Roll no: 31, Div A

Page No.

ALDEL EDUCATION TRUST'S

ST. JOHN COLLEGE OF ENGINEERING AND MANAGEMENT

Name: Harsh. Mendapara.

Sub: - DS

Date

Assignment no 1

Q1] What are various issues in Distributed System.

→ # Distributed System

i) A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system.

ii) It refers to two characteristics features of distributed systems. The first one is that a distributed systems is a collection of computing elements each being able to behave independently of each other.

iii) A computer element which we will generally refer node, can be either hardware device or software process.

iv) A second feature is that users believe they are dealing with a single system. This means that one way or another the autonomous nodes need to collaborate.

Issues in Distributed System

There are various issues in distributed system which are explained below.

1) Heterogeneity

i) The internet enables users to access services and run applications over a heterogeneous

collection of computers and networks.

ii) Internet consists of many different sorts of network their differences are masked by the fact that all of computers attached to them use the internet protocols to communicate with one another.

For eg:- A computer attached to an Ethernet has an implementation of internet protocols over the Ethernet, whereas a computer on a different sort of network will need an implementation of the internet protocols for that network.

2) Openness:-

i) The openness of a computer system is the characteristic that determines whether the system can be extended and reimplemented in various ways.

ii) The openness of distributed systems is determined primarily by the degree to which new resource-sharing services can be added and be made available for use by a variety of client programs.

3) Security:-

i) Many of the information resources that are

made available and maintained in distributed systems have a high intrinsic value to their users.

ii) Their security is therefore of considerable importance.

iii) Security for information resources has three components: confidentiality, integrity and availability.

4) Scalability:-

i) Distributed systems operate effectively and efficiently at many different scales, ranging from a small intranet to the Internet.

ii) A system is described as scalable if it will remain effective when there is a significant increase in the number of resources and the number of users.

5) Failure handling

i) Computer systems sometimes fail. When faults occur in hardware or software, programs may produce incorrect results or may stop before they have completed the intended computation.

ii) Failures in a distributed system are partial

-that is, some components fail while others continue to function.

iii) Therefore the handling of failures is particularly difficult.

6.) Concurrency

i) Both services and applications provide resources that can be shared by clients in a distributed systems.

ii) There is a possibility that several clients will attempt to access a shared resource at the same time.

iii) Object that represents a shared resource in a system must be responsible for ensuring that it operates correctly in concurrent environment. This applies not only to servers but also in applications.

iv) Therefore any programmer who takes an implementation of an object that was not intended for use in a distributed system must do whatever is necessary to make it safe in concurrent environment.

7) Transparency

→ Transparency can be achieved at two differ-

-ent levels. Easiest to do is to hide the distribut -ion from users.

a) location transparency: The users cannot tell where resources are located.

b) Migration transparency :- Resources can move at will without changi -ng their names.

c) Replication transparency :- The users cannot tell how many copies exist.

d) Parellelism transparency :- Activities can happ -en in parallel without users knowing.

8) Quality of service

i) Once users are provided with the functiona -lity that they require of a service, such as file service in distributed system, we can go on to ask about QoS provided.

ii) The main non-functional properties of systems that affect the quality of service experienced by clients and users are reliability, security and performence.

9) Reliability :-

i) A highly reliable system must be highly available, but that is not enough.

ii) Data entrusted to the system must not be lost in any way, and if files are stored redundantly on multiple servers, all the copies must be kept consistent.

10) Performance :-

i) Always the hidden data in the background is the issue of performance.

ii) Building a transparent, flexible, reliable distributed system, more important lies in its performance.

iii) In particular, when running a particular application on a distributed system, it should not be appreciably worse than running the same application on a single processor.

iv) Unfortunately, achieving that is easier said than done.

ALDEL EDUCATION TRUST'S
ST. JOHN COLLEGE OF ENGINEERING AND MANAGEMENT

Date

Q2] Describe any one method of logical clock synchronization.

→ i) Clock synchronization is naturally related to time, although it may not be necessary to have an accurate account of the real time.

ii) It may be sufficient that every node in a distributed systems agress on 'a' current time.

iii) For running make it is adequate that two nodes agree that input 'o' is outdated by new version of input c, for ex. In this case, keeping track of each other's events is matters. For these algorithms, the clocks are referred as logical clocks.

Lamport's logical clocks

i) To synchronize logical clocks, Lamport defined a relation called happens before.

ii) The expression a → b is read "event a happen-s before event b" and means that all process-es agree that first event 'a' occurs, then afterward, event 'b' occurs.

iii) The happen's before relation can be observed directly in two situations.

- If a and b are events in same process, and a occurs before b, the $a \rightarrow b$ is true.
- If a is event of message being sent by one process, and b is event of message being received by another process, then $a \rightarrow b$ is also true. A message cannot be received before it is sent, since it takes a finite, nonzero amount of time to arrive.

iv) The lamport algorithm proposed for ~~assigning~~ assigning times to events.

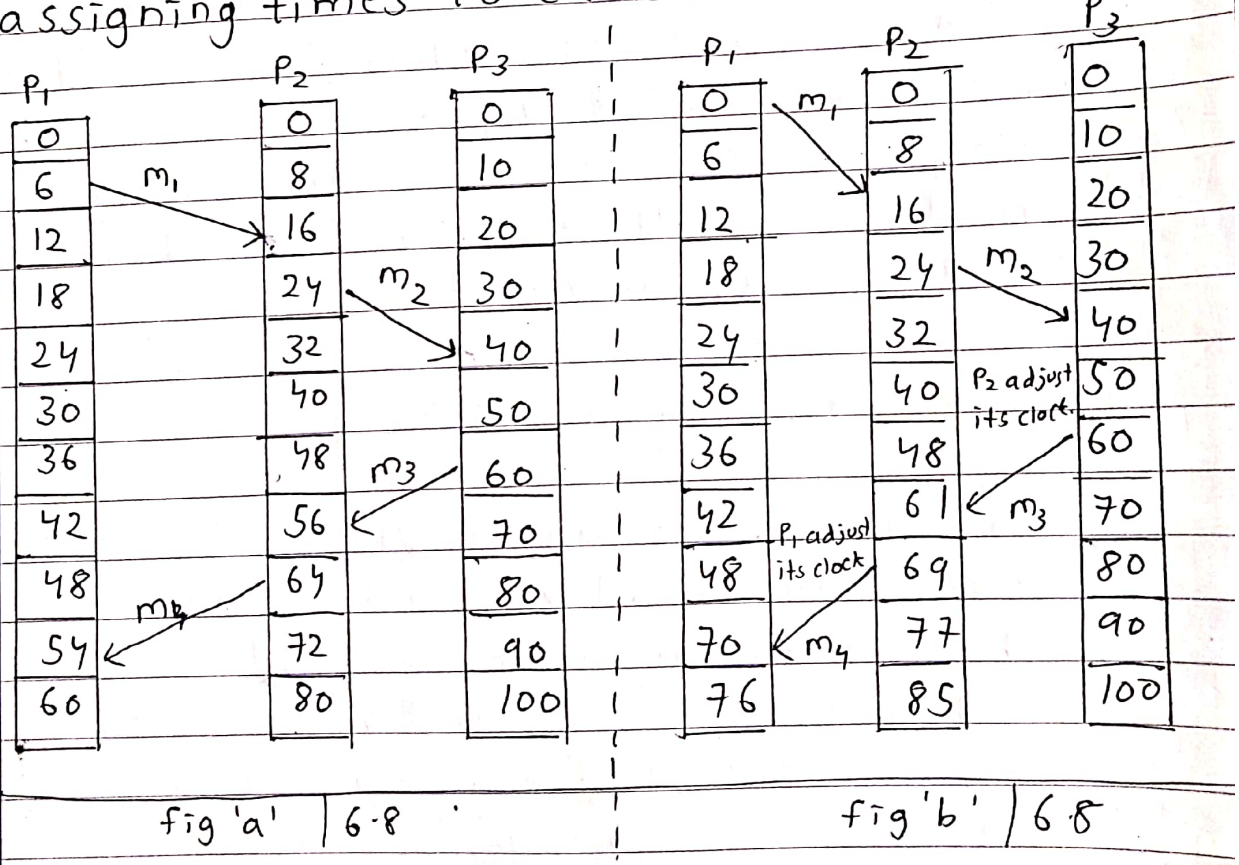


fig 'a' | 6.8 fig 'b' | 6.8

v) Consider the three process shown in fig 'a' and fig 'b'

vi) Fig 'a', three processes, each with its own (logica-l) clock. The clocks run at different rates. Fig 'b', Lamport's algorithm corrects their values

vii) The processes run on different machines, each with its own clock. For the sake of argument, we assume that, a clock is implemented by specific value every T units occurs.

viii) However, the value by which a clock is incremented by 6 units, 8 units in process P_2, and 10 units in process P_3, respectively.

ix) At time 6, process P_1 sends message m_1 to process P_2. How long this message takes to arrive depends on whose clock you believe.

x) In any event, the clock in process P_2 reads 16 when it arrives. If the message carries the starting time 6 in it, process P_2 will conclude that it took 10 ticks to make the journey.

xi) The value is certainly possible. According to this reasoning, message m_2 from P_2 to P_3 takes 16 ticks, again a plausible value.

xii) Now consider message m_3. It leaves process P_3 at 60 and arrives at P_2 at 56. Similarly, message m_4 from P_2 to P_1 leaves at 64 and arrives at 54. These values are impossible. It is situation that must be prevented.

xiii) Lamport's solution follows directly from the happens-before relation. Since m_3 left at 60, it must arrive at 61 or later. Therefore, each message carries the sending time according to sender's clock.

xiv) When a message arrives and the receiver clock shows a value prior to the time the message was sent, the receiver fast forwar-ds its clock to be one more than sending time.

xv) In fig 6.8 we see that m_3 now arrives at 61. Similary, m_4 arrives at 70.

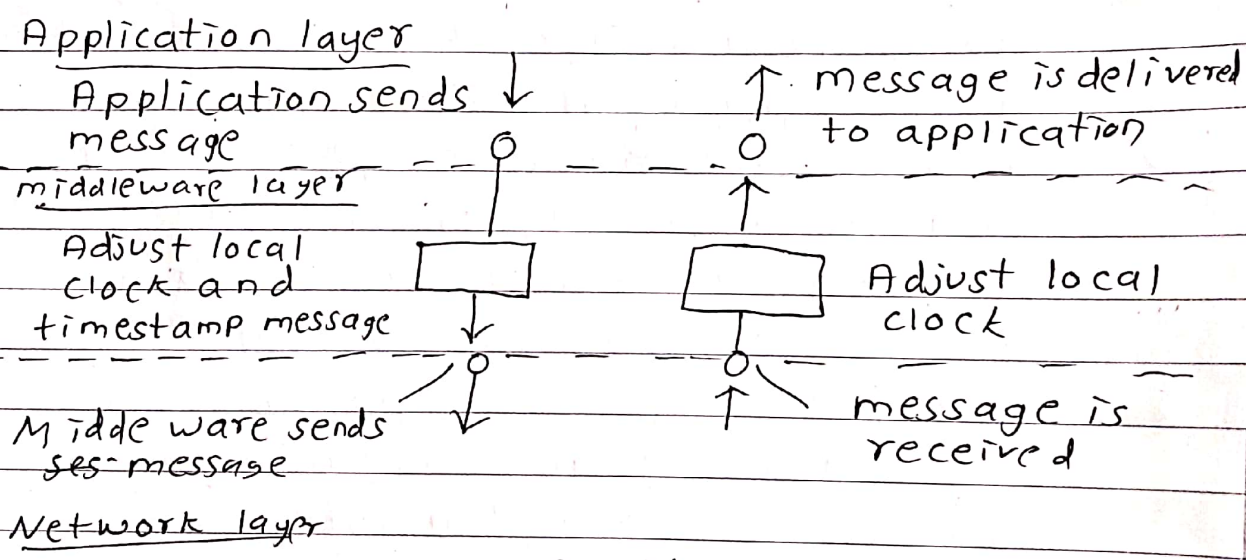


fig. 6.9 Positioning of Lamport's logical clocks in distributed systems.

To implement Lamport's logical clocks, each process P_i maintains a local counter C_i. These counters are updated to foll steps.

1) Before executing an event (i.e, sending a message over network, delivering a message to an application, or some other internal event), P_i increments C_i : $C_i \leftarrow C_i + 1$

2) When process P_i sends a message m to process P_j, it sets m's timestamp $ts(m)$ equal to C_i after having executed the previous step.

3) Upon the receipt of a message m, process P_j adjusts its own local counter as $C_j \leftarrow \max\{C_j, ts(m)\}$ after which it then executes the first step & delivers message to application

ex Total-ordered multicasting

As an application of lamport's logical clocks, consider for situation in which a database has been replicated across several sites.

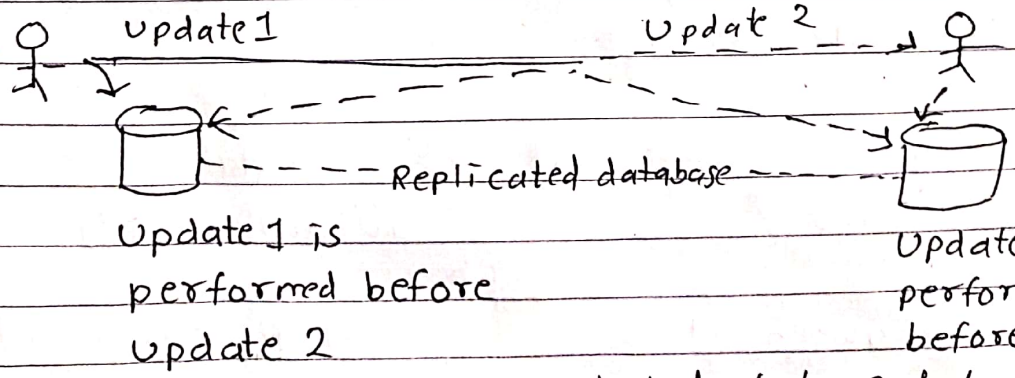


fig 6-10 Updating a replicated database & leaving it is in inconsistent state.

Q3] Difference between message oriented and stream oriented communication.

→

	Message oriented communication	Stream oriented communication
i>	UDP(user datagram protocol) uses manage oriented communication.	i> TCP (transmission contact protocol) uses stream oriented communication
ii>	Data is sent by application in discrete packages called message.	ii> Data is sent by with no particular structure.
iii>	Communication is connection less, data is sent without any set.	iii> Communication is oriented connection established before comm.
iv>	It is unreliable best effort delivery without acknowledgement.	iv> It is reliable, data acknowledged.
v>	Retransmission is not performed.	v> Lost data is reframe automatically.
vi>	Low overhead	vi> High overhead.

vii)	No flow control	vii) Flow control using sent protocol like sliding.
viii)	Transmission speed is very high as compared to stream -oriented.	viii) Transmission speed is lower as compared to message oriented.
ix)	Suitable for applicat -ions like audio, video where speed is criti -cal than loss of messages.	ix) Suitable for applicatio -ns like e-mail system where data must be persistent through delivered rate.

Q4) Explain Berkely physical clock algorithm.

→ i) In many clock synchronization algorithms the time server is passive. other machines periodically ask it for the time.

ii) In Berkely Unix exactly the opposite approach is taken. Here the time server (actually, a time deamon) is active, polling every machine from time to time to ask what time is there.

iii) Based on answers, it computes an average time and tells all the other machines to advance their clocks to new time or slow their clocks down until some specified

reduction is achieved.

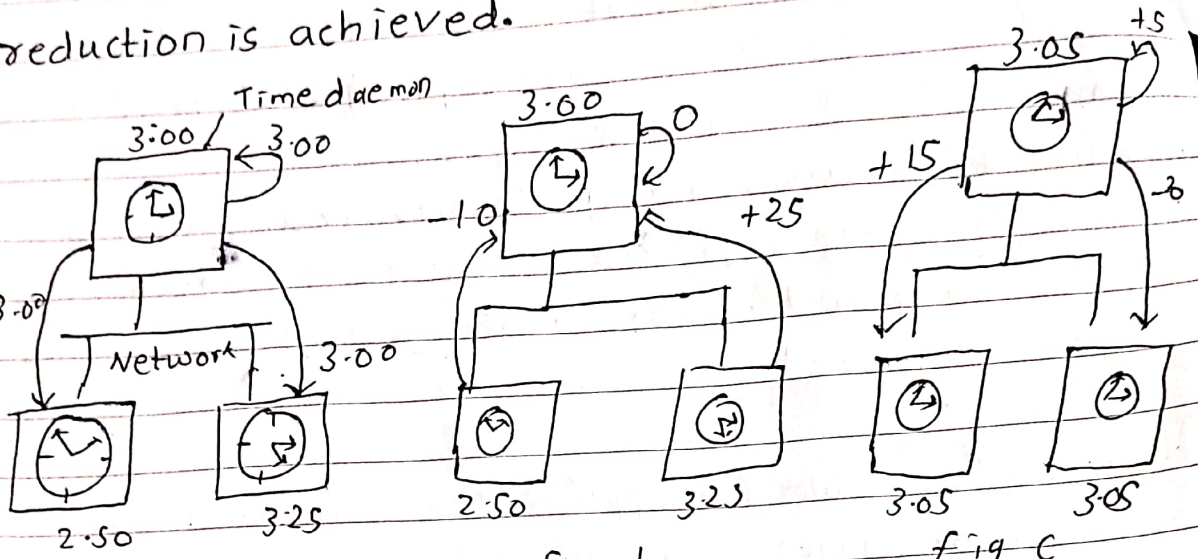


fig a, The time daemon asks all the other machines for their clock values.

fig, b, The machine answers.

fig c, The time daemon tells everyone how to adjust their clock.

iv) This method is suitable for a system in which no machine has UTC receiver. The time daemon's time must be set manually by operator periodically. The method is illustrated above.

v) ~~Fig a at 3:00~~, It is sufficient that all machines agree on the same time.

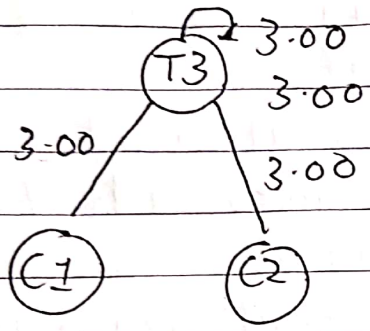
vi) It is not essential that this time also agrees with the real time as announced on the radio every hour.

vii) The Berkeley algorithm is thus typically an internal clock synchronization algorithm

ex:- IP address : 32 bits (4B)
Port address : 16 bits (2B)
ID + Port = socket (48 bits) → 60

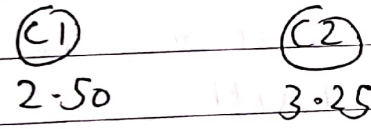
Berkeley

1. Time server broadcast its own time periodically

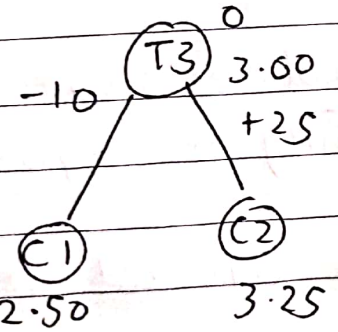


1. Every machine now calculates CV (clock value)

CV(T3) = 0
CV(C1) = 10
CV(C2) = 125



1. Every machine sends CV to time server



Q5] Explain Election Algorithm

→ i) Many distributed algorithms require one process to act as coordinator, initiator, or otherwise perform some specific role.

ii) In, general, it does not matter which process takes on this special responsibility, but one of them has to do it.

iii) Election algorithms attempt to locate the process with the highest identifier and desig-nate it as coordinator.

iv) The goal of an election algorithm is to ensure that when an election starts, it concludes with all processes agreeing on who the new coordinator is to be.

* The bully algorithm

1) A well-known solution for electing a coordina-tor is bully algorithm

2) In the foll, we consider N processes $\{P_0, \ldots P_{N-1}\}$ and let $id(P_k) = k$.

3) When any process notices that co-ordinator is no longer responding to requests, it initiates an election.

4) A process, P_k holds an ~~ect~~ election as follows

- P_K sends an ELECTION message to all processes with higher identifiers:
  $P_{K+1}, P_{K+2}, \ldots, P_{N-1}$
- If no one responds, P_K wins the election and becomes coordinator.
- If one of the higher-ups answers, it takes over and P_K's job is done.

5) At any moment, a process can get an ELECTION message from one of its lower-numbered colleagues.

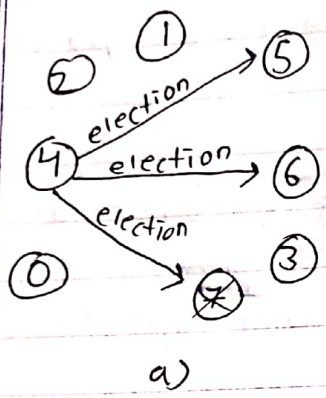
6) When such a message arrives, the receiver sends an OK message back to the sender to indicate that he is alive and will take over.

7) The receiver then holds an election, unless it is already holding one.

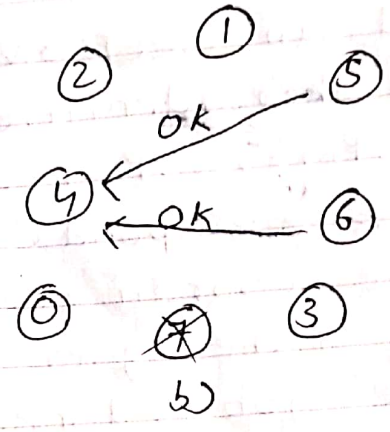
8) Eventually, all processes give up but one and that one is new coordinator.

9) It announces its victory by sending all process a message telling them that starting immedi-tely it is new coordinator.

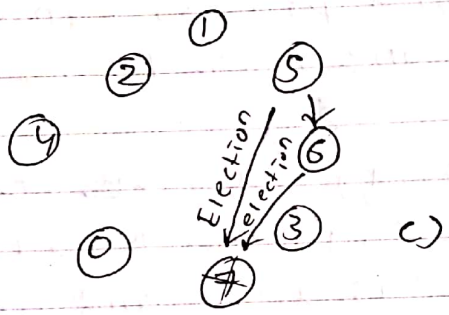
10) If a process that was previously down comes back up, it holds an election. If it happens to be highest-numbered process currently running, it will win the election & take co-ordinator job.



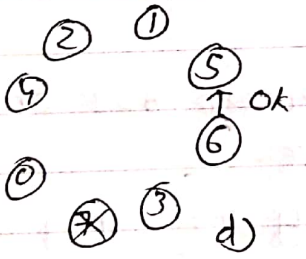
a)



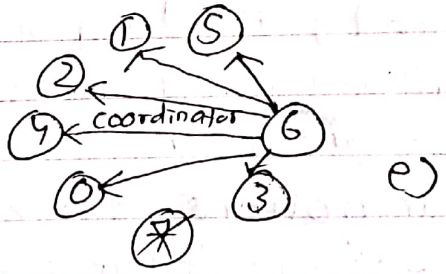
b)



c)



d)



e)

Bully election algorithm

a) Process 4 holds an election
b) Processes 5 and 6 respond, telling 4 to stop
c) Now 5 and 6 each hold an election
d) Process 6 tells 5 to stop.
e) Process 6 wins & tells everyone

Q6] Explain Lamports Distributed mutual algorithm

→ i) Distributed mutual exclusion algorithms can be classified into two different categories.

ii) In token-based solutions mutual exclusion is achieved by passing message between processes known as token.

iii) A permission based approach is second one. In this case, a process wanting to access the resource first requires the permission from other processes.

* Distributed algorithm

1) Using Lamport's logical clocks, and inspired by Lamport's original solution for distributed mutual exclusion. Their solution requires a total ordering of all events in the system.

2) That is, for any pair of events, such as messa-ges, it must be unambiguous which one actually happend first.

3) The algorithm works as follows. When a process wants to access a shared resource, it builts a message containing the name of the resource, its process number and current (logical) time

4) It then sends the message to all other processes, conceptually including itself. The sending of messages is assumed to be reliable that is, no

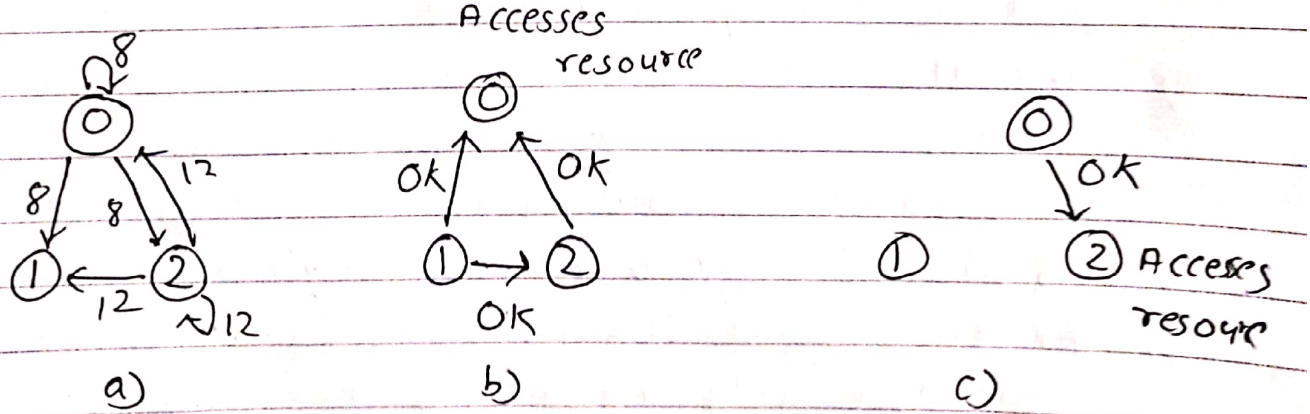
message is lost.

5) When a process receives a request message from another process, the action it takes depends on its own state with respect to the resource named in the message. There are different cases.

i) If the receiver is not accessing the resource and does not want to access it, it sends back an OK message to sender.

ii) If the receiver already has access to the resource, it simply does not reply. Instead, it queues the request.

iii) If the receiver wants to access the resource as well but has not yet done so, it compares the timestamp of incoming message with the one contained in the message that it has sent everyone. The lowest one wins. If incoming message has lower timestamp, the receiver sends back OK message.



- fig a, Two processes want to access a shared resource at the same moment.
- fig b, Po has the lowest timestamp, so it wins.
- fig c, When process Po is done, it sends an OK also, so P2 can now go ahead.

iv) With this algorithm, mutual exclusion is guaran-teed without deadlock or starvation.

v) If the total number of processes is N, then the number of messages that a process needs to send and receive before it can enter its critical section is 2.(N-1) : N-1 requests

vi) ~~(N-1)~~ : messages to all other processes, and subsequently N-1 OK messages, one from each other process.

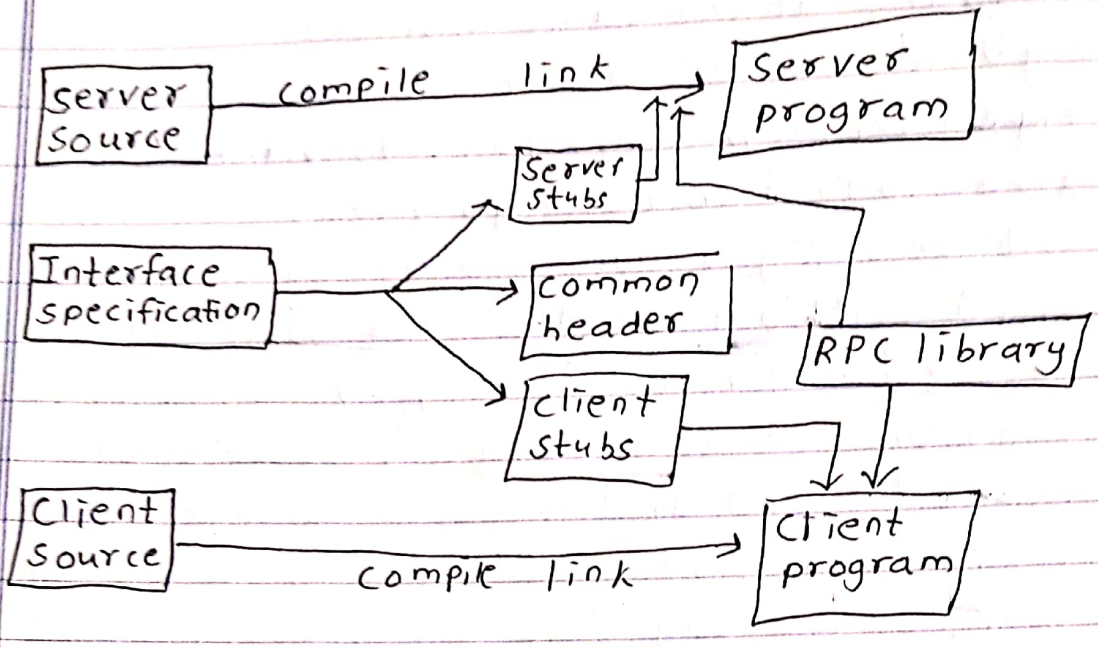
Q7] Explain Marshaling and Unmarshaling mechanis-m in RPC/RMI

→ Marshaling mechanism in RPC/RMI

i) Marshaling 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.

ii) It is similar to serialization & is used to communica

-te to remote objects within an object.



iii) This often involves translating non-portable representations into a portal.

iv) In case of Sun's implementation of RPC, a set of conventions known as external data.

v) In order to hide this process from both the application programmer and author of the RPC library, it is often implemented using automatically generated stubs.

vi) The process basically works like this. The programmer develops the interface for the RPC. The stub generator takes this interface defination and creates server stubs & client stubs, as well as common header file.

vii) The server stubs & client stubs take care of marshalling parameters as well as communication & procedure invocations.

viii) Once this is done, the programmer can build the RPCs and the application. Each is linked against the RPC library which provides the necessary code to implement RPC machinery.

* Unmarshaling mechanism in RPC/RMI

i) Unmarshaling refers to the process of transforming a representation of a object that was used for storage or transmission to a representation of object that is executable.

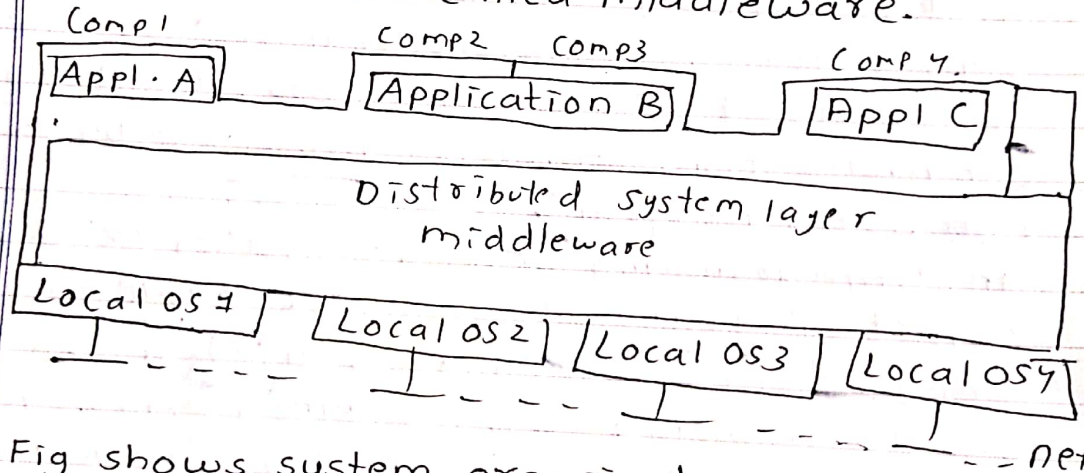
ii) A serialized object which was used for communication cannot be processed by a computer program.

iii) An unmarshaling interface takes the seralized object & transform in executable form.

iv) Unmarshaling is generally used in the receiver one to implementation of RMI (Remote method Invocation and Remote Procedure Call (RPC) mechanisms to unmarshal transmitted objects in executable form.

Q8] Short note: Middleware.

→ i) In order to support heterogenous computers and networks while offering a single system view, distributed systems often are organized by means of a layer of software that is logically placed between a high level layer consisting of users & applications and layer consisting of operating systems & basic communication facilities. It is called middleware.



ii) Fig shows system organized as middleware. The middleware layer extends over multiple machines and offers each application the same interface.

iii) These four networked computers & three applications, of which application B is distributed over computer 2 & 3.

iv) Each application is offered the same interface.

v) The distributed system provide the means for

components of a single distributed application to communicate with each other, but different applications also communicate.

vi) At the same time, it hides as best & reasonable as possible, the difference in hardware & os for each application.

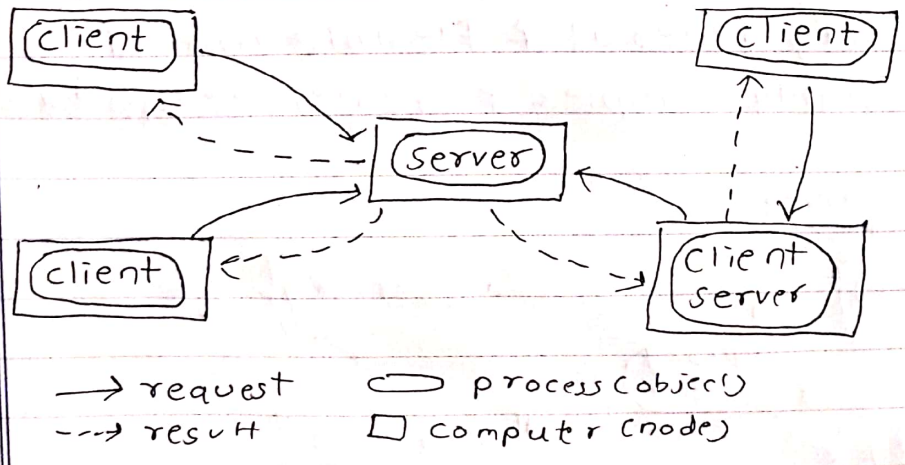
vii) The name 'middleware' stem from the fact that it sits between the client-side request on the front end & back end resource being requested.

viii) The role of middleware is to be enable & ease the access of those back-end resources by the front end.

Q9] Distributed system models.

→ There are three types of models which are given below.

a) Client-server model



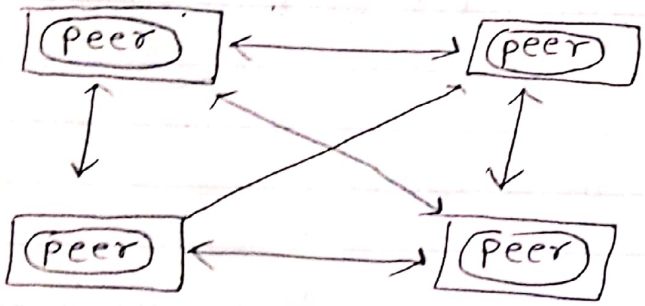
i) The client-server model is usually based on a simple request-reply protocol implemented with send/receive primitives or using RPC/RmI.

ii) The client sends a request message to server asking for some service.

iii) The server does the work and result is returned.

iv) A server can also request-service & act as a client.

b) Peer to Peer



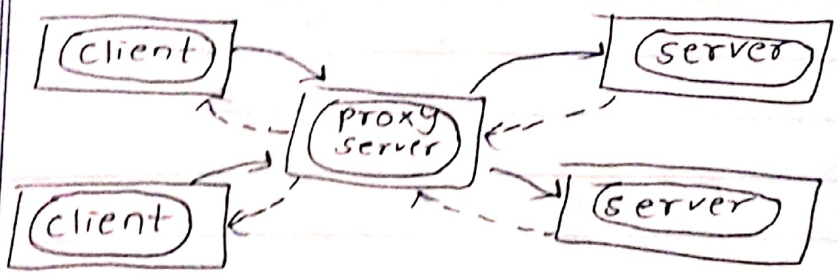
i) Processes (objects) interact without particular distribution between clients & servers.

ii) Large no of data objects are showed

iii) It is very general & flexible model

iv) It is highly complex & creates redundancy.

c) Proxy server



ALDEL EDUCATION TRUST'S
ST. JOHN COLLEGE OF ENGINEERING AND MANAGEMENT

Date

i) A proxy server provides copies of resources which are managed by other servers.

ii) They are usually used as caches for web resources.

iii) When a request is issued by client, the proxy server is first checked if the requested object is available there.

iv) Use of proxy server, increases performance.

Q10] Group communication

→ i) A group is an operating system abstraction for a collective of relative processes.

ii) The group abstraction allows member processes to perform computation on different hosts while providing support for communication & synchronization between them.

iii) Modes of communication are

- Unicast :- point to point
- anycast - nearest one of several identical node
- netcast - one-to-many at a time
- broadcast - 1 to all
- multicast - 1 to many.

iv) Groups are dynamic can be created or destroyed processes can join or leave; one process can belong to zero or many group one entity is delivered to entire group.

v) A reliable multicast protocol allows a group of processes to agree on a set of messages received by the group.

vi) Each message should be received by all numbers of the group or by none.

vii) A reliable multicast protocol is concerned only with message delivery and not by message ordering.

viii) The management of a group needs an effecti -ve & reliable multicast communication mechanism to allow clients obtain services from the groups & ensure consistency among the servers in process of failures.

ix) Failures may occur during a multicast at the recipient process, the communication links or the original process.

x) These are ordering semantics used by multicast ~~proto~~ protocols. FiFo ordering & Total ordering.

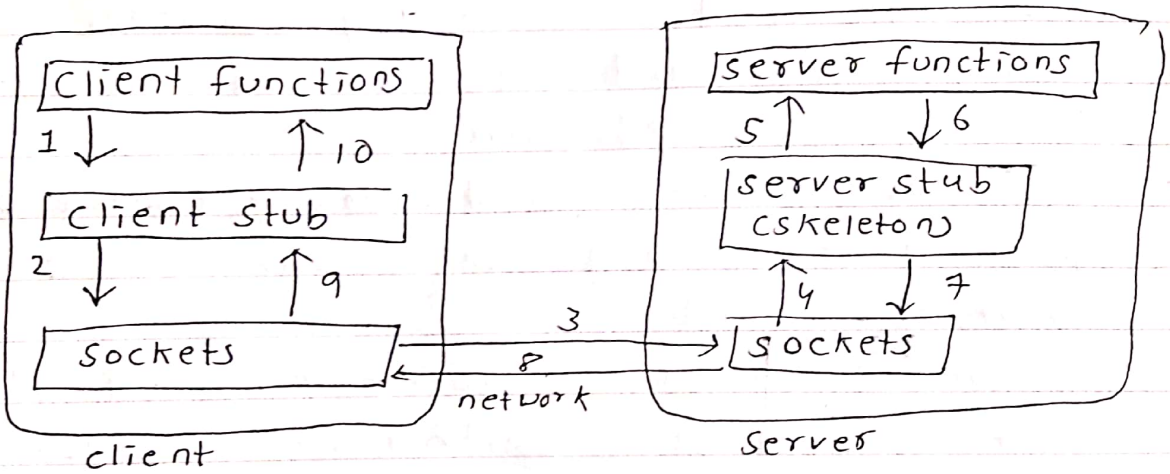
ALDEL EDUCATION TRUST'S
ST. JOHN COLLEGE OF ENGINEERING AND MANAGEMENT

Date

Q11] Write short note on RPC & RMI.

→ * RPC

i) In distributed computing, a remote procedure call is when a computer program causes a procedure (subroutine) to execute in a different address space which is coded as if it were a local procedure call, without the programmer explicitly coding the details for remote interaction.



- The client calls a local procedure, called client stub.
- To the client process, this appears to be the actual procedure, because it is a regular local procedure.
- It just does something different since the real procedure is on the server.
- The client stub packages the parameters to the remote procedure and builds one or

more network messages.

- The packaging of arguments into a network message is called marshaling and requires serializing all the data elements into a flat array-of-bytes format.
- Network messages are sent by the client stub to the remote system via system call.
- A server stub, sometimes called the skeleton, receives the messages on the server.
- It unmarshals the arguments from the message and if necessary, converts them from a standard network format into a machine-specified form.
- The server stub calls the server function, passing it the arguments that is received from the client.
- When the server function is finished, it returns to the server stub with return values.
- The server stub converts return values, if necessary and marshals them into one or more network messages to send to client stub.
- Messages get sent back across the network to the client stub.
- The client stub reads the messages from local kernal. The client code then continues its execution

* RMI (Remote Method Invocation)

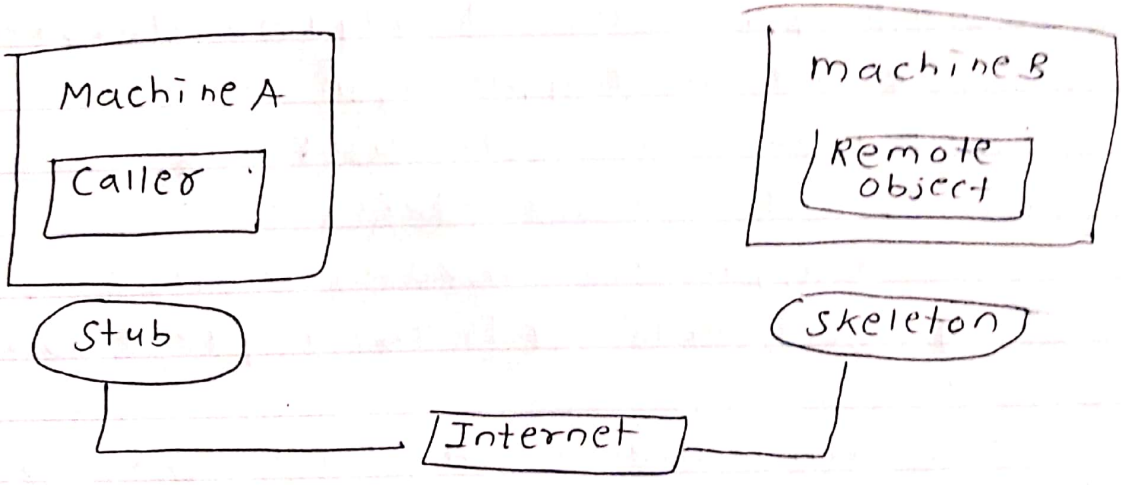
i> The RMI is an API that provides a mechanism to create distributed application in java.

ii> The RMI allows an object to invoke methods on an object running in another JVM.

iii> The RMI provides remote communication between the applications using two objects stub & skeleton.

iv> RMI uses stub & skeleton object for communication with remote object.

v> A remote object is an object whose method can be invoked from another JVM.



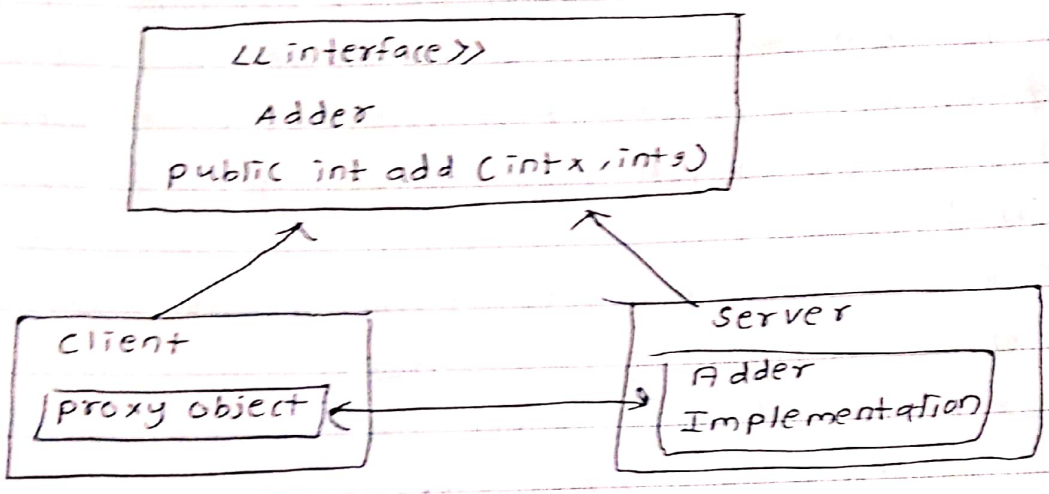
The code for RMI is written in six steps.

1) Create the remote interface
2) Provide the implementation of remote interface
3) Compile the implementation class & create the stub and skeleton objects using rmic tool.

4 Start the registry service by rmi registry tool

5 Create and start the remote application

6 Create and start the client application.



- The client application need only two files, remote interface and client application.
- In RMI application, both client and server interacts with remote interface.
- The client application invokes methods on the proxy object, RMI sends the request to remote JVM.
- The return value is sent back to the proxy object and then to the client application.