DISTRIBUTED COMPUTING

As per syllabus of

MCA SEMESTER 5

(Mumbai University)

Compiled by

MissionMca.com

(For private circulation only)

Q1.) Explain in detail the following.

i) Explain Election algorithms

Distributed algorithms require that there be coordinator process in the entire system that performs some type of coordination activity needed for the smooth running of other processes in the system. If the coordinator process fails due to the failure of the site on which it is located. Election algorithms are meant for electing a coordinator process from among the currently running processes in such a manner that at any instance of time there is a single coordinator for all processes in the system.

Election algorithms are based on the following assumptions:

- Each process in the system has a unique priority number.
- Whenever an election is held, the process having the highest priority number among the currently active processes can take appropriate actions to rejoin the set of active processes.

Two election algorithms are:

i. Bully Algorithm:
In this algorithm it is assumed that every process knows the priority number of every other process in the system. When a process (say P_i) sends a request message to the coordinator and does not receive a reply within a fixed timeout period, it assumes that the coordinator has failed. It then initiates an election by sending an election message to every process with a higher priority number than itself. If P_i does not receive any response to its election message within a fixed timeout period, it assumes that it has the highest priority number. Therefore it takes up the job of the coordinator and sends a message to all processes having lower priority numbers than itself, informing that from now on it is the new coordinator. On the other hand, if P_i receives a response for its election message, this means that some other process having higher priority number is alive. Therefore P_i does not take any further action and just waits to receive the final result of the election it initiated.

When a process (say P_j) receives an election message, it sends a response message to the sender informing that it is alive and will take over the election activity. Now P_j holds an election if it is not already holding one. In this way, the election activity gradually moves on to the process that has highest priority number among the currently active processes and eventually wins the election and becomes the new coordinator.

ii. Ring Algorithm:

In this algorithm it is assumed that all the processes in the system are organized in a logical ring. The ring is unidirectional i.e. messages are passed only in one direction. Every process in the system knows the structure of the ring, so that while trying to circulate a message over the ring, if the successor of the sender process is down, the sender can skip over the successor, or the one after that, until an active member is located.

When a process (say P_i) sends a request message to the current coordinator and does not receive a reply within a fixed period, it assumes that the coordinator has crashed. Therefore it initiates an election by sending an election message to its successor. This message contains the priority number of process P_i. on receiving the election message, the successor appends its own priority number to the message and passes it on to the next active member in the ring. This member appends its own priority number to the message and forwards it to its own successor. In this manner, the election message circulates over the ring from one active process to another and eventually returns back to process P_i. Process P_i recognizes the message as its own election message by seeing that in the list of priority numbers held within the message the first priority number is its own priority number.

When a process(say P_j) recovers after failure, it creates an inquiry message andsends it to its successor. The message contains the identify of process P_j. if the successor is not the current coordinator, it simply forwards the enquiry message to its own successor. In this way, the enquiry message moves forward along the ring until reaches the current coordinator. On receiving an inquiry message, the current coordinator sends a reply to process P_j informing that it is the current coordinator.

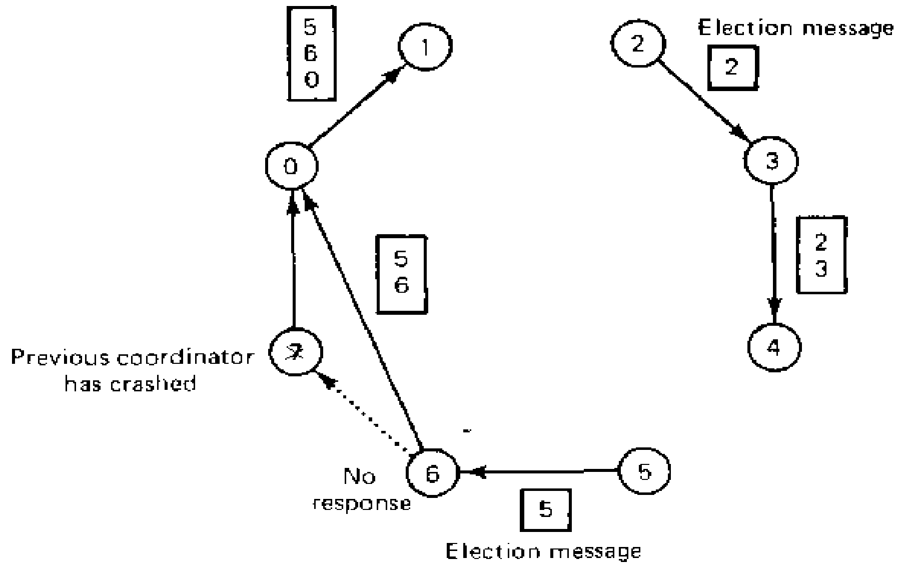


Fig. 3-13. Election algorithm using a ring.

ii) Explain Thrashing

Thrashing is said to occur when the system spends a large amount of time transferring shared data blocks from one node to another, compared to the time spent doing the useful work of executing application processes. It is serious performance problem with DSM systems that allow data blocks to migrate from one node to another. Thrashing may occur in the following situations:

a. When interleaved data accesses made by processes on two or more nodes causes a data block to move back and forth from one node to another in quick succession.
b. When blocks with read-only permissions are invalidated soon after they are replicated.

Following methods may be used to solve the thrashing problem in DSM systems:

a. Providing application-controlled locks. Locking data to prevent other nodes from accessing that data for a short period of time can reduce thrashing.
b. Nailing a block to a node for a minimum amount of time. Another method to reduce thrashing is to disallow a block to be taken away from a node until a minimum amount of time t elapses after its allocation to that node. The time can either be fixed statically or be tuned dynamically on the basis of access patterns.
c. Tailoring the coherence algorithm to the shared-data usage patterns. Thrashing can be minimized by using different coherence protocols for shared data having different characteristics.

iii) Explain Distributed operating system:

A distributed operating system is one that looks to its users like an ordinary centralized operating system but runs on multiple, independent central processing units(CPUs). The key concept here is transparency. In other words, the use of multiple processors should be invisible (transparent) to the user. Distributed operating system features are:

i. System image:

A distributed operating system hides the existence of multiple computers and provides a single-system image to its users. A distributed operating system dynamically and automatically allocates jobs to the various machines of the system processing. Users of a distributed operating system need not keep track of the locations of various resources for accessing them, and the same set of system calls is used for accessing both local and remote resources.

ii. Autonomy.

In distributed operating system, there is a single systemwide operating system and each computer of the distributed operating system runs a part of this global operating system. The distributed operating system tightly interweaves all the computers of the distributed computing system in the sense that they work in close cooperation with each other for the effective utilization of the various resources of the system.

iii. Fault tolerance capability:
In a distributed operating system, most of the users are normally unaffected by the failed machines and can continue to perform their work normally, with only a 10% loss in performance of the entire distributed computing system.

iv) Explain Mutual Exclusion

Mutual Exclusion is a way of making sure that if one process is using a shared modifiable data, the other processes will be excluded from doing the same thing.

Formally, while one process executes the shared variable, all other processes desiring to do so at the same time moment should be kept waiting; when that process has finished executing the shared variable, one of the processes waiting; while that process has finished executing the shared variable, one of the processes waiting to do so should be allowed to proceed. In this fashion, each process executing the shared data (variables) excludes all others from doing so simultaneously. This is called Mutual Exclusion.

Formally, while one process executes the shared variable, all other processes desiring to do so at the same time moment should be kept waiting; when that process has finished executing the shared variable, one of the processes waiting; while that process has finished executing the shared variable, one of the processes waiting to do so should be allowed to proceed. In this fashion, each process executing the shared data (variables) excludes all others from doing so simultaneously. This is called Mutual Exclusion.

Note that mutual exclusion needs to be enforced only when processes access shared modifiable data - when processes are performing operations that do not conflict with one another they should be allowed to proceed concurrently.

v) Explain Immutable Files

Most recent file systems, such as the Cedar File System (CFS), use the immutable file model. In this model, a file cannot be modified once it has been created except to be deleted. The file versioning approach is normally used to implement file updates and each file is represented by a history of immutable versions. Rather than updating the same file, a new version of the file is created each time a change is made to the file contents and the old version is retained unchanged. In practice, the use of storage space may be reduced by keeping only a record of the differences between the old and new versions rather than creating the entire file once again.

Sharing only immutable files make it easy to support consistent sharing. Due to this feature, it is much easier to support file caching and replication in a distributed system with the immutable file model because it eliminates all the problems associated with keeping multiple copies of a file consistent. However, due to the need to keep multiple versions of a file, the immutable file model suffers from two potential problems increased use of disk space and increased disk allocation activity. Some mechanism is normally used to prevent the disk space from filling instantaneously.

vii) Explain Happened before relationship: The happened-before relation (denoted by →) on a set of events satisfies the following conditions:

1. If a and b are events in same process and a occurs before b, then a → b.
2. If a is the event of sending a message by one process and b is the event of receipt of the same message by another process, then a → b. This condition holds by the law of causality because a receiver cannot receive a message until the sender sends it, and the time taken to propagate a message from its sender to its receiver is always positive.
3. If a → b, and b → c, then a → c. That is, happened-before is a transitive relation.

In a physically meaningful system, an event cannot happened before itself, that is, a → a is not true for any event a. This implies that happened-before is an irreflexive partial ordering on the set of all events in the system.

In terms of the happened-before relation, two events a and b are said to be concurrent if they are not related by the happened-before relation. That is, neither a → b nor b → a is true. This is possible if the two events occur in different processes that do not exchange message either directly or indirectly via other processes. This definition of concurrency simply means that nothing can be said about when the two events happened or which one happened first. That is, two events are concurrent if neither can causally affect the other. Due to this reason, the happened-before relation is also known as the relation of causal ordering.

A space-time diagram is used to illustrate the concept of the happened-before relation and concurrent events. In this diagram, each vertical line denotes a process, each dot on a vertical line denotes an event in the corresponding process, and each wavy line denotes a message transfer from one process to another in the direction of the arrow.

From this space-time diagram it is easy to see that for two events a and b, a → a is true if and only if there exists a path from a to b by moving forward in time along process and message line in the direction of the arrow. For example, some of the events of fig. that are related by the happened-before relation are:

$e_{10} \rightarrow e_{11}$ $e_{20} \rightarrow e_{24}$ $e_{11} \rightarrow e_{23}$ $e_{21} \rightarrow e_{13}$

$e_{30} \rightarrow e_{24}$ (since $e_{30} \rightarrow e_{22}$ and $e_{22} \rightarrow e_{24}$)

$e_{11} \rightarrow e_{32}$ (since $e_{11} \rightarrow e_{23}$, $e_{23} \rightarrow e_{24}$, and $e_{24} \rightarrow e_{32}$)

On the other hand, two events a and b are concurrent if and only if no path exists either from a to b or from b to a. For example, some of the concurrent events of fig. are:

$e_{12} \rightarrow e_{20}$ $e_{21} \rightarrow e_{30}$ $e_{10} \rightarrow e_{30}$ $e_{11} \rightarrow e_{31}$ $e_{12} \rightarrow e_{32}$ $e_{13} \rightarrow e_{22}$

ix) Explain Marshaling

Implementation of RPC involves the transfer of arguments from the client process to the server process and the transfer of result from the server process to the client process. These arguments and results are basically language-level data structures, which are transferred in the form of message data between the two computers involved in the call. The transfer of message data between two computers requires encoding and decoding of message data. For RPCs this operation is known as marshaling and basically involves the following actions:

1. Taking the arguments (of a server process) or the result (of a server process) that will form the message data to be sent to the remote process.
2. Encoding the message data on the sender's computer. This encoding process involves the conversion of program objects into the stream form that is suitable for transmission and placing them into the message buffer.
3. Decoding of the message data on the receiver's computer. This decoding process involves the reconstruction of program objects from the message data that was received in stream form.

In order that encoding and decoding of an RPC message can be performed successfully, the order and the representation method (tagged or untagged) used to marshal arguments and results must be known to both the client and server of the RPC. This provides a degree of type safety between a client and a server because the server will not accept a call from the client until the client uses the same interface definition as the server. Type safety is of particular importance to server since it allows them to survive against corrupt call requests.

The marshaling process must reflect the structure of all types of program objects used in the concerned language such as primitive types, structured types, and user-defined types. Marshaling procedures may be classified into two groups:

1. Those provided as a part of RPC software. Normally marshaling procedures for scalar data types, together with procedures to marshal compound types built from the scalar ones, fall in this group.
2. Those that are defined by the users of the RPC system. This group contains marshaling procedures for user-defined types and data types that include pointers.

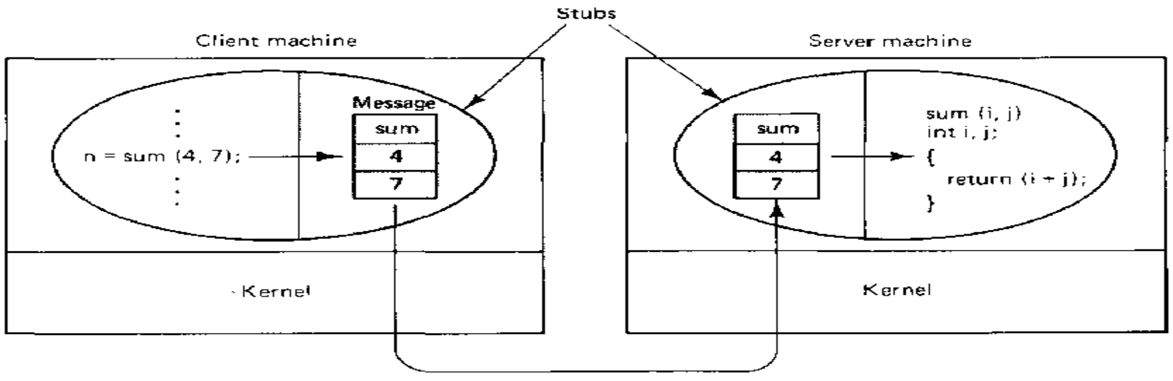


Fig. 2-19. Computing *sum*(4, 7) remotely.

x) Explain DSM

The term Distributed Shared Memory refers to the shared memory paradigm applied to loosely coupled distributed-memory system.

DSM provided a virtual address space shared among processes on loosely coupled processors. That is DSM is basically an abstraction that integrated the local memory of different machined in a network environment into a single logical entity shared by cooperating processes executing on multiple sites. The shared memory itself exists only virtually. Application program can use it in the same way as a traditional virtual memory, except of course that processes using it can run on different machined in parallel. Due to the virtual existence of the shared memory, DSM is sometimes also referred to as Distributed Shared Virtual Memory (DSVM)

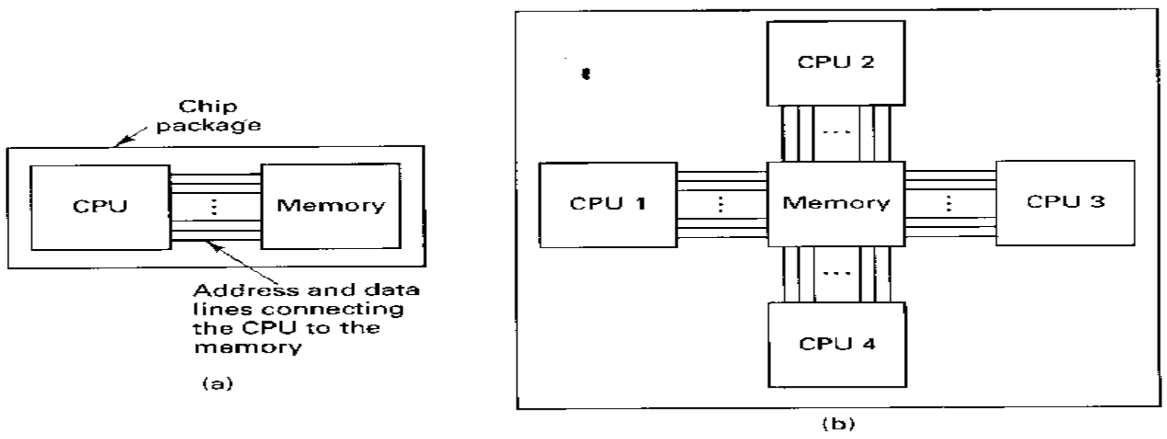


Fig. 6-1. (a) A single-chip computer. (b) A hypothetical shared-memory multiprocessor.

DSM systems normally have an architecture of the form as shown. Each node of the system consists of one or more CPUs and a memory unit. The nodes are connected by a high speed communication network. DSM abstraction presents a large shared memory space to the processors of all nodes. Shared memory exists only virtually. A software memory mapping manager routine in each node maps the local memory onto the shared virtual memory. To facilate mapping operation, the shared memory space is partitioned into blocks. The idea of data caching is also used in DSM system to reduce network latency.

When a process on a node access some data from a memory block of the shared memory space, the local memory mapping manger takes in charge of its request. If the memory block containing the accessed data is resident in the local memory, the request is satisfied by supplying the accessed data from the local memory. Otherwise, a network block fault is generated and the control is passed to the operating system. The operating system then sends a message to the node on which the desired memory block is located to get the block. The missing block is migrated from the remote node to the client process node and the operating system maps it into the application's address space. The faulting instruction is then restarted and can now complete.

Design and implementation issues of DSM:

- Granularity: refers to the block size of DSM that is to the unit of sharing and unit of data transfer in the network.
- Structure of shared memory space: refers to the layout of the shared data in memory, dependent upon type of application DSM is intended to support
- Memory coherence and access synchronization
- Data location and access
- Replacement strategy
- Thrashing
- Heterogeneity

Advantages of DSM

- Simple Abstraction.
- Better portability of Distributed Application Programs.
- Better Performance of Some application.
- Flexible communication Environment.
- Ease of Process Migration

xi) Explain Logical clock

In a distributed system, it is not possible in practice to synchronize time across entities (typically thought of as processes) within the system; hence, the entities can use the concept of a logical clock based on the events through which they communicate.

To determine that an event a happened before an event b, either a common clock or a set of perfectly synchronized clock is needed. Neither of these is available in distributed system. Therefore in a distributed system the happened before relation must be defined without the use of globally synchronized physical clocks.

Lamport provided a solution for this problem by introducing the concept of Logical clock

The logical clocks concepts is a way to associate a timestamp(which may be simply a number of independent of any clock time) with each system event so that events that are related to each other by the happened-before relation(directly or indirectly) can be properly ordered in that sequence. Under this concept, each process Pi has a clock Ci associated with it that assigns a number Ci(a) to physical time. In fact, the logical clock may be implemented by counters with no actual timing mechanism. With each process having its own clock, the entire system of clocks is represented by the function C , which assigns to any event b the number C(b)=Cj(b) if b is an event in process Pj.

The logical clock of a system can be considered to be correct if the events of the system that are related to each other by the happened-before relation can be properly ordered using these clocks. Therefore, timestamps assigned to the events by the system of logical clocks must satisfy the following clock condition;

For any two events a and b, if $a \rightarrow b$ then $C(a) < C(b)$.The following condition must hold true

C1: if a and b are two events within the same process Pi and a occurs before b then $Ci(a) < Ci(b)$

C2: if a is the sending of a message by process Pi and b is receipt of that message by process Pj, then $Cj(a) < Cj(b)$

C3: a clock Ci associated with a process Pi must always go forward, never backward. That is, corrections to time of a logical clock must always be made by adding a positive value to the clock, never by subtracting value.

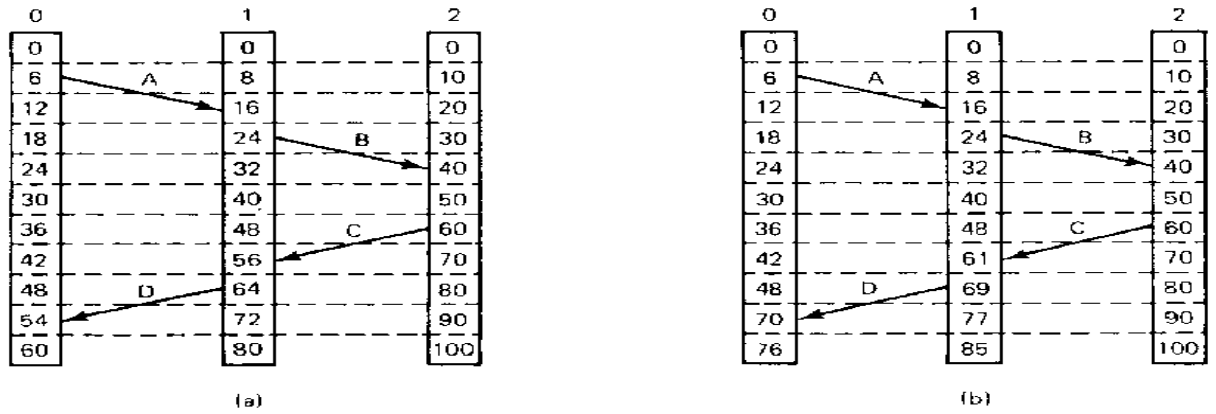


Fig. 3-2. (a) Three processes, each with its own clock. The clocks run at different rates. (b) Lamport's algorithm corrects the clocks.

xii) Explain Orphan call

Last- one call semantics uses the idea of retransmitting the call message based on timeouts until a response is received by the caller. That is , the calling of the remote procedure by the caller, the execution of the procedure by the callee, and the return of the result to the caller will eventually be repeated until the results of the procedure execution is received by the caller.

Last one semantics can be easily achieved in the way described above when only two processors are involved in RPC. However, achieving last one semantics in the presence of crashes turns out to be tricky for nested RPCs that involve more than two modes. For example, suppose process P1 no node N1 calls procedure F1 on node N2 which in turn calls procedure F2 on node N3, while the process on N3 is working on F2, node N1 crashes. Node N1's process will be restarted, and P1's call to F1 will be repeated. The second invocation of F1 will again call procedure F2 on node N3. Unfortunately, node N3 is totally unaware of node N1's crash. Therefore procedure F2 will be executed twice on node N3 and N3 may returns the results of the two execution of F2 in any order, violation last one semantics.

The basic difficulty in achieving last one semantics in such cases is caused by Orphan calls. An orphan call is one whose parent (caller) has expired due to node crash. To achieve last one semantics, these orphan calls must be terminated before restarting the crashed processed. This normally done either by waiting for them to finish or by tracking them down and killing them ("orphan extermination")

xiii) Explain Asynchronous Transfer Mode Technology

Asynchronous Transfer Mode (ATM) is often described as future computing networking paradigm. It is high speed, connection-oriented switching and multiplexing technology that uses short, fixed-length packet to cells to transmit different type of traffic simultaneously, including voice video and data . It is also asynchronous in that information streams can be sent independently without a common clock.

The ATM model is that a sender first establishes a connection (i.e., a virtual circuit) to the receiver or receivers. During connection establishment, a route is determined from the sender to the receiver(s) and routing information is stored in the switches along the way. Using this connection, packets can be sent, but they are chopped up by the hardware into small, fixed-sized units called cells. The cells for a given virtual circuit all follow the path stored in the switches. When the connection is no longer needed, it is released and the routing information purged from the switches.

ATM has its own protocol hierarchy, as shown in Fig. below. The physical layer has the same functionality as layer I in the OSI model. The ATM layer deals with cells and cell transport, including routing, so it covers OSI layer 2 and part of layer 3. However, unlike OSI layer 2, the ATM layer does not recover lost or damaged cells. The adaptation layer handles breaking packets into cells and reassembling them at the other end, which does not appear explicitly in the OSI model until layer 4. The service offered by the adaptation layer is not a perfectly reliable end-to-end service, so transport connections must be implemented in the upper layers, for example, by using ATM cells to carry TCP/IP traffic.

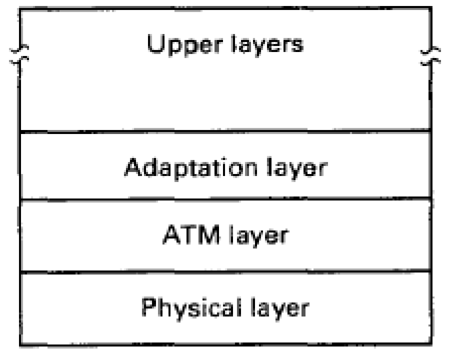


Fig. 2-4. The ATM reference model.

1. <u>The ATM Physical Layer:</u>
   An ATM adaptor board plugged into a computer can put out a stream of cells onto a wire or fiber. The transmission stream must be continuous. When there are no data to be

sent, empty cells are transmitted, which means that in the physical layer, ATM is really synchronous, not asynchronous. Within a virtual circuit, however, it is asynchronous. Alternatively, the adaptor board can use SONET (Synchronous Optical Network) in the physical layer, putting its cells into the payload portion ofSONET frames. The virtue of this approach is compatibility with the internal transmission system of AT&T and other carriers that use SONET.

2. The ATM Layer:

   When ATM was being developed, two factions developed within the standards committee. The Europeans wanted 32-byte cells because these had a small enough delay that echo suppressors would not be needed in most European countries. The Americans, who already had echo suppressors, wanted 64-byte cells due to their greater efficiency for data traffic. The end result was a 48-byte cell, which no one really liked. It is too big for voice and too small for data. To make it even worse, a 5-byte header was added, giving a 53-byte cell containing a 48-byte data field. Note that a 53-byte cell is not a good match for a 774-byte SONET payload, so ATM cells will span SONET frames. Two separate levels of synchronization are thus needed: one to detect the start of a SONET frame, and one to detect the start of the first full ATM cell within the SONET payload.
3. The ATM Adaptation Layer:

   At 155 Mbps, a cell can arrive every 3 use. Few, if any, current CPUs can handle in excess of 300,000 interrupts/sec. Thus a mechanism is needed to allow a computer to send a packet and to have the ATM hardware break it into cells, transmit the cells, and then have them reassembled at the other end, generating one interrupt per packet, not per cell. This disassembly/reassembly is the job of the adaptation layer. It is expected that most host adaptor boards will run the adaptation layer on the board and give one interrupt per incoming packet, not one per incoming cell.

Unfortunately, here too, the standards writers did not get it quite right. Originally adaptation layers were defined for four classes of traffic:

1. Constant bit rate traffic (for audio and video).

2. Variable bit rate traffic but with bounded delay.

3. Connection-oriented data traffic.

4. Connectionless data traffic.

xv) Explain Replication transparency

For better performance and reliability, almost all distributed operating systems have the provision to create replicas of files and other resources on different nodes of the distributed system. In these, both the existence of multiple copies of replicated resource and the replication activity should be transparent to the users. That is, two important issues related to replication transparency are naming of replicas and replication control. It is the responsibility of the system to name the various copies of a resource and to map a user-supplied name of the resource to an appropriate replica of the resource. Furthermore, replication control; decisions such as how many copies of the resource should be created., where should each copy be placed , and when should a copy be created/deleted should be made entirely automatically by the system in a user transparency manner.

Distributed file systems often provide file replication as a service to their clients. In other words, multiple copies of selected files are maintained, with each copy on a separate file server. The reasons for offering such a service vary, but among the major reasons are:

1. To increase reliability by having independent backups of each file. If one server goes down, or is even lost permanently, no data are lost. For many applications, this property is extremely desirable.

2. To allow file access to occur even if one file server is down. The motto here is: The show must go on. A server crash should not bring the entire system down until the server can be rebooted.

3. To split the workload over multiple servers. As the system grows in size, having all the files on one server can become a performance bottleneck. By having files replicated on two or more servers, the least heavily loaded one can be used.

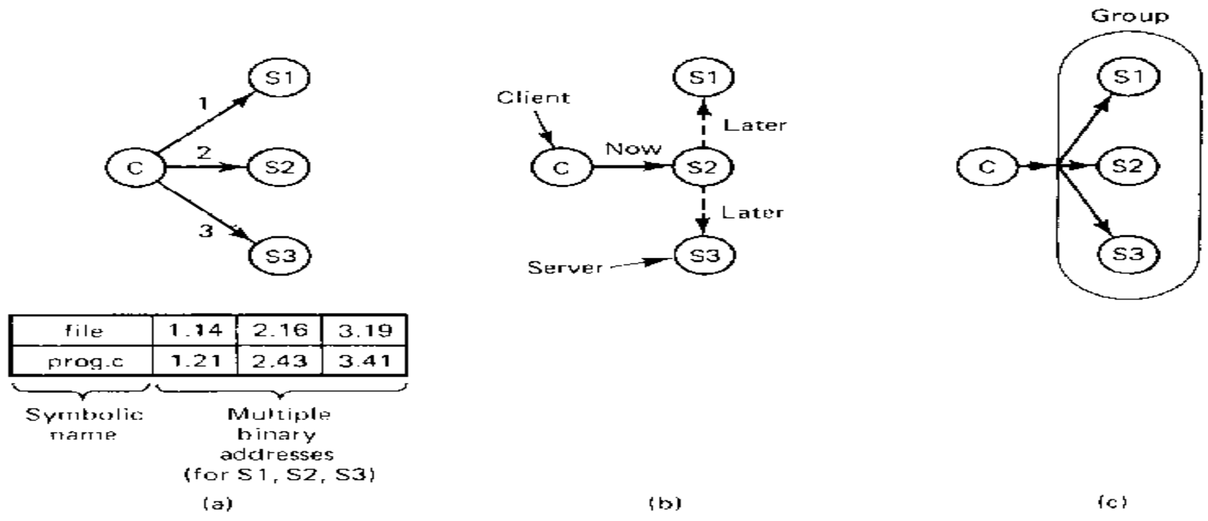


Fig. 5-12. (a) Explicit file replication. (b) Lazy file replication. (c) File replication using a group.

Q 2) What is a stub?

A stub in is a piece of code used for converting parameters passed during a Remote Procedure Call (RPC).

The main idea of an RPC is to allow a local computer (client) to remotely call procedures on a remote computer (server). The client and server use different address spaces, so conversion of parameters used in a function call have to be performed, otherwise the values of those parameters could not be used, because of pointers to the computer's memory pointing to different data on each machine. The client and server may also use different data representations even for simple parameters (e.g., big-endian versus little-endian for integers.) Stubs are used to perform the conversion of the parameters, so a Remote Function Call looks like a local function call for the remote computer.

Stub libraries must be installed on client and server side. A client stub is responsible for conversion of parameters used in a function call and de-conversion of results passed from the server after execution of the function. A server skeleton, the stub on server side, is responsible for de-conversion of parameters passed by the client and conversion of the results after the execution of the function.

Stub can be generated in one of the two ways:

1. Manually: In this method, the RPC implementer provides a set of translation functions from which a user can construct his or her own stubs. This method is simple to implement and can handle very complex parameter types.
2. Automatically: This is more commonly used method for stub generation. It uses an interface description language (IDL) that is used for defining the interface between Client and Server. For example, an interface definition has information to indicate whether, each argument is input, output or both — only input arguments need to be copied from client to server and only output elements need to be copied from server to client.

Q3) Differentiate between strong consistency model and causal consistency model

Points	Strong Consistency Model	Causal Consistency Model
Shared Memory System Support	If the value returned by a read operation o a memory address is always the same as the value written by a read operation	If all write operation that a potentially causally related are seen by all processes in the same order.
Absolute Global Time	Requires the existence of Absolute Global Time so that the memory read/write operation can be correctly ordered.	Does not require Absolute Global Time support.
Implementation	Implementation is practically impossible	Can be implemented practically

Q4). Explain fully the concept of preemptive process migration. What are the different address space transfer mechanisms used in the process transfer?

Process migration is the relocation of a process from its current location (the source node) to another node (the destination node).

A process may be migrated either before it starts executing on its source node – non-preemptive process – or during the course of its execution – pre-emptive process. Preemptive process migration is therefore done in case of the running process.

Preemptive process migration is costlier than non-preemptive process migration since the process environment must also accompany the process to its new node for an already executing process.

Process migration involves the following major steps:

1. Selection of a process that should be migrated
2. Selection of the destination node to which the selected process should be migrated
3. Actual transfer of the selected process to the destination node

The first two steps are taken care of by the process migration policy and the third step is taken care of by the process migration mechanism.
There are various mechanisms for freezing and restarting a process, address space transfer, message-forwarding and mechanisms for handling co-processes.

Address Space Transfer Mechanisms:

A process consists of the program being executed, along with the program's data, stack and state. The migration involves the transfer of
1. Process state
2. Process address space

The various address space transfer mechanisms are as follows:

1. Total Freezing:

   In this method, a process's execution is stopped while its address space is being transferred. This method is used in DEMOS/MP, Sprite and LOCUS. This method is simple and easy to implement. Its main disadvantage is that if a process is suspended for a long time during migration, timeouts may occur, and if the process is interactive, the delay will be noticed by the used.

2. Pre-transferring:

   In this method, the address space is transferred while the process is still running on the source node. Therefore, once the decision has been made to migrate a process, it continues to run on its source node until its address space has been transferred to the destination node. Pre-transferring (pre-copying) is done as an initial transfer of the complete address space followed by repeated transfers of the pages modified during the previous transfer until the number of modified pages is relatively small or until no significant reduction in the number of modified pages is achieved.

   This method is used in the V-System. Here, the freezing time is reduced so migration interferes minimally with the process's interaction with other processes and the user. Although pre-transferring reduces the freezing time of the process, it may increase the total time for migration to the possibility of redundant page transfers.

3. Transfer on Reference:

   This method is based on the assumption that processes tend to use only a relatively small part of their address spaces while executing. Here, the process's address space is left behind on its source node, and as the relocated process executes on its destination node, attempts to reference memory pages results in the generation of requests to copy in the desired blocks from their remote locations. Therefore, in this demand-driven copy-on-reference approach, a page of the migrant process's address space is transferred from its source node to its destination node only when referenced.

   This method is used in Accent. In this method, the switching time of the process from its source node to its destination node is very short once the decision about migrating the process has been made and is virtually independent of the size of the address space. However, this method is not efficient in terms of the cost of supporting remote execution once the process is migrated, and part of the effort saved in lazy transfer must be expended as the process accesses its memory remotely. This method also imposes a continuous load on process's source node and results in process failure if the source node fails or is rebooted.

Q5) What are threads? How are they different from processes?

A thread of execution is the smallest unit of a program. Threads are a popular way to improve application performance through parallelism and are referred to as 'lightweight processes' while processes are referred to as 'heavyweight processes'.

In traditional operating systems, the basic unit of CPU utilization is a process whereas in operating systems with thread facility, a thread is the basic unit and the process consists of an address space and one or more threads of control.

Each process has its own program counter, its own register states, its own stack and its own address space. However, each thread of a process has its own program counter, its own register states, and its own stack. But they share the same address space. Hence, they all share the same global variables. In addition, all threads of a process also share the same set of operating system resources, such as open files, child processes, semaphores, signals, accounting information, and so on.

Due to the sharing of address space, there is no protection between the threads of a process. Protection between processes is needed because different processes may belong to different users. However, a process (hence all its threads) is always owned by a single user. Therefore, protection between threads of a process is not necessary. If protection is required, then it is preferable to put them in different processes, instead of putting them in a single process.

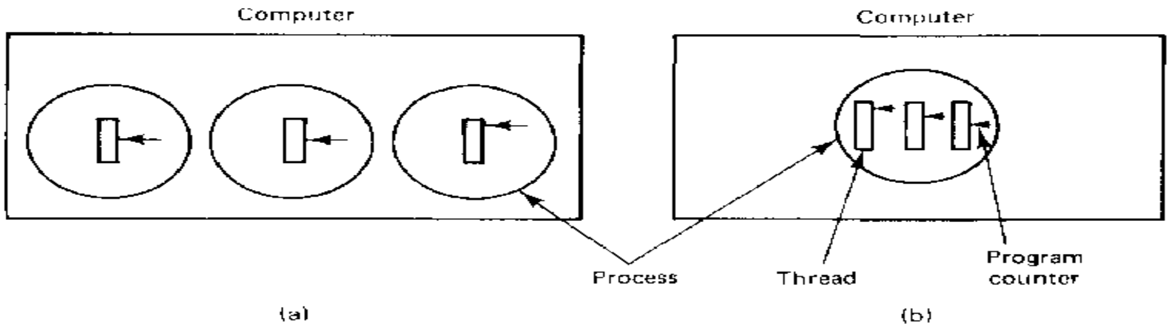


Fig. 4-1. (a) Three processes with one thread each. (b) One process with three threads.

Threads share a CPU in the same way as processes do. That is, on a uniprocessor, threads run in quasi-parallel (time sharing), whereas on a shared-memory multiprocessor, as many threads can run simultaneously as there are processors. Threads can create child threads, can block waiting for system calls to complete, and can change states during their course of execution.

The main motivation for using a multithreaded process instead of multiple single-threaded processes for performing some computation activities are as follows:

1. The overheads involved in creating a new process are in general considerably greater than those of creating a new thread within a process.

2. Switching between threads sharing the same address space is considerably cheaper than switching between processes that have their own address spaces.

3. Threads allow parallelism to be combined with sequential execution and blocking system calls. Parallelism improves performance and blocking system calls make programming easier.

4. Resource sharing can be achieved more efficiently and naturally between threads of a process than between processes because all threads of a process share the same address space.

Thus we can say that a process is a superset of its thread and a thread is a subset of its process.

Q 6) Describe relative advantages and disadvantages of various data locating mechanism that may be used in distributed shared memory system that uses the replicated migration block(RMB) strategy.

High availability is a desirable feature of a good distributed file system and file replication is primary mechanism for improving file availability. A replicated file is file that has multiple copies, with each copy loaded on a separate file is referred to as *replica* of replicated file.

Advantages Of Replication:

1. *Increased Availability:* One of the most important advantages of replication is that it makes and tolerates failure in network gracefully. In particular, the system remains operational and available to the users despite failures. By replicating critical data on server with independent failure modes, the probability of the one copy of the data will be accessible increases. Therefore, alternate copies of their files can be used
2. *Increased Reliability:* Many applications require extremely high reliability of their data stored in the files. Replication is very advantageous for such application because it allows the existence of multiple copies of their files. Due to the presence of redundant information in the system, recovery from the catastrophic failure becomes possible..
3. *Improve Response Time:* Replication also helps in improving the response time because it enable data to access either locally or from a node to which access time is lower than the primary copy access time. The access time differential may arise either because of network topology or because of uneven loading of nodes.
4. *Reduced Network Traffic:* If file's replica is available with a file server that resides on client's node, the client's access request can be serviced locally.
5. *Improved System Throughput:* Replication also enable several clients' requests for access to the same file to be serviced in parallel by different servers, resulting in improved system throughput.
6. *Better Scalability:* As numbers of user of shared file grows, having all access request to the file serviced by a single file server can result in poor performance due to overloading of file server. By replicating file on multiple servers, the same request can now be serviced more efficiently by multiple servers due to workload distribution. .
7. *Autonomous Operation:* In distributed system that provides file replication as service to their clients, all files required by a client for operation during a limited time period may be replicated on the file server residing at the client's node. This will facilitate temporary autonomous operation of client machines.

Disadvantage:

*Multicopy Update Problem:*As soon as the system allows to multiple copies of the same file to exist on different server, it is faced with the problem of keeping them mutually consistent. That is a situation must not be allowed to arise whereby independent conflicting updates have been made to different copies of same files.

Q 7) Why do most RPC systems support call-by-value semantics for parameter passing?

The RPC mechanism is an extension of the procedure call mechanism in the sense that it enables a call to be made to a procedure that does not reside in the address space of the calling process. The called procedure (commonly called remote procedure) may be on the same computer as the calling process or on a different computer.

The RPC model is similar to the well-known and well-understood procedure call model used for the transfer of control and data within a program in the following manner:

1. For making a procedure call, the caller places arguments to the procedure in some well- specified location.
2. Control is then transferred to the sequence of instructions that constitutes the body of the procedure.
3. The procedure body is executed in a newly created execution environment that includes copies of the arguments given in the calling instruction.
4. After the procedure's execution is over, control returns to the calling point, possibly returning a result.

As shown in the following figure, when a remote procedure call is made, the caller and the callee processes interact in the following manner:

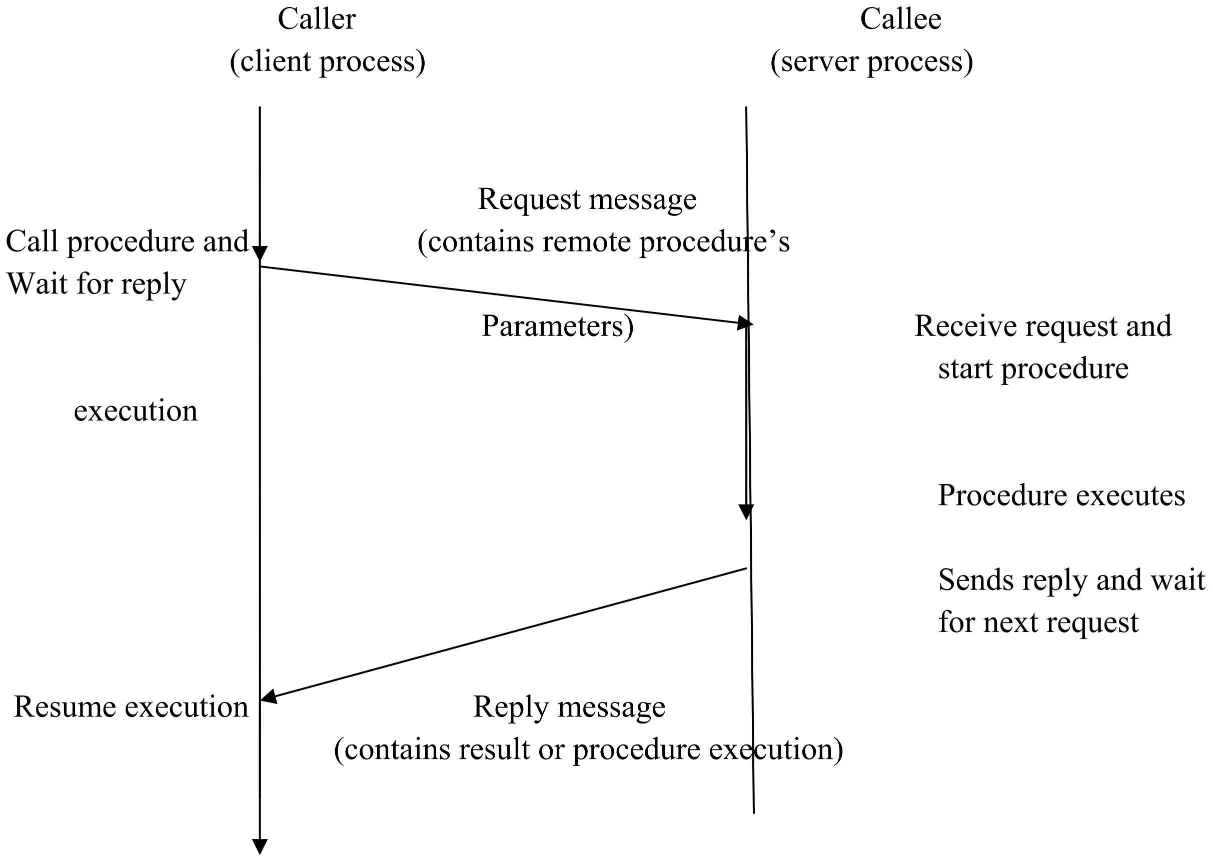


Fig: A typical model of Remote Procedure Call.

1. The caller (commonly known as client process) sends a call (request) message to the callee (commonly known as server process) and waits (blocks) for a reply message. The request message contains the remote procedure's parameters, among other things.
2. The server process executes the procedure and then returns the result of procedure execution in a reply message to the client process.
3. Once the reply message is received, the result of procedure execution is executed, and the caller's execution is resumed.

The choice of parameter- passing semantics is crucial to the design of an RPC mechanism. The two choices are call-by-value and call-by-reference.

Call- by- Value :

In the call by value method, all parameters are copied into a message that is transmitted from the client to the server through the intervening network. This poses no problems for simple compact type such as integers, counters, small arrays, and so on. However, passing larger data types such as multidimensional arrays, trees, and so on , can consume much time for transmission of data that may not be used. Therefore this method is not suitable for passing parameters involving voluminous data.

An argument in favor of the high cost incurred in passing large parameters by value is that it forces the user to be aware of the expense of remote procedure call for large parameter lists. In turn, the users are forced to carefully consider their design of the interface needed between client and server to minimize the passing of unnecessary data. Therefore before choosing RPC parameter-passing semantics, it is important to carefully review and properly design the client- server interfaces so that parameters become more specific with minimal data being transmitted.

Call- by- reference:

In call-by-reference method pointers to parameter are passed from client to server.Thus most RPC mechanisms use the call-by-value semantics for parameter passing because the client and the server exist in different address spaces, possibly even on different types of machines, so that passing pointers or passing parameters by reference is meaningless.

Q 8) What is callback RPC facility? Give an example of an application where this facility may be useful?

In the usual RPC protocol, the caller and callee processes have a client-server relationship. Unlike this, the callback RPC facilitates a peer-to-peer paradigm among the participating processes. It allows a process to be both a client and a server.

Callback RPC facility is very useful in certain distributed applications. For example remotely processed interactive applications that need user input from time to time or under special conditions for further processing requires this type of facility. As shown in the following figure, in such applications, the client process makes an RPC to the concerned server process, and during procedure execution for the client, the server process makes a callback RPC to the client process. On receiving this reply, the server resumes the execution of the procedure and finally returns the result of the initial call to the client. Note that the server may make several callbacks to the client before returning the result of the initial call to the client process.

The ability for a server to call its client back is very important, and care is needed in the design of RPC protocols to ensure that it is possible. In particular, to provide callback RPC facility, the following are necessary:

- Providing the server with client's handle.
- Making the client process wait for the callback RPC.
- Handling callback deadlocks.

Commonly used methods to handle these issues are described below.

<u>Providing the server with the client's handle:</u>

The server must have the client's handle to call the client back. The client's handle uniquely specifies the client process and provides enough information to the server for making a call to it. Typically, the client process uses a transient program number for the callback service and exports the callback service by registering its program number with the binding agent. The program number is then sent as a part of RPC request to the server. To make a callback RPC, the server initiates normal RPC request to the client using the given program number. Instead of having the client just send the server the program number, it could also send its handle, such as the port number. The client's handle could then be used by the server to directly communicate with the client and would save an RPC to the binding agent to get the client's handle.

<u>Making the client process wait for the callback RPC</u>**:**The client process must be waiting for the so that it can process the incoming RPC request from the server and also to ensure that a callback RPC from the server is not mistaken to be the reply of the RPC call made by the client process.

To wait for the callback, a client process normally makes a call to a svc-routine. The svc-routine waits until it receives a request and then dispatches the request to the appropriate procedure.

Handling callback deadlocks:

In callback RPC , since a process may play the role of either a client or server, callback deadlocks can occur. In effect, a callback deadlock has occurred due to the interdependencies of the three processes. Therefore While using a callback RPC, care must be taken to handle callback deadlock situations.

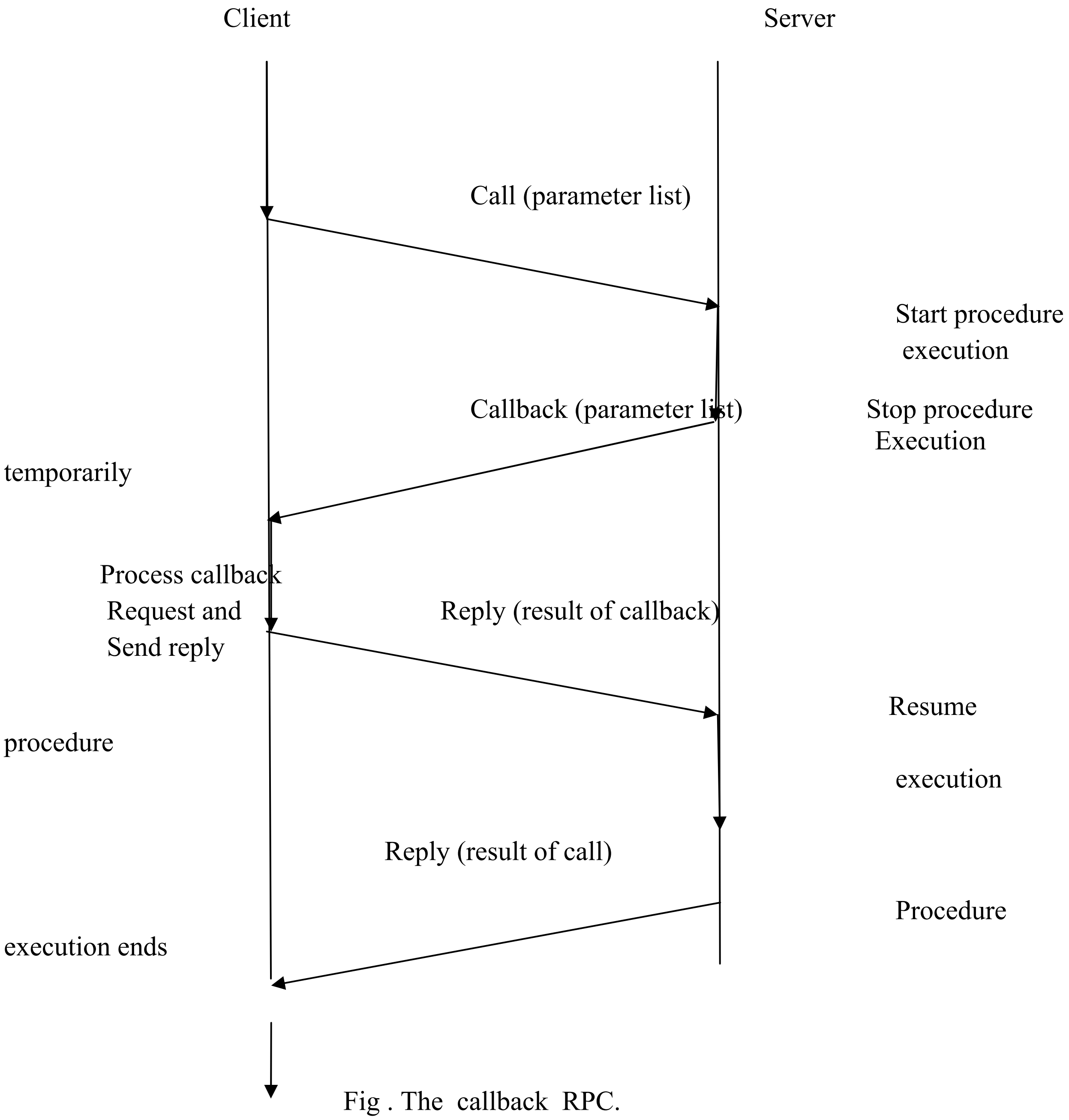


Fig . The callback RPC.

Q 9) A distributed system has 3 nodes N1, N2, N3 each having its own clock. The clock at modes N1, N2, N3 tick 495, 500 and 505 times per millisecond. The system uses external synchronization mechanism in which all nodes receive real time every 20 seconds from an external file source and readjust their clocks. What is the maximum clock skew that will occur in this system?

Ticks for node n1 = 495 x 10^{-3} sec = t1
Ticks for node n2 = 500 x 10^{-3} sec = t2
Ticks for node n3 = 505 x 10^{-3} sec = t3

Duration for real time updates = P = 20 sec

Calculating the deviation for nodes n1 & n2

Skew s_1 (n2 – n1)
= 2PΔt
= 2 x 20 x (t2 – t1)
=40 x (500 – 495) x 10^{-3} sec
=40 x 5 x 10^{-3} sec
=200 x 10^{-3} sec
=<u>20ms</u>

Calculating the deviation for nodes n2 & n3

Skew s_2 (n3 – n2)
= 2PΔt
= 2 x 20 x (t3 – t2)
=40 x (505 – 500) x 10^{-3} sec
=40 x 5 x 10^{-3} sec
=200 x 10^{-3} sec
=<u>20ms</u>

Calculating the deviation for nodes n3 & n1

Skew s_3 (n3 – n1)
= 2PΔt
= 2 x 20 x (t3 – t1)
=40 x (505 – 495) x 10^{-3} sec
=40 x 10 x 10^{-3} sec
=400 x 10^{-3} sec
=<u>40ms</u>

Hence, the maximum allowed clock skew will be 40ms for nodes n1 & n3.

Q 10) What is client server binding? How does a binding agent work in a client-server communication?

The Client/Server Binding is a simple, light-weight, Remote Procedure Call (RPC) mechanism, for use with 30 clients or less, that enables you to implement client/server architecture without having to worry about communications programming.

The Client/Server Binding is included as part of Server Express for use in applications which do not require an extensive, fault-tolerant middleware solution.

The Client/Server Binding removes the requirement for you to include communications code in your application by providing two modules called mfclient and mfserver. These are generic modules which can be used to drive any application.

Based on information contained in a configuration file, these modules manage the communications link and the transfer of data and are able to interact with user-defined programs at each end of the link. The mfclient module is called by a user-written client program (which may, for example, handle the user interface) while mfserver calls a user-written server program (which may, for example, handle data access and business logic).

How the Client/Server Binding Works:The Client/Server Binding works by having a non--dedicated copy of mfserver running on the server tier; this copy of mfserver communicates with any and all clients using an agreed server name. The mfserver module can be run using its built-in defaults, as its only function is to receive requests from the client to establish or terminate a connection.

The flow of information is explained below

The user-written client program (netxcli.cbl) performs any application initialization code, prompts the user for the name of a file, checks that the file exists and then calls mfclient.

1. The first time mfclient is called, it reads configuration information from the configuration file netxdem.cfg.

2. mfserver receives the connection request.

3. mfserver spawns a secondary server for each client. The server name is internally generated, based on the name of the initial server with a numeric ID added (for example,mfserver01). Alternatively, you can specify a name in the configuration file.

4. mfserver sends the secondary server name (mfserver01) back to mfclien**t** and terminates the conversation.

5. mfclient connects with the secondary server (mfserver01), passing parameters obtained from the configuration file via the LNK-PARAM-BLOCK

6. mfserver01 calls the user-written server program (netxserv.cbl), passing the parameters received from mfclient via the Linkage Section.

7. The first time the user-written server program (netxserv.cbl) is called, it performs any application initialization code, and exits the program. Control is returned to mfserver01.

8. mfserver01 returns control to mfclient.

9. mfclient ensures contact has been established and returns control to the user-written client program (netxcli.cbl).

10. The client program (netxcli.cbl) opens the file specified by the user, reads a record from it and calls mfclient to pass the data via the Linkage Section.

11. mfclient passes the data to mfserver01 via its internal buffer.

12. mfserver01 passes the data to the user server program (netxserv.cbl) which creates a file and writes the record to it.

    This read record/write record loop continues until the user-written client program (netxcli.cbl) reaches end-of-file, at which point it closes the file and sends an indication that it is about to terminate to the server. The server responds by sending the name of the directory into which the file was copied.

13. The client program (netxcli.cbl) displays the name of the file which has been created on the server and calls mfclient with LNK-CNTRL-FLAG set to "client-ending" and then terminates.

14. mfclient passes the "client-ending" parameter to mfserver01 which passes it to the user-written server program (netxserv.cbl) which terminates itself.

15. mfclient informs the base server (mfserver), that the secondary server (mfserver01) has terminated.

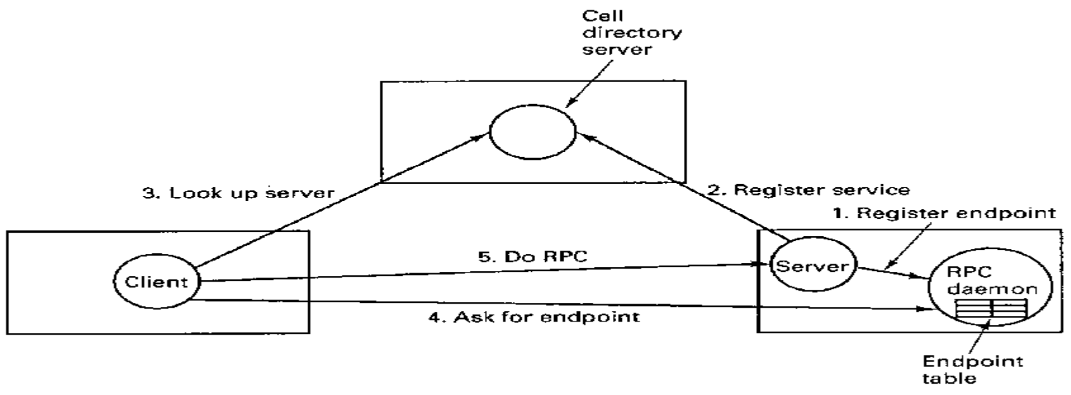


Fig. 10-14. Client-to-server binding in DCE.

Q 11) What are the main differences between the load balancing approach and load sharing approach for process scheduling in Distributed? Systems

Features	**Load balancing Approach**	**Load sharing Approach**
Definition	i. The load balancing algorithm tries to balance the total system load by transferring the workload from heavily loaded nodes to lightly loaded nodes to ensure good overall performance relative to specific metric of system performance	i. The load sharing algorithm attempts to ensure that no node is idle when a node is heavily loaded.
Goal	ii. The goal is to maximize the total system throughput.	ii. The goal is to prevent the nodes from being idle when some other nodes have more than two processes.
Load estimation policies	iii. Various methods are:- →The nodes workload can be estimated based on a measurable parameters which include:- →Total no. of processes on the node at time of load estimation. →Resources demands of these processes. →Instruction mixes of these processes. →Architectures and speed of the nodes processor. →Several Load balancing algorithm uses the total number of processes present on the node as a measure. →Another measure to estimates a nodes workload as the sum of the remaining service times of all processes on that node using methods like:- →**Memoryless method** which assumes that all processes have same expected service time, independent of the time used so far. →**Pastrepats method** assumes that the remaining service time of the process is equal to the time used so far by it. →**Distribution method** assumes the remaining service time is the expected remaining time conditioned by the time	iii. The load sharing algorithm normally employs the simplest load estimation policy of the total number of processes on a node. →But for the modern distributed systems having permanent existence of several processes on an idle node, measuring CPU utilization should be uses as a method of load estimation.

	already used. →The most acceptable method for the use as the load estimation policy in the distributed systems would be to measure the CPU utilization of the nodes.	
Process transfer policy	iv. Most of the load balancing algorithm uses the **threshold policy** to make the decision. The threshold value is the limiting value of nodes workload. The new process at a node is accepted locally for processing if the workload is below threshold value otherwise is transfer to a lightly loaded node. Two methods for determining threshold value:- →**Static policy** in which each node has predefined threshold value depending on its processing ability. →**Dynamic policy** in which the threshold value is calculated as product of the average workloads of all nodes and a predefined constant. Two types of threshold:- →**Single threshold** which have overloaded and underloaded region. In this node accepts new processes if its load is below threshold values and attempts to transfer local processes and rejects remote execution requests if the its load is above threshold value. →**Double threshold** or high-low policy uses two threshold values called high mark and low mark which divides the space into three regions i.e overloaded,	iv. →This algorithm employs **the all-or-none strategy.** →This strategy uses single threshold policy with the threshold value of all node fixed as 1. →The node becomes a candidate for accepting the remote process only when it has no process and a node becomes a candidate for transferring a process as soon as it has more than one process. →Some algorithm use threshold value of 2 instead of 1.

	normal, underloaded . →When the load of the node is in the overloaded region new local processes are sent to run remotely and requests to accept remote processes are rejected. →When the load is in the normal region new local processes run locally and request to accept remote processes are rejected. →When the load is in the underloaded region, new local processes run locally and requests to accept remote processes are accepted.	
Location policy	v. To select a location for the process execution following policies are used:- →**Threshold:-** In this destination node is selected at random and a check is made to determine whether the transfer of the process to that node would place it in a state that prohibits the node to accept remote processes. If not, the process is transferred to the selected node which executes it regardless of the state of process. If the check indicates that the node is in a state that prohibits it to accept remote processes, another node is selected in random. →**Shortest:-** In this Lp distinct nodes are chosen at random and each is polled in turn to determine its load. The process is transferred to the node having minimum load value, unless that node s load is such that it prohibits the node to accept remote process. If none can execute the process it is executes a originating node. →**Bidding:-**	v. In this algorithm, the location policy decides the sender node or the receiver node of a process that is to be moved within the system for load sharing. The location policies are:- →**Sender initiated policy:-** In this the heavily loaded nodes search for the lightly nodes to which work may be transferred. That is in this method when a nodes load become more than threshold value , it either broadcast a message or randomly probes the other nodes one by one to find a lightly loaded node that accept one or more processes, a suitable receiver node is known as soon as the sender node receives reply message from other nodes. →**Receiver initiated policy:-** In this the lightly loaded nodes search for heavily loaded nodes from which work may be transferred. When the nodes load fall below threshold value , it either broadcast a

	Each node in the network is responsible for two roles the manager represents a node having a process in need of location to execute and he contractor represents a node that is able to accept remote processes. To select a node manager broadcast requests-for-bids message and the contractor returns bids to the manager node. The manager chooses the best bid based on speed, performance. When the bid is selected a message is sent to the owner of that bid and the contractor sends back the message whether the bid is accepted or rejected. **→Pairing:-** The 2 nodes that differ greatly in load is temporarily paired with each other and the load balancing operation is carried out between nodes belonging to the same pair by migrating one or more processes from the more heavily loaded node to the other node.	message indicating its willingness to receive processes for executing or randomly probes the other nodes one by one to find heavily loaded nodes that can send processes
State information exchange policy	vi. The policies are:- **→Periodic broadcast:-** In this each node broadcast its state information after elapse of every t units of time. **→Broadcast when state changes**:- In this the node broadcast its state information only when the state of the node changes. **→On demand exchange:-** In this node broadcast a *stateinformationrequest* message when its state switches from normal load region to either underloaded or	vi. In this algorithm the nodes exchange the state information only when its state changes. The two commonly used policy are:- **→Broadcast when state changes:-** In this method a node broadcasts a stateinformationrequest message when it becomes either underloaded or overloaded. **→Poll when state changes:-** In this method when nodes state changes it does not exchange state information with all other nodes but

	overloaded region. On receiving this message the other nodes send their current state of the requesting node. →**Exchange by polling:-** This method is based on the idea that there is no need for node to exchange its information with all other node rather when node needs cooperation it can search for suitable partners by randomly polling the other nodes and the polled nodes.	randomly polls the other nodes one by one and exchanges state information with the polled nodes.
Overheads	vii. The overheads involved in gathering the state information is very large	vii. The overhead is less

Q.12 Discuss the relative advantages and disadvantages of using the full file caching and block caching models for the data caching mechanism of the distributed file system?

<u>Data caching model/mechanism:</u>

In the remote service model, every remote file access request results in network traffic. The data caching model attempts to reduce the amount of network traffic by taking advantage of the locality feature found in file accesses.

- In this model, if the data needed to satisfy the clients access request is not present locally, it is copied from the servers node to the clients node and is cached there.
- The clients request is processed on the clients node itself using cached data.
- Recently accessed data are retained in the cache for sometime so that repeated accesses to the same data can be handled locally.
- A replacement policy such as the least recently used(LRU) is used to keep the cache size bounded.
- A data caching model offers the possibility of increased performance and greater system scalability because it reduces network traffic, contention of the network and contention for the file servers.

<u>Full file caching model:</u>

In this model when an operation requires file data to be transferred across network in either direction between a client and a server, the whole file is moved.

Advantages:

- Transmitting an entire file in response to single request is more efficient than transmitting it page by page in response to several requests because the network protocol overhead is required only once.
- It has better scalability because it has fewer accesses to file servers, resulting in reduced server load and network traffic.
- Disk access routines on the servers can be better optimized if it is known that requests are always for entire files rather than for random disk blocks.
- Once a file is cached at a client's site, it becomes immune to server and network failures.
- The model also offers a degree of intrinsic resiliency.
- It also simplifies the task of supporting heterogeneous workstations because it is easier to transform an entire file at one time from the form compatible with the file system on server workstation to form compatible with the file system of the client workstation.

Disadvantage:

- It requires sufficient storage space on the clients node for storing all the required files in their entirely.
- This fails to work with very large files when the clients run on a diskless workstation.
- When the client's workstation is not diskless files that are larger than local disk capacity cannot be accessed at all.
- If only small fraction of file is needed, moving whole file is wasteful.

Block caching model:

In this model, the file data transfers across the network between client and a server take place in units of file blocks. A file block is a contiguous portion of a file and is usually fixed in length.

This model is also called as page-level transfer model.

Advantages:

- It does not require client nodes to have large storage space.
- It also eliminates the need to copy an entire file when only small portion of the file data is needed.
- This model can be used in the systems having diskless workstations.
- It provides large virtual memory for client nodes that do not have their own secondary storage devices.

Disadvantages:

- When an entire file is to be accessed, multiple server requests are needed in this model, resulting in more traffic and more network protocol overhead.
- This model has poor performance as compared to the file level transfer model when the access requests are such that most files have to be transferred entirely.

Q 13) Write Short Notes on:

i) Munin: A release consistent DSM system

In addition to sequential consistency, release consistency is also promising and attractive for used in DSM.Structure of shared memory space:

The shared memory space of munin is structured as collection of shared variable including program data structure. The shared variables are declared with keyword shared so that compiler can recognize them. Programmers can annote a shared variable with one of the standard annotation types.

Each shared variable by default is placed by compiler on a separate page that is the unit of data transfer across the network by MMU hardware. However programmers can specify that multiple shared variable having same annotation type be placed in same page. placing of different of annotation type in same page is not allowed because the consistency protocol used for a page depends on annotation type of variables contained in the page obviously variable of size larger than size of a page occupy multiple pages.

<u>Implementation of release consistency:</u>

Release consistency application must be modeled around critical section. Therefore a DSM system that support release consistency must have mechanism and programming language construct for critical section. Munin provides two such synchronization mechanism- A locking mechanism and a barrier mechanism.

-<u>The locking mechanism</u> uses lock synchronization variable with acquireLock and releaseLock primitives. The acquireLock primitive with a lock variable as its parameter is executed by a process to enter a critical section and releaseLock primitive with a same lock parameter is executed by same process to exit from critical section. To ensure release consistency, write operation on shared variable must be performed within a critical section but read operation can be performed within a critical section or outside.

-<u>The barrier mechanism</u> uses barrier synchronization variables with a WaitAtBarrier primitive for accessing the variables. They are implemented by using centralized barrier server mechanism.

Annotation for shared memory:

The release consistency of Munin allows application to have better performance than in a sequentially consistent DSM system. For further performance improvement, Munin defines several standard annotations for shared variables and uses a consistency protocol for each type that is most suited to that type.

1>Read only:

2>Migratery:

3>Mired Shared:

4>Write Shared

5>Producer Consumer

6>Result

7>Reduction

8>Conventional

ii) Mach Operating system.

Mach is a microkernel based operating system developed at Carnegie-Mellon university under the leadership of Richard Rashid and with the support of DARPA, the US department of Defense Advance Research Project Agency. It is based on previously developed OS at CEMU called accent.

Therefore, many of the basic concept in MACH are based on accet work. However,as compared to accent, Mach has many improved features including finer grained parallelism by the use of thread, multi-processor support, better inter processor communication mechanism and more flexible and efficient memory management scheme.

The first version of Mach was released in 1984 for the DEC VAX computer family, including the VAX 11/784, four CPU multiprocessor.

>Main features of Mach:

1>Open system architecture:

2>Compatibility with BSD UNIX:

3>Network transparency:

4>Flexible memory management:

5>Flexible IPC:

6>High performance:

7>Simple program interface:

>System Architecture Layers:

1>Microkernel Layer

2>User level server layer

3>Transparent shared library layer

4>Application code layer:

iv) Name Space

The naming system is one of the most important components of a distributed operating system because it enables other services and objects to be identified and accessed in uniform manner. A naming system employs one or more naming convention for name assignment to objects. For example, a naming system may use one naming convention for assigning human oriented names to objects.

The set of names complying with a given naming convention is said to form a name space.

Types of name space:

1) Flat Name Space
   The simplest name space is a flat name space where names are character string exhibiting no structure. Names defined in a flat name space are called primitive or flat names. Flat names space do not have any structure, it is difficult to assign unambiguous meaningful names to large set of objects. Therefore, flat names are suitable for use either for small spaces having names for only a few objects or for system-oriented names that need not be meaningful to the users.
2) Partitioned Name Space

When there is need to assign unambiguous meaningful names to a large set of objects, a naming convention that partitions the name space into disjoint classes in normally used. When partitioning is done systematically, the name structure reflects physical or organizational association. Each partition of a partitioned name space is called a domain of the name space.

A name defined in a domain is called a simple name. In a partitioned name space, all objects cannot be uniquely identified by simple name, and hence compound names are used for the purpose of unique identification. A compound name is composed of one or more simple names that are separated by a special delimiter character such as /, $, @, %.

A commonly used type of partitioned name space is the hierarchical name space, in which name space is portioned into multiple levels and is structured as an inverted tree. In this type of name space, the number of levels may be fixed or arbitrary.

v) File Replication

High availability is a desirable feature of a good distributed file system and file replication is a primary mechanism for improving file availability. A replicated file is a file that has multiple copies, with each copy located on a separate file server.

Advantages of Replication:

1) *Increased availability.*
   - It makes and tolerates failures in the network gracefully.
   - The system remains operational and available to the users despite failure.
   - Alternate copies of a replicated data can be used when the primary copy is unavailable.

2) *Increased reliability.* Many applications require extremely high reliability of their data stored in files. Replication is very advantageous for such application because it allows the existence of multiple copies of their files.

3) *Improved response time.* It helps in improving response time because it enables data to be accessed either locally or from a node to which access time is lower than the primary copy access time.

4) *Reduced network traffic.* If a file's replica is available with a file server that resides on a client's node, the client access request can be serviced locally, resulting in reduced network traffic.

5) *Improved system throughput.* Replication also enables several clients request for access to the same file to be serviced in parallel by different servers, resulting in improved system throughput.

6) *Better scalability.* By replicating the file on multiple servers, the request can now be serviced more efficiently by multiple servers due to workload distribution. This result in better scalability.

7) *Autonomous operation.* In a distributed system that provides file replication as a service to their client, all files required by client for operation during a limited time period may be replicated on the server residing at the client's node. This will facilitate temporary autonomous operation of client machine.

<u>The replication process is of two types</u>:

1) *Explicit replication.* In this type, users are given the flexibility to control the entire replication process. That is when a process creates a file, it specifies the server on which the file should be placed.
2) *Implicit/lazy replication.* In this type, the entire replication process is automatically control by system without user's knowledge. That is, when a process creates a file, it does not provide any information about its location.

vi) AMOEBA:-

Amoeba is a microkernel-based distributed operating system developed at Vrije University and the Center for Mathematics and Computer Science in Amsterdam, The Netherlands.It was started in 1981 by Andrew S. Tanenbaum as a research project in distributed and parallel computing. Since then, it has evolved over the years to acquire several attractive features.

- Transparency
- Parallel Programming Support
- Capability-Based, Object-Oriented Approach
- Small-kernel Approach
- High Performance
- High Reliability
- UNIX Emulation

It is also used system model called processor-pool. Kernel complexity in terms of system call interface is Simple. Supports multiple threads in a single process. Threads are managed and scheduled by Kernel. Automatic load balancing facilities are providing. A basic IPC mechanism is RPC. It is used untyped messages. Mechanism used for locating ports or processes is Hint cache and Broadcasting. Supported network protocols are TCP/IP and FLIP. It is not support the Virtual memory mechanism. It is also provide support for distributed shared memory on the object-based.

vii) Group Communication in IPC

The most elementary form of message-based interaction is one to one, communication(also known as point to point , or unicast communication) in which a single-sender process sends a message to a single-receiver process. However, for performance and ease of programming, several highly parallel distributed applications require that a message – passing system should also provide group communication facility. Depending on a single or multiple senders and receivers, the following three types of group communication are possible:

1. One to Many(single sender and multiple receivers)
2. Many to One(multiple senders and single receiver)
3. Many to Many(multiple sender and multiple receivers)

1. One to Many Communication:-In this scheme, there are multiple receivers for a message sent by a single sender. It is also known as multicast communication. A special case of multicast communication is broadcast communication, in which the message is sent to all processors connected to a network.
2. Many-to-One Communication:-In this scheme, multiple senders send messages to a single receiver. The single receiver may be selective or nonselective. A selective receiver specifies a unique sender ; a message exchange takes place only if that sender sends a message. On the other hand, a nonselective receiver specifies a set of senders, and if any one sender in the set sends a message to this receiver, a message exchange takes place.
3. Many-to-Many Communication:-In this scheme, the one-to-many and many-to-one schemes are implicit. In addition, an important issue related to many-to-many communication scheme is that of ordered message delivery.

viii) Lightweight Remote Procedure Call

This presents a lightweight communication facility (based on RPC) designed to handle communication between protection domains on a single machine.

- The Case for Lightweight RPC (LRPC):
    - Monolithic kernels are insulated from user programs, but few protection boundaries exist within the OS itself, which makes it difficult to modify, debug, and validate.
    - Capability systems consist of fine-grained objects sharing an address space but with their own protection domains. These offer flexibility, modularity, and protection.
    - The common case for communication is between domains on the same machine as opposed to across machines.
    - Most communication is simple, involving few arguments and little data, since complex data is often hidden behind abstractions (Both this and the above point were backed with some questionably representative examples).
    - While RPC is robust enough to handle both local and remote calls, it has high overhead. A lighter weight solution is necessary.
- Lightweight RPC features:
    - *Simple control transfer:* Conventional RPC involves multiple threads that must send signals and switch context. In LRPC, the kernel changes the address space for the caller's thread and lets it continue to run in the server domain (at the same priority?).
    - *Simple data transfer:* Shared buffers are pre-allocated (when the caller imports the LRPC module) for the communication of arguments and results. The caller copies the data onto the A-stack, after which no other copies are required.
    - *Simple stubs:* RPC has general stubs, but many of its features are infrequently needed. LRPC uses a stub generator that produces simple stubs in Modula2+ assembly language.
    - *Design for concurrency:* Idle processors in multi-processor machines cache domain contexts to reduce context-switch overhead. Counters are used to keep the highest activity LRPC domains active on idle processors.
- Performance:
    - 3X speedup for single processor.
    - Additional speedups for multiprocessor machines.

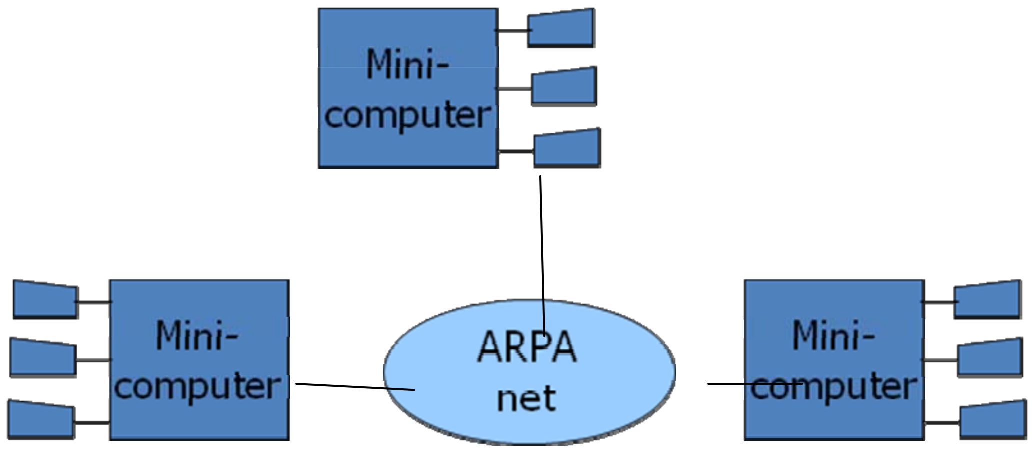
This offers an optimization that can be used to significantly improve the structure of operating systems through improving the common case of local cross-domain calls (with simple data). In general, their techniques and results seem useful and valuable.

ix) Distributed Computing System models

Minicomputer Model:

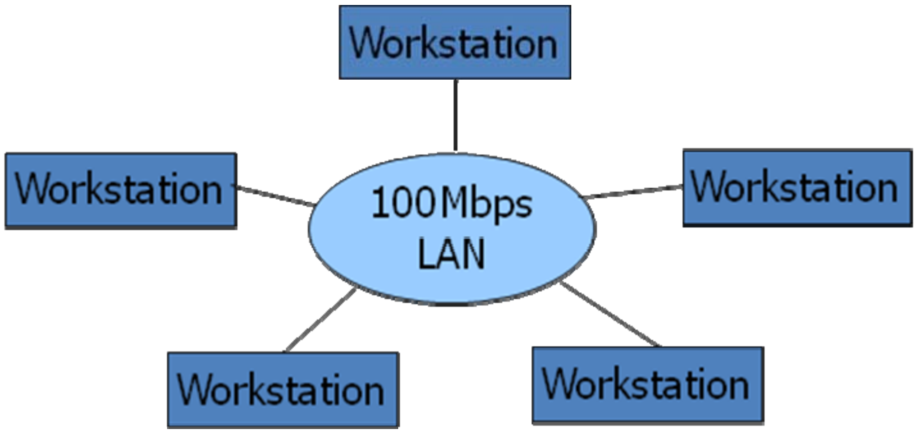
1. It consists of a few minicomputers interconnected by a communication network.
2. Each minicomputer usually has multiple users logged on to it simultaneously. For this several interactive terminals are connected to each minicomputer.

3. Each user is logged onto one minicomputer, with remote access to other minicomputers.
4. The network allows the user to access remote resources that are available on some machine other than the user is currently logged.
5. Minicomputer model can be used for resource sharing like information databases.



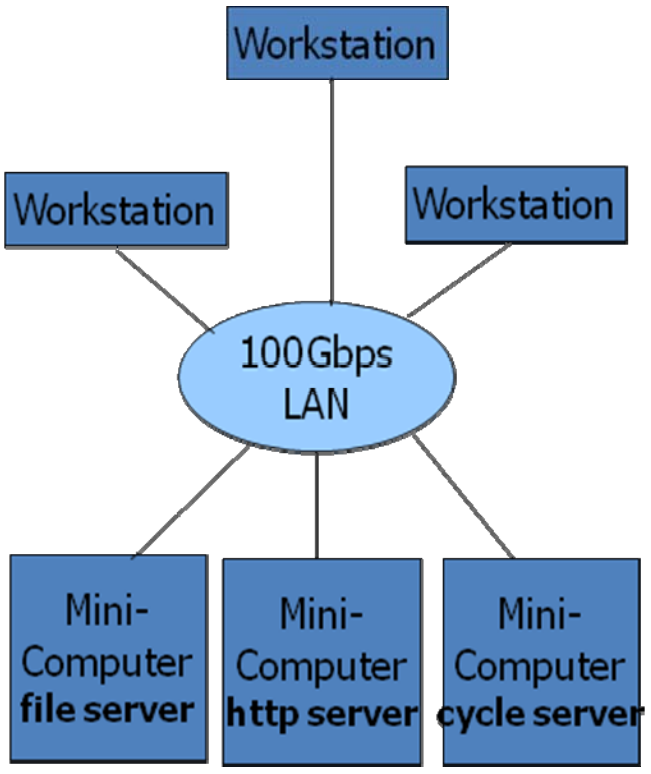
Workstation Model:

1. A workstation is a high-end microcomputer designed for technical or scientific applications.
2. Each workstation is equipped with its own disk and is intended primarily to be used by one person at a time (single-user computer), but they are commonly connected to a local area network and run multi-user operating systems.
3. Large number of workstations can lie idle resulting in a waste of CPU time.
4. Using a high speed LAN allows idle workstations to process jobs of users who are logged onto other workstations, which have less computing powers.



<u>Workstation-Server Model:</u>

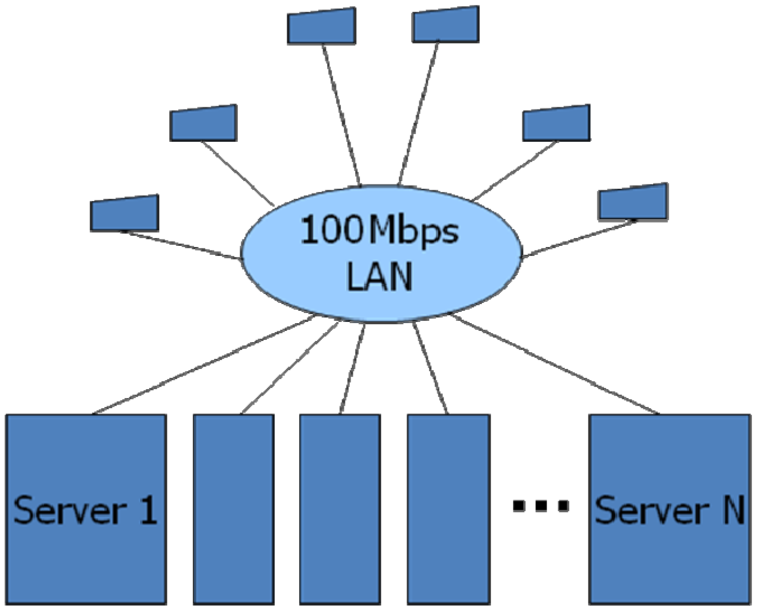
1. It is a network of personal workstations each with its own disk and local file system.
2. Workstation with its own local disk is usually called a disk-full workstation and a workstation without a local disk is called a diskless workstation.
3. When diskless workstations are used, file system to be used must be implemented by a disk-full workstation or by a minicomputer equipped with a disk for file storage.
4. Other minicomputers are used for providing other services like database, print etc


Workstation
Workstation
Workstation
100Gbps LAN
Mini-Computer file server
Mini-Computer http server
Mini-Computer cycle server

.

Processor-Pool Model:

1. When an user needs a very large amount of computing
power for a short time, instead of allocating a processor to
each user, processors are pooled together to be shared by
the users as needed.
2. Pool of processors contain a large number of microcomputers
and mini-computers.
3. Each processor in the pool has its own memory to load and
run a system program or an application program of the
distributed computing system.



Hybrid Model:
1.Advantages of the workstation-server and processor-pool models are combined to build a
hybrid model.
2. It is built on the workstation-server model with a pool of processors.
3. Processors in the pool can be allocated dynamically for large computations, that cannot be
handled by the workstations, and require several computers running concurrently for efficient
execution.
4. This model is more expensive to implement than the
hybrid or the processor-pool model

x) Release consistency:

1. Memory synchronization basically involves the following operations:

a. All changes made to the memory by the process are propagated to other nodes.

b. All changes made to the memory by other processes are propagated from other nodes to the process's node.

2. This model provides a mechanism to clearly tell the system whether a process is entering a critical section or exiting from a critical section so that system can decide and perform either first or second operation when a synchronization variable is accessed by the process.

3. This is achieved by using two synchronization variables (called acquire and release) instead of single synchronization variable. Acquire is used by the process to tell the system that it is about to enter the critical section. So that system perform only second operation when this variable is accessed.

4. On the other hand, release is used by a process to tell system that it has just exited the critical section, so that system perform only first operation when this variable is accessed.

5. Release consistency is also realized by using the synchronization mechanism based on barriers instead of critical sections. A barrier defines end of a phase of execution of a group of concurrently executing processes. All processes in the group must complete their execution up to a barrier before any process is allowed to proceed with its execution following the barrier.

6. A barrier can be implemented by using centralized barrier server. When a barrier is created it is given a count of number of processes that must be waiting on it before they can all be released.

7. For supporting release consistency, the following requirement must be met:

a. All accesses to acquire and release synchronization variables obeys processor consistency semantics.

b. All previous acquires performed by a process must be completed successfully before the process is allow to perform a data access operation on the memory.

c. All previous data access operations performed by a process must be completed successfully before a release access done by process is allowed.

xi) Stable storage:

1. Stable storage is a classification of computer data storage technology that guarantees atomicity for any given write operation and allows software to be written that is robust against some hardware and power failures.
2. To be considered atomic, upon reading back a just written-to portion of the disk, the storage subsystem must return either the write data or the data that was on that portion of the disk before the write operation.
3. Most computer disk drives are not considered stable storage because they do not guarantee atomic write; an error could be returned upon subsequent read of the disk where it was just written to in lieu of either the new or prior data.
4. Multiple techniques have been developed to achieve the atomic property from weakly atomic devices such as disks. Writing data to a disk in two places in a specific way is one technique and can be done by application software.
5. Most often though, stable storage functionality is achieved by mirroring data on separate disks via RAID technology (level 1 or greater). The RAID controller implements the disk writing algorithms that enable separate disks to act as stable storage.
6. The RAID technique is robust against some single disk failure in an array of disks whereas the software technique of writing to separate areas of the same disk only protects against some kinds of internal disk media failures such as bad sectors in single disk arrangements.

xii) Name cache:

1. In large distributed system a substantial portion of network traffic is naming related, hence it is very desirable for a client to be able to cache the result of name resolution operation for a while rather than repeating it every time the value is needed.
2. Name cache have substantial positive effect on distributed system due to the following characteristics of name service related activities:

   a. High degree of locality of name lookup.
   b. Slow update of name information database.
   c. One-use consistency of the cached information is possible.

3. Types of Name cache:

   a. <u>Directory cache:</u> in this type each entry consists of directory page. This type of cache is normally used in those systems that use the interactive method of name resolution. All recently used directory pages that are brought to the client node during name resolution are cached for a while.

b. Prefix cache: this type of name cache is used in those naming system that use the zone-based context distribution mechanism. In this each entry consist of name prefix and the corresponding zone identifier.
c. Full-name cache: in this each entry consists of an object's full pathname and the identifier and the location of its authoritative name server. Therefore requests for accessing an object whose name is available in the local cache can be directly sent to the object's authoritative name server.

xiii) File model

Different file system use different conceptual models of file. The two most commonly used criteria for file modeling are structure and modifiability.

- Unstructured and Structured Files

According to the simplest model, a file is an **unstructured sequence** of data. In this model there is no substructure known to the file server and the content of each file of the file system appears to the file server as a uninterrupted sequence of bytes. The operating System is not interested in the information stored in the files. The interpretation of the meaning and structure of the data stored in the files are entirely up to the application programs.

Most modern operating systems use the **unstructured file model**. This is mainly because sharing of a file by different application is easier with the unstructured file model. Since a file has no structure in unstructured model, different application can interpret the contents of a file in different ways.

Another file model is rarely used nowadays is the structured file model. In this model, a file appears to the file server as an ordered sequence of records. Records of different files of the same file system can be of different size. Therefore, many types of files exists ina file system, each having different properties.

Structured files are of two types-

1. Files with un-indexed records
   A file record is accessed by specifying its position within the file .
2. Files with indexed records
   Records have one or more key fields. In the file system that allows indexed records, a file is maintained as a B-tree or other suitable data structure.

- Mutable and Immutable Files

Most existing operating system use mutable file model. In this model, an update performed on the file overwrites on its old contents o produce new contents. That is, a file is represented as a single stored sequence that is altered by each update operation.

Some more recent file system, such as the Cedar File System, uses the immutable file model. In this model, a file cannot be modified once it has been created except to be deleted. The file versioning approach is normally used to implement file updates, and each file is represented by history of immutable versions. That is, rather than updating the same file ,a new version of file is created each time a change is made to the file contents and the old version is retained unchanged. Sharing only immutable files makes it easy to support consistent sharing. Due to this feature, it is much easier to support file caching and replication in a distributed system with the immutable file model because it eliminate all the problems associated with keeping multiple copies of file consists.

xiv) VMTP

The Versatile Message Transport Protocol (VMTP) is a transport protocol that has been especially designed for distributed operating system and has been used for V-system. It is a connectionless protocol that has special feature to support request/response behavior between a client and one or more server processes. It is based on the concept of message transaction that consist of a request message sent by a client to one or more servers followed by zero or more response messages sent back to the client by the servers, at most one per server. Most message transactions involve a single request message and a single response message.

For better performance, a response is used to serve as an acknowledgement for the corresponding request, and a response is usually acknowledged by the next request from the same client. Using special facilities, a client can request for an immediate acknowledgement for is response.

To support transparency and to provide group communication facility, entities in VMTP are identified by 64- bit identifiers that are unique, stable, and independent of the host address. The VMTP provides a **selective retransmission mechanism.** The packets of message are divided into packet groups that contain up to a maximum of 16 kilobytes of segment data. The data segment is viewed as a sequence of segment blocks of 512 bytes allowing the portion of the segment in a packet group to be specified by a 323-bit mask.

VMTP uses a rate-based flow control mechanism. In this mechanism, packets in a packet group are spaced out with interpacket gaps to reduce arrival rate at the receiver. The mechanism allows client and server to explicitly communicate their desired interpacket gap times and to make adjustment based on selective retransmission requests. An optimization used in VMTP is o differentiate between idempotent and nonidempotent operations. Idempotent operation is one whose execution is can be repeated any number of times without there being any side effects. When a response is non-idempotent, VMTP prevents the server from executing a request more than once.

xv) Atomic Multicast

It has an all-or-nothing property. That is, when a message is sent to a group by atomic multicast, it is either received by all the processes that are members of the group or else it is not received by any of them. Atomic multicast is not always necessary. For example, application for which the degree of reliability requirement is 0-reliable, 1- reliable, or m-out-of-n–reliable do not need atomic multicast. On the other hand, application for which the degree of reliability is all-reliable needs multicast facility. Therefore, a flexible message passing system should support both atomic and non-atomic multicast facilities.

A simple method to implement atomic multicast is to multicast a message, with the degree of reliability requirement being all-reliable. In this case, the kernel of the sending machine sends the message to all members of the group and waits for acknowledgement from each member. After the time out period, the kernel retransmits the message to all those members from whom an acknowledge message has not yet been received. The timeout retransmission of message is repeated until an acknowledgement is received from all members of the group. When all acknowledgements have been received, the kernel confirms to the sender that the atomic multicast process is complete. This method works fine only as long as the machines of the senders process and the receivers process do not fail during an atomic multicast operation.

An atomic multicast is in general very expensive as compared to a normal multicast due to the large number of messages involved in its implementation. Therefore, a message-passing should not use the atomicity property as a default property of multicast message but should provide this facility as an option.

Q14) What are idempotent operations? Give three examples. How will you make an operation idempotent?

Definition:

1. In computing, an idempotent operation is one that has no additional effect if it is called more than once with the same input parameters.

Pure functions (those whose return value only depends on the input parameters) are always idempotent (Python syntax examples):

```
def sq(x):
   return x * x
```

Functions that only set a value are idempotent:

2. Idempotent operations are often used in the design of network protocols, where a request to perform an operation is guaranteed to happen at least once, but might also happen more than once. If the operation is idempotent, then there is no harm in performing the operation two or more times.

Examples:

1. Looking up some customer's name and address in a database are typically idempotent, since this will not cause the database to change. Similarly, changing a customer's address is typically idempotent, because the final address will be the same no matter how many times it is submitted. However, placing an order for a car for the customer is typically not idempotent, since running the method/call several times will lead to several orders being placed. Cancelling an order is idempotent, because the order remains cancelled no matter how many requests are made.

A composition of idempotent methods or subroutines, however, is not necessarily idempotent if a later method in the sequence changes a value that an earlier method depends on – idempotence is not closed under composition. For example, suppose the initial value of a variable is 3 and there is a sequence that reads the variable, then changes it to 5, and then reads it again. Each step in the sequence is idempotent: both steps reading the variable have no side effects and changing a variable to 5 will always have the same effect no matter how many times it is executed. Nonetheless, executing the entire sequence once produces the output (3, 5), but executing it a second time produces the output (5, 5), so the sequence is not idempotent.

2. In the HyperText Transfer Protocol (HTTP), idempotence and safety are the major attributes that separate HTTP verbs. Of the major HTTP verbs, GET, PUT, and DELETE are idempotent (if implemented according to the standard), but POST is not. These verbs represent very abstract operations in computer science: GET retrieves a resource; PUT stores content at a resource; and DELETE eliminates a resource. As in the example above, reading data usually has no side effects, so it is idempotent .Storing a given set of content is usually idempotent, as the final value

stored remains the same after each execution. And deleting something is generally idempotent, as the end result is always the absence of the thing deleted.

3. In Event Stream Processing, idempotence refers to the ability of a system to produce the same outcome, even if an event or message is received more than once.

4. In load-store architecture, instructions that might possibly cause a page fault are idempotent. So if a page fault occurs, the OS can load the page from disk and then simply re-execute the faulted instruction. In a processor where such instructions are not idempotent, dealing with page faults is much more complex.

Making an operation idempotent:

One way to make the operations idempotent is to use the exactly-once semantics. One way to ensure the exactly-once semantics is to use a unique identifier for every request that the client makes and to set up a reply cache in the kernel's address space on the server machine to cache replies.

In this case, before forwarding a request to the server for processing, the kernel of the server machine checks to see if a reply already exists in the reply cache for the request. If yes, this means that this a duplicate request that has already been processed. Therefore the previously computed result is extracted from the reply cache and a new response is sent to the client. Otherwise the request is a new one. In this case, the kernel forwards the request to the appropriate server for the processing, and when the processing is over, it caches the request identifier along with the result of processing in the reply cache before sending a response to the client.

Q 15) Why is process migration important in a distributed system? What are desirable features of a good process migration mechanism? Explain the mechanism of migration with an example.

Process migration deals with the movement of a process from its current location to the processor to which it has been assigned.

Process migration is the relocation of a process from its current location(the source node) to another node(the destination node).

A process any be migrated either before it starts executing on its source node or during the course of its execution. The former is known as non-preemptive process migration and the latter is known as preemptive process migration. Preemptive process migration is costlier than non-preemptive process migration since the process environment must also accompany the process to its new node for an already executing process.

Process migration involves the following major steps:
1. Selection of a process that should be migrated
2. Selection of the distinction node to which the selected process should be migrated
3. Actual transfer of the selected process to the destination node.

The first two steps are taken care of by the process migration policy and the third step is taken care of by the process migration mechanism. The policies for the selection of a source node, a destination node and the process to be migrated on resource management.

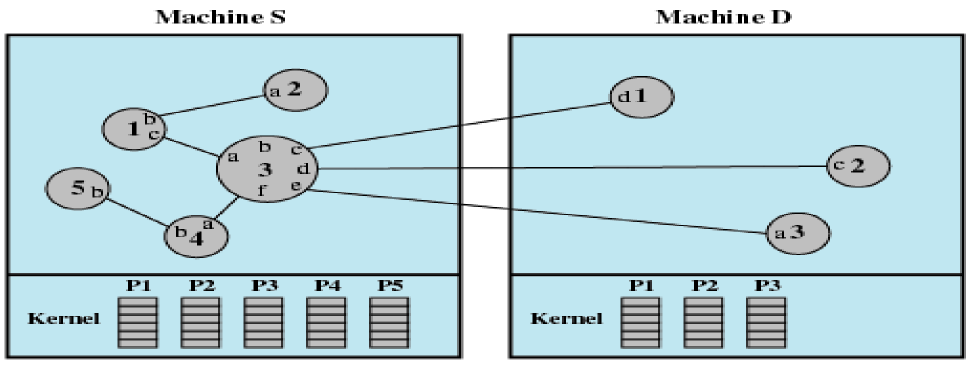
Goals:

- Load sharing
- Efficient interaction with other processes and data
- Access to special resources
- Survival

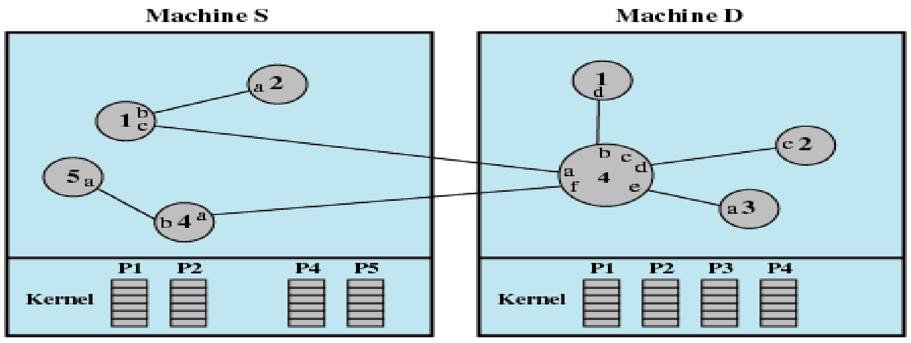
Migration Process:

- Destroy the process on A and create it on B
- Move at least the PCB
- Update any links between P and other processes and data . . .

Including any outstanding messages and signals, open files, etc.



(a) Before migration



(b) After migration

<u>Example:</u>

A bank is distributed over two branches.

To close a checking account at a bank, the account balance (global state of account) needs to be known

Deposits may not have cleared

Fund transfers may be pending

Checks may not have been cashed

Ask all correspondents to state pending activity

Close the account when all reply

Situation is analogous to determining the global state of a system

Q 16) What is a critical section? How will you implement mutual exclusion algorithm? Describe Ricart and Agrawala's algorithm for mutual exclusion?

Critical Section:

The sections of a program that need exclusive access to shared resources are referred to as critical sections.

For mutual exclusion, means are introduced to prevent processes from executing concurrently within their associated critical sections.

An algorithm for implementing mutual exclusion must satisfy the following requirements.

1. Mutual exclusion:
   If a resource is held by a process, any other requesting for that resource must wait until the resource has been released.
2. No starvation:

   When a process is not blocked but doesn't make any progress nonetheless because it repeatedly tries - but is always outcompeted by another process.

❖ Ricart and Agarwala's algorithm for mutual exlusion:

- In the distributed approach, the decision making for mutual exclusion is distributed across the entire system.
- 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. The message contains the following information:
  - The process identifier of the process.
  - The name of the critical section that process wants to enter.
  - A unique timestamp generated by the process for the request message.
- On receiving a request message, a process either immediately sends back a reply message to the sender or defers sending a reply based on the following rules:
  - If the receiver process is itself currently in the critical section, it simply queues the request message and defers sending a reply.
  - If the receiver process is currently not executing in the critical section but is waiting for its turn to enter the critical section, it compares the timestamp in the received request message with the timestamp in its own request message that it has sent to other processes. If the timestamp of the received request message is lower, it means that the sender process made a request before the receiver process to enter the critical section. Therefore, the

receiver process immediately sends back a reply message to the sender. On the other hand, if the receiver process's own request message has a lower timestamp, the receiver queues the received request message and defers sending a reply message.

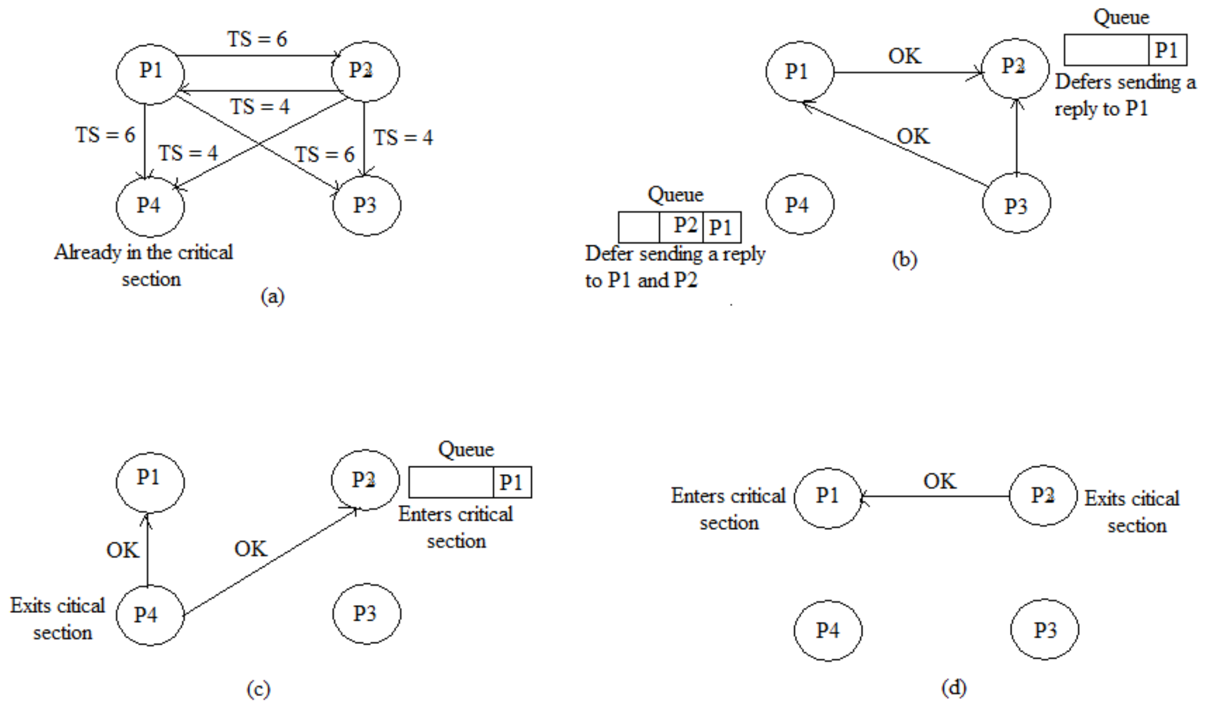
  - o If the process neither is in the critical section nor is waiting for its turn to enter the critical section, it immediately sends back a reply message.

- A process that sends out a request message keeps waiting for reply messages from other processes.
- It 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 and deletes them from its queue.

To illustrate how the algorithm works, let us consider the example of figure 1.

- There are four processes P1, P2, p3 and p4. While process P4 is in critical section, processes P1 and P2 want to enter the same critical section. To get permission from other processes, processes P1 and P2 send request messages with the timestamps 6 and 4 respectively to other processes.

Fig 1



- Now let us consider the situation in fig 1(b).Since process P4 is already in the critical section, it defers sending a reply message to P1 and P2 and enters them in its queue.

Process P3 is currently not interested in the critical section, so it sends a reply messages to both P1 and P2.

- Process P2 defers sending a reply message to P1 and enters P1 in its queue because the timestamp (4) in its own request message is less than the timestamp (6) in P1's request message.
- On the other hand, P1 immediately replies to P2 because the timestamp (6) in its own request message is greater than the timestamp (4) in P2's request message.
- Next consider the situation in Fig 1.c.When process P4 exits the critical section, it sends a reply message to all processes in its queue. Now since process P2 has received a reply message from all other processes, it enters the critical section. However, process P1 continues to wait since it has not yet received a reply message from the process P2.
- Finally, when process P2 exits the critical section, it sends a reply message to P1 (fig 1.d). Now since process P1 has received a reply message from all other processes, it enters the critical section.

Q 17) What is the difference between a procedural call and remote procedural call (RPC)? Explain RPC model fully with a diagram?

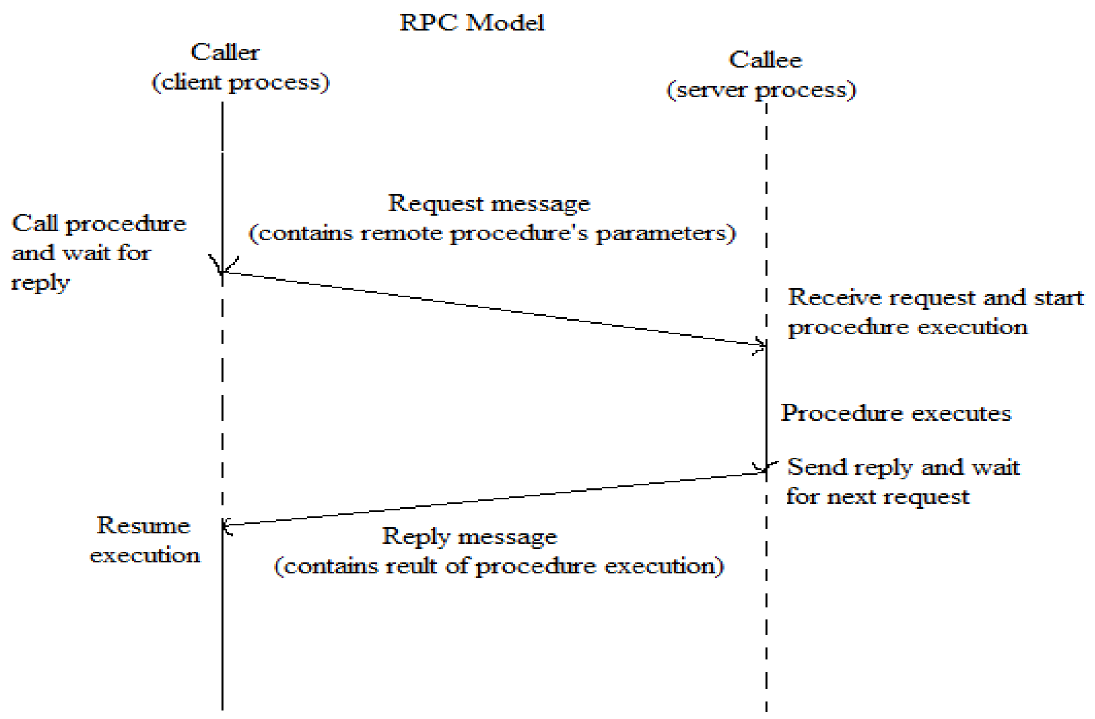
Difference between procedure call and remote procedure call:

Local procedure call	**Remote procedure call**
1. Procedure call model is used to call a procedure that reside in the same address space	RPC is used to call a procedure that resides in different address space.
2. Parameters can be passed by reference since the subroutine and the calling program share the same address space	Parameters cannot be passed by reference, since the subroutine and the calling program don't share the same address space
3. Overhead is less	There can be considerable overhead for calling an RPC. For example, the overhead might be 100 times the overhead of a local procedure call
4. The called (remote) procedure has access to any variables or data values in the calling program's environment.	The called (remote) procedure cannot have access to any variables or data values in the calling program's environment.
5. LPC is less vulnerable to failure.	RPC's are more vulnerable to failures since they involve two different processes and possibly a network and two different computers.
6. Local procedure call consumes less time than remote procedure call.	Remote procedure calls consume much more time (100 – 1000 times more) than local procedure call.

<u>Remote Procedure Call (RPC):</u>

- Remote Procedure Call is used to execute subroutines on other hosts. The ideas is to provide a simple mechanism for distributed computing without the need to include socket code (or something similar) in the program.
- RPC facility provides a valuable communication mechanism that is suitable for building fairly large distributed applications.

- RPC model:
    - When a remote procedure call is made, the caller and the callee processes interact in the following manner:


RPC Model
Caller
(client process)
Callee
(server process)
Call procedure and wait for reply
Request message
(contains remote procedure's parameters)
Receive request and start procedure execution
Procedure executes
Send reply and wait for next request
Resume execution
Reply message
(contains reult of procedure execution)

- The caller (commonly known as the client process) sends a call (request) message to the callee (commonly known as the server process) and waits for a reply message. The request message contains the remote procedure parameters, among other things.
- The server process executes the procedure and then returns the result of procedure execution in a reply message to the client process.
- Only the reply message is received, the result of procedure execution is extracted, and the caller's execution is resumed.

Implementation of Remote procedure call:

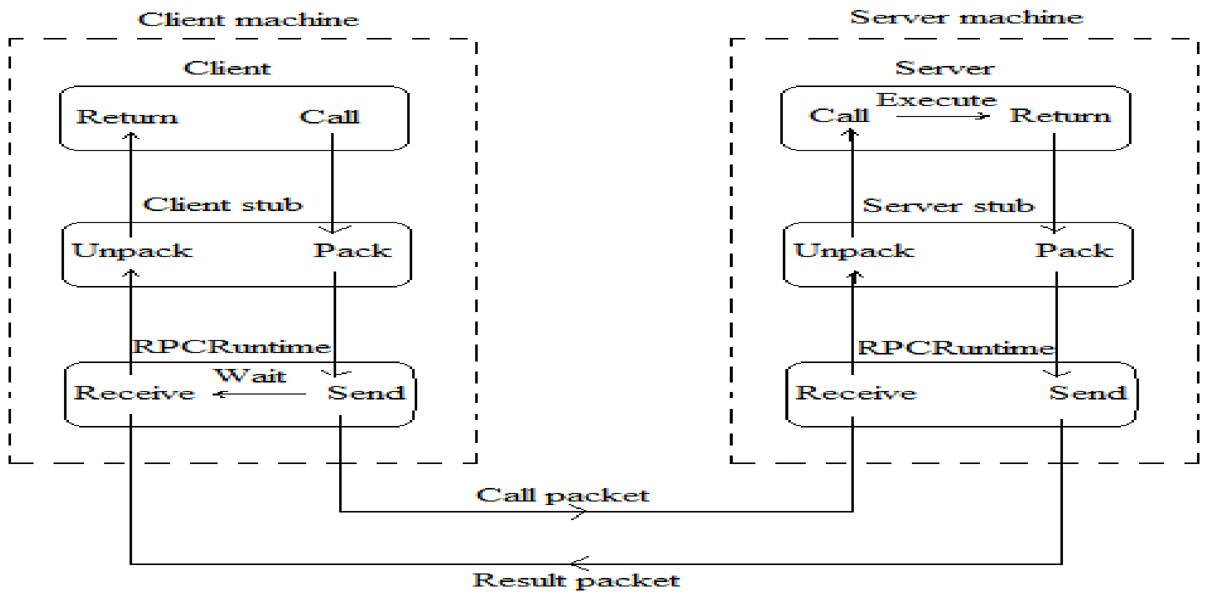


Fig: Implementation of RPC mechanism

The **client** calls the procedure.

Client Stub:

- Client calls the client stub, which packages the arguments into network messages. Packaging is called **marshaling.**
- On receipt of the result of procedure execution, it unpacks the result and passes it to the client.

RPC runtime:

- The RPC runtime handles the transmission of messages across the network between client and server machines.
- It is responsible for retransmission, acknowledgments, packet routing, and encryption.

Server Stub:

- It unpacks (unmarshals) the request message and makes a local procedure call to invoke the appropriate procedure in the server.
- It packs the result from the server and asks the local RPCRuntime to send it to the client stub.

Server:

- The **server process** executes the procedure and returns the result to the server stub

Q 18) Solve any one of the below.

1….Discuss the relative advantages of full-file caching and block-caching models for the data caching mechanisms of distributed file system.

An important issue in file systems that use the data-caching model is to decide the unit of data transfer (fraction of file data that is transferred to and from client as a result of a single read or write operation).

Four commonly used data transfer model are as follows.

1) File-level (full-file caching) transfer model.
2) Block-level transfer model.
3) Byte-level transfer model
4) Record-level transfer model.

1) **Full-file caching**: - In this model, when an operation requires file data to be transferred across the network in either direction (client/server) the whole file is moved.

Advantage:

1) Transferring entire file in response to single request is more efficient than transmitting it page by page in response to several requests; because the network protocol overhead required only once.

2) It has better scalability because it requires fewer accesses to file servers; which results in reduced server load and network traffic.

3) Disk access routine on server can be better optimized if it is known that requests are always for entire file rather than for random disk blocks.

4) Once entire file is cached at client's site, it becomes immune to server and network failures.

5) Simplifies the task of supporting heterogeneous workstations; because it's easier to transform an entire file at one time from the form compatible with the file system of server workstation to the form compatible with the file system of the client workstation.

2) **Block-level transfer model:** - In this model, file data transfers across the network between client and server take place in units of file blocks (portion of file which is usually fixed in length).

Advantage:

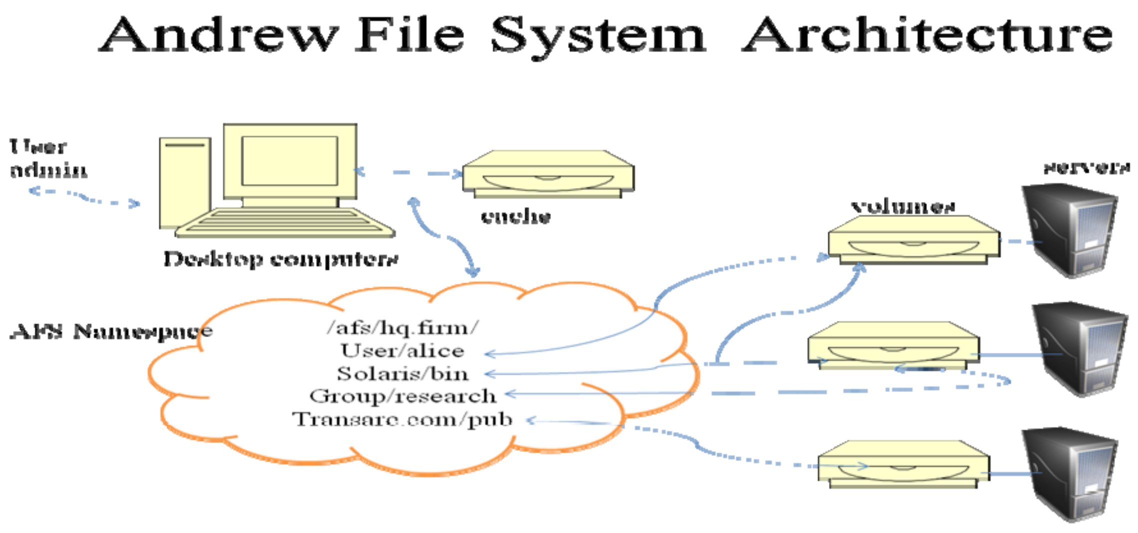
1) It does not require client nodes to have large storage space.

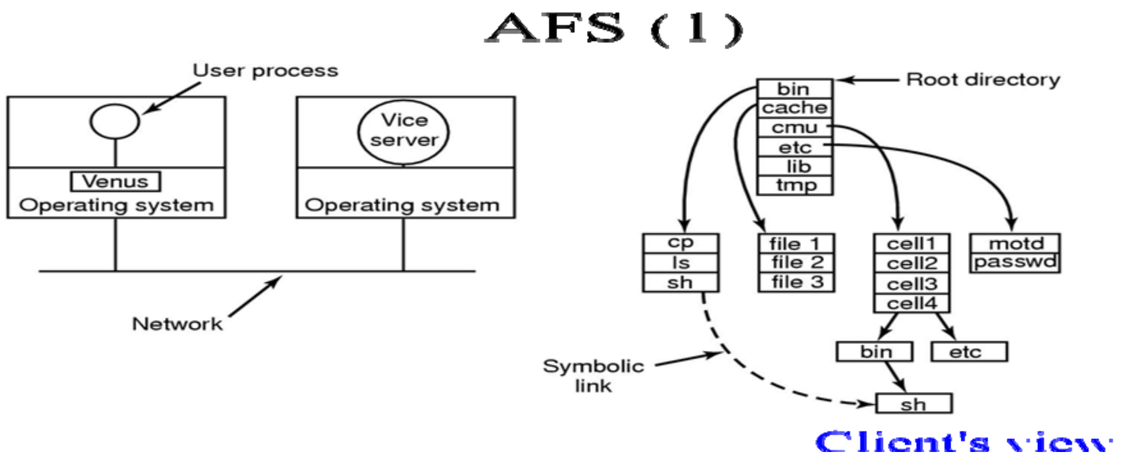
2) Eliminates the need to copy entire file when only a small portion of the file data is needed; therefore this model can be used in system having diskless workstations.

3) It provides large virtual memory for client node that do not have their own secondary storage devices.

2….Describe the architecture of Andrews file system in detail. Explain the concept of virtual file system layer in AFS.

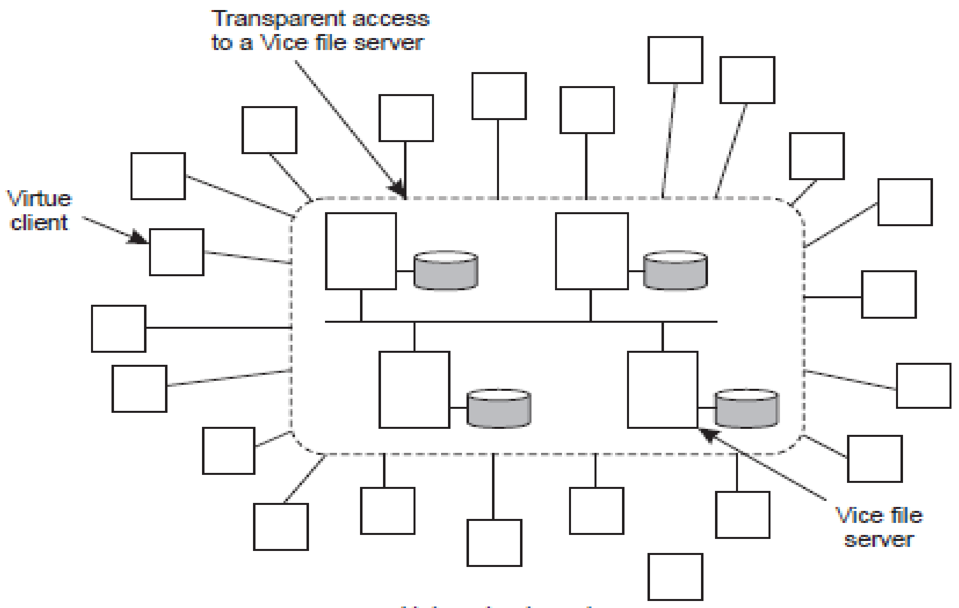
AFS (Andrew File System) is a distributed, networked file system that enables efficient file sharing between clients and servers.

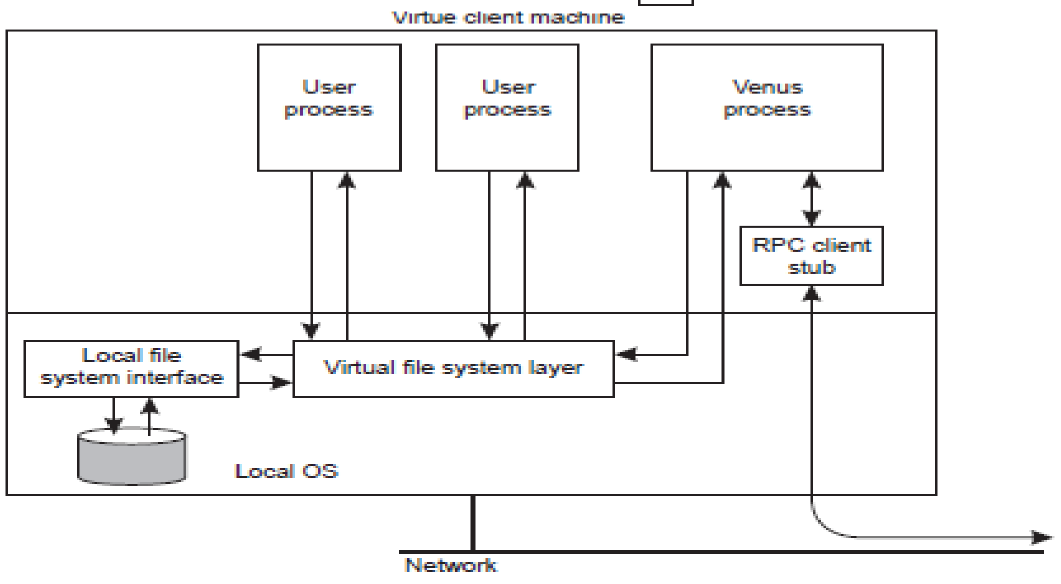




System Architecture:

1) Client: User-level process Venus (AFS daemon)
2) Trusted servers collectively called vice.
3) Cache on local disk





DESIGN & ARCHITECTURE

1) Disconnected operation:

2) All client updates are logged in a Client Modification Log (CML)

3) On re-connection, the operations registered in the CML are replayed on the server

4) CML is optimized (e.g. file creation and removal cancels out)

5) On weak connection, CML is reintegrated on server by trickle reintegration

6) Trickle reintegration tradeoff: Immediate reintegration of log entries reduces chance for optimization, late reintegration increases risk of conflicts

7) File hoarding: System (or user) can build a user hoard database, which it uses to update frequently used files in a hoard walk

8) Conflicts: Automatically resolved where possible; otherwise, manual correction necessary

9) Conflict resolution for temporarily disconnected servers

Features

- File sharing — easily and securely share documents and files over the network with colleagues, workgroups, and even other institutions.
- File backup and restore — all data is backed up nightly. Backups are kept for 30 days.
- File access — easily and securely access your files from other computers.
- File permissions — use AFS file permissions to provide access to individuals or groups.
- Websites — store and serve web pages directly and securely.
- File security — files are protected by the Kerberos authentication system.

Q 19) What are different types of memory consistency in a distributed shared memory? What is difference between sequential and release consistency? Which is preferred and why?

A consistency model basically refers to the degree of consistency that has to be maintained for the shared- memory data for the shared-memory data for the memory to work correctly for a certain set of applications .It is defined as a set of rules that application must obey if they want the dsm system to provide the degree of consistency guaranteed by the consistency model.

The different types memory consistency in a distributed shared memory are as follow:

1) Strict Consistency Model

2) Sequential Consistency Model

3) Causal Consistency Model

4) Pipelined Random-Access Memory Consistency Model

5) Processor Consistency Model

6) Weak Consistency Model

7) Release Consistency Model.

1) <u>Strict Consistency Model</u>:

The strict consistency model is the strongest form of memory coherence, having the most stringent consistency requirement. A shared memory system is said to support the strict consistency model if the value returned by a read operation on a memory address is always the same as the value written by the most recent write operation to that address , irrespectively of the locations of the processes performing the read and write operation .That is ,all writes instantaneously become visible to all process.

2) <u>Sequential Consistency Model</u>:

The sequential consistency model was proposed by Lamport. A shared-memory system is said to support the sequential consistency model if all processes see the same order of all memory access operation on the shared memory.The exact order in which the memory access operation are interleaved does not matter.

3) Causal Consistency Model:

The causal consistency model was proposed by Hutto and Ahamad, relaxes the requirements of the sequential consistency model for better concurrency. Unlike the sequential consistency model in the causal consistency model ,all process see only those memory references operations that are not potentially causally related. A memory references operations (read/write)is said to be potentially causally related to another memory reference operation if the first one might have been influenced in any way of the second one.

4) Pipelined Random-Access Memory Consistency Model:

The pipelined random access memory consistency model, proposed by Lipton and Sandberg, provides a weaker consistency semantics than the consistency semantics that consistency model described. It only ensures that all write operations performed by a single process are seen by all other process in the order in which they performed as if all write operations performed by a single process are in a pipeline.

5) Process Consistency Model:

The processor consistency model , proposed by Goodman, is very similar to an PRAM consistency model with an additional restriction of memory coherence. That is a processor consistent memory is both coherent and adheres to the PRAM consistency model .Memory coherences means that for any memory location all processes agree same order of all write operation location.

6) Weak Consistency Model:

The weak consistency model , proposed by Dubois et al. [1988], is designed to the advantage o f the following two characteristics common to many application:

1. It is not necessary to show the change in memory done by every time write operation to other process.The results of several write operation write operation can be combined and sent to other processes only when they need it.

2. Isolated accesses to shared variables are rare.

7) Release Consistency Model:

The release consistency model provides a mechanism to clearly tell the system whether a process is entering a critical section so that the system can decide and perform only either the first or the second operation when a synchronization variables is accessed by a process.

<u>The difference between Sequential consistency model and Release consistency model are as follow</u>

The sequential consistency model was proposed by Lamport. A shared-memory system is said to support the sequential consistency model if all processes see the same order of all memory access operation on the shared memory. The exact order in which the memory access operations are interleaved does not matter. A DSM system supporting the sequential consistency model can be implemented by ensuring that no memory operation is started until all the operation is started until all the previous ones have been completed. A sequentially consistency memory provides one-copy/single-copy semantics because all the processes sharing a memory location always see exactly the same contents stored in it. This is the most intuitively excepted semantics for memory coherence. Therefore, sequential consistency is acceptable by most applications.

Release consistency may also be realized by using the synchronization mechanism based on barriers instead of critical sections. A barriers defines the end of a phase executing of a group of concurrently executing processes. All processes in the group must complete their execute up to barriers before any process is allowed to proceed with the executing following the barriers.

The preferred one is Release consistency model because a barrier can be implemented by a centralized barrier server. when a barriers is created it is given a count the number of processes that must be waiting on it before they can all be released. Each process of a group of concurrently executing processes send a message to the barriers server when it arrives at a barriers and then blocks until reply is received from the barrier sever. The barriers server does not send any replies until all processes in the group have arrived at the barriers.

Q 20) In a fault tolerant communication between client-server, how will you implement exactly-once semantics in the following cases:-

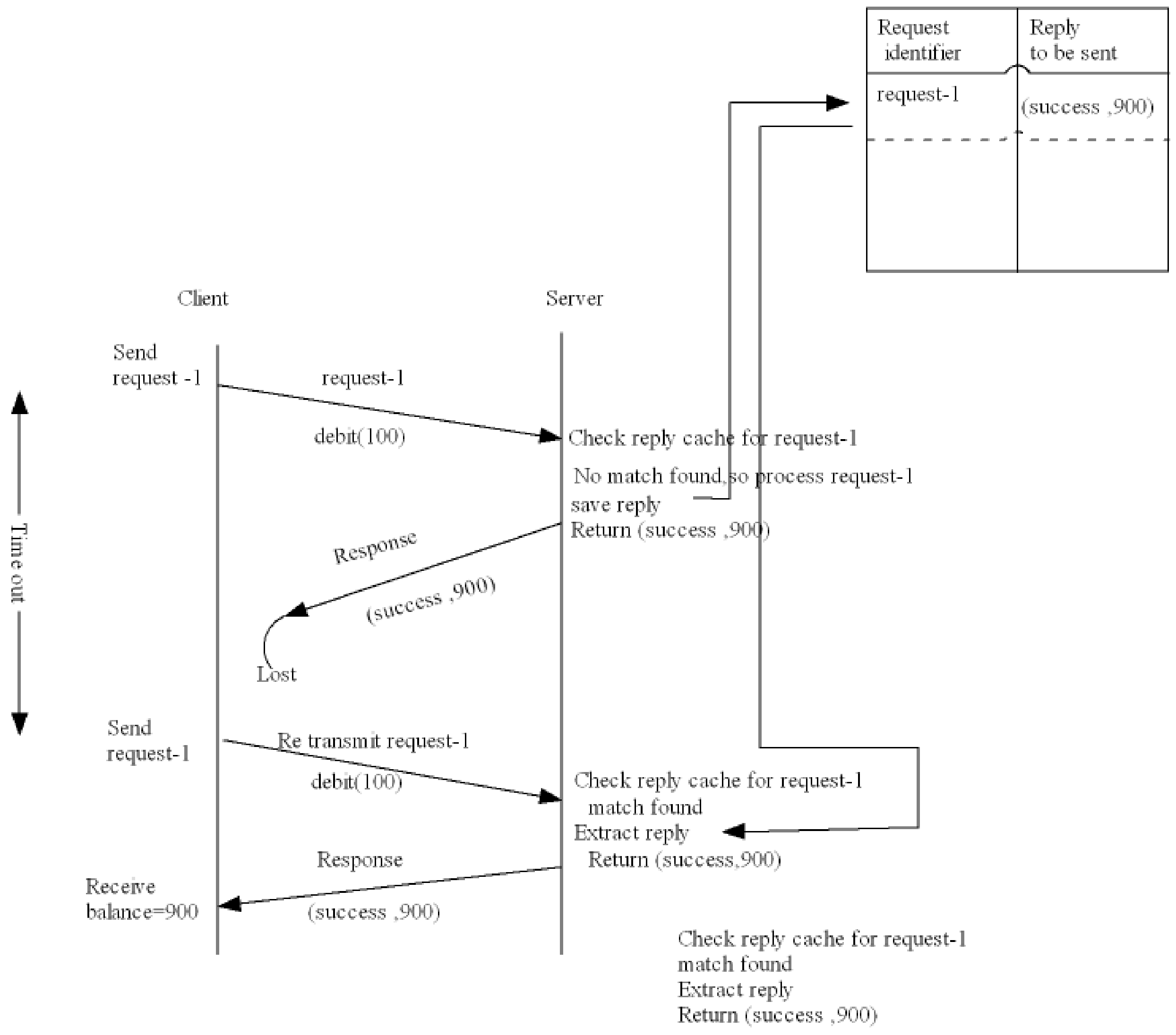
i. **The client-server machines are reliable but the communication links connecting them are unreliable,**
ii. **The client-server machines are unreliable but server and the communication links are reliable,**
iii. **The client is unreliable but the server and the communication links are reliable**

iv. **The client and links are reliable but server is unreliable.**

One way to implement exactly-once semantics is to use a unique identifier for every request that the client makes and to set up a reply cache in the kernel's address space on the server machine to cache the replies .in this case, before forwarding a request to a server for processing ,the kernel of the server machine checks to see if a reply already exist in the reply cache for the request .If yes ,this means that this is a duplicate request that has already been processed .Therefore the previously computed result is extracted from the reply cache and a new response message is sent to the client .Otherwise ,the request is a new one.

An example of implementing exactly once semantics is shown in fig below .The client requests are numbered, and a reply cache has been added to the server machine. the client makes request -1 ;the server machine's kernel receives request -1 and the checks the reply cache to see if there is a cached reply for request-1

There is no match, so it forwards the request to the appropriate server. The server processes the request and returns the result to the kernel .The kernel copies the request identifier and the result of execution to the reply cache and sends the result in the form of response message to the client. This reply is lost and the client times out on request-1 and retransmits request-1.The server machines kernel receives request-1 once again and checks the reply cache to see if there is a cached reply for request-1.This time match is found so it extracts the result corresponding to request-1 from the reply cache and once again sends it to the client as a response message. Thus the reprocessing of duplicate message is avoided.



Example of exactly -once semantic using request identifier and reply cache

Q 21) Explain with diagrams what is meant by Absolute Ordering, Consistent Ordering and Casual Ordering of messages?

Ans. An important issue related to Many-to-Many communication scheme is that of ordered message delivery.

Ordered message delivery ensures that all messages are delivered to all receivers in an order acceptable to the application. This property is needed by many applications for their correct functioning.

To ensure ordered message delivery, a special message handling mechanism is required in Many-to-Many communication scheme.

The commonly used semantics for ordered delivery of multicast messages are Absolute Ordering, Consistent Ordering and Casual Ordering.

Absolute Ordering:

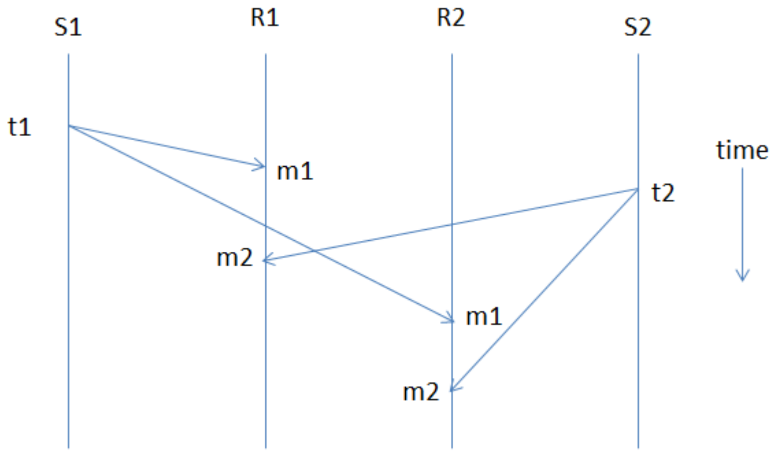


Fig: Absolute Ordering of messages

- This semantic ensures that all messages are delivered to all receiver processes in the exact order in which they were sent.
- One method to implement this is to use global timestamps as message identifiers. i.e. the system is assumed to have a clock at each machine and all clocks are synchronized with each other, and when a sender sends a message, the clock value(timestamp) is taken as the identifier of that message and embedded in the message.
- The kernel of each receiver's machine saves all messages meant for a receiver in a separate queue.
- A sliding-window mechanism is used to periodically deliver the message from the queue to the receiver. i.e. a fixed time interval is selected as the window size, and periodically

all messages whose timestamp values fall within the current window are delivered to the receiver.

- Messages whose timestamp values fall outside the window are left in the queue because of the possibility that a tardy message having timestamp value lower than that of any of the messages in queue might still arrive.
- The window size is properly chosen taking into consideration the maximum possible time that may be required by a message to go from one machine to any other machine in the network

<u>Consistent Ordering:</u>

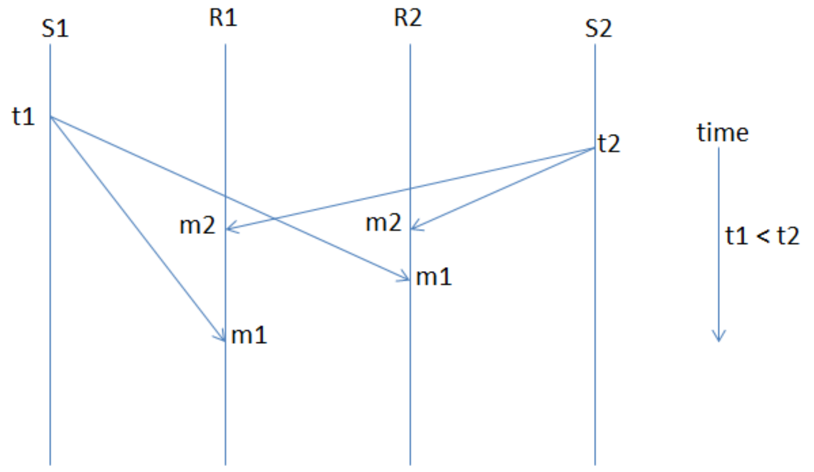


Fig: Consistent Ordering of Messages

- Absolute ordering is not really what applications need to function correctly as it requires globally synchronized clocks, which are not easy to implement.
- Instead of supporting absolute ordering semantics, most systems support Consistent Ordering semantics.
- This semantics ensures that all messages are delivered to all the receiver processes in the same order. However, this order may be different from the order in which messages were sent.
- One method to implement consistent ordering semantic is to make the many-to-many scheme appear as a combination of many-to-one and one-to-many schemes. i.e. the kernels of sending machines send messages to single receiver(known as <u>Sequencer</u>) that assigns a sequence number to each message and then multicasts it. The kernel at each receiver's machine saves all incoming messages in a separate queue.

- Messages in the queue are delivered immediately to the receiver unless there is a gap in the message identifiers, in which case messages after the gap are not delivered until the ones in the gap have arrived.
- A problem with this method is poor reliability- single point of failure.
- A distributed algorithm for implementing consistent ordering semantics that overcomes the above problem is ABCAST Protocol of ISIS system.

ABCAST Protocol:

This assigns a sequence number to a message by distributed agreement among the group members and the sender and works as follows.

1. The sender message assigns a temporary sequence to the message and sends it to all the members of the multicast group. The sequence number must be larger than any previous sequence number used by the sender. A simple counter can be used by the sender to assign sequence numbers to its messages.
2. On receiving the message, each member of the group returns a proposed sequence number to the sender. A member (i) calculates its proposed sequence number by using the function—

$$\max(F_{max}, P_{max}) + 1 + i/N$$

   where, F_{max}= largest final sequence number agreed upon so far for a message received by group.

   P_{max}= largest proposed sequence number by this member.

   N= total number of members in the multicast group.
3. When the sender has received the proposed sequence numbers from all the members, it selects the largest one as final sequence number for the message and sends it to all members in a *commit* message. The chosen final sequence number is guaranteed to be unique because of the term i/N in the function used for calculation of a proposed sequence number.
4. On receiving the *commit* message, each member attaches the final sequence number to the message.
5. Committed messages with final sequence numbers are delivered to the application programs in order of their sequence numbers.

Casual Ordering:

- An application can have a better performance if the message-passing system used supports weaker ordering semantics that is acceptable to the application.
- Casual ordering semantics is one such weak ordering semantic that is acceptable to many applications.
- This semantic ensures that if the event of sending one message is casually related to event of sending another message, the two messages are delivered to all receivers in the correct order.

- However, if two message-sending events are not casually related, the two messages may be delivered to the receivers in any order.
- Two message-sending events are said to be casually related if they are correlated by the *happened-before* relation. i.e. two message-sending events are casually related if there is any possibility of the second one being influenced in any way by the first one.
- The basic idea is that when it matters, messages are always delivered in proper order, but when it does not matter, they may be delivered in any arbitrary order.
- The figure below shows an example of Casual Ordering Message Semantic.

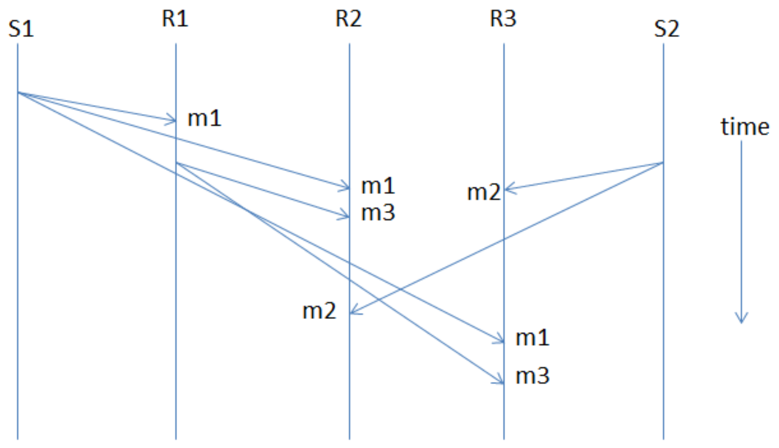


Fig: Casual Ordering of Messages

- In this, sender S1 sends message m1 to receivers R1, R2, and R3 and sender S2 sends message m2 to receivers R2 and R3.
- On receiving m1, receiver R1 inspects it, creates a new message m3, and sends it to R2 and R3. (NOTE: the event of sending m3 is casually related to the event of sending m1 because the contents of m3 might have been derived in part from m1; hence the two messages should be delivered to both R2 and R3 in proper order, m1 before m3. Since m2 is not casually related to either m1 or m3, m2 can be delivered at any time to R2 and R3 irrespective of m1 or m3.
- One method to implement this semantic is the CBCAST Protocol of ISIS system.

CBCAST Protocol:
This protocol works as follows.

1. Each member process of a group maintains a vector of *n* components, where *n* is the total number of members in the group. Each member is assigned a sequence number from 0 to *n*, and the *i*th component of the vectors corresponds to the member with sequence number *i*. The value of the *i*th component of a member's vector is equal to the number of the last message received in sequence by this member from member *i*.
2. To send a message, process increments the value of its own component in its own vector and sends the vector as part of the message.
3. When the message arrives at the receiver's site, it is buffered by the runtime system. The runtime system tests for 2 conditions to decide whether the message can be delivered to

the user process or its delivery must be delayed to ensure casual-ordering semantics. The two conditions to be tested are:

a. $S[i] = R[i] + 1$

   This condition ensures that receiver has not missed any message from the sender. This is needed because two messages from the same sender are always casually related.

b. $S[j] \leq R[j] \neq i$

   This condition ensures that the sender has not received any message that the receiver has not yet received. This is needed to make sure that the sender's message is not casually related to a message missed by the receiver.

If the message passes these two tests, the runtime system delivers it to the user process. Otherwise, message if left in the buffer and the test is carried out again for it when a new message arrives.

Q 22) Discuss all the issues involved in distributed system design

Transparency, is the quality of a distributed operating system to be seen and understood as a **single-system image**, and is the greatest overriding consideration in the high-level conceptual design of a distributed operating system. While a simple concept, the consideration of transparency directly effects decision making in every aspect of design of a distributed operating system. Depending on the degree to which transparency is implemented into a system, certain requirements and/or restrictions may be imposed upon the many other design considerations, and their relationships.

Transparency allows a user to accomplish a system-related objective with absolute minimal knowledge of the particular internal details related to the objective. A system or application may expose as much, or as little transparency in a given area of functionality as deemed necessary. That is to say, the degree to which transparency is implemented can vary between subsets of functionality in a system or application. There are many specific areas of a system that can benefit from transparency; access, location, performance, naming, and migration to name a few.

For example, a distributed operating system may present access to a hard drive as "C:" and access to a DVD as "G:". The user does not require any knowledge of device drivers or methods of direct memory access techniques possibly used behind-the-scenes; both devices work the same way, from the user's perspective. This example demonstrates a high-level of transparency; and displays how low-level details are made somewhat "invisible" to the user through transparency. On the other hand, if a user desires to access another system or server, a host name or IP address may be required, along with a remote-machine user login and password. This would indicate a low-degree of transparency, as there is detailed knowledge required of the user in order to accomplish this task.

Generally, transparency and user-required knowledge form an inverse relation. As transparency is designed and implemented into various areas of a system, great care must be taken not to adversely affect other areas of transparency and other basic design concerns. Transparency, as a design concept, is one of the grand challenges in design of a distributed operating system; as it is a factor in the necessity for a complete upfront understanding.

- **Location transparency** - Location transparency comprises two distinct sub-aspects of transparency, Naming transparency and User mobility. Naming transparency requires that nothing in the physical or logical references to any system entity should expose any indication of the entities location, or its local or remote relationship to the user. User mobility requires the consistent referencing of system entities, regardless of the system location from which the reference originates. Transparency dictates that the relative location of a system entity—either local or remote—must be both invisible to, and undetectable by the user.

- **Access transparency** - Local and remote system entities must remain indistinguishable when viewed through the user interface. The distributed operating system maintains this perception through the exposure of a single access mechanism for a system entity, regardless of that entity being local or remote to the user. Transparency dictates that any differences in methods of accessing any particular system entity—either local or remote—must be both invisible to, and undetectable by the user.

- **Migration transparency** - Logical resources and physical processes migrated by the system, from one location to another in an attempt to maximize efficiency, reliability, availability, security, or whatever reason, should do so automatically controlled solely by the system. There are a myriad of possible reasons for migration; in any such event, the entire process of migration—before, during, and after—should occur without user knowledge or interaction. Transparency dictates that both the need for, and the execution of any system entity migration must be both invisible to, and undetectable by the user.

- **Replication transparency** - A system's elements or components may need to be copied to strategic remote points in the system in an effort to possibly increase efficiencies through better proximity, or provide for improved reliability through the duplication of a back-up. This duplication of a system entity and its subsequent movement to a remote system location may occur for any number of possible reasons; in any event, the entire process—before, during, and after—should occur without user knowledge or interaction. Transparency dictates that both the necessity and execution of replication, as well as the existence of replicated entities throughout the system must be both invisible to, and undetectable by the user.

- **Concurrency transparency** - The distributed operating system allows for simultaneous use of system resources by multiple users and processes, who are kept completely unaware of the concurrent usage. Transparency dictates that both the necessity for concurrency, and the multiplexed usage of system resources must be both invisible to, and undetectable by the user. [DSC, pg 23]

- **Failure transparency** - In the event of a partial system failure, the system is responsible for the automatic, rapid, and accurate detection and orchestration of a remedy. These measures should exhibit minimal user imposition, and should initiate and execute without user knowledge or interaction. Transparency dictates that users and processes be exposed to absolute minimal imposition as a result of partial system failure; and any system-employed techniques of detection and recovery must be both invisible to, and undetectable by the user. [DSA, pg 30]

- **Performance Transparency** - In any event where parts of the system experience significant delay or load imbalance, the system is responsible for the automatic, rapid, and accurate detection and orchestration of a remedy. These measures should exhibit minimal user imposition, and should initiate and execute without user knowledge or interaction. While reasonable and predictable performance are important goals in these situations, there should be no expressed or implied concepts of fairness or equality among affected users or processes. Transparency dictates that users and processes be exposed to absolute minimal imposition as a result of performance delay or load imbalance; and any system-employed techniques of detection and recovery must be both invisible to, and undetectable by the user. [DCD, pg 23]

- **Size/Scale transparency** - A system's geographic reach, number of nodes, level of node capability, or any changes therein should exists without any required user knowledge or interaction. Transparency dictates that system and node composition, quality, or changes to either must be both invisible to, and undetectable by the user. [DCD, pg 23]

- **Revision transparency** - System occasionally have need for system-software version changes and changes to internal implementation of system infrastructure. While a user may ultimately become aware of, or discover the availability of new system functions or services, their implementation should in no way be the prompt for this discovery. Transparency dictates that the implementation of system-software version changes and changes to internal system infrastructure must be both invisible to, and undetectable by the user; except as revealed by administrators of the system. [DSA, pg 30]

- **Control transparency** - All system information, constants, properties, configuration settings, etc. should be completely consistent in appearance, connotation, and denotation to all users and software applications aware of them. [TLD, pg 84]

- **Data transparency** - No system data-entity should expose itself as peculiar with respect its location or purpose in the system, as a result of user interaction. [TLD, pg 85]

- **Parallelism transparency** - Arguably the most difficult aspect of transparency, and described by Tanenbaum as the "Holy grail" for distributed system designers. A system's parallel execution of a task among various processes throughout the system should occur without any required user knowledge or interaction. Transparency dictates that both the need for, and the execution of parallel processing must be both invisible to, and undetectable by the user.

Inter-process communication

Inter-Process Communication (IPC) is the implementation of general communication, process interaction, and dataflow between threads and/or processes both within a system node, and between all nodes in a distributed operating system. The distributed nature of a system's nodes and the multi-level considerations of intra-node and inter-node requirements provide the base-line for high-level IPC design considerations. However, IPC in a distributed operating system is a low-level implementation. IPC is the low-level critical complement to the high-level concept of transparency. Many of the requirements and restrictions imposed on a system as a result of transparency will be accomplished directly or indirectly through IPC. In this sense, IPC is the greatest underlying concept in the low-level design considerations of a distributed operating system.

Process management

Process management provides policies and mechanisms for effective and efficient sharing of a system's distributed processing resources between that system's distributed processes. These policies and mechanisms support operations involving the allocation and de-allocation of processes and ports to processors, as well as provisions to run, suspend, migrate, halt, or resume execution of processes. While these distributed operating system resources and the operations on them can be either local or remote with respect to each other, the distributed operating system must still maintain complete state of and synchronization over all processes in the system; and do so in a manner completely consistent from the user's unified system perspective.

As an example, load balancing is a common process management function. One consideration of load balancing is which process should be moved. The kernel may have several mechanisms, one of which might be priority-based choice. This mechanism in the kernel defines *what can be done*; in this case, choose a process based on some priority. The system management components would have policies implementing the decision making for this context. One of these policies would define what priority means, and how it is to be used to choose a process in this instance.

Resource management

Systems resources such as memory, files, devices, etc. are distributed throughout a system, and at any given moment, any of these nodes may have light to idle workloads. **Load sharing** and load balancing require many policy-oriented decisions, ranging from finding idle CPUs, when to move, and which to move. Many algorithms exist to aid in these decisions; however, this calls for a second level of decision making policy in choosing the algorithm best suited for the scenario, and the conditions surrounding the scenario.

Reliability

One of the basic tenets of distributed operating systems is a high-level of **reliability**. This quality attribute of a distributed operating system has become a staple expectation. Reliability is most often considered from the perspectives of availability and security of a system's hardware, services, and data. Issues arising from availability failures or security violations are considered faults. Faults are physical or logical defects that can cause errors in the system. For a system to be reliable, it must somehow overcome the adverse effects of faults.

There are three general methods for dealing with faults: **fault avoidance**, **fault tolerance**, and **fault detection and recovery**. Fault avoidance is considered to be the proactive measures taken to minimize the occurrence of faults. These proactive measures can be in the form of transactions, replicated resources and processes, and primary back-ups of complete servers. Fault tolerance is the ability of a system to continue some meaningful level of operation in the face of a fault. In the event a fault does occur, the system should detect the fault and have the capability to respond quickly and effectively to recover full functionality. In any event, any actions taken should make every effort to preserving the single system image.

Performance

Performance is arguably the quintessential computing concern, and in the distributed operating system, it is no different. Many benchmark metrics exist for performance; throughput, job completions per unit time, system utilization, etc. Each of these benchmarks are more meaningful in describing some scenarios, and less in others. With respect to a distributed operating system, this consideration most often distills to a balance between process parallelism and IPC. Managing the task granularity of parallelism in a sensible relation to the messages required for support is extremely effective. Also, identifying when it is more beneficial to migrate a process to its data, rather than copy the data, is effective as well. Many process and resource management algorithms, and algorithms in this space work to maximize performance.

Synchronization

Cooperating concurrent processes have an inherent need for synchronization. Three basic situations that define the scope of this need:

- one or more processes must synchronize at a given point for one or more other processes to continue,
- one or more processes must wait for an asynchronous condition in order to continue,
- or a process must establish mutual exclusive access to a shared resource.

There are a multitude of algorithms available for these scenarios, and each have many variations. Unfortunately, whenever synchronization is required the opportunity for process deadlock usually exists.

Flexibility

Flexibility in a distributed operating system is made possible through the modular characteristics of the microkernel. With the microkernel presenting an absolute minimal—but complete—set of primitives and basic functionally cohesive services, The higher-level management components can be composed in a similar cohesive manner. This capability leads to exceptional flexibility in the management components collection; but more importantly, it allows the opportunity to dynamically swap, upgrade, or install additional instances of components above the kernel.

Q 23) Enumerate the desirable features of a good message passing system. Also explain buffering mechanism used in IPC.

A message passing system is a subsystem of a distributed operating system that provides a set of message-based IPC protocols and does so by shielding the details of complex network protocols and multiple heterogeous platform from programmers.

Desirable features of a good message-passing System:

1) Simplicity **:-**
   A message-passing system should be simple and easy to use.
   It must be straightforward to construct new applications and to communicate with existing ones by using the primitives provided by the message-passing system.
   It should also be possible for a programmer to designate the different modules of a distributed application and to send and receive message between them in a way as simple as possible.
   Clean and simple semantics of the IPC protocols of a message-passing system make it easier to build distributed applications and get them right.

2) Uniform semantics **:-**
   In a distributed system, a message passing system may be used for the following two types of interprocess communication:
   - Local communication: in which the communication processes are on the same node.
   - Remote communication: in which the communication process are on different nodes.

   An important issue in design of message passing system is that the semantics of remote communications should be as close as possible to those of local communications.
   This is important requirement for ensuring that the message-passing system is easy to use.

3) Efficiency **:-**
   Efficiency is normally a critical issue for a message-passing system to be accepted by the users. if the message-passing system is not efficient,interprocess communication may become so expensive that application designers will strenuously try to avoid its use in their applications.
   An IPC protocol of a message-passing system can be made efficient by reducing the number of message exchanges, as far as particable, during the communication process.
   Some optimizations normally adopted for efficiency include the following:

- Avoiding the costs of establishing and terminating connections between the same pair of processes for each and every message exchange between them.
- Minimizing the costs of maintain the connections.
- Piggybacking of acknowledgement of previous message with the next message during a connection between a sender and a receiver that involves several message exchanges.

4) Reliability**:**
Distributed systems are prone to different catastrophic events such as node crashes or communication link failures.
A reliable IPC protocol can cope with failure problems and guarantees the delivery of a message. Handling of messages usually involves acknowledgement and retransmission on the basis of timeouts.
Another issue related to reliability is that of duplicate messages. Duplicate messages may be sent in the event of failures or because of timeouts. A reliable IPC protocol is also capable of detecting and handling duplicates. Duplicate handling usually involves generating and assigning appropriate sequence numbers to messages.
A good message-passing system must have IPC protocols to support these reliability features.

5) Correctness :
Correctness is a feature related to IPC protocols for group communication.
Although not always required, correctness may be useful for some applications.
Issues related to correctness are
   - Atomicity:-ensures that every message sent to a group of receivers will be delivered to either all of them or none of them.
   - Ordered delivery: - ensures that messages arrive at all receivers in an order acceptable to the application.
   - Survivability: - guarantees that messages will be delivered correctly despite partial failures of processes, machines, or communication links.

6) Flexibility **:-**
The IPC primitives should be such that the users have the flexibility to choose and specify the types and levels of reliability and correctness requirements of their applications.
Moreover, IPC primitives must also have the flexibility to permit any kind of control flow between the cooperating processes, including synchronous and asynchronous send/receive.

7) Security :-
   A good message-passing system must also be capable of providing a secure end-to-end communication. That is,a message in transit on the network should not be accessible to any user other than those to whom it is addressed and the sender.
   Steps necessary for secure communication include the following:
   - Authentication of the receiver(S) of a message by the sender
   - Authentication of the sender of a message by its receiver(S)
   - Encryption of a message before sending it over the network.

8) Portability :-
   There are two different aspects of portability in a message-passing system:
   - The message-passing system should itself be potable. That is it should be possible to easily construct a new IPC facility on another system by reusing the basic design of the existing message-passing system.
   - The application written by using the primitives of the IPC protocols of the message-passing system should be portable.

Buffering mechanism:-

Messages can be transmitted from one process to another by copying the body of the message from the address space of the sending process to the address space of the receiving process.

In some cases receiving process may not be ready to receive message transmitted to it but wants the operating system to save that message for later reception. In these cases, the operating system will rely on the receiver having a buffer in which messages can be stored prior to the receiving process executing specific code to receive the message.

In interprocess communication, the message-buffering strategy is strongly related to synchronization strategy.

Buffering strategies are:

1) Null buffer (no buffering) :
   In case of no buffering, there is no place to temporarily store the message. Hence one of the following implementation strategies may be used:
   The message remains in the senders process`s address space and execution of the send is delayed until the receiver executes the corresponding receive.
   The message is simply discarded and the timeout mechanism is used to resend the message after a timeout period. That is, after executing send the sender process waits for an acknowledgement from the receive process.

2) Single-message buffer :-

It is suitable for synchronous communication between two processes.

In this, a buffer having a capacity to store single message is used on the receiver`s node. This is because in systems based on synchronous communication, an application module may have at most one message buffer strategy to keep the message ready for use at location of the receiver.

3) Unbounded-capacity buffer :-

   In the asynchronous mode of communication, since sender does not wait for the receiver to be ready there may be several pending messages that have not yet been accepted by the receiver.

   Therefore an unbounded-capacity message buffer that can store all unrecived messages in needed to support asynchronous communication with the same assurance that all the message sent to receiver will be delivered.

4) Finite-bound (multiple-message) buffer :-

   When the buffer has finite bounds, a strategy is also needed for handling the problem of a possible buffer overflow. The buffer overflow problem can be dealt with one of the following ways:

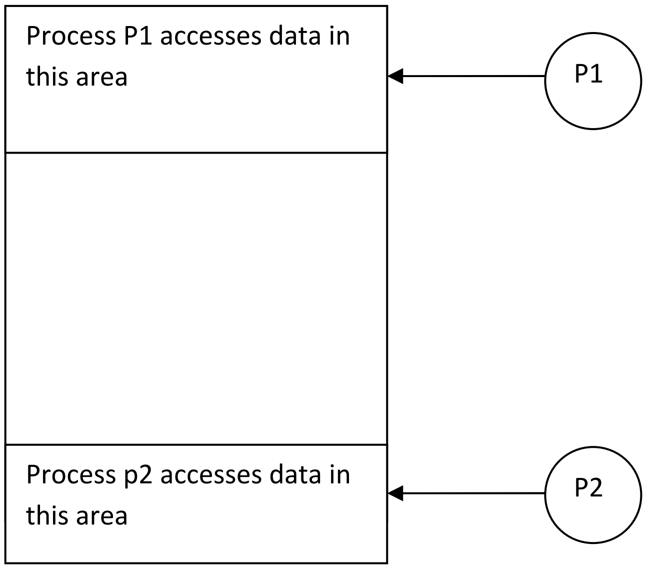
   Unsuccessful communication

   Flow-controlled communication

   Message communication based on multiple-message buffering capability provided better concurrency and flexibility.

Q 24) What is false sharing? Can it be eliminated? Give the pros and cons of using a large block size in designing DSM system.

False sharing occurs when two different processes access two unrelated variables that reside in the same data block.


Process P1 accesses data in this area
P1
Process p2 accesses data in this area
P2

A data block fig: false sharing

In such situation, even though the original variables are not shared, the data block appears to be shared by the two processes. The larger is the block size, the higher is the probability of false sharing, due to fact that same data block may contain different data structures that are used independently.

We can eliminate the false sharing by using the small data size block or page sized block.

Pros of using large block size :

1) Paging overhead:-
Paging overhead is less for large block sizes as compared to the paging overhead for small block sizes.

2) Directory size:-
Larger block sized result into reduced directory management overhead .

Cons of using large block size data:-

1) False sharing:-
When the block size is larger the probability of false sharing is always large, cause the larger data size block contain the data with different data structure.

2) Thrashing:-
In large data sized block as different regions in the same block may be updated by processes on different nodes, causing data block transfers that are not necessary with smaller block sizes.

Q 25) What is clock synchronization? Compare and contrast the various algorithms used for clock synchronization in a distributed system.

A distributed system consists of a collection of processes that are spatially separated and run concurrently. These processes must share system resources economically. Such sharing can be cooperative or competitive. One process may influence the action of other process when it competes for resources. For e.g., for a resource that cannot be used simultaneously for multiple processes, a process must wait if another process is using it.

Both, competitive and cooperative sharing requires certain rules of behavior that guarantee that correct interaction occurs. These rules are implemented in form of Synchronization Mechanism.

Clock Synchronization

Every Computer needs a timer mechanism to keep track of current time, and also for various accounting purposes such as calculating time spent by a process in CPU utilization, disk I/O, etc.

For applications running processes concurrently on multiple nodes of the system, the clocks of the nodes must be synchronized with each other.

E.g.: For a distributed Online Reservation System to be fair, the only remaining seat booked simultaneously by two different nodes should be offered to the client who booked first, even if the time difference between the two bookings is very small.

This cannot be achieved if the clocks of both the system are not synchronized.In a Distributed system, synchronized clocks also enable one to measure the duration of distributed activities that start on one node and terminate on another.

It is the job of a distributed operating system designer to devise and use suitable algorithms for properly synchronizing clocks of a distributed system.

A Distributed system requires the following types of synchronization:

1. Synchronization of the Computer clocks with real time (or external) clocks.
    a. This type of synchronization is mainly required for real time applications. That is, external clock synchronization allows the system to exchange information about the timing of events with other systems and users.
    b. An external time source that is often used as a reference for synchronizing computer clocks with the real-time is the coordinated universal time (UTC).
2. Mutual (or internal) synchronization of the clocks of different nodes of the system.
    a. This type of synchronization is mainly required for those applications that require a consistent view of time across all nodes of a distributed system as well as for the

measurement of the duration of distributed activities that terminate on a node different from the one on which they start.

Clock synchronization algorithms may be classified as

1. Centralized
2. Distributed

Centralized Algorithms
In centralized algorithms, one node has a real-time receiver, which is usually called "time server node". The goal is to keep the clocks of all other nodes synchronized with the clock time of this "time server node".
This algorithm is of two types:

a. Passive Time Server Centralized Algorithm.
b. Active Time Server Centralized Algorithm.

<u>Passive Time Server Centralized Algorithm</u>**.**

In this method, each node periodically sends a message (time=?) to the time server. When the time server receives the message, it quickly responds with a message (time=T), where T is the current time in the time server node.

<u>Active Time Server Centralized Algorithm</u>**.**

In Passive Time Server approach, the time server only responds to requests for time from other nodes. On the other hand, in the active time server approach, the time server periodically

<u>Distributed Algorithms</u> broadcasts its clock time (time=T). The other nodes receive the broadcast message and use the clock time in the message for correcting their own clocks.

<u>Drawbacks of Centralized Algorithms:</u>

1. Subject to single-point failure. If the time server node fails, the clock synchronization operation cannot be performed.

2. It is not generally acceptable to get all time requests serviced by a single time server.

Externally synchronized clocks are also internally synchronized. If each node's clock is independently synchronized with real time, all the clocks of the system remain mutually synchronized.

Following two approaches are used for internal synchronization in practice:

1. Global Averaging Distributed Algorithms

2. Localized averaging distributed algorithms

Global Averaging Distributed Algorithms

In this approach, the clock process at each node broadcasts its local clock time in the form of a special “resync” message when its local time equals $\mathbf{T_0 + iR}$ for some integer *i,* where T0 is fixed time in the past agreed upon by all the nodes and R is a system parameter.

i.e.; a resync message is broadcast from each node at the beginning of every fixed-length resynchronization interval.

Localized Averaging Distributed Algorithms

The global averaging algorithms require the network to support broadcast and also generates large amount of message traffic. Hence they are suitable only for small networks.
In localized approach, the nodes of a distributed system are logically arranged in some kind of pattern, such as a ring or a grid. Periodically, each node exchanges its clock time with its neighbors in the ring, grid, or other structure and then sets its clock time to the average of its own clock time and the clock times of its neighbors.

Q26) What is a deadlock? Explain the four necessary conditions for a deadlock to occur.

It may happen that some of the processes that entered the waiting state because the requested resources are not available at the time of request, will never again change the state because the resources they had requested are held by other waiting processes this situation is called deadlock, and the processes involved are said to be deadlocked. Hence, deadlock is the state of permanent blocking of a set of processes each of which is waiting for an event that only other process in the set can cause. All the processes in the set block permanently because all the processes are waiting and hence none of them will ever cause any of the events that could wake up any of the other members of the set.

Following conditions are necessary for a deadlock situation to occur in a system.

1. Mutual –exclusion condition: If a resource is held by a process, any other process requesting for that process must wait until the resource has been released.
2. Hold-and-wait condition: Processes are allowed to request for new resources without releasing the resources that they are currently holding.
3. No-preemption condition: A resource that has been allocated to a process becomes available for allocation to another process only after it has been voluntarily released by the process holding it.
4. Circular-wait condition: Two or more processes must form a circular chain in which each process is waiting for a resource that is held by the next member of the chain.

All four conditions must hold simultaneously in a system for a deadlock to occur. If any one of them is absent, no deadlock can occur.

Q 27) Explain process migration. Discuss the relative advantages and disadvantages of preemptive and non-preemptive process migration.

<u>Process Migration:</u>

Process Migration is the relocation of a process from its current location (the source node) to another node (the destination) .The flow of execution of a migrating process is shown in following figure.

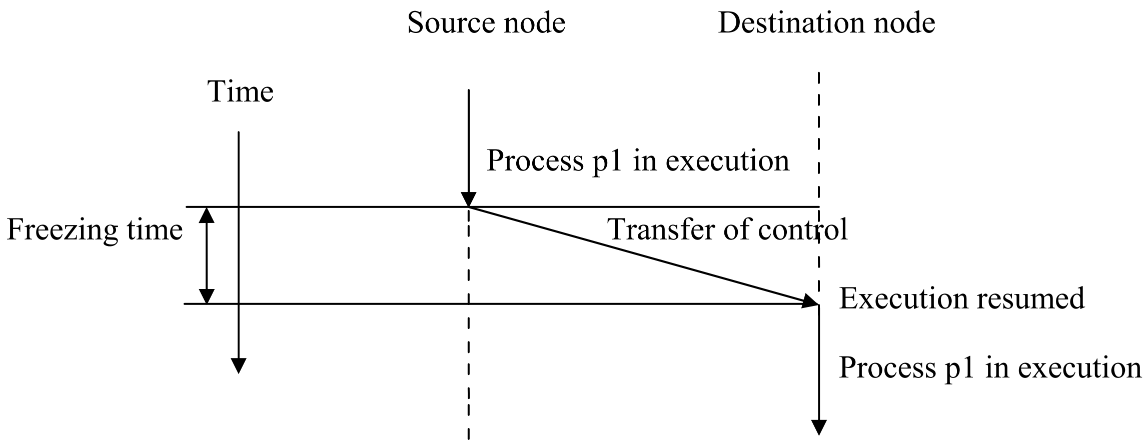


Fig. flow of execution of a migrating process

A process may be migrated either before it starts executing on its source node or during the course of its execution. The former is known as non-preemptive process migration and the latter is known as preemptive process migration. Preemptive process migration is costlier than non-preemptive process migration since the process environment must also accompany the process to its new node for an already executing process.

<u>Process migration involves the following major steps:</u>

1. Selection of a process that should be migrated

2. Selection of the destination node to which the selected process should be migrated

3. Actual transfer of the selected process to the destination node

The first two steps are taken care of by the process migration policy and the third step is taken care of by the process migration mechanism.

Process Migration mechanism

Migration of a process is complex activity that involves proper handling of several sub-activities in order to meet the requirements of a good process migration mechanism.

The four major sub-activities involved in process migration are as follows:

1. Freezing the process on its source node and restarting it on its destination node

2. Transferring the process's address space from its source code to its destination node

3. Forwarding messages meant for the migrant process

4. Handling communication between cooperating processes that have been separated (placed on different nodes) as a result of process migration.

Mechanism for freezing and restarting a process

In preemptive process migration, the usual process is to take a 'snapshot" of the process's state on its source node and reinstate the snapshot on the destination node. For this, at some point during migration, the process is frozen on its source node, its state information is transferred to its destination node, and the process is restarted on its destination node using this state information. By freezing the process, we mean that the execution of the process is suspended and all external interactions with the process are deferred. Although the freezing and restart operations differ from system to system.

Address Space Transfer Mechanisms

A process consists of the program being executed, along with the program's data, stack, and state. Thus, the migration of a process involves the transfer of the following types of information from source to the destination node:

- Process's state, which consists of the execution status (contents of registers), scheduling information, information about main memory being used by the process (memory tables), i/o states (i/o queue, contents of i/o buffers, interrupt signals, etc.), a list of objects to which the process has a right to access (capability list) and so on.

- Process address space (code, data ,and stack of the program)

  Mechanisms used are total freezing, pretransferring, or transfer on reference.

  Message Forwarding Mechanism

In moving a process, it must be ensured that all pending, en-route, and future messages arrive at the process's location. The messages to be forwarded to the migrant process's new location are classified into various types.

Mechanisms used:

- Mechanism of resending the message
- Origin site mechanism
- Link traversal mechanism
- Link update mechanism

Mechanism for handling coprocesses

In system that allows process migration, another important issue is the necessity to provide efficient communication between a process and its sub processes which might have been migrated and placed on different nodes. The two mechanisms to take care of this problem are:

- Disallowing separation of coprocesses.
- Home node or origin site concept.

The relative advantages and disadvantages of preemptive and non-preemptive process migration:

- A preemptive process migration facility allows the transfer of an executing process from one node to another. On the other hand, in system supporting only non-preemptive migration facility, a process can only be transferred prior to beginning its execution.
- Preemptive process migration is costlier than non-preemptive process migration since the process state, which must accompany the process to its new node, becomes much more complex after execution begins.

Q 28) Discuss the desirable features of a distributed file system. How are these features incorporated in NFS?

1. Purpose of files
   a. Permanent storage of information
   b. Sharing of information
2. File system is responsible for organization, storing, naming, retrieval, sharing and protection of files.
3. Distributed File System more complex as users and storage devices are physically dispersed. It supports:
   a. Remote information sharing
   b. User mobility
   c. Availability/ replication
   d. Allow use of Diskless workstations

<u>Desirable Features of DFS</u>

Transparency

a. Structure transparency: Multiplicity of file servers should be transparent to clients.

b. Access transparency: Both local and remote files should be accessible in the same way.

c. Naming transparency: Name of a file should not give hint of its location.

d. Replication transparency: Client should be unaware of both the existence of multiple copies of a file and their location.

2. User mobility
   i. Users should be able to access any file in the shared name space from any workstation. The performance characteristics of the system should not discourage users from accessing their files from workstations other than the one at which they usually work.

3. Performance
   a. Acceptable performance is hard to quantify, except in very specific circumstances. Our

   b. Goal is to provide a level of file system performance that is at least as good as that of a lightly. Loaded timesharing system at CMU. Users should never feel the need to make explicit file placement decisions to improve performance.

4. Simplicity and ease of use: The most important issue that the semantics of the distributed file system should be easy to understand.

5. Scalability
   a. It is inevitable that the system will grow with time. Such growth should not cause serious disruption of service, nor significant loss of performance to users.

6. High availability
   a. Single point network or machine failures should not affect the entire user community. We are willing, however, to accept temporary loss of service to small groups of users.

7. High reliability: In a good distributed file system, the probability of loss of stored data should be minimize as far as practicable. This is, user should not feel compelled to make backup copies of their files because of the unreliability of the system.
8. Data integrity: A file is often shared by multiple users. For a shared file, the file system must guarantee the integrity the data stored. That is, concurrent access request from multiple users who are computing to access the file must be properly synchronized by the use of some form of concurrency control mechanism.

9. Security
   Workstations are not trustworthy. The building of a distributed operating system from untrustworthy components is a much harder problem.

10. Heterogeneity
    a. It seems a more difficult proposition to span a spectrum of workstations with a distributed operating system than with a distributed file system.

Q 29 Differentiate between synchronous and asynchronous type of communication.

1. **Asynchronous Communication**
   - Advantages
     1. Doesn't require synchronization of both communication sides
     2. Cheap, timing is not as critical as for synchronous transmission, therefore hardware can be made cheaper
     3. Set-up is very fast, well suited for applications where messages are generated at irregular intervals
     4. Allows more parallelism
   - Disadvantages
     1. Large relative overhead, a high proportion of the transmitted bits are uniquely for control purposes and thus carry no useful information
     2. Not very reliable

2. **Synchronous Communication**
   - Advantages
     1. Simple & easy to implement
     2. Reliable
   - Disadvantages
     1. Limits concurrency
     2. Can lead to communication deadlock
     3. Less flexible as compared to asynchronous
     4. Hardware is more expensive

Q.30 Explain with diagrams how logical clocks are implemented with counters and physical clocks.

Timer mechanism used to keep track of current time, accounting purposes, measure duration of distributed activities that starts on one node and terminate on another node etc.

Computer Clocks

1. A quartz crystal that oscillates at fixed frequency.
2. A counter register whose value is decremented by one for every oscillation of quartz crystal.
3. A constant register to reinitialize counter register when its value becomes zero & an interrupt is generated.

Logical Clocks

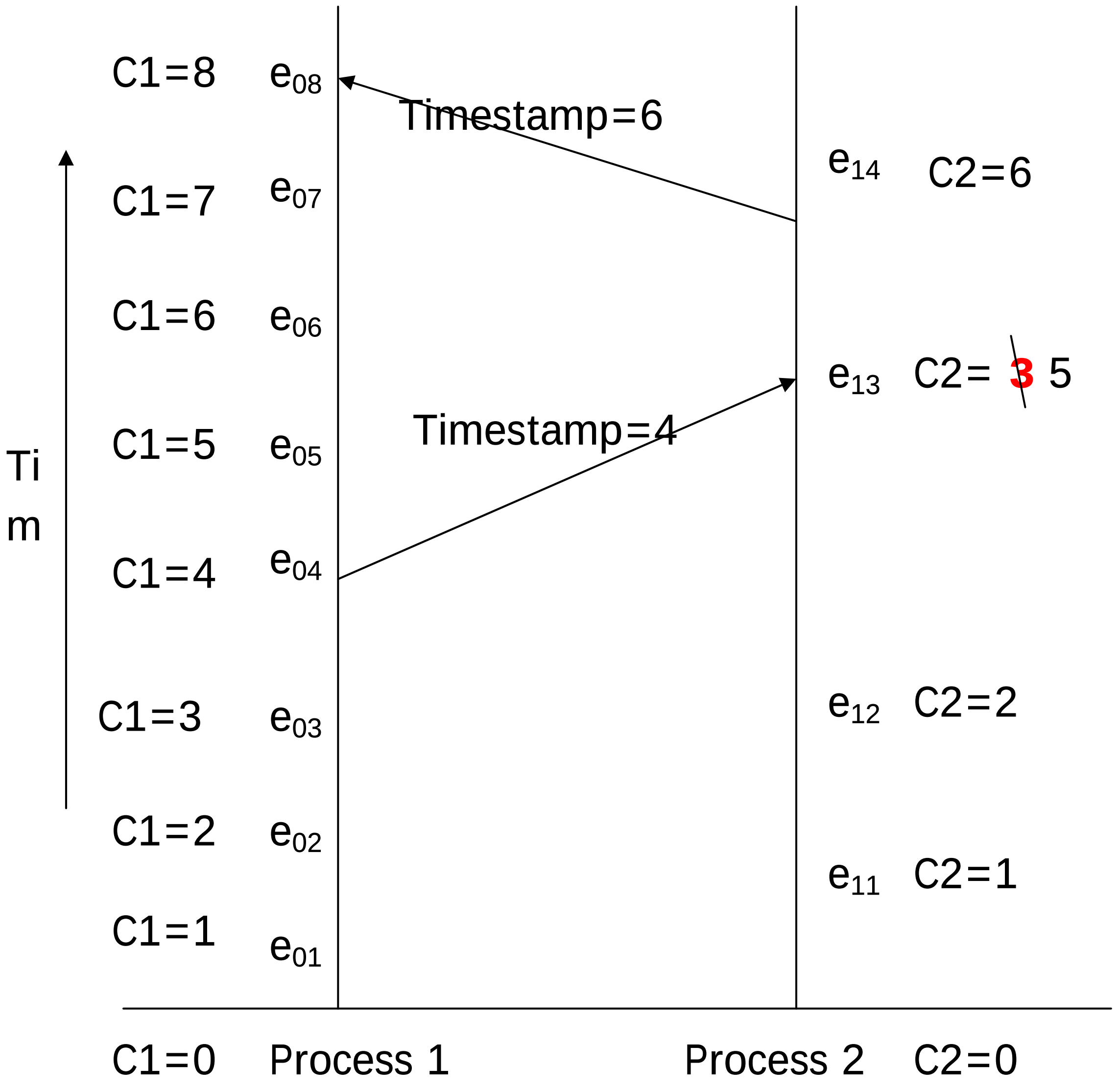
1. Need globally synchronized clocks to determine a → b
    a. Rather use Logical clock concept
        i. Associate timestamp with every event
        ii. Timestamps assigned to events by logical clocks must follow clock condition : if a → b, then C(a) < C(b)
2. Conditions to follow to satisfy clock condition
    a. If a occurs before b in process Pi then Ci(a) < Ci(b)
    b. If a is the sending of a message by process Pi and b is the receipt of that message by process Pj then Ci(a) <Cj(b)
    c. Clocks should always go forward, that is corrections should be positive value
3. To meet the above conditions : Lamport's algorithm
4. The implementation rules of lamport's algorithm
    a. IR1 : Each process Pi increments Ci between any two events

    b. IR2 : If event a is the sending of message m by process Pi the message m contains a timestamp Tm = Ci(a). Upon receiving the message m, a process Pj sets Cj greater than or equal to its present value but greater than Tm

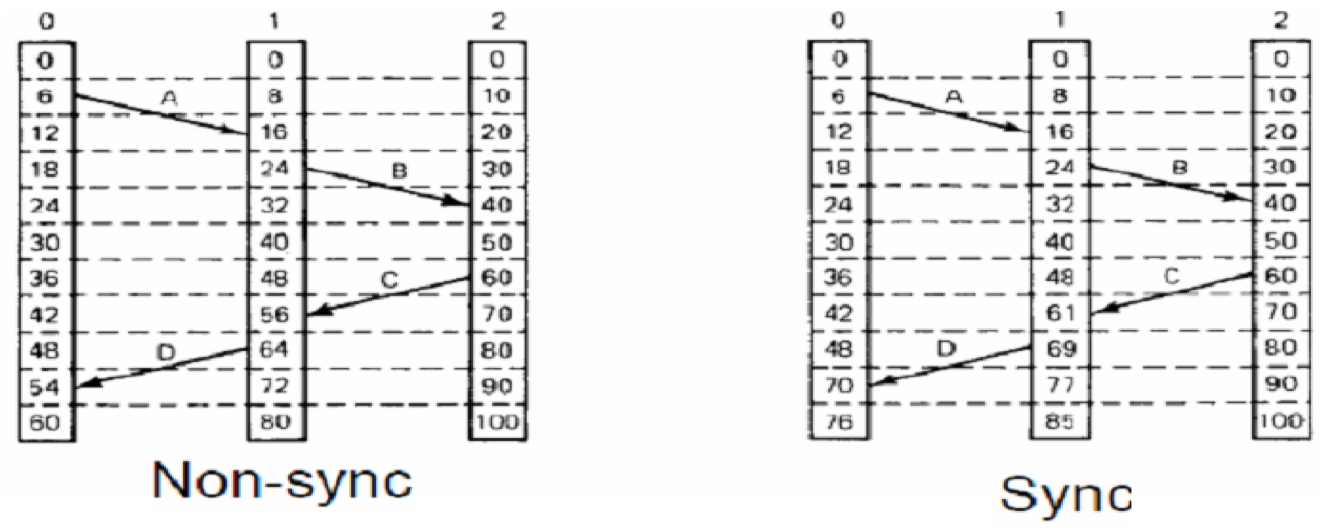
Implementation of Logical Clock using Counters

- All processors have counters acting as logical clocks.
- Counters initialized to zero & incremented by 1 whenever an event occurs in that process.
- On sending of a message, the process includes the incremented value of the counter in the message.

- On receiving the message, counter value incremented by 1 & then checked against counter value received with message.
  - If counter value is less than that of received message, the counter value set to (timestamp in the received message + 1). Ex. e13.
  - If not, the counter value is left as it is. Ex. e08



Implementation of Logical Clock using Physical Clocks



Example:

1. Three processes that run on different machines, each with its own clock, running at its own speed.
2. When the clock has ticked 6 times in process 0, it has ticked 8 times in process 1 and 10 times in process 2.
3. Each clock runs at a constant rate, but the rates are different due to differences in the crystals.
4. At time 6: process 0 sends message A to process 1.
5. The clock in process 1 reads 16 when it arrives.
6. If the message carries the starting time 6 in it, process 1 will conclude that it took 10 ticks to make the journey.
7. According to this reasoning, message B from 1 to 2 takes 16 (40 – 24) ticks, again a plausible value.
8. Message from 2 to 1 leaves at 60 and arrives at 56. Impossible!
9. Message D from 1 to leaves at 64 and arrives at 54. Impossible!

Lamport's solution:

1. Each message carries the sending time, according to the sender's clock.
2. When a message arrives and the receiver's clock shows a value prior to the time the message was sent, the receiver fast forwards its clock to be one more than the sending time.
3. Since C left at 60, it must arrive at 61 or later.
4. On the right we see that C now arrives at 61. Similarly, D arrives at 70.

Q 31) Give a mechanism for consistency ordering of messages in One-to-Many communications, Many-to-One communications, and Many-to-Many communications

Mechanism for consistency ordering of messages:

Since, Absolute- ordering semantics requires globally synchronized clocks, which are not easy to implement. Moreover, absolute ordering is not really what many applications need to function correctly.

For instance, in the replicated database updation example it is sufficient to ensure that both servers receive the update messages of the two senders in the same order even if this order is not the real order in which the two messages were sent.

Therefore, instead of supporting absolute ordering semantics, most systems support consistent-ordering semantic. This semantics ensures that all messages are delivered to all receiver process in the same order. However, this order may be different from the order in which messages were sent.

One method to implement consistent-ordering semantics is to make the many-to-many scheme appear as a combination of many-to-one and one-to-one schemes.

That is, the kernels of the sending machines send messages to a single receiver (known as *sequencer*) that assigns a sequence number to each message and then multicasts it. The kernel of each receiver's machine saves all incoming messages meant for a receiver in a separate queue are delivered immediately to the receiver unless there is a gap in the message identifiers, in which case messages after the gap are not delivered until the ones in the gap have arrived.

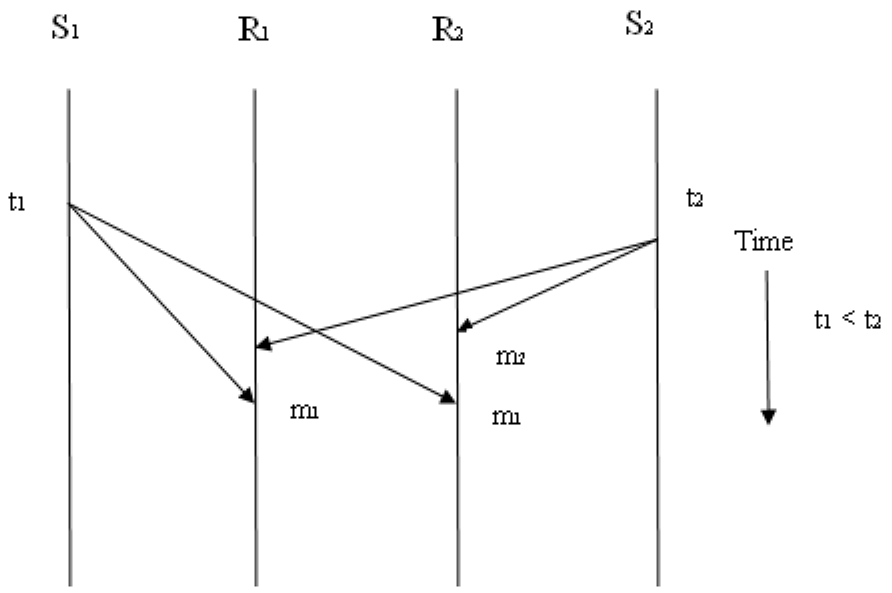


Fig. Consistent ordering of messages.

The sequencer-based method for implementing consistent-ordering semantics is subject to single point of failure and hence has poor reliability. A distributed algorithm for implementing consistent-ordering semantics that does not suffer from this problem is the ABCAST protocol of the ISIS system.

It assigns a sequence number to a message by distributed agreement among the group members and the sender and works as follows:
The sender assigns a temporary sequence number to the message and sends it to all the members of the multicast group. The sequence number assigned by the sender must be larger than any previous sequence numbers to its messages.

- On receiving the message, each member of the group returns a proposed sequence number to the sender. A member (*i*) calculates its proposed sequence numbers by using the function

  $$\max(F_{max}, P_{max}) + 1 + i/N$$

  where F_{max} is the largest final sequence number agreed upon so far for a message received by the group (each member makes a record of this when a final sequence number is agreed upon), P_{max} is the largest proposed sequence number by this member, and N is the total number of members in the multicast group.
- When the sender has received the proposal sequence number from all the members, it selects the largest one as the final sequence number for the message and sends it to all members in a commit message. The chosen final sequence number is guaranteed to be unique because of the term *i*/N in the function used for the calculation of a proposed sequence number.
- On receiving the commit message, each member attaches the final sequence number to the message.
- Committed messages with final sequence numbers are delivered to the application programs in order of their final sequence numbers. Note that the algorithm for sequence number assignment to a message is a part of the runtime system, not the user processes.

It can be shown that this protocol ensures consistent ordering semantics.

Q 32.) What is the difference between sequential consistency and release consistency? State their relative advantages. How can sequential consistency be implemented

Difference between sequential consistency and release consistency

No.	Sequential consistency	Release consistency
1	Proposed by Lamport.	Proposed by Gharachorloo.
2	A shared memory system is said to support the sequential consistency model if all processes see the same order of all memory access operations on the shared memory.	Release consistency is like weak consistency, but there are two operations "lock" and "unlock" for synchronization.
3	Example of sequentially consistent execution: P1: W(x)1 P2: R(x)0 R(x)1	Example (valid or not?): P1: L W(x)1 W(x)2 U P2: L R(x)2 U P3: R(x)1
4	The results are the same as if operations from different processors are interleaved, but operations of a single processor appear in the order specified by the program.	Release consistency provides a mechanism to clearly tell the system whether a process is entering a critical section
5	Sequential consistency is inefficient.	Release consistency is efficient.
6	Very restrictive and hence suffers from low concurrency.	Provides better concurrency and also supports the intuitively expected semantics.
7	No extra burden on the programmers to the use of synchronization variable.	They require programmers to use the synchronization variables properly which imposes burden on the programmers.

Relative advantages:-

1. Sequential Consistency Model
   - The most commonly used model in DSM systems is the sequential consistency model because it can be implemented, it supports the most intuitively expected semantics for memory coherence, and it does not impose any extra burden on the programmers.
   - Another important reason for its popularity is that a sequentially consistency DSM system allows existing multiprocessor programs to be run on multicomputer architectures without modification.
   - This is because programs written for multiprocessors normally assume that memory is sequentially consistent.

2. Release Consistency Model
   - Release consistency model, which use explicit synchronization variables, seem more promising for use in DSM design because they provide better concurrency and also support the intuitively expected semantics.

Implementing Sequential Consistency Model

Protocols for implementing the Sequential consistency model in DSM systems depend to a great extent on whether the DSM system allows replication and/or migration of shared-memory data blocks. The designer of a DSM system may choose from among the following replication and migration strategies

1. Nonreplicated, nonmigrating blocks (NRNMBs)
2. Nonreplicated, migrating blocks (NRMBs)
3. Replicated, migrating blocks (RMBs)
4. Replicated, nonmigrating blocks (RNMBs)

The protocols that may be used for each of these categories are described below.

1. Nonreplicated, nonmigrating blocks (NRNMBs)

   This is the simplest strategy for implementing a sequentially consistent DSM system. In this strategy, each block of the shared memory has a single copy whose location is always fixed. All access requests to a block from any node are sent to the owner node of the block, which has the only copy of the block. On receiving a request from a client node, the memory management unit (MMU) and OS software of the owner node perform the access request on the block and return a response to the client. (Fig. 31.1)

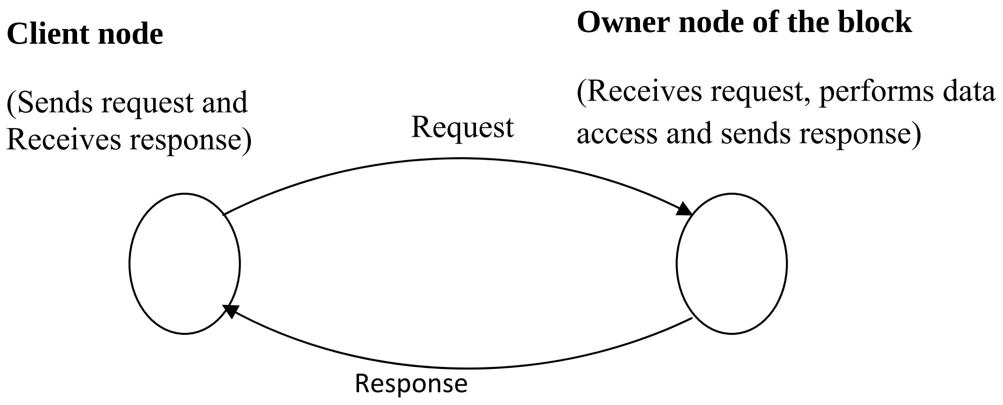

Client node
(Sends request and Receives response)
Owner node of the block
(Receives request, performs data access and sends response)
Request
Response

Fig: Non-replicated, non-migrating block (NRNMB) strategy

   Enforcing sequential consistency is trivial in this case because a node having a shared block can merely perform all the access requests (on the block) in the order it receives them.

   Although the method is simple and easy to implement, it suffers from the following drawback:

- Serializing data access creates a bottleneck.

- Parallelism, which is a major advantage of DDSM, is not possible with this method.

2. Nonreplicated, migrating blocks (NRMBs)

   In this strategy each block of the shared memory has a single copy in the entire system. However, each access to a block causes the block to migrate from its current node, to the node from where it is accessed. Therefore, unlike the previous strategy in which the owner node of a block always remains fixed, in this strategy the owner node of a block changes as soon as the block is migrated to a new node. When a block is migrated away, it is removed from any local address space it has been mapped into. In this strategy only the processes executing on one node can read or write a given data item at only one time. Therefore the method ensures sequential consistency.

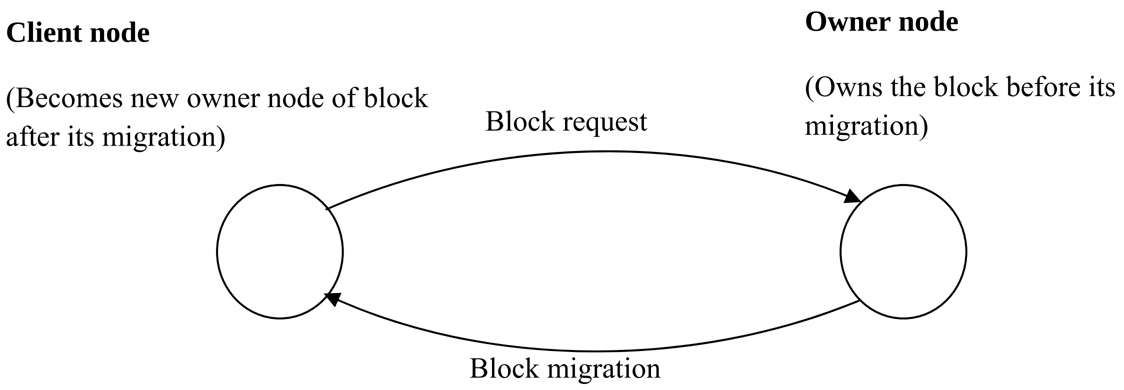

Client node
(Becomes new owner node of block after its migration)
Owner node
(Owns the block before its migration)
Block request
Block migration

Fig: Non-replicated, migrating block (NRMB) strategy

The method has the following advantages-

No communication costs are incurred when a process accesses data currently held locally. It allows the applications to take advantage of data access locally. If an application exhibits high locality of reference, the cost of data migration is amortized over multiple accesses.

However, the method suffers the following drawbacks-

- It is prone to thrashing problem. That is, a block may keep migrating frequently from one node to another, resulting in few memory accesses between migrations and thereby poor performance.
- The advantage of parallelism cannot be availed in this method also.

3. Replicated, migrating blocks (RMBs)

   A major disadvantage of the nonreplication strategies is lack of parallelism because only the processes on one node can access data contained in a block at any given time. To increase parallelism, virtually all DSM systems replicate blocks.

With replicated blocks, read operations can be carried out in parallel at multiple nodes by accessing the local copy of the data.

Therefore, the average cost of read operations is reduced because no communication overhead is involved if a replica of the data exists at the local node.

However, replication tends to increase the cost of write operations because for a write to a block all its replicas must be invalidated or updated to maintain consistency.

Nevertheless, if the read/write ratio is large, the extra expense for the write operations may be more than offset by the lower average cost of the read operations. Replication complicates the memory coherence protocol due to the requirements of keeping multiple copies of a block consistent.

The two basic protocols that may be used for ensuring sequential consistency in this case are as follows:

1. Write-invalidate

   All copies of a piece of data except one are invalidated before a write can be performed on it. When a write fault occurs at a node, its fault handler copies the accessed block from one of the block's current nodes to its own node, invalidates all other copies of the block by sending an invalidate message containing the block address to the nodes having a copy of the block, changes the access of the local copy of the block to write and returns to the faulting instruction.

   In this method, after invalidation of a block, only the node that performs the write operation on the block holds the modified version of the block.

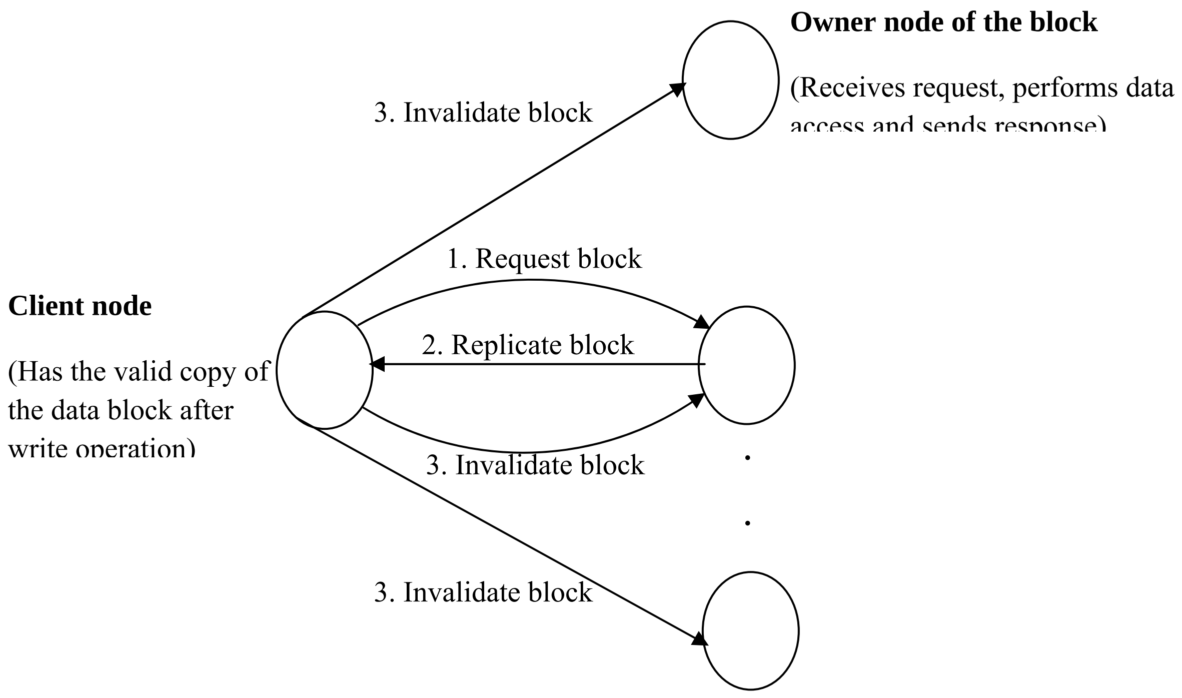


Fig: Write-invalidate memory coherence approach for replicated migrating block (RMB) strategy

2. Write-update

   A write update operation is carried out by updating all copies of the data on which the write is performed. When a write fault occurs at a node, the fault handler copies the accessed block from one of the block's current nodes to its own node, updates all copies of the block by performing the write operation on the local copy of the block and sending the address of the modified memory location and its new value to the nodes having a copy of the block and then returns to the faulting instruction.

   The write operation completes only after all the copies of the block have been successfully updated. After a write operation completes, all the nodes that had a copy of the block before the write also have a valid copy of the block after the write.

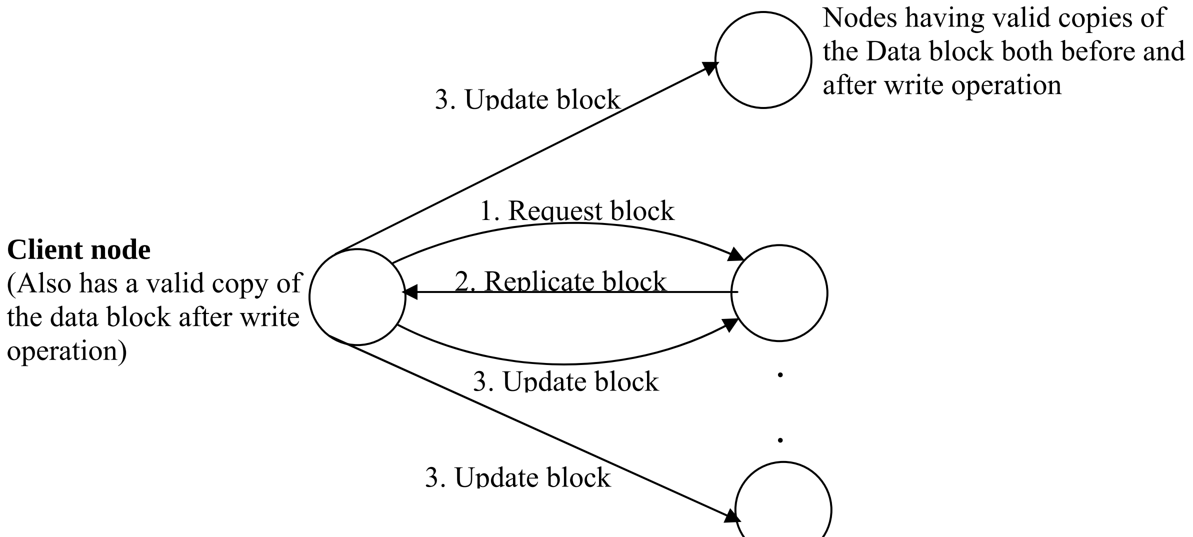

Nodes having valid copies of the Data block both before and after write operation
3. Update block
1. Request block
Client node
(Also has a valid copy of the data block after write operation)
2. Replicate block
3. Update block
3. Update block

Fig.: Write-update memory coherence approach for replicated migrating block (RMB) strategy

4. Replicated, nonmigrating blocks (RNMBs)

   In this strategy, a shared-memory block may be replicated at multiple nodes of the system, but the location of each replica is fixed. A read or write access to a memory address is carried out by sending the access request to one of the nodes having a replica of the block containing the memory address.

   All replicas of a block are kept consistent by updating them all in case of a write access. Sequential consistency is ensured by using a global sequencer to sequence the write operations of all nodes.

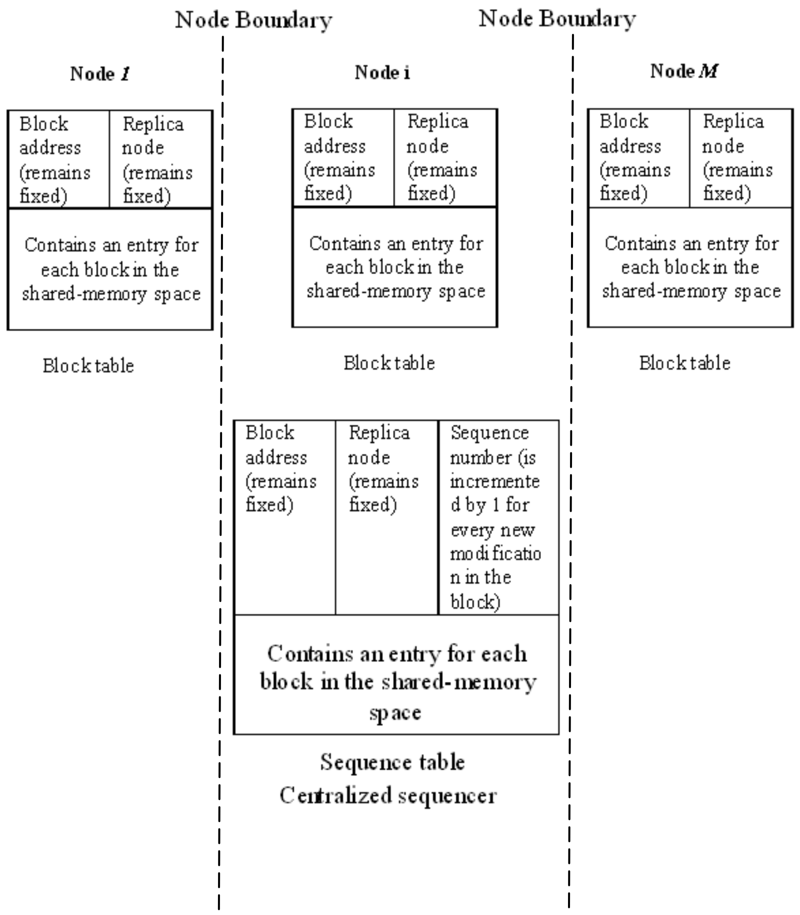
   The RNBM strategy has the following characteristics:

   - The replica locations of a block never change.
   - All replicas of a data block are kept consistent.
   - Only a read request can be directly sent to one of the nodes having a replica of the block containing the memory address on which the read

request is performed and all write requests have to be first sent to the sequencer.

Based on these characteristics, the best approach of data locating for handling read/write operations in this case is to have a block table at each node and a sequence table with sequencer (Fig).

The block table of a node has an entry for each block in the shared memory. Each entry maps a block in the shared-memory space. Each entry of the sequence table has three fields – a field containing the block address, a replicaset field containing a list of nodes having a replica of the block, and a sequence number field that is incremented by 1 for every new modification performed on the block.



Node Boundary
Node Boundary
Node 1
Node i
Node M
Block address (remains fixed)
Replica node (remains fixed)
Contains an entry for each block in the shared-memory space
Block table
Block address (remains fixed)
Replica node (remains fixed)
Sequence number (is incremented by 1 for every new modification in the block)
Contains an entry for each block in the shared-memory space
Sequence table
Centralized sequencer

Fig. Structure and locations of block table and sequence table in the centralized sequencer data-locating mechanism for RNMB strategy.

Q 33) What is an idempotent operation? Which of the following operations are idempotent? If not, give semantics to make them idempotent:-

i. **Read_next_record(filename)**
ii. **Read_record(filename, rec_name)**
iii. **T=time(x)**
iv. **B=mean(a1,a,a3)**
v. **Sqrt(144)**

Idempotent operation :

- Idempotency basically means "repeatability".
- An idempotent operation produces the same results without any side effects no matter how many times it is performed with the same arguments.
- That is whenever this operation is performed we must get the same result if we have used the same arguments.
- On the other hand , operations that do not necessarily produce the same results when executed repeatedly with the same arguments are said to be nonidempotent.
- An example of idempotent operation is a simple *GetSum* procedure.
  For example , if we execute this operation as *GetSum(10,20)* then after execution it will give the answer as 30 which is the sum of 10 and 20. No matter how many times this operation is performed , we are going to get the same result as 30 if both arguments are same i.e. 10 and 20.
- For non-idempotent operation let us consider the following routine of a server process that debits a specific amount from a bank account and returns the balance amount to a requesting client:

```
debit(amount)
if(balance >=  amount)
        (balance = balance – amount ;
        return ("success",balance);)
else    return("failure",balance) ;
end ;
```

- If the first request asks the server to debit an amount of 100 from the balance. Server receives request and processes it. Suppose initial balance was 1000 , so the server sends a reply ("success",900) to client. For some reason if this reply could not delivered to the client , then after some timeout period it retransmits the request . The server processes same request again and send a reply ("success",800) , which is not correct. So we have seen that multiple executions of non-idempotent routine produce undesirable results.

i. Read_next_record(filename)

- This operation is not idempotent operation.
- When we execute it for first time as Read_next_record("test.txt") it will give the result as first record from test file. If we execute it again it will give the next record from test file.
- The number of time we execute this operation it gives different results for same filename.
- To make it idempotent:
    - First read the old pointer position.
    - Change the current pointer position to old value and then execute the operation.

ii. Read_record(filename, rec_name)

- This is idempotent operation.
- Whenever we execute this routine with same arguments we get the same record whose name is mentioned in the arguments.

iii. T=time(x)

- This is an idempotent operation.
- Since the time for x i.e. when x is used or generated will be same always hence it will give the same result for multiple executions.

iv. B=mean(a1,a,a3)

- This is idempotent operation.
- Suppose we execute the operation with arguments 15,10 and 20. We will get the result in B as 15.

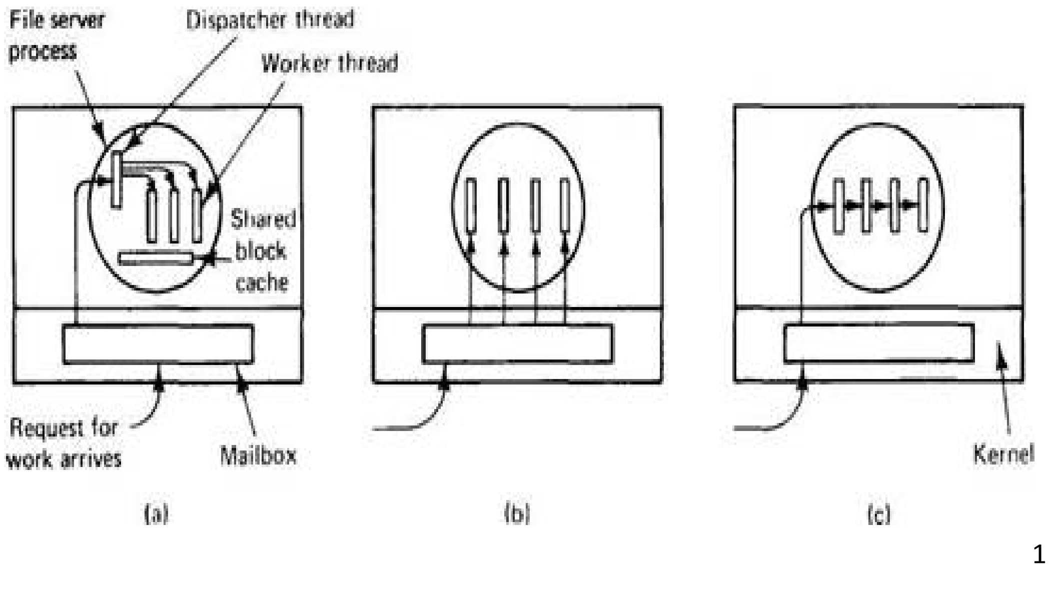
v. Sqrt(144)

- Sqrt is an idempotent operation.
- Number of times it is executed it gives the same result as 12.

Q 34) Differentiate between process and thread concepts. Give suitable examples for each one of the following:-

- vi. **A process using multiple threads that are organized in the dispatcher-worker model.**
- vii. **A process using multiple threads in a team model.**
- viii. **A process using multiple threads in a pipeline model.**

Process	**Thread**
1. In traditional operating systems the basic unit of CPU utilization is a process.	1. In operating systems with threads facility, the basic unit of CPU utilization is a thread.
2. Each process has its own program counter, its own register states, its own stack and its own address space.	2. Each thread of a process has its own program counter, its own register states, and its own stack. But all the threads of a process share the same address space.
3. Protection between processes is needed because different processes may belong to different users.	3. Due to sharing of address space, there is no protection between the threads of a process.
4. Traditional processes are referred to as heavyweight processes.	4. Threads are referred to as lightweight processes.



i. **A process using multiple threads that are organized in the dispatcher-worker model:**

In this model, the process consists of a single dispatcher thread and multiple worker threads. The dispatcher thread accepts requests from clients and, after examining the request, dispatches the request to one of the free worker threads for further processing of the request. Each worker thread works on a different client request. Therefore multiple client requests can be processed in parallel. (refer fig. (a))

ii. **A process using multiple threads in a team model:**

In this model, all threads behave as equals in the sense that there is no dispatcher worker relationship for processing clients' requests. Each thread gets and processes clients' request on its own. This model is often used for implementing specialized threads within a process. That is, each thread of the process is specialized in servicing a specific type of request. Therefore, multiple types of requests can be simultaneously handled by the process. (refer fig. (b))

iii. **A process using multiple threads in a pipeline model:**

This model is useful for applications based on the producer-consumer model, in which the output data generated by one part of the application is used as input for another part of the application. In this model, the threads of a process organized as a pipeline so that the output data generated by the first thread is used for processing by the second thread, the output of the second thread is used for processing by the third thread, and so on. The output of the last thread in the pipeline is the final output of the process to which the threads belong.
(refer fig. (c))

Q35). Why are election algorithms required? Describe Bully algorithm in detail along with suitable diagrams. Is it more efficient than the ring algorithm?

Election algorithms are meant for electing a coordinator process from among the currently running processes in such a manner that at any instance of time there is a single coordinator for all processes in the system.Therefore, whenever initiated, an election algorithm basically finds out which of the currently active processes has the highest priority number and then informs this to all other active processes.

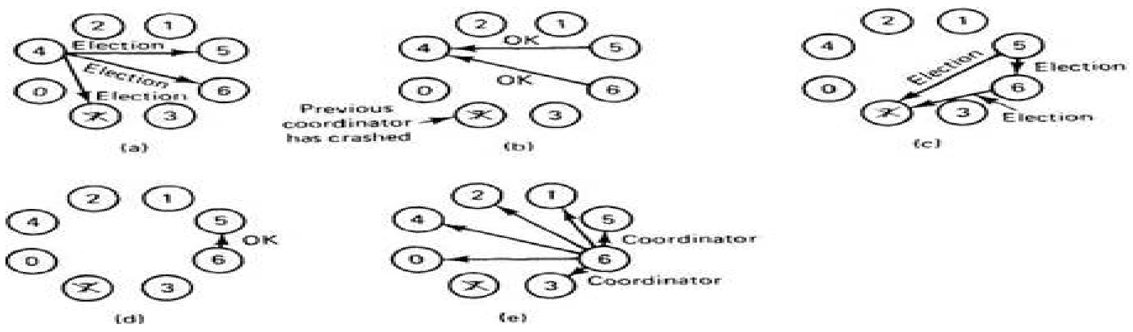
Bully Algorithm:This algorithm was proposed by Garcia-Molina. When the process notices that the coordinator is no longer responding to requests, it initiates an election. A process, P, holds an election as follows:

1. P sends an ELECTION message to all processes with higher numbers.
2. If no one responds, P wins the election and becomes the coordinator.
3. If one of the higher-ups answers, it takes over. P’s job is done.

At any moment a process can get an ELECTION message from one of its lower numbered colleagues. 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. The receiver then holds an election, unless it is already holding one. Eventually, all processes give up but one, and that one is the new coordinator. It announces its victory by sending all processes a message telling them that starting immediately it is the new coordinator.

If a process that was previously down comes back up, it holds an election. If it happens to the highest numbered process currently running, it will win the election and take over the coordinator’s job. Thus the biggest guy in town always wins, hence the name “bully algorithm”.

In the figure we see an example of how the bully algorithm works. The group consists of eight processes, numbered from 0 to 7. Previously process 7 was the coordinator, but it has just crashed. Process 4 is the first one to notice this. So it sends ELECTION messages to all the processes higher than it, namely 5, 6 and 7, as shown in fig.(a). Processes 5 and 6 both respond with OK, as shown in fig.(b). Upon getting the first of these responses, 4 knows that its job is over. It knows that one of these will take over and become the coordinator. It just sits back and waits to see who the winner will be.



In fig.(c), both 5 and 6 hold elections, each one only sending messages to those processes higher than itself. In fig.(d), process 6 tells 5 that it will take over. At this point 6 knows that 7 is dead and that (6) it is the winner. It there is state information to be collected from disk or elsewhere to pick up where the old coordinator left off, 6 must now do what is needed. When it is ready to take over, 6 announces this by sending a COORDINATOR message to all running processes. When 4 gets this message, it can now continue with the operation it was trying to do when it discovered that 7 was dead, but using 6 as the coordinator this time. In this way the failure of is handled and the work can continue.

If process 7 is ever restarted, it will just send all the others a COORDINATOR message and bully them into submission.

The bully algorithm is not efficient than ring algorithm because of the following two reasons:1) In the bully algorithm, when the process having the lowest priority number detects the coordinator's failure and initiates an election, in a system having total n processes, altogether n-2 elections are performed one after another for the initiated one. That is, all the processes, except the active process with the highest priority number and the coordinator process that has just failed, perform elections by sending messages to all processes with higher priority numbers. Hence, in the worst case, the bully algorithm requires $O(n^2)$ messages. However, when the process having the priority number just below the failed coordinator detects that the coordinator has failed, it immediately elects itself as the coordinator and sends n-2 coordinator messages in the best case.

On the other hand in the ring algorithm, irrespective of which process detects the failure of the coordinator and initiates an election, an election always requires 2(n-1) messages.

1)In the bully algorithm, a failed process must initiate an election on recovery. Therefore, once again depending on the priority number of the process that initiates the recovery action, the bully algorithm requires $O(n^2)$ messages in the worst case, and n-1 messages in the best case.

On the other hand, in the ring algorithm, a failed process does not initiate an election on recovery but simply searches for the current coordinator. Hence, the ring algorithm requires only n/2 messages on an average for recovery action.

Q 36) 1….Enumerate the various server creation semantics involved in RPC?

In RPC, the remote procedure to be executed as a result of a remote procedure call made by a client process lies in a server process that is totally independent of the client process. The client and the server process have separate lifetimes, run on different machines and have their own address space. Based on the time duration for which RPC servers survive, they may be classified as:

1. Instance-per-call servers
2. Instance-per-transaction/session servers
3. Persistent servers

Instance-per-call servers:

Servers belonging to this category exist only for the duration of a single call. A server of this type is created by the RPC runtime on the server machine only when a call message arrives. This server is deleted after the call has been executed.

This approach is not commonly used because:

1. The servers are stateless because they are killed as soon as they have serviced the call for which they were created. Therefore, any state that has to be preserved across server calls must be taken care of either-

   Supporting operating system- making RPC expensive.

   Client process- leading to loss of data abstraction across client-server interface.
2. When a distributed application needs to successively invoke the same type of server several times, this approach appears more expensive, since resource allocation and deallocation has to be done many times.

Instance-per-session servers:

Servers belonging to this category exist for the entire session for which a client and a server interact. Since a server of this type exists for the entire session, it can maintain intercall state information, and the overhead involved in server creation and destruction for a client-server session that involves a large number of calls is also minimized.

Normally, there is a server manager, registered with the binding agent, for each type of service. Client specifies the type of service to the binding agent which returns address of the service manager to the client. Server manager spawns a new server and passes its address to the requesting client. Client directly interacts with the server for the entire session and informs the server manager of the corresponding type when it no longer needs that server.

A server of this type can retain useful state information between calls and so can present a cleaner, more abstract interface to its clients.

Persistent servers:

A persistent server generally remains in existence indefinitely. A persistent server is usually shared by many clients.

Servers of this type are usually created and installed before the clients that use them. Each server independently exports its service by registering itself with the binding agent. On client request for a particular type of service, the binding agent selects a server of that type arbitrarily or based on some in-built policy and returns the address of the selected server to the client. Client then directly interacts with the server.

Persistent servers may be used for improving the overall performance and reliability of the system.

2….List the main differences and similarities between threads and processes.

Differences:

1. A process has its own address space. All threads of a process share the same address space. They also share global variables and same set of operating system resources
2. Processes are referred to as heavyweight processes. Threads are referred to as lightweight processes.
3. Protection between processes is needed because different processes belong to different users. Protection between threads of a process is not necessary.
4. Overheads involved in creating a new process are greater than creating a new thread.
5. Switching between threads is comparatively cheaper than switching between processes.
6. Resource sharing can be achieved more efficiently and naturally between threads than processes
7. Threads help in improving performance through parallelism.

Similarities:

1. Like process, threads share CPU and only one thread active (running) at a time.
2. Like process, threads within a process execute sequentially.
3. Like process, thread can create children.
4. Like process, if one thread is blocked, another thread can run.
5. Like process, threads have their own program counter, register states, stack.

Q.37)

1…. How do you make a distributed system transparent to a user? Name different types Of Transparencies?

One of the main goals of distributed operating system is to make the existence of multiple computers invisible (transparent) and provide a single system image to its user.

That is, a distributed operating system must be designed in such a way that a collection of distinct machines connected by a communication subsystem appears to its user as a virtual uniprocessor.

Achieving complete transparency is a difficult task and requires that several different aspects of transparency be supported by the distributed operating system. Eight forms of transparency identified by [ISO 1992] are as follows:

1. **Access Transparency :**

   Access transparency means that users should not need or be able to recognize whether a resource (hardware or software) is remote or local. This implies that the distributed operating system should allow users to access remote resources in the same way as local resources. That is, the user interface which takes the form of a set of system calls, should not distinguished between local and remote resources, and it should be the responsibility of the distributed operating system to locate the resource and to arrange for servicing user request in a user-transparent manner.

   This requirement calls for a well-designed set of system calls that are meaningful in both centralized and distributed environment and a global resource naming facility. Due to need to handle communication failures in distributed systems, it is not possible to design system calls that provide complete access transparency. The distributed shared memory mechanism is also quite useful in providing access transparency; however it is suitable only for limited types of distributed applications due to its performance limitation.

2. **Location Transparency :**

   The two main aspect of location transparency are as follows:

   A. **Name Transparency :**

      This refers to the fact that the name of a resource (hardware and software) should not reveal any hint as to the physical location of the resource. That is, the name of a resource should be independent of the physical connectivity or topology of the system or the current location of the resource. Furthermore, such resources, which are capable of being moved from one node to another in a distributed system (such as file), must be allowed to move without having their names changed. Therefore, resources names must be unique system wide.

B. User Mobility :

This refers to the fact that no matter which machine a user is logged onto, he or she should be able to access a resource with the same name. That is, the user should not be required to use different names to access the same resource from two different nodes of the system. In a distributed system that supports user mobility, user can freely log on to any machine in the system and access any resources without making any extra effort.

Both name transparency and user mobility requirements calls for a system wide, global resource naming facility.

3. Replication Transparency :

For better performance and reliability, almost all distributed operating system have the provision to create replicas (additional copies) of files and other resources on different nodes of the distributed system. In these systems, both the existence of multiple copies of a replicated resource and the replication activity should be transparent to users. That is, two important issues related to replication transparency are naming of replicas and replication control. It is responsibility of system to name the various copies of a resource and to map a user-supplied name of the resource to an appropriate replica of the resource. Furthermore, replication control decisions such as how many copies of the resource should be created, where should each copy be placed, and when should a copy be created/deleted should be made entirely automatically by the system in a user-transparent manner.

4. Failure Transparency :

Failure transparency deals with masking from the user's partial failures in the system. A distributed operating system having failure transparency property will continue to function, perhaps in degraded form, in the face of partial failures.

Complete failure transparency is not achievable with the current state of the art distributed operating systems because all types of failures cannot be handled in a user-transparent manner. Moreover, an attempt to design a completely failure-transparent distributed system will result in a very slow and highly expensive system due to the large amount of redundancy required for tolerating all types of failures. The design of such a distributed system, although theoretically possible, is not practically justified.

5. Migration Transparency :

For better performance, reliability and security reasons, an object that is capable of being moved (such as a process or a file) is often migrated from one node to another in a distributed system. The aim of migration transparency is to ensure that the movement of the object is handled automatically by the system in a user-transparent manner. Three important issues in achieving this goal are as follows:

a) Migration decisions such as which object is to be moved from where to where should be made automatically by the system.

b) Migration of an object from one node to another should not require any change in its name.

c) When the migrating object is a process, the inter process communication mechanism should ensure that a message sent to the migrating process reaches it without the need for the sender process to resend it if the receiver process moves to another node before the message is received.

6. **Concurrency Transparency :**

In a distributed system, multiple users who are spatially separated use the system concurrently. In such a situation, it is economical to share the system resources among the concurrently executing user processes. However, since the number of available resources in a computing system is restricted, one user process must necessarily influence the action of other concurrently executing user processes, as it competes for resources.

For providing concurrency transparency, the resource sharing mechanism of the distributed operating system must have following four properties:

I. An event ordering property ensures that all access requires to various system resources are properly ordered to provide a consistent view to all users of the system.

II. A mutual exclusion property ensures that at any time at most one process accesses a shared resource, which must not be used simultaneously by the multiple processes if program operation is to be correct.

III. A no starvation property ensures that if every process that is granted a resource, which must not be used simultaneously by multiple processes, eventually releases it, every request for that resource is eventually granted.

IV. A no deadlock property ensures that a situation will never occur in which competing processes prevent their mutual progress even though no single one request more resource than available in the system.

7. **Performance Transparency :**

The aim of performance transparency is to allow the system to be automatically reconfigured to improve performance, as loads very dynamically in the system. As far as practicable, a situation in which one processor of the system is overloaded with jobs while another processor is idle should not be allowed to occur. That is, the processing capabilities of the system should be uniformly distributed among the currently available jobs in the system.

This requirement calls for the support of intelligent resource allocation and process migration facilities in distributed operating system.

8. **Scaling Transparency :**

The aim of scaling transparency is to allow the system to expand in scale without disrupting the activities of the users. This requirement calls for open-system architecture and the use of scalable algorithm for designing the distributed operating system components.

2….Differentiate between stateful and stateless servers. Explain two situations where stateless servers are useful. When is a stateless server required ?

Stateful Server	**Stateless Server**
1) The stateful approach does depend on the history of the serviced requests.	1) The stateless approach does not depend on it
2) A stateful server keeps state between connections.	2) A stateless server does not keep state between connections.
3) So, when you send a request to a stateful server, it may create some kind of connection object that tracks what information you request. When you send another request, that request operates on the state from the previous request. So you can send a request to "open" something. And then you can send a request to "close" it later. In-between the two requests, that thing is "open" on the server.	3) When you send a request to a stateless server, it does not create any objects that track information regarding your requests. If you "open" something on the server, the server retains no information at all that you have something open. A "close" operation would make no sense, since there would be nothing to close.
4) Stateful server is harder to code.	4) Stateless server is straightforward to code.
5) SMB is a stateful protocol. A client can open a file on the server, and the server may deny other client's access to that file until the client closes it.	5) HTTP and NFS are stateless protocols. Each request stands on its own.
6) A stateful server loses all its volatile state in a crash.	6) With stateless server, the effects of server failure and recovery are almost unnoticeable. A newly reincarnated server can respond to a self-contained

	request without any difficulty.
7) Stateful information can reduce the data exchanged (from losses), and thereby affecting the response time.	7) Penalties for using the robust stateless service: => Longer request messages. => Slower request processing. => Additional constraints imposed on DFS design - operations need to be idempotent (can be executed several times without problems) because of retransmissions.

When is a stateless server required?

The use of stateless servers in many distributed application is justified by the fact that stateless servers have a distinct advantage over stateful servers in the event of failure. For example, with stateful servers, if a server crashes and then restarts, the state information that it was holding may be lost and the client process might continue its task unaware of the crash, producing inconsistent result. Similarly, when a client process crashes and then restarts its task, the server is left holding state information that is no longer valid but cannot easily be withdrawn. Therefore, the client of a stateful server must be properly designed to detect server crashes so that it can perform necessary error handling activities. On the other hand, with stateless server, a client has to only retry a request until the server responds; it does not need to know that the server has crashed or that the network temporarily went down. Therefore, stateless servers, which can be constructed around repeatable operations, make crash recovery very easy.

Q 38.) Differentiate between monolithic kernel and microkernel approaches for designing Distributed operating system.

The most important design factor that influences the flexibility of a distributed operating system is the model used for designing its kernel. The kernel of an operating system is its central controlling part that provides basic system facilities.

Monolithic Kernel	Micro Kernel
1. The services provided by kernel are: process management, memory management, device management, file management, name management and inter-process communication.	1. The services provided are: inter-process communication, low-level device management, limited amount of low-level process and some memory management.
2. Kernel is larger	2. Kernel is smaller
3. Larger kernel size reduces overall flexibility and configurability of the resulting operating system.	3. The micro-kernel is highly modular in nature, due to is small sized kernel.
4. Comparatively difficult to design, implement and install.	4. Easy to design, implement and install.
5. Since it is difficult to implement, it also becomes difficult to modify the design.	5. Since most services are implemented as user-level server processes, it is easy to modify the design or add new services.
6. The system needs to be stopped and booted again every time a new service is added or changed.	6. There is no need to stop and boot the system, every time a new service is added or changed.
7. Server processes share the same address space.	7. Each server process has its own independent address space.
8. No message passing or context switching are required while the kernel is performing a job.	8. Message passing, context switching is required for inter-process communication
9. Requests are serviced faster in this model as serves processes share the same address space.	9. Requests are serviced slower as compared to monolithic kernel due to the additional performance overhead caused by inter-process communication methods (message passing etc.)

Q 39) Difference between PRAM consistency model and processor consistency model

PRAM consistency model	processor consistency model
• Proposed by Lipton and sandberg	proposed by Goudman
• Ensures that all write operations Performed by a single process are seen By all other processes in the order in which They were performed.	ensures that all write operations performed on the Same memory location(no matter by which process they are performed)
• Consistence semantics is stronger Than processor consistency model.	consistency semantics is weaker than PRAM
• All processes do not agree on the Same order of memory reference Operations.	all the processes agree on the same order of memory reference operations (memory coherence)

Q40). What are advantages of Distributed computing systems

Distributed computing systems are more complex and difficult to build than traditional centralized systems. However despite the increased complexity and difficulty in building distributed computing systems their use and installations are rapidly increasing because their advantages outweigh their disadvantages.

THE MAJOR ADVANTAGES OF DISTRIBUTED COMPUTING SYSTEMS ARE:-

1. INHERENTLY DISTRIBUTED APPLICATIONS

Distributed computing systems come into existence in some very natural ways .For example several applications are inherently distributed in nature and require a distributed computing system for their realization. For instance in an employee database of a nationwide organization the data pertaining to a particular employee are generated at employees branch office in addition to the global need to view the entire database, there is local need for frequent and immediate access to locally generated data at each branch office .Applications such as these require some processing power to be available at many distributed locations for collecting, processing, accessing data for which distributed computing systems are needed. Other example of an inherently distributed application is worldwide airline reservation system. Computerized banking system etc.

2. INFORMATION SHARING AMONG DISTRIBUTED USERS

Another reason for emergence of distributed computing systems was a desire for efficient person-to-person communication facility by sharing information over great distances. In dcs information generated by one of the users can be easily and efficiently shared by users working at other nodes of the system.

This facility is useful in many ways. For example a project can be performed by two or more users who are geographically far from each other but whose computers are part of distributed computing system.thus in this case they can co-operate with each other even if they are away from each other. For example by transferring files of the project, logging onto each others remote computers to run programs etc. The use of distributed computing system by a group of users to work co operatively is known as computer-supported cooperative working (cscw) or groupware . Groupware is an emerging technology that holds major promise for software developers.

3. RESOURCE SHARING

Information is not the only thing that can be shared in dcs. Sharing of software resources such as software libraries and databases as well as hardware resources such as printers , hard disks and plotters can also be done in a very effective way among all the computers and users of distributed computing system. For example in a dcs based on workstation-server model the workstations may have no disk or only a small disk for temporary storage, and access to permanent files on a large disk can be provided to all the workstations by a single file server.

4. BETTER PRICE-PERFORMANCE RATIO

This is one of the most important advantages of distributed computing system. With the rapidly increasing power and reduction in price of microprocessors, combined with increasing speed of communication networks, distributed computing systems potentially have better price-performance ratio than a single large centralized system. For example how a small no of CPUs in a dcs based on processor –pool model can be effectively be used by large no of users from inexpensive terminals thus giving a high price-performance ratio as compared to centralized system. Also dcs facilitate resource sharing among multiple users.

5. SHORTER RESPONSE TIMES AND HIGHER THROUGHPUT

Due to multiplicity of processors,distributed computing systems are expected to have better performance than centralized systems. Two of the most common performance metrics are response time and throughput of user procceses.ie the multiple proccesors of dcs can be utilized properly for providing shorter response time and higher throughput. Distributed computing systems with very fast communication networks are increasingly being used as parallel computers to solve single complex problems rapidly. Another method used to increase performance in dcs is to distribute load evenly among the multiple processors by movging jobs from currently overloaded processors to lightly loaded ones.

6. HIGHER RELIABILITY

A reliable system prevents loss of information even in the event of component failures. The multiplicity of storage devices and proccesors in dcs allows the maintainence of multiple copies of critical information within the system and execution of important computations redundantly to protect them against catastrophic failures. In comparison to centralized system , a distributed system also enjoys the advantage of increased availability.

The advantage of higher reliability comes at cost of performance. Therefore it is necessary to maintain a balance between the two.

7. EXTENSIBILITY AND INCREMENTAL GROWTH

Another major advantage of distributed computing systems is that they are capable of incremental growth. That is it is possible to gradually extend the power and functionality of a distributed system by simply adding additional resources. Extensibility is also easier in distributed computing systems because addition of new resources to an existing system can be performed without significant disruption of normal functioning of system. Properly designed distributed computing systems that have property of extensibility and incremental growth are called open distributed systems.

8. BETTER FLEXIBILITY IN MEETING USER'S NEEDS

Different types of computers are usually more suitable for performing different types of computations. For example computers with ordinary performance are suitable for ordinary data processing jobs while high performance computers for complex mathematical computations. In a centralized system the users have to perform all the computations on the available computer only.however a distributed computing system may have a pool of different types of computers , in which case the most appropriate one can be selected for processing users job depending upon nature of job.