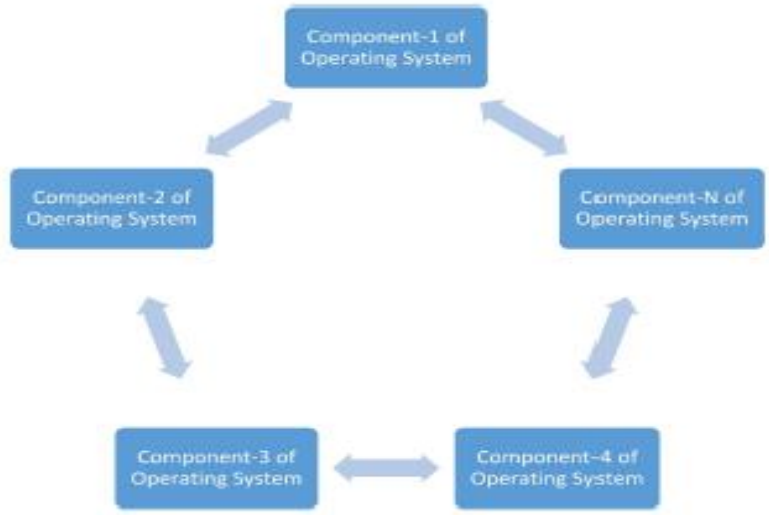
DISTRIBUTED OPERATING SYSTEM

INTRODUCTION TO DISTRIBUTED OPERATING SYSTEM:

The distributed operating system consists of different computers which are connected with a network within a virtual shell. A user interacts with the virtual shell in order to perform different tasks as and when needs arises but the distributed operating system architecture delegates the responsibility of performing a task to one or more computers within the virtual shell.



View of Distributed Operating System

The dotted circle (boundary) in the picture gives an idea about the user's view of distributed operating system and the internal details are not visible to a user. However, the developer's view of the distributed operating system will be component available within the dotted circle. The Figure shows the availability of components from 1 up to N which can vary from one scenario to other.

The components within a distributed operating system can be computers of different operating system which are located at different sites for execution. All the components are connected using a strong local area network as the same plays a vital role in execution of a task within a distributed architecture.

The operating system generally provides different functions like process management, input/ output management, memory management, file organization, security & protection and network management. The distributed operating system has different components which are

responsible from different functions of an operating system and all these components are connected with a network.

Whenever a user submits a command to a distributed operating system, the instruction may require the services of one or more than one components of an operating system. The user gets a feel as if the whole system is a single unit but internally the whole system consists of sub systems which work in tandem in order to achieve the centralized objective of a distributed operating system.

EVOLUTION OF DISTRIBUTED OPERATING SYSTEM

The computers were having very less computing power in the early days of computing when the computers were introduced. The pace of execution along with processing power was increasing many-fold with the advances in technology.

Nowadays, minicomputers were used to process different volumes of data. With the advancement in technology and increase in computer speed along with the processing power of a processor, more complex processing along with huge volumes of data was performed on personal computers.

However, data analysis where the volume of data was mammoth was performed on main-frame computers. The main-frame computers were not appropriate options for processing of different task in some case which led researchers in exploring option to find different alternatives and one of the alternatives provided by the research community was distributed operating system architecture. The distributed architecture was about delegating the responsibility among one or more computers connected together within a network.

This feature of distributed architecture allowed an end user to get higher speed and better processing power. The architecture helped researchers to increase the processing power to manifold by integrating the resources of different independent computers as a single virtual computer. The distributed architecture is not only about hardware rather it is amalgamation of hardware utilization along with appropriate software which help the system to perform with efficiency and effectiveness.

The distributed architecture was further enhanced with the advent of RPC remote procedure calls. The RPCs gained prominence due to the availability of TCP/IP protocol which is known as Transmission Control Protocol/ Internet Protocol. The RPCs helps an individual to execute different commands and instructions on any computer remotely provided the computer is part of a network which a user has permissions to access. The feature has led to more prominence and relevance to the need of distributed operating system architectures in providing higher processing power to an end-user.

Initially the architectural framework was designed to perform batch processing which was later on improved with the inception of minicomputers. The mini computers enabled the researchers to incorporate the concept of master/slave where a computer was nominated as master and other connected nodes were treated as slaves. The complete execution of a process from inception till maturity was monitored by master and all the slaves will execute the orders of the master as per the requirement of a process or task.

As the size of computers was reduced dramatically and with the increase in the processing power of microcomputers the processing within a distributed operating system become more easy, effective and efficient. Nowadays the networks that allow user to connect to internet has reached newer heights in terms of performance, scalability and security which has resulted in leveraging from the advancements in networks by incorporating the high end servers to help in distributed computing with the help of strong, reliable and secure networks.

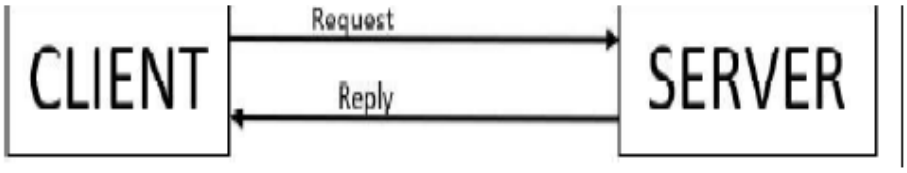
MODELS OF DISTRIBUTED OPERATING SYSTEM

The models in distributed operating system can also be termed as architectural models. These models give a brief idea about the connection of different components within a distributing operating system of distributed computing. The information about the architectures helps the researchers in devising policies and procedures to make these systems more scalable, robust, effective and efficient. The different architectural models of distributed operating system are listed below:

1. Interaction Model
2. Failure Model
3. Security Model

Interaction model

In the interaction model, the issues related to the interaction of process are included like timing of events. An example of an interaction model can be a client-server architecture where a client submits a request to the server and the server responds with a reply.



In a distributed operating system environment, all the clients need to interact with the server in order to process a request. The server in turn manages and monitors the processing within all the components of a distributed operating system connected with a network. However, the communication between the client and the server result in different issues which need to be addressed by the developers appropriately.

The interaction model can be categorized into two main categories which are given below:

(i) Synchronous Model
(ii) Asynchronous Model

The Failure Model

Failure Model uses the specifications of different faults that occur during the processing of a process and while communicating between the different components of a process. The failure model handles different situation where failures do occur provided the system is able to identify and detect a failure like Omission failure,Byzantine (Arbitrary) Failure and timing failures etc.

The Security Model

The Security Model uses the specifications of different threats that occur during the processing of a process and while using the communication channels which play a pivotal part in execution of any process in a distributed operating system.

Security threats in a distributed operating system are generally attacks that are intended to break the communication channel or to make a communication process unsuccessful which will result in failure of a distributed operating system.

Some of the commonly occurred threats are list below:

1. Unauthorized connection to network
2. Identity Threat
3. Denial of Service
4. Modification/ Deletion of message in a communication channel
5. Trojan Horse
6. Virus
7. Worm
8. Server overloading

In order to make any system secure the recommendation is always to use pro-active approach which is similar to the approach of "Prevention is better than cure" where the impact of threats can be mitigated by properly monitoring the processing of tasks and by taking preventive measuring for mitigating the impact of threats.

ISSUES IN DESIGNING DISTRIBUTED COMPUTING SYSTEM

The design issues in a distributed system help engineers in developing the distributed system effectively. The various design issues in the development of distributed systems are stated as follows:

1. Transparency
2. Flexibility
3. Reliability
4. Performance
5. Scalability

Transparency

A distributed system is said to be transparent if the users of the system feel that the collection of machines is a timesharing system and belongs entirely to him.

Transparency can be achieved at two different levels.

In the first level, the distribution of the system is hidden from the users: for example, in UNIX, when a user compiles his program using the make command, compilation takes place in parallel on different machines, which use different file systems. The whole system can be made to look like a single-processor system.

In the other level, the system is made to look transparent to the programs. The system call interface can be designed in such a way that the existence of multiple processors can be hidden. This process is more difficult than the first process.

There are various types of transparency in the distributed system, which are stated as follows:

Location transparency:

It implies that in a true distributed system, the user cannot tell the location of hardware and software resources, such as CPU, printer, database and files: for example, if a user wants to change his address in the database, then he can update the database with his new address. But while doing so, he does not need to know where the database is stored.

Migration transparency:

It implies that the resources should move from one location to another without changing their names: for example, consider that there is a file with the hierarchy fun/games. An user decides that playing games is fun and changes the file from fun/games to games/fun. Then if some other client will boot the system, he will not see the file as fun/games but as games/fun.

Replication transparency:

It means that the OS in a distributed system has the authorization to create additional copies of files and resources without involving the user. It means that the user does not know about the number of the copies of the files that exist: for example, consider a collection of servers, which are logically connected to each other to form a ring.

Each server maintains the directory tree structure of files. If a client sends a request to read a file to any of the servers, it will reply only if it contains the file; otherwise, it forwards the request to the next server. The next server repeats the same process. In replication transparency, servers can make multiple copies of files, which are heavily used.

Concurrency transparency:

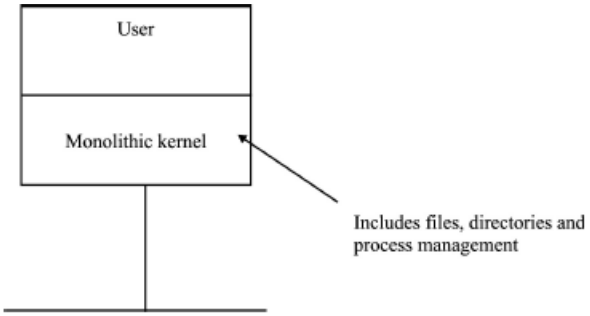
It suggests that multiple users can use the resources automatically. The problem arises when two or more users try to access the same resource or update the same file concurrently. In such a situation, the system locks the resource or the file if someone else starts accessing it. The lock is released only after the user has finished accessing the resource or the file.

Parallelism transparency:

It refers to the parallel execution of activities without the user's knowledge: for example, if the user wants to evaluate the boards of the chess program, multiple situations have to be evaluated in parallel. After evaluating the results, they all send the result to the system and the user can see the result on the screen.

Flexibility

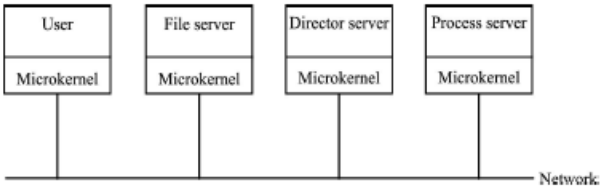
Flexibility in distributed systems is important because this system is new for engineers, and thus there may be false starts and it might be required to backtrack the system. The design issues might prove wrong in the later stages of development. There are two different schemes for building distributed systems. The first one, called monolithic kernel, states that each machine should run a traditional kernel, which provides most of the services itself.



Monolithic Kernel

Monolithic kernel is the centralized OS, which has networking ability and remote services. The system calls are made by locking the kernel, then the desired task is performed and the kernel is released after returning the result to the user. In this approach, the machines have their own disks and maintain their own local file system.

The other one, called microkernel, states that the kernel should provide very little services and most of the OS services should be provided from the user level servers.



Microkernel

Most of the distributed systems are designed to use microkernel because it performs very few tasks. As a result, these systems are more flexible. The services provided by the microkernel are stated as follows

- It provides inter process communication.
- It manages the memory.
- It performs low-level process management and scheduling.
- It also performs low-level I/O.

These services are provided by the microkernel because it is expensive for the user-level servers to provide these services. However, it does not provide the file system, directory system, process management and system calls. If the user wants to search for a name, he sends the request to the server, which returns the result after searching.

The advantages of this system are stated as follows:

- It is easy to install, implement and debug.
- There is a well-defined interface for each service.
- Each service is equally accessible to all the clients.
- It is more flexible as users have the ability to add, delete and modify the services because it does not involve stopping or booting the kernel.
- The users can write their own services.

The microkernel is more powerful as compared to the monolithic kernel because there can be multiple file servers in the distributed system in which one supports MS-DOS and the other one uses the UNIX file system. Individual users can decide to use either of them or both in the microkernel, whereas in the monolithic kernel, the users do not have any choice.

The disadvantage of using the microkernel is its poor performance. The monolithic kernel is faster as it locks the kernel to perform the task rather than sending the request to the server to perform the task as in the case of the microkernel.

Reliability

Distributed systems are more reliable than single-processor systems because if one system in a distributed system stops functioning, other systems can take over. Some other machine can take up the job of the machine that is not working. There are various issues related to reliability, which are stated as follows:

Availability:

It refers to the fraction of time during which the system is usable. The availability of a system can be ensured with the design, which does not require simultaneous use of key components. In other words, the components, which are required very often, should not be used concurrently.

The resources or files, which are used frequently, can be replicated. If any one of them is occupied by one process or fails, the other processes do not have to wait indefinitely. The data, which is required frequently, can be stored on multiple servers for quick access. But it must be ensured that all the copies are consistent with each other. If one file is updated, all the other copies should also be updated.

Security:

It implies that anyone can access the data stored on a distributed system. As a result, it should be protected from unauthorized access. This problem also persists in single-processor systems. But in single-processor systems, the users are required to log in, and thus they are authenticated and the system can check the permission of the user. But in a distributed system, the system has no provision for determining the user and his permission. Anyone can send the request for any file or data in the distributed system. As a result, the issue of security has to be kept in mind while designing the distributed systems.

Fault tolerance:

It refers to masking of failure of one of the servers from the users. Suppose while processing some task, one of the servers crashes and quickly reboots. Then, the user will not be able to know what has happened and the data will be lost. Distributed systems should be designed in such a way that even if one of the servers crashes, another server takes up the job and continues evaluating it. If there is cooperation between the servers, the user will not be able to know that the server has crashed and his work will also be completed, though with a little degradation in the performance.

Performance:

Building a system, which is flexible and reliable, is of no use if the system slower than a single-processor system. To measure the performance of the system, various performance metrics are used, such as number of jobs per hour, system utilization and amount of network capacity consumed. The result of any standard used to measure the performance of the system is dependent on the nature of the standard. A standard, which involves a large number of CPU-related computations, will give results different from that of which involves scanning and searching a file for some pattern.

In a distributed system, performance is measured by the communication between two users. Sending a message from one system to another and getting a reply does not take much time. Time is wasted in waiting for the protocols, which are used to handle the messages. To increase the performance of the system, it is required to run as many activities as possible at a time. But this requires sending a large number of messages. To solve this problem, it is important to analyse the grain size of all the computations. First of all, small computations, such as addition of two numbers, has to be performed and then complex computations should be performed.

This is because large complex computations require more CPU cycles as compared to simple computations. There are some processes in distributed systems, which require simple computations but involve high interaction with one another. These processes are said to have fine-grained parallelism. On the other hand, processes, which involve large computations and require low interaction with one another, are said to exhibit coarse-grained parallelism.Processes, which have coarse grained parallelism, are preferred over processes having fine-grained parallelism.

Better reliability can be achieved when the servers cooperate on a single request: for example, when the request arrives at the server, it can send a copy of the message to the other server for reliability. If the first server crashes before completing the task, the other server can continue the task. When the task is completed, the server has to inform the first server that the task has been completed. This provides more reliability to the system but involves extra messages across the network, which does not produce any output.

Scalability

Distributed systems are designed to work with a few hundred CPUs. There may be situations in future when the systems are much bigger than the systems, which are presently used. As a result, the solution, which works well, may not work well for the system containing very large number of CPUs: for example, consider that the postal, telephone and telegraph administration decides to install a terminal in every house and business in its area.

The terminal will allow the people to access the online database containing all the telephone numbers in the area. Thus, there is no need for printing and distributing telephone directory. When all the terminals are placed, they can also be used for e-mails. Using this system, the users can access all the databases and services, such as electronic banking and ticket reservation.

There are certain bottlenecks in developing such a large system, which are stated as follows:

Centralized components:

There should not be any centralized components in the systems: for example, if there is a single centralized mail server for all the users of the system, the traffic over the network will increase. The system will not be able to tolerate faults and also if any one of the systems fails, the whole system will crash.

Centralized tables:

If the data of the users is stored in the centralized tables, the communication lines will be blocked. Thus, the system will become prone to faults and failures.

Centralized algorithms:

If the messages in such a large system are sent using a single algorithm, it will take much time to reach the destination due to the large number of users and traffic. In such a large system, only decentralized algorithms should be used.

The characteristics of decentralized systems are stated as follows:

- No machine has the complete information about the system.
- The decisions made by the machines are based on local information.
- Failure of any one system does not affect the overall system.

INTRODUCTION TO MESSAGE PASSING

FEATURES

The characteristics of any communication in a distributed system should ensure the effectiveness, efficiency and correctness along with optimal usage of resources required for execution of a process from inception till maturity. Keeping in mind the importance of inter-process communication in distributed operation system, following are the basic characteristics that a good inter-process communication system should have.

(i) Ease of Use:

The approach used for inter-process communication should be simple and easy to use for different routines or processes of a distributed operating system. A programmer or developer should be able to write the programmes in order to communicate between two or more process in a distributed operating system and these interfaces written should be easy to understand in order to provide ease-of-use feature to a distributed operating system. The ease-of-use feature should not become an obstacle in the performance and efficiency of a distributed operating system. The simplicity of the inter-process communication should allow developers to develop programs for sharing of messages and the internal communication should be handled by the inter-process communication routines.

(ii) Effective Communication:

The inter-process communication system should be effective, efficient and correct while passing messages among two or more computers in a distributing operating system. The message to be communicated should not get modified or lost while communicating from one computer system to the other. It should ensure that the message communicated among any two computer system is complete & correct along with the optimal usage of resources. Any communication between two or more processes is the backbone for a

distributed computing environment therefore any delay that occurs in inter-process communication will result in decrease in efficiency of a distributed operating system.

Therefore, the effective communication is a pivotal feature which a distributed operating system should have in order to complete any process requested by a user. The inter-process communication should be designed with the objective to reduce the over-heads which are required to be managed while establishing a connection between any two nodes within a network of computers.

(iii) Reliable Communication:

The inter-process communication system should use a reliable mode of communication between any two nodes within a distributed system. For example, if a computer system or a node is lost due to some technical error should not result in making the complete system non-interactive because this will result in failure of the whole distributed operating system. Therefore, the inter-process communication system should handle the situation in such a fashion that the operations of a distributed operating system do not get affected.

(iv) Uniformity in Communication:

The inter-process communication system should have uniformity in different message passing mechanisms within a distributed operating system. This feature will help developers and users of the operating system to develop routines to interact within the sub-nodes or computers available in a distributed operating system. This characteristic will facilitate the ease-of-use feature in any inter-process communication system. The communication within a distributed operating system can be either local or remote. In local the communication between any two process occurs at the node locally however, in case of remote communication any two process can communicated remotely provided the processes are connected to a network which helps the process in getting connected remotely.

(v) Flexible & Portable Communication:

The inter-process communication should be able to accommodate new changes in order to provide hassle free connectivity within the nodes of a distributed operating system. The network of nodes available should have the flexibility feature imbibed in order to adapt to the new developments or changes. Any change or adaption of new technological developments should not hamper the process of communication between two or more nodes. The feature of flexibility will facilitate the process of making the inter-process communication portable as well.

The inter-process communication system will be portable when the system has been designed to ensure the adherence of ease-of-use, flexibility and uniformity characteristic within a distributed operating system. The portable feature in inter-process communication will allow a message passing mechanism to be used in any other environment where the working of the message passing mechanism will not stop rather it will continue to work effectively and efficiently. This feature will help the developers as well as users to re-use the message passing mechanisms in different distributed computing environments.

(vi) Secure Communication:

The message communicated between any two nodes within a network of computers of a distributed operating system can be modified. These types of threats may lead to breakdown of the communication between any two or more nodes. Therefore, a secure communication system must ensure that the message from a sender to a received should not be viewed or modified by any one during the process of communication within a distributed operating system.

(vii) Complete and Correct data:

The message transferred from sender to receiver node may get lost or manipulated while transferring the message. It works as a motivation for people to develop mechanisms which will ensure that the message or data transferred from the sender to the receiver is correct and complete. In other words, it can be said that the data transferred from the sender to receiver must adhere to the basic properties and the same are listed below:

a. Atomicity: When a message is transferred from a sender to a receiver either it should reach the receiver as a complete message or it should not deliver any message to the receiver which explains the concept of atomicity in inter-process communication. Therefore, the distributed operating system needs to have the feature of ensuring the atomicity in message transfer which will result in correct and complete transfer of message.

b. Consistency: The message transferred from a sender to a receiver is complete and accurate. In case any changes are made at the sender node accordingly all the parameters dependent on the modification are appropriately modified in order to accommodate the change. The modifications made at the sender node should also be delivered at the received node with all the changes that were applied at the sender node.

c. Isolation: In case more than one message is passed from a sender to different receiver nodes, the distributed computing message passing procedures should ensure that all the appropriate messages destined for the nodes are delivered correctly at the destined node only. The message or data transferred from a sender should not be delivered at a wrong node which will lead to reduction in performance and reliability of a message passing mechanism in a distributed operating system.

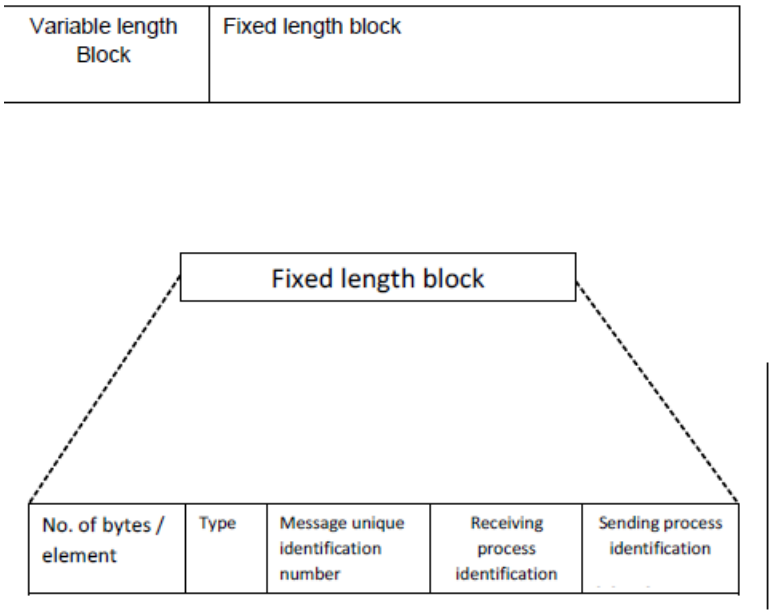
While transferring more than one message or data packet from a sender to a receiver the sequence of the messages or data packets is also very important while reframing the messages or data packets transferred from a sender to a receiver. Therefore, a message passing mechanism should ensure that the messages are delivered at the receiver node in the same sequence as was marked by the sender node while sending the message or a data packet. This feature will ensure the completeness of the message at the receiver node as per the format and sequence that has been prepared by the sender node.

d. Durability: While transferring message from a sender to a receiver the acknowledgement and reply to message is also very important in order to complete the process of communication between any two nodes. In case the process of message transfer is not complete between any two nodes that resources occupied by these nodes will not be released which will increase the probability of reaching a deadlock state. Therefore, the inter-process communication in a distributed operating system should have a feature to ensure that the process of communication between any two nodes should be monitored for any errors. If the process of communication has not generated any errors then the message transfer process should be completed and the resources acquired by the processes should be released accordingly. However, in case the message passing process has not been completed then appropriate measures need to be taken to ensure that the resources should not remain in acquired state which will result in reduction of performance of whole system.

ISSUES IN MESSAGE PASSING

The design of an inter-process communication needs to address some basic issues while designing communication process procedures used for communication between any two or more nodes. The messages are generally transferred from sender to receiver in the form of data packets. The sender node will send the data packet which will include of two basic components fixed length header and variable length block. The fixed length header includes different information related to sender process address, receiving process address, message unique identification number, type of data, number of bytes/ element.

The information available in the fixed length header helps a data packet to reach the correct destination and given correct information to the receiver about the originating process as the same is used for sending the acknowledgement from the receiver. Once the process of receiving a data packet is complete then an acknowledgement is sent to the originating process in order to complete the transfer of message between a sender and a receiver.



Parameters Available in "Fixed Length Block" of a Data Packet

The list of the basic challenges that are encountered by a distributed operating system during inter process communication are given below:

(i) Naming and Name Resolution:

Every process in a communication system is assigned a unique identification number know as Process-ID (process identification number). The computer network system should have a naming system which allows a process to names in order to resolve any conflicts or in order to manage the process execution in a distributed operating system or inter-process communication. The implementation of a naming system can be implemented either using distributed or non-distributed approach. The method of selection have a direct impact on the effectiveness and efficiency of a distributed operating system.

(ii) Routing Strategies:

The main activity in an inter-process communication is how a data packet will be sent from sender to receiver. Which path the data packet should select in order to reach from source to destination. The data packet may be required to pass through different nodes or computers in order to reach the destination node. Needless to mention the message will be decrypted or viewed only at the destination node not at any of the nodes it passes through in order to reach the destination. The path a data packet selects from the source node to the destination node is known as route. The methods and mechanisms used in an inter-process communication for identifying a route from source to destination is known as routing strategy. The prime concern of any inter-process communication must be to select a routing strategy which will be effective, efficient, secure and should use the resources optimally.

The commonly used routing strategies are given below:

a. Fixed routing
b. Virtual Circuit
c. Dynamic Routing

(iii) Connection Strategies:

The back bone of any communication between a sender and receiver is the physical link or a physical connection. The method or technique used to establish a physical connection between a sender node and a receiver node is known as connection strategy. If a connection strategy is not selected appropriately will lead to different issues like delay in communication, loss of message in communication or modification of a message during the process of communication. The different connection strategies used in distributed operating systems that need to be selected based on the requirements of an operating system are given below:

a. Circuit Switching;
b. Message Switching;
c. Packet Switching

The message within an inter-process communication system is send from the source node to the destination node in a format which includes information about the different attributes like Address, Sequence number, structural information and data within a block. The address parameter corresponds of sender address and received address and the structural information corresponds of information about the type and size of information. The actual data will be available at the end of the block and in some cases the last element of the block will carry pointer to the actual data.

SYNCHRONIZATION

The basic building blocks of a distributed operating system are cooperation, exchange of data between different nodes of a distributed operating system. In order to exchange data between any two nodes, the computer system at the destination should accept the data that has been sent by a sender which is possible only when synchronization between the sender and receiver is implemented. The process of synchronization gives the details of the timing between sender and receiver which helps the sender and receiver to communicate.

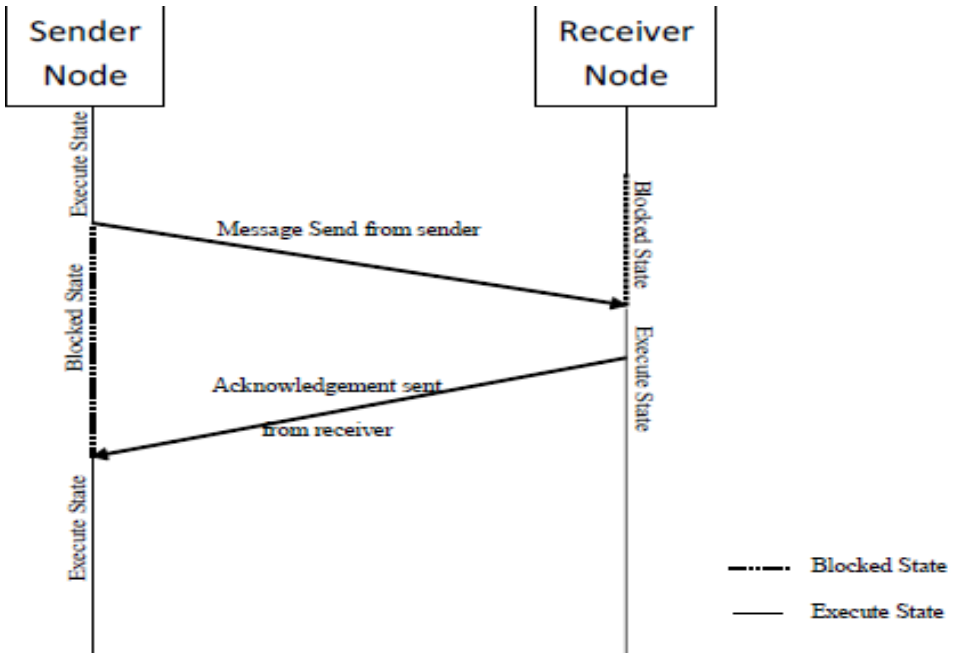
The distributed operating system is fully dependent on the communication between different independent computers in a distributed system network which increases the priority and importance of synchronization in distributed operating systems. The process of synchronization can be categorized in two basic categories like blocking and non-blocking synchronization.

A communication process is said to be complete only when the message is transferred to the receiver in its original form. The message which is transferred in the form of packets can be send using any one of the two methods of synchronization where if the sender node is blocked after sending the message to the receiver and remains in the block state till the receiver acknowledges the receipt of the message is known as blocking type of synchronization. However, if the sender is not blocked after sending the data packet from sender node rather the control is immediately transferred to the sender node is known as non-blocking type of synchronization.

Both blocking and non-blocking type of synchronization are discussed in detail in the following section

(a) Blocking: In every communication process a message is passed from a sender to a receiver. In the blocking synchronization method the sender after sending the message will wait for the acknowledgement from the receiver and during the wait period the sender will be blocked till acknowledgement is received. Similarly, the receiver after sending the acknowledgement will wait for a message from the sender in order to proceed further. The process of message communication from a sender to a receiver is shown graphically

In the figure given below where the dotted lines represent the block state and the solid line represents the execute state. The sender node state changes from the solid line to a dotted line when a message is send from a sender node and the sender node is blocked. The receiver node needs to send the acknowledgement on receipt of the message which results in unblocking of the sender node.



The sender or the receiver node is blocked while the message is being transmitted from sender to receiver and in some cases may result in permanent blocking of either sender or receiver. In order to avoid any situation of permanent blocking of sender or a receiver a timeout is fixed either at the system level or at the process execution level.

The timeout is used to fix a time limit for keeping a sender in a blocking stage while waiting for the acknowledgement. In case a sender does not receive any acknowledgement the sender will be clocked till it will reach the timeout limit. Once the timeout limit is exceeded the sender will be unblocked and the system can proceed with further execution of other process which will result in mitigation of reaching a deadlock state. If both the sender and receiver are using the blocking method of synchronization the method of communication is called as synchronous communication.

The synchronous method of communication is better the asynchronous communication method as it is simple to implement and the method ensures that the message has reached the receiver from sender. However, the synchronous communication method may lead to different state which can result in a deadlock state where the whole system will not be able to respond.

(b) Non-Blocking:

In the non-blocking synchronization method the sender after sending the message is not blocked and the sender will not wait for the acknowledgement from the receiver in order to proceed further with execution. The sender is allowed to go ahead once the message from the sender is moved to buffer. In an inter-process communication process a message is sent from a sender to a receiver via a buffer. A receiver will proceed with other process executions after executing the receive statement. A receiver in non-blocking will not wait for the message from the sender in order to proceed further with other process executions. If both the sender and receiver are using the non- blocking method of synchronization the method of communication is called as asynchronous communication.

In the non- blocking synchronization how and when will a receiver node know that the message is available in the buffer? A message may be lost in the buffer and receiver will not receive the message or a receiver has to continuously check the buffer for any messages which are destined for the receiver node.

A method of continuously checking the buffer status for any messages by the receiver is followed in non-blocking synchronization method and is known as **polling**.

Another method to intimate the receiver about the availability of a message in a buffer is known as **Interrupt method**.

In the interrupt method an interrupt is generated using software once the message is available in the buffer and the same is sent to the receiver. The receiver will receive the interrupt and will start with the process of retrieving the message from the buffer. However, the interrupt method requires for efforts and resources in order to implement the synchronization and at times it becomes tough to implement the same using software in distributed operating systems.

In both the blocking and non-blocking methods of synchronization some challenges are to be addressed in order to make the system more robust, effective and efficient along with utilizing the resources optimally. However it is always recommended to use a perfect blend of both blocking and non-blocking methods to complete the communication process between two nodes in an inter-process communication system.

BUFFERING

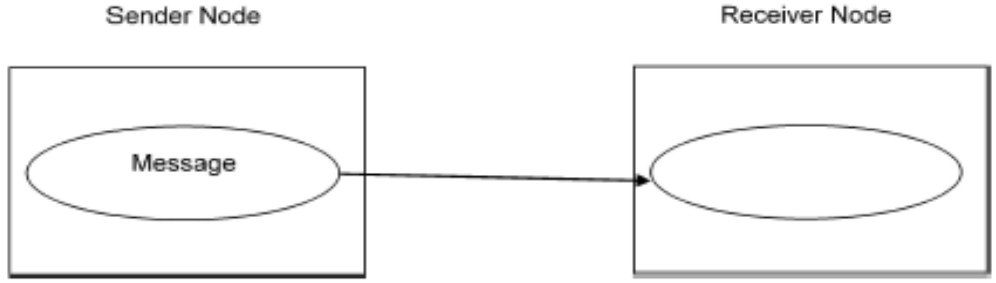
The message that is sent from a sender to receiver will either use synchronous or asynchronous communication method. However, when a message is sent from a sender it needs to be temporarily stored in a memory area until the receiver node receives the message. The message can be stored in a memory area that is available at the sender node or at the memory area which is managed by the operating system.

This memory area which is used to store the message till the receiver node receives it is known as buffer and the process is known as buffering.

Different types of buffering are used based on the requirement of a process within a distributed operating system and some of them are given below:

1. Null Buffering:

   This type of buffering doesn't use any buffer rather the send process remains in suspended mode till the receiver node in a position to receive the message. Once the process of send message starts the receiver starts the receiving the message and accordingly an acknowledgement is sent once the message is delivered. The sender node on receipt of acknowledgement sends a message to the received in order to unblock the receiver node for further processing

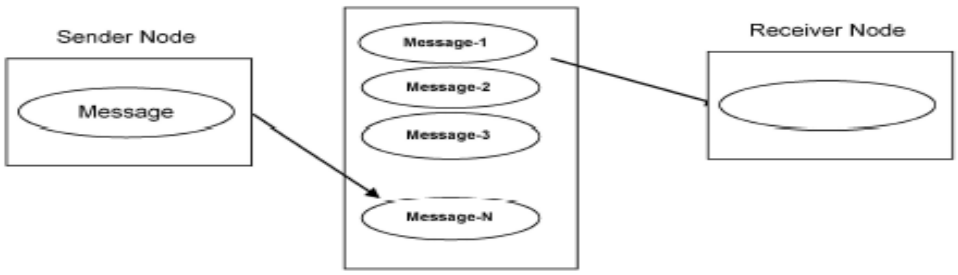


2. Single Message Buffering:

This type of buffering uses a single buffer either at the receiver node address space in order to ensure that the message is readily available to the receiver as and when the receiver node is ready to accept the same. The single message buffer performs better in some situations as the message is available in the buffer which helps the while system in reducing the blocking duration at different nodes. The single message buffer method reduces the delays in communication in comparison with the Null buffering method of communication.

3. Multiple Message Buffering:

The multiple message buffering communication mechanism is generally used in asynchronous type of communication in inter process communication within distributed operating system. The multiple message buffer as shown in the figure below works as a mail box which is either stored at the receiver's address space or operating system address space. A sender executes the send process in order to send a message and the same is received by the receiver from the mail box as and when the receiver processes the receive message process.



The multiple message buffering is a good mechanism for message passing however, the method is prone to buffer overflow problem. The buffer overflow problem is handled in two different mechanisms where a message send request will generate an error message in case the buffer is full and similarly, the receive message request will generate an error message when the buffer is empty.

The second method to handle this problem is to use the multiple message buffer in an controlled fashion where the send message request is processed and the message is stored in the multiple message buffer. The multiple message buffer is blocked till the acknowledgement from the receiver is received to acknowledge the receipt of the message at the receiver node. This mechanism is also treated as forced synchronous mode of communication between different nodes and this mechanism may lead to unexpected occurrence of deadlocks in the communication process within a distributed operating system.

MULTI-DATAGRAM MESSAGES

The inter-process communication between different nodes within a distributed operating system is an essential part of a network based operating system. The messages transferred from a sender to a receiver in a network is in the form of packets where the data packets correspond of different information attributes like process identifier, address, sequence number, structural information and actual data.

A datagram is a self-sufficient and independent packet of data associated with a packet switched network. It carries adequate information to be routed from the source to the destination and the network.

Datagrams provide a connectionless communication service across a packet-switched network. Every network allows a maximum allowable size of a datagram that can be transmitted from one node to the other and the same is known as **MTU maximum transfer unit**. In case the size of the message is smaller than the maximum transfer unit then the datagram is known as “single datagram” message.

However, in case the size of the datagram is more than the maximum transfer unit then the datagram is divided into smaller datagrams in order to communicate these multiple datagrams from sender to receiver

.

The multiple datagrams communicated from sender to receiver are known as multi-datagrams. These multi-datagrams include extra attributes within a packet which carry the information about the sequence of the datagrams and mechanism used for fragmenting. This extra information is used at the receiver end to combine all the multiple datagrams into a single block of information.

ENCODING AND DECODING

A message transmitted from the source node to destination node is in the form of single datagram or multi-datagram. The message delivered at the destination or receiver node should be complete and correct. Therefore, the structure of the datagram should also be known at the receiver node in order to understand the complete properties of the datagram received.

The receiver node should have the complete information related to the datagram available at sender node in order to maintain the consistency and integrity of data. To ensure this the datagram to be sent to the receiver node should be converted into a form which can be transmitted through the communication channel and accordingly on receipt of the packet the same should be converted back to the original form at the receiver node.

The process of converted the original data packet into a stream that is compatible with the communication channel is known as encoding of the message and the process of reverted the received message to the original message at the receiver node is known as decoding.

The received node encodes the original message and sends the same through the communication channel or buffer to the receiver node where the encoded message is decoded to get the original message back at the receiver node. Different methods are used for encoding and decoding and the two basic representation of encoding and decoding process are Tagged representation and untagged representation.

In the tagged representation all the details about the object along with the data value is encoded and then send through any communication channel or buffer to the receiver. At the receiver node the encoding done using tagged method reverted back to its original form. The data packet received by the receiver node is simple to decode and implement as the information about all the properties of the data packet along with the data is available.

However, in untagged representation program objects do contain only data which does not make the data packet delivered at the receiver node as self-explanatory. The receiver node must have the information about the encoding method or mechanism used at the sender node in advance to decode the data packet to its original form.

PROCESS ADDRESSING AND FAILURE HANDLING

INTRODUCTION TO PROCESS ADDRESSING

The process of communication from any two nodes in a distributed operating system cannot be complete without the address of the source node and target node is known. The address of a node or a computer while communicating messages from one node to another is known as addressing or naming. The addressing or naming helps the network to identify the nodes with unique value which is very important in establishing connection between any two nodes of a distributed system

The two basic addressing modes used in distributed operating system are given below:

1. Explicit Addressing: when the message is explicitly destined for a process then the message is sent using the explicit mode of addressing. In this addressing mode, the process-identification (x) along with message (y) is sent to the receiver node (z). The receiver node (z) will only receive the message from process-identification (x). If any other message from process-identification (k) is available it will not be received by the receiver node (z).

2. Implicit Addressing:

when a message is destined for any receiver node that requires the services of the message then implicit mode of addressing is used. In this addressing mode, the service-identification (x1) along with message (y) is intended to be sent to any receiver node where the service identification(x1) is offered .The implicit mode of addressing allows a sender node to send the message to more than one node within a network provided the sender has to name a service rather than a process.

This type of addressing mode is feasible for client-to-server communication where a client requests for a service and sends the message to all the available servers. The server in turn will receive the message and treat it as a process. Similarly a server can also send a message to all clients to access a service and in turn only clients which are allowed to use the service can receive the message from the server and acknowledge the same.

The naming of a node is carried out using different methods where the address of the machine is used as the address of the node and in some cases the address of the machine along with the process identification number is used as the address of a node.

Suppose a node has been assigned an address 159 which for humans is a numerical number and the same number is understood by a machine in the form of machine instruction which is always in 1's and 0's. In case any node intends to send a message to the node which has been given 159 number will be sending the message by creating a data packet where the address will be mentioned as 159.

Once the data packet is broadcasted in the network the node with 159 address will accept the data packet and send the acknowledgement to source node. The kernel of an operating system will have information about the task of sending the message from the sender node to the receiver node if only one process is existing within a network. The statement will not be true in case of distributed operating system where at a given point of time one or more number of processes will be existing and all the processes are required to be monitored by the kernel of an operating system.

Therefore the address method that has been discussed above will not work for distributed operating system. Another method of address can include more than one segments in the address for example if a machine address is 159 and the process identifier of requesting process is 29 then we can have an address like 159@29 which is self-explanatory to the operating system as the first component gives information about the address of the machine and the second component of the address gives the information about the process identification.

The machine address and process identification are unique numbers which are assigned to a machine and a process respectively within a network. Therefore, 159@29 gives the information to the network that the message is destined to machine number 159 where process number 29 requires the message. In this addressing system the load balancing becomes difficult if the number of running processes is more in number. The problem of load balancing can be reduced by including third attribute in the address which can contain the information about the machine address of the last known location.

Therefore, the addressing mechanism will have three segments i.e. Machine_Number@Process_Identification_Number @Machine_Number_lastknown.

Once a process is created the first two segments will have constant value till the ompletion time of the process. However, the third segment of the address will keep on changing as per the movement of the message from one node to the other. The addressing mechanism with three segments is also known as link-based addressing. This addressing mechanism will be overloaded, if a process has moved from one node to many nodes while reaching the destination as the overheads in this regard will increase as the message is moving via many nodes. The second problem in this addressing mechanism is that it does not support portability.

In order to mitigate the impact of the above mentioned problems a two level addressing mechanism can be used which include a high level address and a low level address.

The high-level address will be an ASCII string which will be independent of the machine that is being used in order to enhance portability and migration of a process from one system on a network to other system on another network. The low-level address will contain the information about the machine and the process locally. The low-level address will contain the machine identification number and the process identification number. However, this addressing mechanism requires support of the naming server as the ASCII string as high-level address needs to be translated to low-level address which will be stored on naming server.

In this addressing mechanism the onus lies on the name server as the same is responsible for generating the low-level address based on the high-level address.

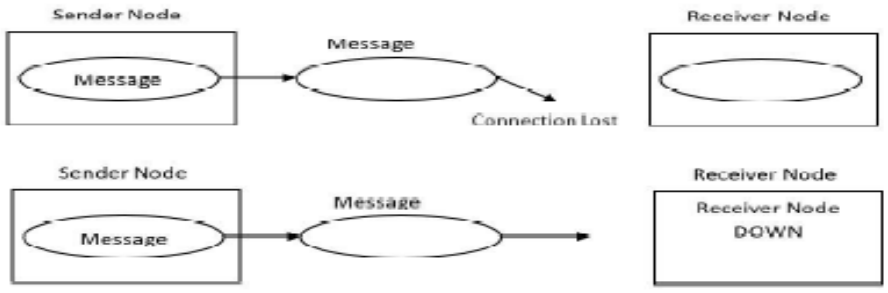
In case the name server response time is more the whole system will take more time complete the processing of a process.

FAILURE HANDLING

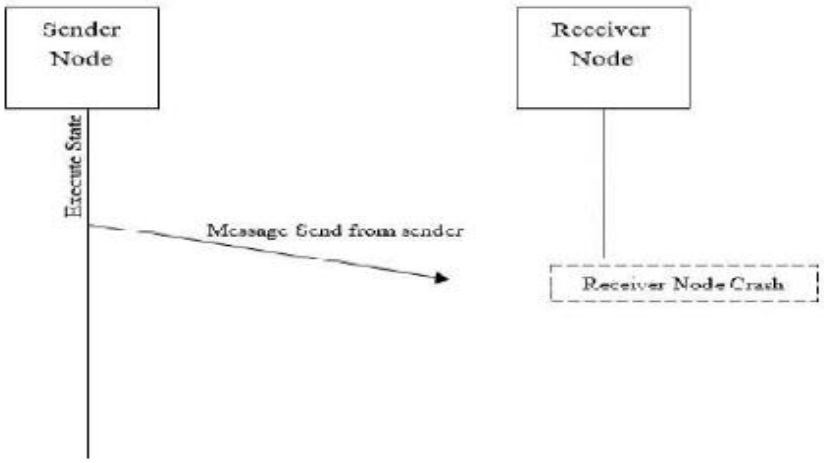
The messages sent from the sender node are not received at the receiver node due to different failures which may occur in a distributed operating system. The scalable and robust design is always pro-active in order to give seamless services to users interacting with a distributed operating system.
The failures that can occur while communicating a message within any two or more nodes of a network are discussed below:

1. Request Message Lost:

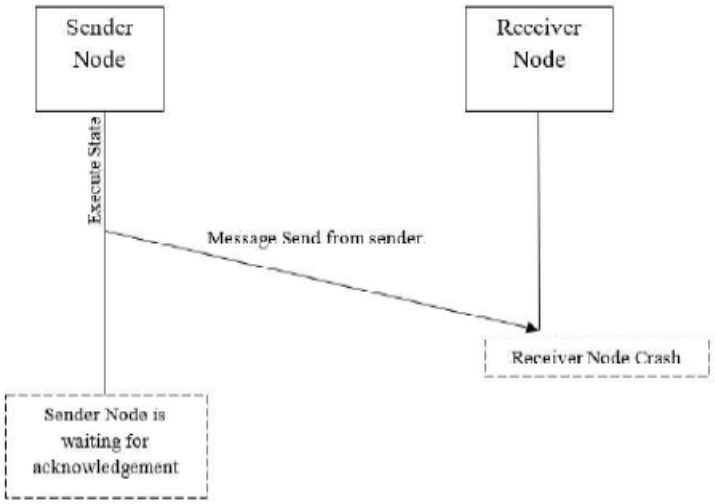
This problem can occur if the sender node sends a message and the receiver is down due to breakdown of the communication link between the sender node and the receiver node.



In case the receiver node is down or gets crashed the communication between the sender node and the receiver node will not be possible and the same is shown in figure given below:



In case the receiver node receives the message but crashes prior to sending the acknowledgement will result failure of communication as the sender node will not receive the acknowledgement. The scenario of receiver crash after receiving the message is shown in figure given below:



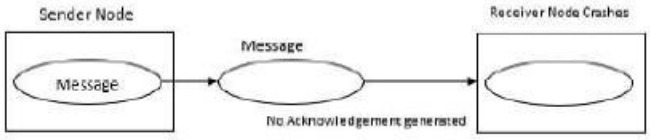
The impact of this problem is mitigated by implementing the concept of timeout. In case the sender is not able to receive acknowledgement from the receiver node, the kernel of the operating system will wait for a particular time limit known as timeout. Once the time limit has reached the sender node tries to send the message again and if the acknowledgement is not received the kernel of the operating system initiates the process of freeing the resources acquired by message sending process.

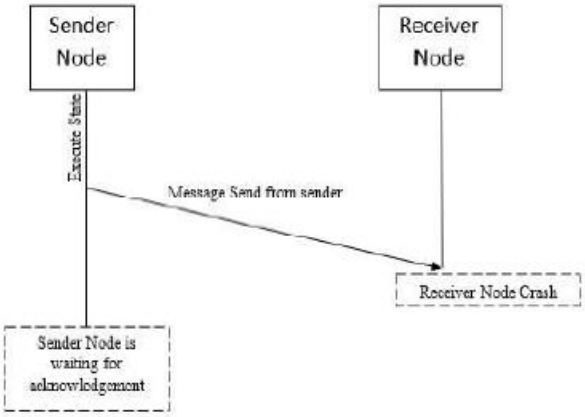
2. Node/Computer Crashes: The problem occurs when either a sender node or the receiver node crashes in the process of communication. The kernel of the operating system initiates the process of freeing the resources after waiting for timeout. In case the sender node crashes after sending the message and the receiver node does not receive the message, the kernel of the operating system waits for the timeout limit. After the expiry of the timeout the kernel frees the communication channel and clears the message and the processes associated with the message.

Similarly, in case the receiver node crashes after sending the acknowledgement to the sender node which in turn has to reply to the receiver node in order to complete the process and free the resources acquired by the process at the receiver end. The kernel of the operating system will initiate the process of freeing the resources acquired by the process after the timeout period is over.

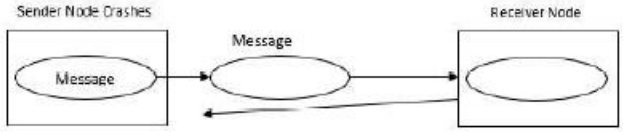
The two main cases where a node may crash are given below:

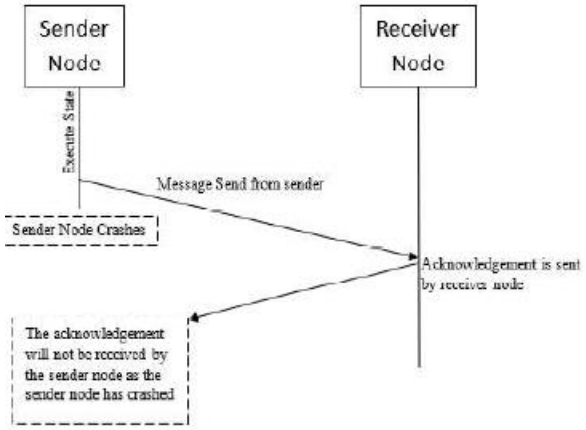
(a) Receiver node crashes after receiving message
(b) Sender node crashes after sending the message





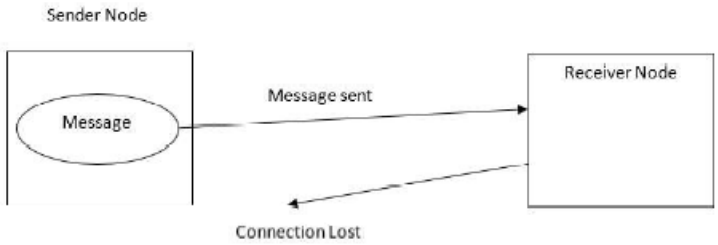
(a) Pictorial Representation of Receiver Node Crashes after Receiving the Message





(b) Pictorial Representation of Sender Node Crashes after Receiving the Message

3. Response Message Lost: This problem generally occur when a receiver node receives the message and the acknowledgement is not received by the sender due to breakdown of the communication link or sender node is down after sending the message to the receiver node.



The impact of this problem is mitigated by implementing the concept of time out. In case the sender is not able to receive acknowledgement from the receiver node, the kernel of the operating system will wait for a particular time limit known as timeout. Once the time limit has reached the sender node tries to send the message again and if the acknowledgement is not received the kernel of the operating system initiates the process of freeing the resources acquired by message sending process.

The whole system of communication is susceptible to errors or failures which need to be handled properly in order to mitigate the impact of communication failures on distributed operating system. In order to handle the situations which may lead to a failure different tools and techniques are used in distributing computing environments.

One of the failure handling technique is to create checkpoints for all the message passing routines which are required for process completion. The checkpoints give the information about the state of the message passing sub routine to the kernel in order to decide whether a commit or a rollback is required in order to complete the process of communication between any two nodes within a network.

The commit point gives information to the kernel about the exact status of a message passing process prior to the node crash in a network. The status information in commit is then implemented and stored permanently in the system in order to ensure the integrity and consistency of data within a distributed operating system.

The rollback point gives the information about the status of a message passing routine within a distributed operating system and the same information helps a kernel to roll back all the changes made as per the information available in the rollback point.

The rollback of all the computations is implemented by the operating system by reinstating the status to the previously stored commit point. Another method to handle failures in communication process is by using retransmission or a message between a sender node and a receiver node where the retransmission of a message is implemented after the expiry of timeout. In this method the kernel of the corresponding machine will wait for the acknowledgement or reply from the corresponding nodes till the timeout limit is reached and after the expiry of the timeout limit the kernel of the machine will retransmit the message. As an example if a sender node sends a message to the receiver node and the sender node is not receiving the acknowledgement from the receiver due to any failure that has occurred during the communication process.

The sender node will not retransmit the message unless the timeout is reached. After the timeout limit is reached the sender node will retransmit the message to sender node. Similarly, if a receiver node has received a message and has sent an acknowledgement to the receiver.

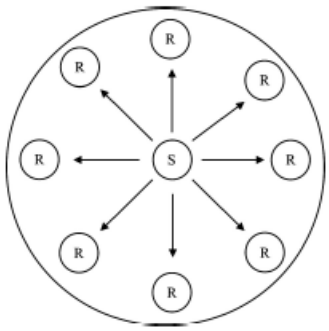
The receiver node in turn has to send a reply to the receiver node in order to change the status of the communication process to complete. In case the receiver node does not receive the reply from the sender node, it will wait for the timeout limit to expire. Once the timeout limit expires the receiver node will retransmit the acknowledgement to the sender node in order to complete the communication process and free the resources associated with the process.

GROUP COMMUNICATION

The communication between two processes in RPC is established by calling remote procedures.This technique is not suitable for an environment in which multiple members are involved: for example, consider that multiple servers are cooperating on a single, fault tolerant file system. In this system, the client has to send the message to all the servers so that the task is performed even if one of the servers crashes. RPC cannot handle a communication involving one sender and multiple receivers. It has to perform separate RPCs with everyone.

Introduction to Group Communication

A group consists of a collection of processes, which perform the task together in the system. If a message is sent to the group, all the processes receive it. Different members of a group can communicate in two ways, one-to-many communication and one-to-one communication. In the one-to-many communication, one sender and many receivers are involved. One-to-many communication involving one sender and many receivers is shown in



One-to-Many Communication

These groups are dynamic and therefore, new groups can be created and old groups can be deleted at any time. A process can join or leave any group and can be the member of many groups at a time. While sending a message to the group, the process does not need to know the number of members in the group.

The implementation of communication in groups depends on the hardware. A network address can be assigned to the network, which can be used to send the message in the group. When the message is sent to that particular address, it is automatically delivered to all the members. This process is called multicasting.

Multicasting provides a simple technique to implement groups by assigning different multicast addresses for each group. Networks, which do not have multicasting,broadcast the packet. In broadcasting, the packet is sent to all the members of all the groups. Broadcasting the packet is less efficient because when the packets are delivered to every group, the software is required to check whether or not the packet is intended for the computer.

If the packet is not delivered, then the packet has to be discarded. Thus, extra time is required for this purpose If neither multicasting nor broadcasting can be implemented, then the group communication can be achieved by sending separate packets to each member of the group. However, n packets are required for a total of n members in all the groups. Sending of message from a single sender to a single receiver is called unicasting or point-to-point communication. The process of transmitting data from a single user to a single receiver is shown in the below picture.

Point-to-Point Communication

This process is efficient if the size of groups is small and is not suitable in case the size of the groups is big.

Design Issues related to Group communication

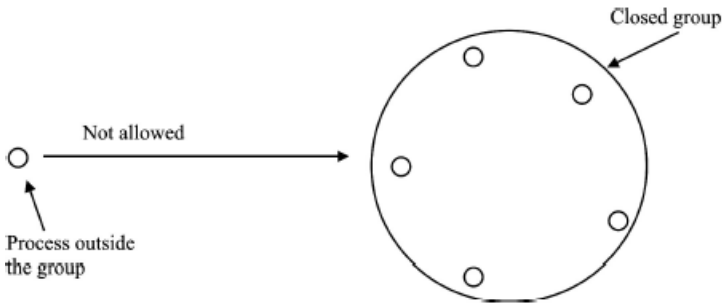
Different types of group communications can be designed to establish communication among multiple users of the system. There are a lot of design possibilities used in the designing of the group communication system. The regular message passing and primitives based communications are examples of such design possibilities. Designing of the group communication system is entirely dependent on the internal organization of a group.

The types of group communication systems are stated as follows:

- Closed group: This group refers to the group in which outsiders are not allowed to send messages to the group as a whole.
- Open group: This group refers to the group in which an outsider can send message to any group involved in the network.
- Peer group: This group refers to the group in which every member of the group is connected to the other members of that group.
- Hierarchical group: This group refers to the group in which one member of the group acts as a coordinator of the other members of that group.

Closed Groups versus Open Groups

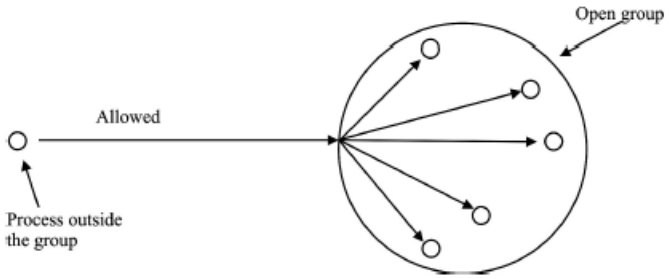
Group communication system can be categorized on the basis of the permission given to a sender, who sends message to a group. Closed group system and open group system are examples of such group systems. In closed groups, only the members of the group can send a message to the other members of the group. A computer, which is not the member of the group, cannot send the message in the group.



Closed Group

Closed groups are used for parallel processing of data: for example, a collection of processes running for computing the moves in a chess game or a group of processes copying a file from one location to another form a closed group. These groups do not interact with the computers outside the groups.

In open groups, any computer can send the message to any other computer or member in the group.

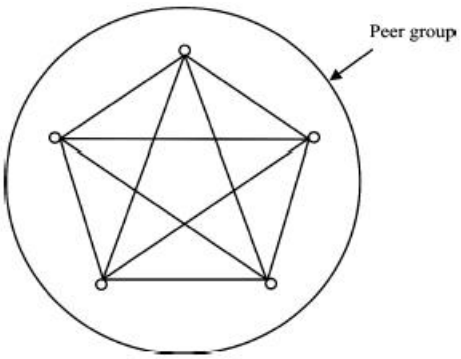


Open Group

Open groups are used when coordination between servers is required for performing some task. In this situation, it is required that the computers, which are not members of the groups, should be able to send the message to the other groups.

Peer Groups versus Hierarchical Groups

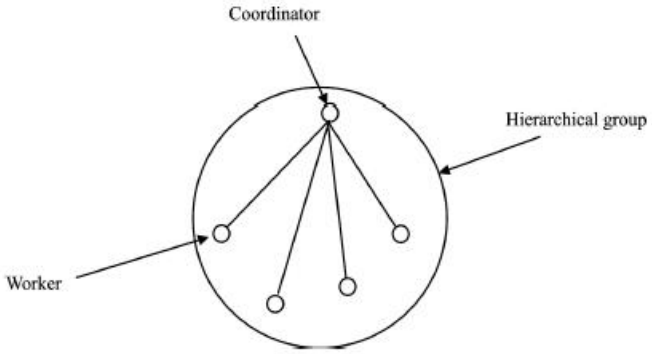
The members of the group can also be differentiated on the basis of internal structure of the group. In some groups, all the members are considered equal and can make the decisions collectively. This type of group is called peer group.



Peer Group

A peer group is symmetric and does not fail. If any one member crashes, the other members can take over the job. The disadvantage of this group is that decision-making is very complicated in this group. Voting is performed to decide anything. In other types of groups, an hierarchy of members exists. One process is made the coordinator, which makes the decisions for all the members of the group.

This type of group is called hierarchical group.



Hierarchical Group

When a request for any task is generated by an external client or any member of the group, the coordinator decides which member is best suited to complete the task. In this system, if the coordinator crashes, the whole system comes to a halt. But, if any other member crashes, then it does not affect the coordinator but the performance of the system decreases.

Group Membership

When group communication is employed, then a method is required to create and delete groups and also one method is required to allow the members to leave and join groups. One method can be to have a group server to which all the requests of this type can be sent. The group server maintains the database for all the groups and their members.

However, there is a problem with this technique as well. If the group server crashes, then the group management also fails and all the groups have to be reconstructed from the beginning by terminating all the processes. The other method can be to have the membership in a distributed technique. Any computer can send the message to any group to become the member of the group. In case of closed groups also, the members have to send the message to join the group. If any member wishes to leave the group, then it can simply send the message to all the other members.

There are a few problems associated with both of these techniques related to the membership of groups. These problems are stated as follows:

If any one of the members crashes, then the other members will not be able to know whether the member has really crashed or left the group. The other members have to be certain that the member, which is not responding, has really crashed or left the group.

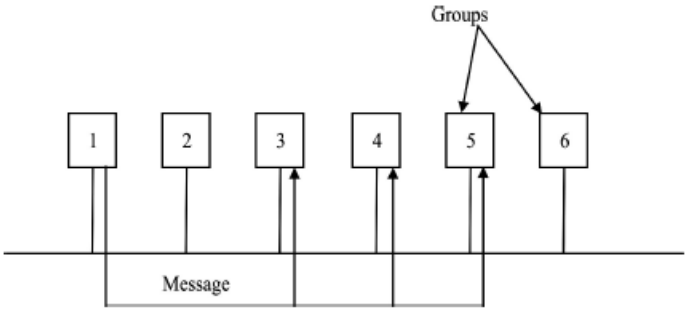
One more matter of concern is there, i.e. the process of leaving and joining the group has to be synchronous with the messages. In other words , the computer should start receiving messages as soon as it joins the group and should also be able to send messages to the other members of the group.

Besides as soon as the member leaves the group neither it should receive any message from the group nor the members of the group should receive any message from it.

The final issue is that if many members in the group crash, then some mechanism is required to reconstruct the whole group. Since in such a situation, the group will no longer be able to complete the task, one of the working members has to take up the job of reconstructing the whole group. There could be a problem if more than one member starts reconstructing the group. A mechanism is required to control this problem as well.

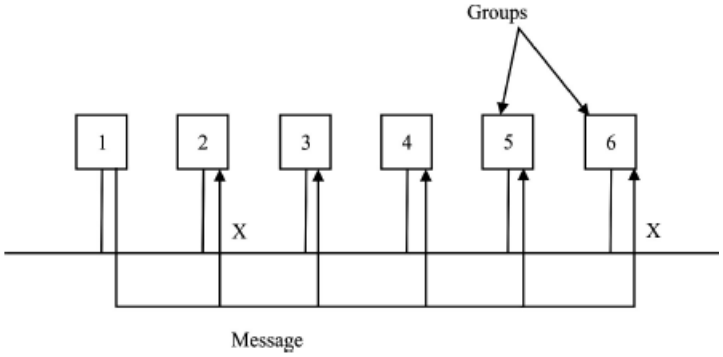
Group Addressing

If any computer wants to send a message to a group, then there must be some mechanism to specify the group to which it wants to send the message. The groups can be provided with some unique address to address the groups. If multicast is supported by the system, then the address of the group can be associated with the multi cast address. In this way, every message, which is sent to the group, can be multi casted. In this technique, the message is sent only to those groups, which need the message.



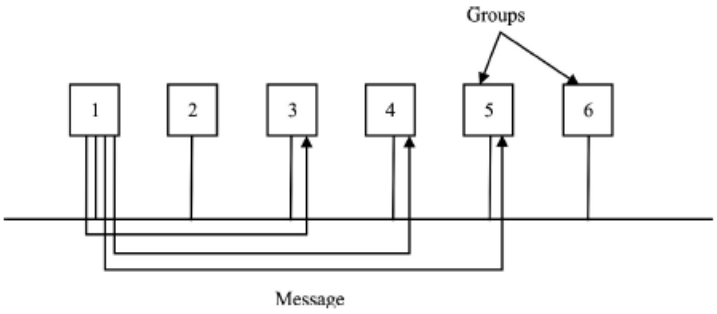
Group Multicasting

If the system supports broadcasting but not multicasting, then the message can be broadcasted to all the groups. The groups who do not need the message just discard the message.



Group Broadcasting

If neither of the two techniques is supported b the system, then the kernel has to send the message individually to the groups intended to. This process uses point-to-point communication called unicast for sending the message to all the groups.



Point-to-Point Messaging

In all the three processes, the computer sends the message to the group address and it is delivered to all the members. OS decides the process of sending the message to the groups. The sending computer is not aware of the process used to send the message to the groups.

Another method of sending the message to the group can be to provide an explicit list of the entire destination, such as IP address to the sender. If this method is used, then the location to the destination contains a pointer to the list of addresses as a parameter. In this method, each member of the groups is required to have the information about the other members of the group. If the members in the group are added or removed, then the list of members has to be updated. This task of updating the list is performed by the kernel.

Send and Receive Primitives in Group Communication

The send and receive primitives are also used in the group communication system. In group communication system, n number of replies can be transferred in order to answer a request message. To deal with a number of members of the group, explicit calling of one-way send and receive primitive is used. The send primitive has two parameters, the destination address and the message. If the destination address is related to a process, then message is sent to that process and if the destination parameter contains the address of a group, then the message is delivered to all the members of a group. The receive primitive is used to check whether or not the message is received by the destination. The send and receive primitives can be of any type, such as buffered, un-buffered, blocking and non-blocking.

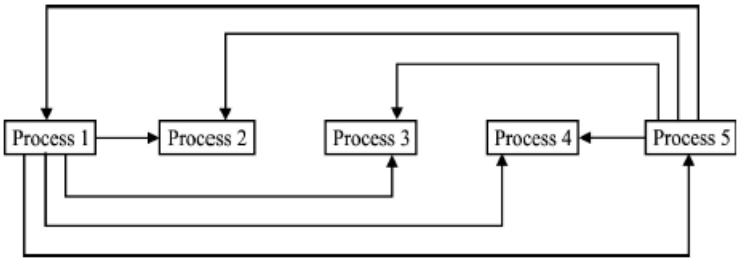
Atomicity

Atomicity refers to a condition when a message is sent to a group and it is received correctly either by all the members of the group or by none of the member of the group. It is one of the most important properties of the group communication system. Sometimes this property is known as all-or-nothing or atomic broadcast.

Atomicity helps in making communication easier among all the members of a distributed system. In other words, when a process of a system having atomicity sends a message to a group, then it does not have to bother about the proper delivery of the message. It also enhances the fault tolerance capability of a distributed system by holding information about the failure of the machine involved in the system.

Message Ordering

Message ordering is another important property of the group communication system. It refers to the right order of the messages received by the different members of a group. To understand the right order of the messages, consider an example of a group, which consists of five processes namely, Process 1, Process 2, Process 3, Process 4 and Process 5. Process 1 and Process 5 have to send message to the other processes of the group. First, Process 1 sends messages to Process 2, Process 3, Process 4 and Process 5 and then Process 5 sends messages to Process1, Process 2, Process 3 and Process 4. Figure 2.23 shows sending of messages in a group.



Message Ordering between Processes

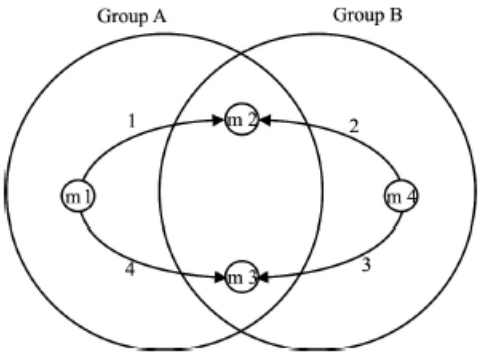
In the Message Ordering between Processes picture Process1 sends message to Process2 and then other processes; whereas Process 5 sends message to Process 4 first. In this case, Process 4 receives the message of Process 5 before receiving the message of Process 1. If both the processes, i.e. Process 1 and Process 5 send messages related to updation of the database, then the order of receiving messages can cause problems with the database.

To avoid such type of problems, a proper message ordering is maintained by the system so that the order of receiving the message at the destination end is similar to the order of sending the messages.

Overlapping of Groups

A computer can be the member of any number of groups at a time. This can lead to problems and inconsistency in the groups as well: for example, consider the situation in which there are two groups A and B. Group A consists of members m1, m2 and m3 and group B consists of members m1, m2 and m4 respectively. If the two groups send messages simultaneously to all the members in the group, it can lead to inconsistency in the two groups. Suppose the messages are sent using unicasting so that the member m2 receives the message first from group A and then from group B and m3 first receives the message from group B and then from group A.

This process is shown in the below picture.



Overlapping of Groups

REFERENCE:

Pradeep K. Sinha, ”Distributed operating system concepts and design”, PHI, New Delhi, 2007