive methods are for transmitting datagrams between a pair of sockets. These methods throws IOExceptions.
setSoTimeout method allows the timeout to be set and ~~throws InterruptedIOExcep~~ receive method will block for the time specified & then throw 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

Date ___/___/___
Page ______

decides how much data to collect before transmitting it as one or more IP packets.
(2) Lost messages - 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 enables the recipient to detect and reject duplicates, or to reorder messages that do not arrive in sender order.
(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 contents 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 property of reliable 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 property, TCP streams use timeouts and retransmission 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.

→ Java API for TCP streams -

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

ServerSocket - to create a socket at a server 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 -

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

Socket can throw an UnknowHostException 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 in a message.

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 structured 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 sender's ordering and the ordering specified.
- receiver translates if needed.
- 15 primitive types - short (16 bit), long (32-bit), unsigned short, unsigned long, float (32 bit), double (64 bit), char, boolean, octet (8 bit) and any

basic & constructed type

Date ___/___/___

Page ______

- Constructed types are -

(1) sequence - length (unsigned long) followed by elements in order

(2) string - length (unsigned long) followed by characters in order

(3) array - array elements in order (no length specified because it is fixed)

(4) enumerated - unsigned long (the values are specified by the order declared)

(5) union - type tag followed by the selected member

→ CORBA CDR message -

index in sequence of bytes →

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]

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 types of the data items in a message.

→ Marshalling in CORBA -

Marshalling operations can be generated automatically from the specification of the types of data items to be transmitted in message.

(2) 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.

```
public class Person implements Serializable {
    private String name;
    private string place;
    private int year;
    public Person (String aName, String aPlace, $int aYear) {
```

Date ___/___/___

Page ___

```
    name = aName;
    place = aPlace;
    year = aYear;
  }
  // 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 –

Serialized values				Explaination
Person	8-byte version number		h2	class name, version no.
3	int year	java.lang.String name	java.lang.String place	number, types and name of instance variables
1934	5 smith	6 London	h1	values of instance variables

The true serialized form contains additional type markers; h0 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 Reflection –

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.

(3) Extensible Markup language (XML) -

It is markup language defined by world wide web consortium (W3C) for general use on the web. In general, the term markup language refers to a textual encoding that represents both a text and details as to its structure or its namespace.

Tag relate to the structure of the text that is enclose. 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 -

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

XML definition of the Person structure.

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

Attributes consists 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 &.

Eg - <place> <![CDATA [King's Cross]]> </place>

(first line) XML Prolog - Specifies the XML version, encoding and standalone status.

Eg - <?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 — refer to the file containing the namespace definitions.

Eg - xmlns:pers = "http://www.cdk4.net/person"

prefix to refer to the elements

→ XML schemas -

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 types and default values. Eg - XML schema for the Person structure -

```
<xsd:schema xmlns:xsd = URL of XML schema definitions>
  <xsd:element name = "person" type = "personType" />
    <xsd:complexType name = "personType" >
      <xsd:sequence>
        <xsd:element name = "name" type = "xs:string" />
        <xsd:element name = "place" type = "xs:string" />
        <xsd:element name = "year" type = "xs:positiveInteger" />
      </xsd:sequence>
    <xsd:attribute name = "id" type = "xs:positiveInteger" />
    </xsd:complexType>
</xsd:schema>
```

Document type definitions (DTDs) defines 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.

Representation of remote object reference -

32 bits	32 bits	32 bits	32 bits	
INTERNET ADDRESS	PORT NUMBER	TIME	OBJECT NUMBER	INTERFACE OF REMOTE OBJECT

Date ___/___/___

Page ______

(3) 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 servers 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 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 - It may be permanent or temporary assigned by Internet authority from the range 224.0.0.1 to 224.0.0.255.

For temporary → { Uses 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 is ~~omission~~ omission failure and ordering problem

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

MulticastSocket methods are joinGroup, leaveGroup and setTimeToLive
↳ default is 1.

Date ___/___/___

Page ________

Reliability and ordering of multicast -

Effects of reliability and ordering of failure semantics of IP multicast are -

(1) Fault tolerance based on replicated services -

Ordering of the requests might be important, servers can be inconsistent with one another.

(2) Finding the discovery servers in spontaneous networking - Not too problematic

(3) Better performance through replicated data -

Loss and out-of-order updates could yield inconsistent data, sometimes this may be ~~in~~tolerable.

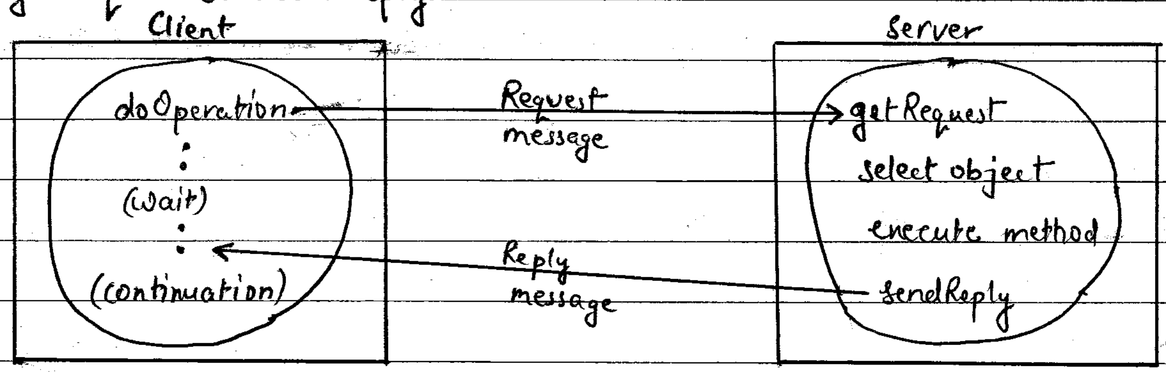
(4) Propagation of event notification - Not too problematic

(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, getRequest and sendReply.



REQUEST - REPLY COMMUNICATION

Operations of the request-reply protocol -

public byte[] doOperation (RemoteObjectRef o, int methodId, byte[] arguments);

public byte[] getRequest();

public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);

→ Request-reply message structure -

messageType	int (0=Request, 1=Reply)
requestId	int
objectReference	RemoteObjectRef
methodId	int or Method
arguments	// array of bytes

→ Message identifiers - Consist of 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 integer (Eg → $2^{32}-1$) 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 values in the sequence of integers.

→ Failure model of the request-reply protocol -

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

→ Timeout - Return immediately from doOperation with an indication to the client that the doOperation is failed. No getting a reply → timeout and retry

→ Discarding duplicate request messages - Protocol is designed to recognize successive messages with the same request identifier and to filter out duplicates.

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

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

→ RPC Exchange Protocols -

Three protocols, which produce differing behaviours in the presence of communication failures, are used ~~to impl~~ for implementing various types of RPC.

request (R) protocol, request-reply (RR) protocol & request-reply-acknowledgment reply ~~proto~~ protocol

NAME	Messages sent by CLIENT	SERVER	CLIENT
R	Request		
RR	Request	Reply	
RRA	Request	Reply	Acknowledgement Reply

→ 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 increases

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

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

The protocol allows for content negotiation and password-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 server 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 results. Multipurpose Internet Mail Extensions (MIME) is a standard for sending multipart data containing, for eg text, images & sounds in email messages.

→ HTTP Methods -

GET → requests the resource whose URL is given as argument.
HEAD → Return all the information about the data.
POST → specifies 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.

→ Message Contents -

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 concepts of stubs.

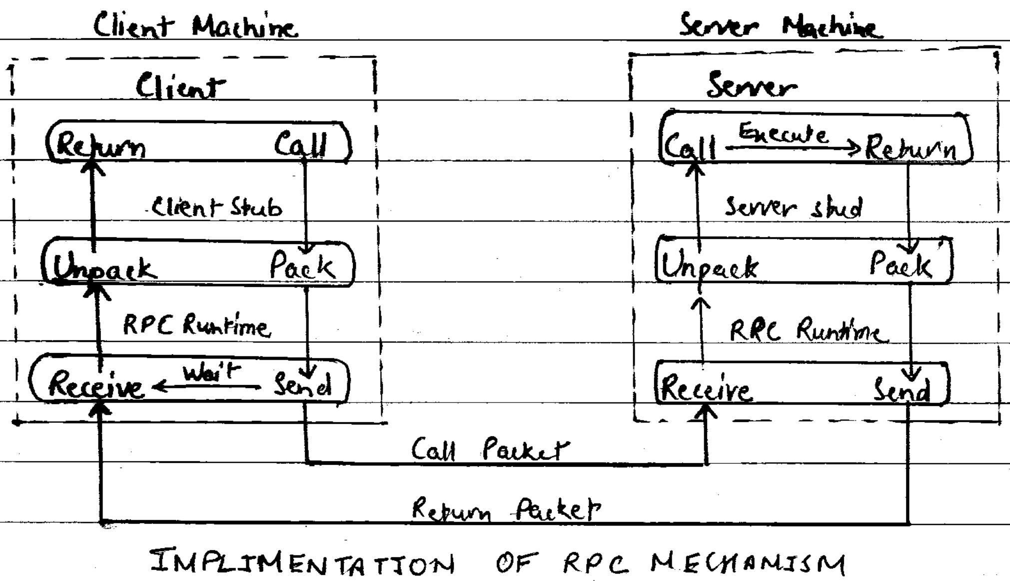
→ Stubs - It provides a normal / local procedure call abstraction by concealing the underlying RPC mechanism.

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

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

→ Thus implementation of RPC involves the five elements of program - Client, Client stub, RPC runtime, Server stub and Server.

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



IMPLIMENTATION OF RPC MECHANISM

Date ___/___/___

Page ________

→ Client - It call the local procedure, called client hub for remote services

→ Client Stub - It is responsible for two task -

(1) On receipt of a call request from the client

→ it packs a specifications of the target procedure & the arguments into a message.

→ asks 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, acknowledgement, routing and encryption.

→ Server Stub - It is responsible for two task -

(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 of procedure execution to the server hub.

(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 or her own ~~stubs~~ 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 play role in reducing data storage and controlling amount of data

**** RPC system is independent of transport protocols and is not concerned as to how a message is passed from one process to another.

transferred 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 combined with client and server programs, without making any changes to the existing compilers

(2) Client stub and server stub procedures

(3) The appropriate marshalling and unmarshaling operations.

(4) A header file that supports the data types.

③ 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.

Basic components →

RPC call message format

Message	Message	Client	Remote procedure identifier			Arguments
Identifier	Type	Identifier	Program Number	Version Number	Procedure No	

Message Identifier: identify lost and duplicate message
Message Type: 0 → call message, 1 → reply message
Client Identifier: for authentication 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 -

(i) The server finds that the call messages is not intelligible to it.

(ii) Client is not authorized to use the service.

(iii) Remote procedure identifier is missing.

Date __/__/__

Page ____

(iv) The remote procedure is not able to decode the supplied argument

(V) Occurence of exception condition.

RPC reply message format

Message Identifier	Message Type	Reply Status (Successful)	Result

Successful reply message format

Message Identifier	Message Type	Reply Status (Unsuccessful)	Reason for failure

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 ~~con~~ 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 1 s.

(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 drifts from the real-time clock

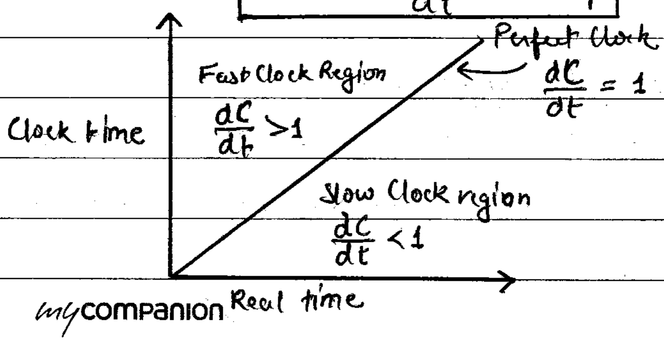
Clock based on a quartz crystal, drift rate is approximately 10^{-6}, giving a difference of 1 second every 1,000,000 seconds or 11.6 days.

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

$$\boxed{1-\rho \le \frac{dC}{dt} \le 1+\rho}$$

where $C \rightarrow$ time value of a clock

$\rho \rightarrow$ maximum drift rate.



time interval between two synchronization

$$\boxed{\Delta t \le \frac{\delta}{2\rho}}$$

where $\delta \rightarrow$ time diff between two clocks

myCOMPANION

** The difference in time values of two clocks is called clock skew.

Date ___/___/___

Page ______

→ Type of clock synchronization -

(1) Synchronization of the computer clock with real-time (or external) 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 time 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 $\boxed{T+(T_1-T_0)/2}$, where

T → current time, T_0 → when the client node sends the message.

T_1 → when it receives the "time=T" message.

Two cases -

[1] If ~~value~~ additional information is available then, $\boxed{T+(T_1-T_0-I)/2}$, where I → 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_1-T_0 are made. Minimum value of T_1-T_0 is considered to be the most accurate one then ~~$T+\min(T_1-T_0)$~~ $\boxed{T+[\min(T_1-T_0)]/2}$

(ii) 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 message for correcting their own clocks.

myCOMPANION

fault-tolerant average -

Time server chooses a subset of all clocks values having ~~values~~ do not ~~differed~~ differ from one another by more than a specified amount and then the average is taken.

Node's clock is readjusted to the time $\boxed{T+T_0}$ where $T \rightarrow$ current time

$T_0 \rightarrow$ Priori 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, Berkely algorithm can be used.

→ Berkely 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.

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

→ 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 receiver so that each node's clock can be independently synchronized with real time.

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

~~One~~ Two types of distributed algorithms are -

(i) Global Averaging Distributed Algorithms -

In this approach, the clock process at each node broadcasts its local clock time in front of a special "resync" message when its local time equals $\boxed{T_0 + iR}$

$T_0 \rightarrow$ fixed time in the past agreed upon by all node, $i \rightarrow$ some integer

$R \rightarrow$ system parameter depends on factors like No. of nodes, maximum allowable drift etc.

Broadcasting nodes waits for time T during which it collects "resync" 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 w.r.t other nodes on the basis of times at which it received "resync" messages.

Calculate fault tolerant average of estimated ~~guess~~ skews & use it to correct its own local clock before restart of next "resync" interval.

→ Two algorithms used -

(i) Take the average of the estimated skews 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 m lowest estimated skews and then calculating the average of the remaining skews and then ~~calculating the average~~ used as the correction for the local clock.

(ii) Localized Averaging Distributed Algorithms -

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

→ Case Study: Distributed Time Service (DTS) -

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

DTS uses the the usual client-server structure: DTS clients, daemon process called DTS clerks, request the correct time from some number of servers, receive responses, and then reset their clocks as necessary to reflect this new knowledge.

Components of DCE DTS are -

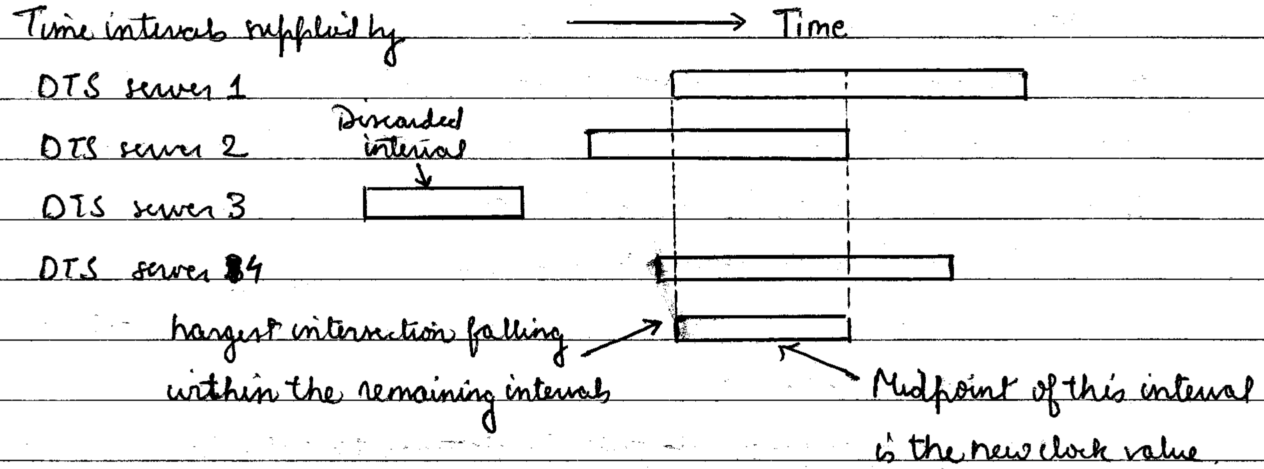
(1) DTS clerks

(2) Time server - three types -

~~(a)~~ Local time server maintains the time synchronization of a given LAN. The global time server and courier time server 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 -



② Mutual Exclusion -

Mutual Exclusion are introduced to prevent process from executing concurrently with their associated critical sections.

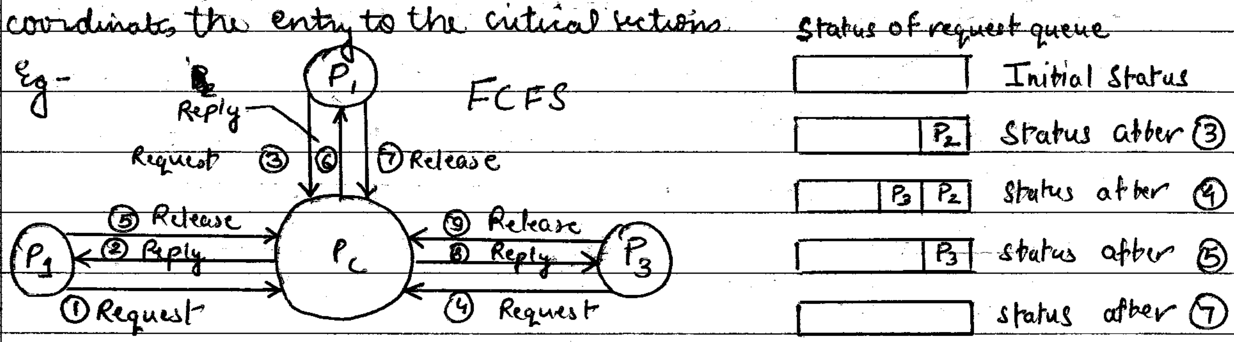
An algorithm for implementing mutual exclusion must satisfy the requirements that is mutual exclusion and No starvation.

Three approaches for implementing mutual exclusion in distributed systems are -

(1) Centralized Approach -

One of the processors in the system is elected as coordinator and coordinates the entry to the critical sections.

Eg -



Advantages - simple to implement & requires only three messages per critical section.

Drawbacks - 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 -

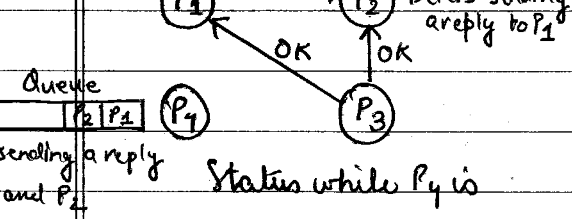
(1) Receiver process defers sending a reply if the receiver process itself executing in the critical section and queues the request message.

(2) If the receiver process waiting for its turn to enter the critical section, it compares the timestamp of itself and the received request message. If the timestamp of received request message is lower, the receiver process sends the reply and if the timestamp of receiver process is lower, it defers sending a reply and queues the request message.

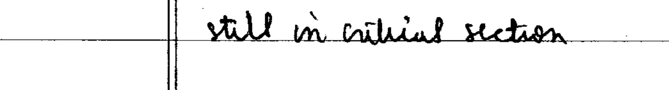
(3) If the receiver process neither in the critical section nor is waiting for turn, then it immediately reply back a reply message.

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.

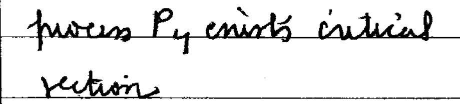
Eg -



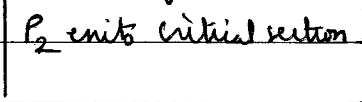
Status when processes P_1 and P_2 send request messages to other processes while process P_4 is already in the critical section.



Status while P_4 is still in critical section



Status after process P_4 exits critical section



Status after process P_2 exits critical section

2(n-1) messages per critical section entry [n processes, n-1 request messages, n-1 reply messages].

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

(3) <u>Token-Passing Approach</u> -

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

When a process receives a token, if it <u>wants to enter the critical section</u>, it keep the token, enter the critical section and exits from the critical section and then pass the token along the ring (Note → one critical section at a time) or if it <u>does not want to enter the critical section</u>, it just pass the token along the ring.

<u>Drawbacks</u> - Process failure (logical ~~link~~ ring breaks), lost token

③ <u>Election Algorithms</u> -

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

(1) Each process in the system has a unique priority number

(2) Whenever an election is held, highest priority number process is elected.

(3) On recovery, failed process can rejoin the set of active processes.

Towo election algorithms are given as -

(1) <u>The Bully Algorithm</u> -

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

→ <u>sends an election message</u> to all processes with higher id's & wait for answers (except the failed coordinator / process)

If <u>no answer in time T</u> then, it becomes coordinator and sends coordinator message (with its id) to all processes with lower id's.

<u>else</u> 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 messages to all higher-id processes (including 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 and bullies the current coordinator into submission.

(2) A Ring Algorithm - (Ring structure)

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

→ starts ~~the~~ an election message → mark itself as participant → places its id's in an election message and sends the message to its neighbour & neighbours do the same.

→ The election message returns back to the process then -

select highest priority number process as coordinator and inform others about the new coordinator along the ring.

→ When coordinator process recovers from failure then it ~~inquires~~ creates an inquiry message and sends along the ring until it reaches the current coordinator which starts ~~sending~~ informing other process of the new coordinator

	Bully Algorithm -	Ring Algorithm -
~~In case of~~ Assigning new coordinator	worst case → $O(n^2)$ messages	worst case → $2(n-1)$ messages
	best case → $n-2$ messages	best case → $2(n-1)$ messages
Recovery of failed coordinator process	worst case → $O(n^2)$ messages	worst case → $n/2$ messages
	best case → $n-1$ messages	best case → 1 message

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