St. Francis Institute of Technology
Department of Computer Engineering

Academic Year: 2021-2022 **Semester: VIII**
Subject: Distributed Computing **Class / Branch / Division: BE/CMPN/A**
Name: Rebecca Dias Roll Number:18

Experiment No: 02

Aim: Case study on Distributed Operating System.

Pre-requisites: Networking

Theory:

1. What is DOS?

A distributed operating system (DOS) is an essential type of operating system. Distributed systems use many central processors to serve multiple real-time applications and users. As a result, data processing jobs are distributed between the processors.

It connects multiple computers via a single communication channel. Furthermore, each of these systems has its own processor and memory. Additionally, these CPUs communicate via high-speed buses or telephone lines. Individual systems that communicate via a single channel are regarded as a single entity. They're also known as loosely coupled systems.

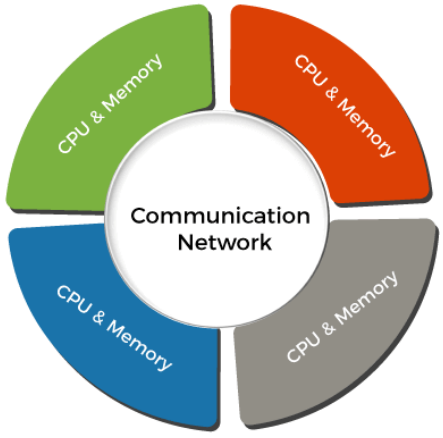

Fig No:2.1 Distributed Operating System

This operating system consists of numerous computers, nodes, and sites joined together via LAN/WAN lines. It enables the distribution of full systems on a couple of center processors, and it supports many real-time products and different users. Distributed operating systems can share their computing resources and I/O files while providing users with virtual machine abstraction.

2. Need for DOS.

An important goal of a distributed system is to make it easy for users (and applications) to access and share remote resources. … For example, it is cheaper to have a single high-end reliable storage facility be shared then having to buy and maintain storage for each user separately.

We need Distributed Operating System because:

- To decrease the load on the single system.
- If one system stops it should not affect the other system.
- A system that shares a workload that makes calculations easy.
- The size of the system can be set according to requirements.

3. Features and Applications of DOS

There are various important goals that must be met to build a distributed system worth the effort. A **distributed system** should easily connect users to resources, it should hide the fact that resources are distributed across a network, must be open, and must be scalable.

i. Connecting Users and Resources:

The main goal of a distributed system is to make it easy for users to access remote resources, and to share them with other users in a controlled manner. Resources can be virtually anything, typical examples of resources are printers, storage facilities, data, files, web pages, and networks. There are many reasons for sharing resources. One reason is economics.

ii. Transparency:

An important goal of a distributed system is to hide the fact that its process and resources are physically distributed across multiple computers. A distributed system that is capable of presenting itself to users and applications such that it is only a single computer system is called transparent.

The concept of transparency can be applied to many aspects of a distributed system as shown in table.

Different Forms of Transparency –

Sr.No.	Transparency	Description
(1)	Access	Hide data representation.
(2)	Location	Hide location
(3)	Migration	Move place information.
(4)	Relocation	Hide moved place relocation.
(5)	Replication	Hide that a resource is replication.
(6)	Concurrency	Shared data bases access
(7)	Failure	Hide fact about resource failure.
(8)	Persistence	Hide fact about memory location.

iii. Openness:

Another important goal of distributed systems is openness. An open distributed system is a system that offers services in standards that describable the syntax and semantics of those service instances, standard rules in computer networks control

the format, content, and meaning of messages sent and received. Such rules are formalized in the protocols. In distributed systems, services are typically specified through interfaces, often called interface definition languages (IDL). Interface definitions written in IDL almost always capture only the syntax of services. They accurately specify the names of functions that are available with the types of parameters, return values, possible exceptions that can be raised and so on.

iv. Scalability:

The uncertain trend in distributed systems is towards larger systems. This observation has implications for distributed file system design. Algorithms that work well for systems with 100 machines can work for systems with 1000 machines and none at all for systems with 10, 000 machines. for starters, the centralized algorithm does not scale well. If opening a file requires contacting a single centralized server to record the fact that the file is open then the server will eventually become a bottleneck as the system grows.

v. Reliability:

The main goal of building distributed systems was to make them more reliable than single processor systems. The idea is that if some machine goes down, some other machine gets used to it. In other words, theoretically the reliability of the overall system can be a Boolean OR of the component reliability. For example, with four file servers, each with a 0.95 chance of being up at any instant, the probability of all four being down simultaneously is 0.000006, so the probability of at least one being available is (1-0.000006) = 0.999994, far better than any individual server.

vi. Performance:

Building a transparent, flexible, reliable distributed system is useless if it is slowed like molasses. In particular application on a distributed system, it should not deteriorate better than running some application on a single processor. Various performance metrics can be used. Response time is one, but so are throughput, system utilization, and amount of network capacity consumed. Furthermore, the results of any benchmark are often highly dependent on the nature of the benchmark. A benchmark involves a large number of independent highly CPU-bound computations which give radically different results than a benchmark that consists of scanning a single large file for same pattern.

Applications of Distributed Operating System

The applications of distributed OS are as follows −

- Internet Technology
- Distributed databases System
- Air Traffic Control System
- Airline reservation Control systems
- Peep-to-peer networks system
- Telecommunication networks

- Scientific Computing System
- Cluster Computing
- Grid Computing
- Data rendering

4. Case study amoeba

A) Introduction to Amoeba

Originated at a university in Holland, 1981 Currently used in various EU countries Built from the ground up. UNIX emulation added later - Goal was to build a transparent distributed operating system resource, regardless of their location, are managed by the system, and the user is unaware of where processes are actually run.

B) The Amoeba System Architecture

Assumes that a large number of CPUs are available and that each CPU ha 10s of Mb of memory CPUs are organised into processor pools

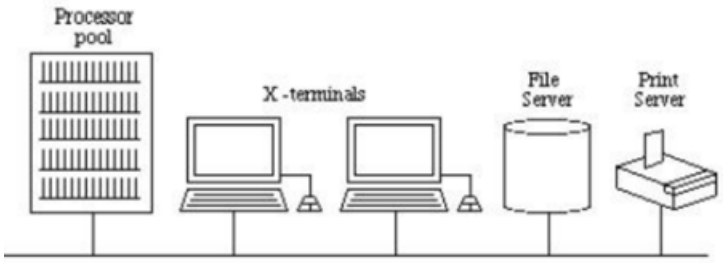

Fig No:2.2 System Architecture

CPUs do not need to be of the same architecture (can mix SPARC, Motorola PowerPC, 680x0, Intel, Pentium, etc.). When a user types a command, system determines which CPU(s) to execute it on CPUs can be timeshared. Terminals are X-terminals or PCs running X emulators. The processor pool doesn't have to be composed of CPU boards enclosed in a cabinet, they can be on PCs, etc., in different rooms, countries. Some servers (eg, file servers) run on dedicated processors, because they need to be available all the time.

C) The Amoeba Microkernel

The Amoeba microkernel is used on all terminals (with an on-board processor), processors, and servers The microkernel manages processes and threads, provides low-level memory management support, supports inter-process communication (point-to-point and group) and handles low-level I/O for the devices attached to the machine.

D) The Amoeba Servers: Introduction

OS functionality not provided by the microkernel is performed by Amoeba servers , To use a server, the client calls a stub procedure which Marshalls parameters, sends the message, and blocks until the result comes back

i) Server Basics

- Amoeba uses capabilities. Every OS data structure is an object, managed by a server.
- To perform an operation on an object, a client performs an RPC with the appropriate server.
- specifying the object, the operation to be performed and any parameters needed.
- The operation is transparent (client does not know where server is, nor how the operation is performed)

ii) Capabilities

- To create an object the client performs an RPC with the server.
- Server creates the object and returns a capability, to use the object in the future, the client must present the correct capability.
- The check field is used to protect the capability against forgery.

iii) Object protection

- When an object is created, server generates random check field, which it stores both in the capability and in its own tables.
- The rights bits in the capability are set to on.
- The server sends the owner capability back to the client Creating a capability with restricted rights.

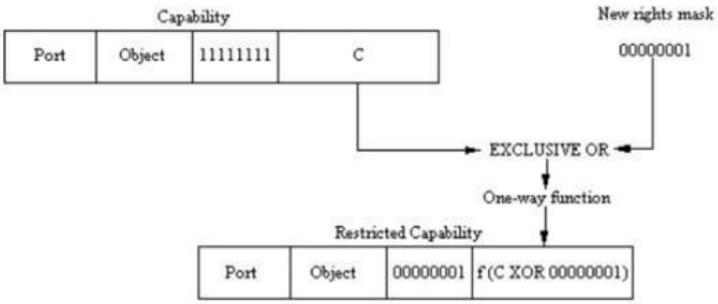

Fig No:2.3 Ameoba server

iv) Process Management

- All processes are objects protected by capabilities Processes are managed at 3 levels by process servers, part of the microkernel by library procedures which act as interfaces by the run server, which decides where to run the processes Process management uses process descriptors
- Contains:
- platform description process owner's capability etc

v) Memory Management

- Designed with performance, simplicity and economics in mind Process occupies contiguous segments in memory. All of a process is constantly in memory. Process is never swapped out or paged.
- Communication
- Point-to-point (RPC) and Group

E) The Amoeba Servers

i) The File System

- Consists of the Bullet (File) Server, the Directory Server, and the Replication Server

ii) The Bullet Server

- Designed to run on machines with large amounts of RAM and huge local disks
- Used for file storage
- Client process creates a file using the create call
- Bullet server returns a capability that can be used to read the file with
- Files are immutable, and file size is known at file creation time.
- Contiguous allocation policies are used.

iii) The Directory Server

- Used for file naming
- Maps from ASCII names to capabilities
- Directories also protected by capabilities
- Directory server can be used to name ANY object, not just files and directories

iv) The Replication Server

- Used for fault tolerance and performance
- Replication server creates copies of files, when it has time

F) Other Amoeba Servers

i) The Run Server

- When user types a command, two decisions have to be made.
- On which architecture should the process be run?
- Which processor should be chosen?
- Run server manages the processor pools

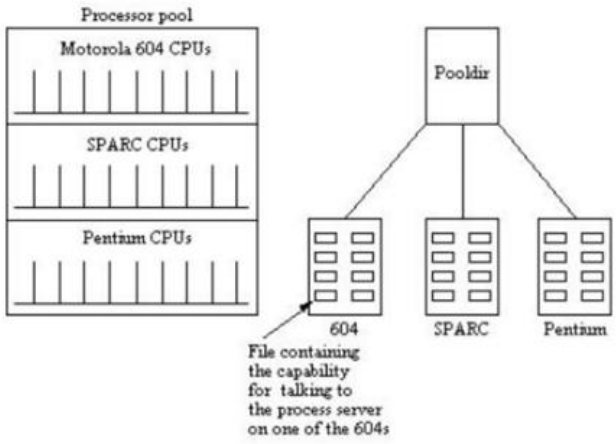

Fig No:2.4 Run Server

- Uses processes process descriptor to identify appropriate target architecture
- Checks which of the available processors have sufficient memory to run the process Estimates which of the remaining processor has the most available compute power

ii) The Boot Server

- Provides a degree of fault tolerance Ensures that servers are up and running
- If it discovers that a server has crashed, it attempts to restart it, otherwise selects another processor to provide the service
- Boot server can be replicated to guard against its own failure

Conclusion:

Thus, the case study completed on distributed operating system. The Amoeba OS is the distributed operating system. The aim of the Amoeba project is to build a timesharing system that makes an entire network of computers appear to the user as a single machine.

Reference:

1) https://www.javatpoint.com/distributed-operating-system

2) https://www.tutorialspoint.com/distributed-operating-system

3) https://www.geeksforgeeks.org/features-of-distributed-operating-system/

4) https://www.tutorialspoint.com/difference-between-network-operating-system-and-distributed-operating-system

5) https://www.tutorialspoint.com/what-is-a-distributed-operating-system