

# Communication in DOS

Rajeev Ranjan, Rahul Yadav  
*Student, Department of Information Technology  
Dronacharya College of Engineering*

**Abstract-** The paper discusses issues related to distributing operating system along with its security over the remote server. Also we will discuss the implementation and designing of distributed system. In this paper we will discuss the issues related to message passing techniques, speed of communication, reliability etc. and the protocols used in communicating the processes across the web.

**Index Terms-** message passing, parallel computing, synchronous or asynchronous, multicasting, redundancy, datagram, handshake.

## I. INTRODUCTION

A distributed operating system is an operating system which manages a collection of independent computers and makes them appear to the users of the system as a single computer.

Usually a distributed operating system utilizes the resources of multiple computers and devices and allows for remote execution. You can run a program on one machine, but send the output to a different one. So in theory you have the computing power of every computer in the network at your disposal, not just one.

A distributed OS will share the scheduling of processes as a collective group for the networked machines. A program running on one system may have subprocesses running on other systems in the network, all cooperating and running as if on the same computer.

In a distributed system, there's no shared memory, so the entire nature of interprocess communication must be completely rethought from scratch.

All communication in distributed system is based on **message passing**.

Communication (IPC) in distributed system is related to a set of methods for the exchange of data among multiple threads and/or processes. Processes may be running on one or more computers connected by a network.

## II. ISSUES

1. Processes which execute on CPUs sharing memory can easily communicate, but what about processes executing on CPUs which don't share memory?
2. How to implement message passing (communication)
3. Speed of communication
4. Reliability of communication
5. Transparency of communication
6. Locating the right process to communicate with
7. Consistency of Communication unified

**1. Message passing** in computer science is a form of communication used in concurrent computing, parallel computing, object-oriented programming, and interprocess communication. In this model, processes or objects can send and receive messages (comprising zero or more bytes, complex data structures, or even segments of code) to other processes. By waiting for messages, processes can also synchronize.

Message passing is the paradigm of communication where messages are sent from a sender to one or more recipients. Forms of messages include (remote) method invocation, signals, and data packets. When designing a message passing system several choices are made:

- Whether messages are transferred reliably
- Whether messages are guaranteed to be delivered in order
- Whether messages are passed one-to-one (unicast), one-to-many (multicast or broadcast), many-to-one (client-server), or many-to-many (All-to-All).
- Whether communication is synchronous or asynchronous.

### III. SPEED OF COMMUNICATION

we consider very high-speed, connection-oriented communication in distributed systems where each node system has a limited-size queue for connection requests. An important example of this type of system is an HIPPI-based interconnection system of a supercomputer complex. For such systems, we present a distributed connection management policy and propose several possible service disciplines. We develop an analytic model to evaluate the interconnection system under different system configurations, connection management policies, and service disciplines. In this evaluation, we consider separately systems where nodes want to communicate with any one of a pool of identical servers, and systems where a node needs to access a specific one of a set of distinct servers.

The recent advent of Smart Grid has given rise to advances in communication systems for distribution systems. Modern numerical overcurrent relays have the technology available to utilize these communication channels for both high speed-assisted tripping and sectionalizing. Assisted tripping and sectionalizing allows the utility to operate their distribution system in a network as opposed to radial feeders. A networked system is much more reliable and customers experience fewer outages since there are multiple sources readily available. Assisted tripping and sectionalizing quickly isolates the fault and eliminates the need for long clearing times and complex coordination typically associated with classical time overcurrent protection.

#### **Several Problem Areas in high speed networks**

- Multicast distribution: What techniques should be used to efficiently implement logical channels for multicast switching in a network with many subscribers, in order to support applications ranging from selective distribution to dynamically reconfigurable group communication?
- Distributed switching: How can signalling protocols, addressing, routing and clock distribution be provided for distributed switching? What network structures are suitable for interconnecting switching elements? What are the advantages and disadvantages with using circuits, cells, and packets as the basis for switching, and how can variable length packet switching be provided in cell-based networks?

- Network resource management and routing: What traffic control policies are most suitable; can feedback schemes react fast enough, and are reservation-based policies cost-efficient? How are resources managed to support real-time multicast transmission?
- Fault-tolerance in high-speed networks: How are networks organized to be resilient to link and node failures? What are the best ways to reconfigure high-speed networks after faults, and how are faults detected and located in optical networks?
- Impact of new optical components on protocols: What medium access and switching techniques are best suited for multi-wavelength distribution techniques? How are protocols best designed for cost-efficient use of optical switches and amplifiers?
- End-system integration: Does the close integration of communication and computation impose new requirements on optical fiber access techniques, switching, and network protocols? Is it possible and advantageous to combine network resource management with management of resources in the end-equipment (e.g. CPU scheduling)? How are protocols designed to enable efficient end-system interfaces?
- High-speed switching: How will switching fabrics and circuit architectures change when moving from current low and medium rate switching systems towards 10 Gbit/s and 40 Gbit/s per port switching systems? How to dimension and design such switching systems? What impact will C&C environment applications have on switch architectures and switch dimensioning. Due to high operational speed, what constraints will implementations pose, and how are such constraints coupled to system design of switching systems and architectures? What implementation techniques and architectures need to be used for optical interfaces to such switches as well as to the overall switch fabric itself?

### IV. RELIABILITY OF COMMUNICATION

Making a distributed system reliable is very important. The failure of a distributed system can result in anything from easily repairable errors to catastrophic meltdowns. A reliable distributed system is designed to be as fault tolerant as possible. Fault tolerance deals with making the system function in the presence of faults (see Fault-Tolerant

Systems). Faults can occur in any of the components of a distributed system

This article gives a brief overview of the different types of faults in a system and some of their solutions

### Component Faults

There are three types of component faults: transient faults, intermittent faults and permanent faults. A transient fault occurs once and then disappears. If the operation is repeated then the system will behave normally. An intermittent fault arises, then goes away, rises again and so on. A common cause of an intermittent fault is a loose contact on a connector. These faults are very annoying and hard to fix because of their sporadic nature. Lastly there are permanent faults caused by faulty components. The system will not work until the component is replaced. Burnt out chips, software bugs and processor failure (explored in Processor Faults) are all examples of permanent faults.

#### Processor Faults

A special type of component is the processor, and it can fail in three ways, fail-silent, Byzantine and slowdown. All lead to a different kind of failure. A fail-silent, also known as fail-stop, fault occurs when a processor stops functioning. It no longer accepts input and outputs nothing, except perhaps to say it is no longer functioning. Byzantine faults occur when a faulty processor continues to run, giving wrong answers and maybe working with other faulty processors to give the impression that they are working correctly. Compared with fail-silent faults, Byzantine faults are hard to diagnose and more difficult to deal with. A slowdown fault occurs when a certain processor executes very slowly. That processor might be labeled as failed by the other processors. However the “failed” processor may return to its normal speed and begin to issue orders, causing problems within the distributed systems.

### Network Failures

Network failures keep processors from communicating with each other. We will look at the failures resulting in total loss of communication along parts of the network. Two problems arise from this: one-way links and network partitions. One-way links cause problems similar to processor slowdown (see Processor Faults). For example, processor A can send messages to processor B but cannot receive messages from B. Processor C can talk to both A and

B. So each processor has a different idea of which processors have failed. Processor A might think that B has failed since it does not receive any messages from it. Processor C thinks both A and B are working properly since it can send and receive messages from both.

Network partitions occur when a line connecting two sections of a network fail. So processors A and B can communicate with each other but cannot communicate with processors C and D and visa versa. Let us say processor A updates a file and processor C updates the same file but in a different way. When the partition is fixed, the file will not be consistent among all processors. It is not clear to the processors how to make the data consistent again.

### Solutions To System Failures

Before we explore some of the common solutions to system failures, we must learn the difference between synchronous and asynchronous systems. In a synchronous system the amount of time required for a message to be sent from one system to another has a known upper bound. Therefore, processor A sends a message to processor B and waits a given time for a response. If A does not receive a response within that time, it knows an error has occurred and it will send the message again. After a set number of resends, B is labeled as failed. In an asynchronous system, none of this is true. A processor will wait an infinite time for a response from the other processor. Many solutions for fault tolerance cannot be implemented in an asynchronous system. A processor experiencing slowdown is impossible to differentiate from a dead processor (see Impossibility of Consensus).

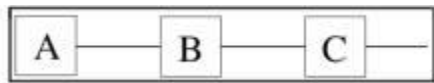
The most common approach to handling faults is redundancy. There are three types of redundancy: information redundancy, time redundancy, and physical redundancy. Information redundancy involves adding extra bits to allow recovery from distorted bits. An example is adding a Hamming code to data in order to recover from noise in the transmission line. With time redundancy an action is performed and if need be it is performed again. An aborted transaction can be redone without harm to the system. For more information refer to Chapter 3 of Tanenbaum [2]. Time redundancy is the most frequently used solution for intermittent and transient

faults. Physical redundancy involves adding more components to a system in case a component fails.

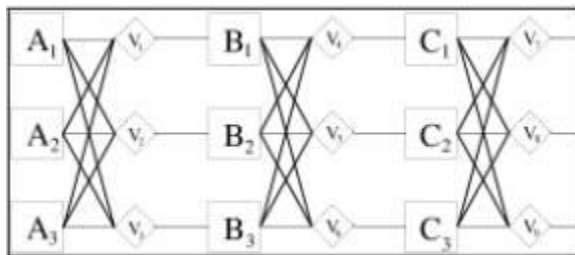
### Physical Redundancy

There are two types of physical redundancy: active replication and primary backup. The advantages and disadvantages of each must be weighed when determining which type of physical redundancy will be implemented.

First we will try to understand active replication by looking at Triple Modular Redundancy (TMR) in a circuit. Consider a circuit with devices *A*, *B*, and *C* linked in sequence. If all devices are working properly then the final result will be correct. But if one of the devices is faulty then the final result will probably be incorrect.



Now we will look at a circuit utilizing TMR. First devices *A*, *B* and *C* are replicated three times and then three voters are added after each stage of the circuit. Why are three voters required in this system?



The answer is that the voters are components too and might fail. Each voter has three inputs but only one output. The majority of inputs become the output of the voter. If two or all inputs are the same then that becomes the output. If all three inputs are different then the output is undefined.

Now let us see if the system will be fault tolerant when different components fail. First, consider a simple case where *A1* fails. The voters will pass on the value of *A2* and *A3* since that value is in the majority. The voters pass on this value to *B1*, *B2* and *B3*, which receive the same value as they would have if no fault had occurred. The system is fault tolerant. Now we will see what happens if one voter, *V1*, fails. This would mean that *B1* would get the wrong input but *B2* and *B3* would have the correct input. So in

the next series of voters, the error will be ruled out and the circuit will act like there was no failure. One should note that an error in *V1* behaves exactly the same as an error in *B1*.

Now let *A1* and *A2* fail. Assuming they fail with the same result, this result will be passed through the voter so *B1*, *B2* and *B3* contain the wrong value. If *A1* and *A2* fail with different results then there will be three different inputs to the voter. The output of *V1*, *V2* and *V3* will be undefined. So the TMR system is not perfect but what is?

When considering active replication, it is important to consider how much replication is needed. The answer is related to the amount of fault tolerance wanted. A system is said to be *k* fault tolerant if faults in *k* processors can produce the same outputs as a fully functioning system. With fail silent faults, *k* + 1 processors are needed to achieve *k* fault tolerance. But with Byzantine failures, 2*k* + 1 processors are required. Refer to Tanenbaum [2] for an application of active replication in a distributed system.

The other type of physical redundancy is called primary backup. This type of fault tolerance involves one server, which is the primary server, and an unused backup server. If the primary fails, then the backup server becomes the primary. The client operating system but not the application programs will notice the switch of control. When compared to active replication, primary backup has one main advantage: simplicity. Messages are sent to the primary server only, as opposed to a whole group of servers. Second, this type of physical redundancy only requires two machines, a primary and a backup. Of course when a backup server becomes a primary server, a new backup is needed instantly. A large disadvantage to primary backup fault tolerance is that it handles Byzantine failures poorly. There is no check routine to make sure the primary is functioning correctly. Another disadvantage is that a primary backup system must always be in agreement so that the backup can take over the functions of primary. Also recovery from a primary failure is time consuming and complex.

An important decision involving the primary backup approach is when and how to switch to the backup server. One solution is for the backup server to send messages asking if the primary server is still functioning. If the server does not respond in a

certain time then the backup will become the primary. This is not an ideal solution for asynchronous systems because the server could be running slowly resulting in repeated actions. Another solution is a hardware mechanism that the backup can use to reboot the primary. An alternate solution is to use a dual-ported disk shared by the primary and backup. The primary server will write the request to the disk then do the work and then write the results to the disk. This way if the primary fails at any time, the backup can read the disk and find out where the primary crashed. It can finish the job for the primary and there will not be any repeated messages. For further information on redundancy see Replicated Objects.

## V. TRANSPARENCY IN DISTRIBUTED SYSTEM

4.1 Integrating system components and resources into what appears to the user as a single local system

### Goal of Transparency

Hide all irrelevant system-dependent details from the user and system programmer and create the illusion of a simple and easy to use system

### Make system simple and effective

Conflicting goals requires trade-offs:

An overly simple system does not allow the user to take full advantage of the system preventing its use to be as effective as possible. Giving the user access to all system level details will maximize the potential effectiveness of the system but greatly adding to the complexity of its use.

### ACCESS TRANSPARENCY

Access to the system is the same regardless whether it is local or remote. The physical separation is concealed from the user.

### MIGRATION TRANSPARENCY

Resource may move yet access is identical as it was before. Also referred to as Location Independence.

### CONCURRENCY TRANSPARENCY

Allows sharing of objects without interference.

### SIZE TRANSPARENCY

Incremental growth of the system is kept hidden from the user and has no effect on the user's perception of the system.

### PERFORMANCE TRANSPARENCY

Attempt to achieve consistent and predictable performance levels so the user is unaware of changes in system structure or workload.

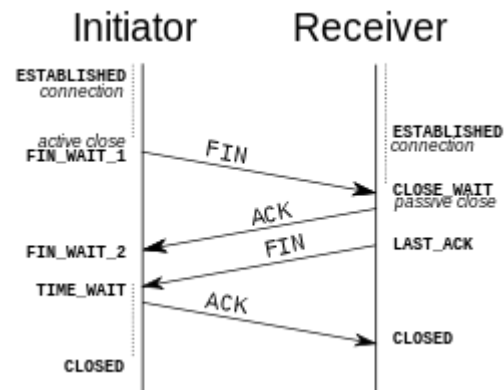
## FAILURE TRANSPARENCY

Failures are handled gracefully minimizing impact on user and hiding most of its effects.

## VI. PROTOCOLS USED IN COMMUNICATIONS IN DISTRIBUTED NETWORK

### 5.1 TCP

The **Transmission Control Protocol (TCP)** is one of the core protocols of the Internet protocol suite (IP), and is so common that the entire suite is often called *TCP/IP*. TCP provides reliable, ordered, error-checked delivery of a stream of octets between programs running on computers connected to a local area network, intranet or the public Internet. It resides at the transport layer.



### TCP PROTOCOL OPERATION

TCP protocol operations may be divided into three phases. Connections must be properly established in a multi-step handshake process (*connection establishment*) before entering the *data transfer* phase. After data transmission is completed, the *connection termination* closes established virtual circuits and releases all allocated resources.

A TCP connection is managed by an operating system through a programming interface that represents the local end-point for communications, the *Internet socket*. During the lifetime of a TCP connection the local end-point undergoes a series of state changes:<sup>[12]</sup>

#### LISTEN

(server) represents waiting for a connection request from any remote TCP and port.

#### SYN-SENT

(client) represents waiting for a matching connection request after having sent a connection request.

#### SYN-RECEIVED

(server) represents waiting for a confirming connection request acknowledgment after having both received and sent a connection request.

#### ESTABLISHED

(both server and client) represents an open connection, data received can be delivered to the user. The normal state for the data transfer phase of the connection.

#### FIN-WAIT-1

(both server and client) represents waiting for a connection termination request from the remote TCP, or an acknowledgment of the connection termination request previously sent.

#### FIN-WAIT-2

(both server and client) represents waiting for a connection termination request from the remote TCP.

#### CLOSE-WAIT

(both server and client) represents waiting for a connection termination request from the local user.

#### CLOSING

(both server and client) represents waiting for a connection termination request acknowledgment from the remote TCP.

#### LAST-ACK

(both server and client) represents waiting for an acknowledgment of the connection termination request previously sent to the remote TCP (which includes an acknowledgment of its connection termination request).

#### TIME-WAIT

(either server or client) represents waiting for enough time to pass to be sure the remote TCP received the acknowledgment of its connection termination request. [According to RFC 793 a connection can stay in TIME-WAIT for a maximum of four minutes known as a MSL (maximum segment lifetime).]

#### CLOSED

(both server and client) represents no connection state at all.

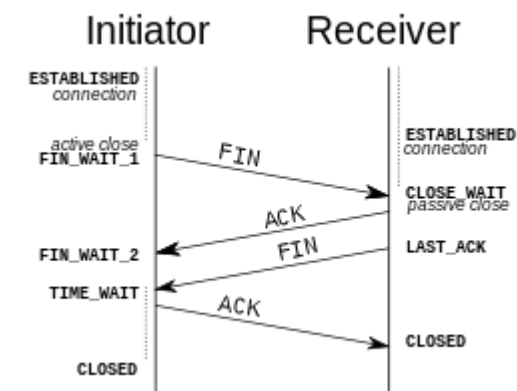
#### Connection establishment

To establish a connection, TCP uses a three-way handshake. Before a client attempts to connect with a server, the server must first bind to and listen at a port to open it up for connections: this is called a passive open. Once the passive open is established, a client may initiate an active open. To establish a connection, the three-way (or 3-step) handshake occurs:

1. **SYN:** The active open is performed by the client sending a SYN to the server. The client sets the segment's sequence number to a random value A.
2. **SYN-ACK:** In response, the server replies with a SYN-ACK. The acknowledgment number is set to one more than the received sequence number i.e. A+1, and the sequence number that the server chooses for the packet is another random number, B.
3. **ACK:** Finally, the client sends an ACK back to the server. The sequence number is set to the received acknowledgement value i.e. A+1, and the acknowledgement number is set to one more than the received sequence number i.e. B+1.

At this point, both the client and server have received an acknowledgment of the connection. The steps 1, 2 establish the connection parameter (sequence number) for one direction and it is acknowledged. The steps 2, 3 establish the connection parameter (sequence number) for the other direction and it is acknowledged. With these, a full-duplex communication is established.

#### Connection termination



### Connection termination

The connection termination phase uses a four-way handshake, with each side of the connection terminating independently. When an endpoint wishes to stop its half of the connection, it transmits a FIN packet, which the other end acknowledges with an ACK. Therefore, a typical tear-down requires a pair of FIN and ACK segments from each TCP endpoint. After both FIN/ACK exchanges are concluded, the side which sent the first FIN before receiving one waits for a timeout before finally closing the connection, during which time the local port is unavailable for new connections; this prevents confusion due to delayed packets being delivered during subsequent connections.

### 5.2 UDP

The **User Datagram Protocol (UDP)** is one of the core members of the Internet protocol suite (the set of network protocols used for the Internet). With UDP, computer applications can send messages, in this case referred to as *datagrams*, to other hosts on an Internet Protocol (IP) network without prior communications. UDP uses a simple transmission model with a minimum of protocol mechanism.<sup>[1]</sup> It has no handshaking dialogues, and thus exposes any unreliability of the underlying network protocol to the user's program. As this is normally IP over unreliable media, there is no guarantee of delivery, ordering or duplicate protection. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram.

UDP is suitable for purposes where error checking and correction is either not necessary or performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system.<sup>[2]</sup> If error correction facilities are needed at the network interface level, an application may use the Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP) which are designed for this purpose.

A number of UDP's attributes make it especially suited for certain applications.

- It is **transaction-oriented**, suitable for simple query-response protocols such as the

Domain Name System or the Network Time Protocol.

- It provides **datagrams**, suitable for modeling other protocols such as in IP tunneling or Remote Procedure Call and the Network File System.
- It is **simple**, suitable for bootstrapping or other purposes without a full protocol stack, such as the DHCP and Trivial File Transfer Protocol.
- It is **stateless**, suitable for very large numbers of clients, such as in streaming media applications for example IPTV
- The **lack of retransmission delays** makes it suitable for real-time applications such as Voice over IP, online games, and many protocols built on top of the Real Time Streaming Protocol.
- Works well in **unidirectional** communication, suitable for broadcast information such as in many kinds of service discovery and shared information such as broadcast time or Routing Information Protocol

### IP PROTOCOL

The **Internet Protocol (IP)** is the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries. Its routing function enables internetworking, and essentially establishes the Internet.

IP, as the primary protocol in the Internet layer of the Internet protocol suite, has the task of delivering packets from the source host to the destination host solely based on the IP addresses in the packet headers. For this purpose, IP defines packet structures that encapsulate the data to be delivered. It also defines addressing methods that are used to label the datagram with source and destination information.

### CONSISTENCY MODEL

- A Consistency Model is a contract between the software and the memory
  - it states that the memory will work correctly but only if the software obeys certain rules
- The issue is how we can state rules that are not too restrictive but allow fast execution in most common cases

- These models represent a more general view of sharing data than what we have seen so far!

### STRICT CONSISTENCY

- Strict consistency is the strictest model
  - a read returns the most recently written value (changes are instantaneous)

### SEQUENTIAL CONSISTENCY

- Sequential consistency (*serializability*): 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
- Sequential consistency is inefficient: we want to weaken the model further

#### Causal Consistency

- Causal consistency: writes that are potentially causally related must be seen by all processors in the same order. Concurrent writes may be seen in a different order on different machines
  - causally related writes: the write comes after a read that returned the value of the other write

#### Pipelined RAM (PRAM) or FIFO Consistency

- PRAM consistency is even more relaxed than causal consistency: writes from the same processor are received in order, but writes from distinct processors may be received in different orders by different processors **Processor consistency**: PRAM consistency plus writes to the same memory location are viewed everywhere in the same order

### WEAK CONSISTENCY

. Weak consistency uses synchronization variables to propagate writes to and from a machine at appropriate points:

- accesses to synchronization variables are sequentially consistent
- no access to a synchronization variable is allowed until all previous writes have completed in all processors

- no data access is allowed until all previous accesses to synchronization variables (by the same processor) have been performed

That is:

- accessing a synchronization variable “flushes the pipeline”
- at a synchronization point, all processors have consistent versions of data

### DISTRIBUTED SHARED MEMORY:

**Distributed Shared Memory (DSM)**, in Computer Architecture is a form of memory architecture where the (physically separate) memories can be addressed as one (logically shared) address space. Here, the term **shared** does not mean that there is a single centralized memory but **shared** essentially means that the address space is shared (same physical address on two processors refers to the same location in memory).<sup>[1]</sup> **Distributed Global Address Space (DGAS)**, is a similar term for a wide class of software and hardware implementations, in which each node of a cluster has access to shared memory in addition to each node's non-shared private memory. Software DSM systems can be implemented in an operating system (OS), or as a programming library and can be thought of as extensions of the underlying virtual memory architecture. When implemented in the OS, such systems are transparent to the developer; which means that the underlying distributed memory is completely hidden from the users. In contrast,

Distributed Shared Memory (DSM), in computer science, refers to a wide class of software and hardware implementations, in which each node of a cluster has access to a large shared memory in addition to each node's limited non-shared private memory.

Software DSM systems can be implemented within an operating system, or as a programming library. Software DSM systems implemented in the operating system can be thought of as extensions of the underlying virtual memory architecture. Such systems are transparent to the developer; which means that the underlying distributed memory is completely hidden from the users. In contrast, Software DSM systems implemented at the library or



language level are not transparent and developers usually have to program differently. However, these systems offer a more portable approach to DSM system implementation.

Software DSM systems also have the flexibility to organise the shared memory region in different ways. The page based approach organises shared memory into pages of fixed size. In contrast, the object based approach organises the shared memory region as an abstract space for storing sharable objects of variable sizes.

Shared memory architecture may involve separating memory into shared parts distributed amongst nodes and main memory; or distributing all memory between nodes. A coherence protocol, chosen in accordance with a consistency model, maintains memory coherence.

## VII. CONCLUSION

This paper has clearly demonstrate thatthat it is possible to build an efficient, high-performance distributed operating system. By having a microkernel, most of the key features are implemented as user processes.

The objectbased nature of the system, and the use of capabilities provide a unifying theme that holds the various pieces together. From some past years paper it has been conclude that it's perfmance remains good and reliable.

## REFERENCE

1. Bal, H.E., and Tanenbaum, A.S. Distributed Programming with Shared Data, *Computer Languages*, vol. 16, pp. 129-146, Feb. 1991.
2. Howard, J.H., Kazar, M.L., Menees, S.G., Nichols, D.A., Satyanarayanan, M., and Sidebotham, R.N.: Scale and Performance in a Distributed File System. *ACM Trans. on Comp. Syst.* 6, (Feb. 1988), pp. 55-81.
3. Douglass, F., Kaashoek, M.F., Tanenbaum, A.S., and Ousterhout, J.K.: A Comparison of Two Distributed Systems: Amoeba and Sprite. Report IR-230, Dept. of Mathematics and Computer Science, Vrije Universiteit, (Dec. 1990).
4. Mullender, S.J., van Rossum, G., Tanenbaum, A.S., van Renesse, R., van Staveren, J.M.

Amoeba — A Distributed Operating System for the 1990s. *IEEE Computer* 23, (May 1990), pp. 44-53.

## LIST OF WEB PAGES

1. [www.yahooanswers.com](http://www.yahooanswers.com)
2. [www.wikipedia.com](http://www.wikipedia.com)
3. [www.ghacks.com](http://www.ghacks.com)
4. [www.ehow.com](http://www.ehow.com)