Unit-9

Distributed Operating System

Introduction

A distributed operating system is an operating system that runs on several machines whose purpose is to provide a useful set of services, generally to make the collection of machines behave more like a single machine. The distributed operating system plays the same role in making the collective resources of the machines more usable that a typical single-machine operating system plays in making that machine's resources more usable. Usually, the machines controlled by a distributed operating system are connected by a relatively high-quality network, such as a high-speed local area network. Most commonly, the participating nodes of the system are in a relatively small geographical areas, something between an office and a campus.

The Distributed Os involves a collection of autonomous computer systems, capable of communicating and cooperating with each other through a LAN / WAN. A Distributed Os provides a virtual machine abstraction to its users and wide sharing of resources like as computational capacity, I/O and files etc. A distributed operating system (DOS), are systems which model where distributed applications are running on multiple computers, linked by communications.

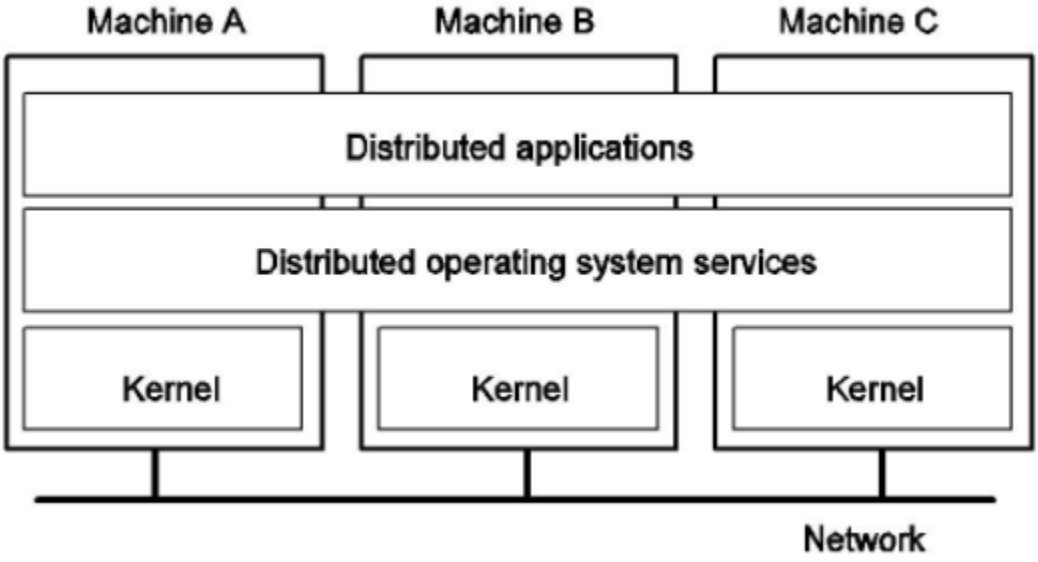


Fig: Distributed system

Applications of Distributed Operating Systems:

- Telecommunication Networks
- Network Applications
- Real Time Process Control
- Parallel Computation

Advantages of Distributed System

1. **Scalability:**
   As computing occurs on each node independently, it is simple and inexpensive to add more nodes and functionality as required.
2. **Reliability:**
   Most distributed systems are made from many nodes that work together which ultimately make them fault tolerant. The system doesn't experience any disruptions if a single machine fails.
3. **Performance:**
   These systems are regarded to be very efficient as the work load can be broken up and sent to multiple machines, therefore reducing data processing.
4. **Data sharing:**
   Nodes can easily share data with other nodes as they are connected with each other.
5. **No domino effect in case of a node failure:**
   The failure of one node in a DOS does not have a domino effect and enables all other nodes fail. Other nodes can still communicate with each other despite the failure.
6. **Shareable:**
   Resources, for instance like printers, can be shared with multiple nodes rather than just being constrained to just one node.

Disadvantages of Distributed Operating System

1. **Scheduling:**
   The system has to decide which jobs need to be executed, when they should be executed, and where they should be executed. A scheduler will have limitations, this may lead to under-utilised hardware and unpredictable runtimes.
2. **Latency:**
   The more widely distributed a system is the more latency can be experienced with communications. This therefore results in teams/developers to make tradeoffs between availability, consistency and latency.
3. **Observability:**
   It can be a real challenge to gather, process, present, and monitor hardware usage metrics for large clusters.
4. **Security:**
   It is difficult to place adequate security in DOS, as the nodes and the connections need to be secured.
5. **Data loss:**
   Some data/messages may be lost in the network while moving from one node to another.
6. **Complicated database:**
   In comparison to a single user system, the database connected to a DOS is relatively complicated and difficult to handle.
7. **Overloading:**
   If multiple nodes in DOS send data all at once, then the system network may become overloaded.

8. **Expensive:**
   These systems are not readily available, as they are regarded to be very expensive.
9. **Complex software:**
   Underlying software is highly complex and is not understood very well compared to other systems.

Network Operating System

A network operating system (NOS) is a computer operating system (OS) that is designed primarily to support workstations, personal computers and, in some instances, older terminals that are connected on a local area network (LAN).

Network Operating System runs on a server and gives the server the capability to manage data, users, groups, security, applications, and other networking functions. The basic purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks.

Some examples of network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.

Advantages

1. Centralized servers are highly stable.
2. Security is server managed.
3. Upgradation of new technologies and hardware can be easily integrated into the system.
4. It is possible to remote access to servers from different locations and types of systems.

Disadvantages

1. High cost of buying and running a server.
2. Dependency on a central location for most operations.
3. Regular maintenance and updates are required.

Difference between network operating system and distributed operating system

The main difference between these two operating systems is that in network operating system each node or system can have its own operating system on the other hand in distribute operating each node or system have same operating system which is opposite to the network operating system.

Network Operating System	Distributed Operating System
1. Network Operating System's main objective is to provide the local services to remote client.	1. Distributed Operating System's main objective is to manage the hardware resources.
2. In Network Operating System, Communication takes place on the basis of files.	2. In Distributed Operating System, Communication takes place on the basis of messages and shared memory.
3. Network Operating System is more scalable than Distributed Operating System.	3. Distributed Operating System is less scalable than Network Operating System.
4. In Network Operating System, fault tolerance is less.	4. While in Distributed Operating System, fault tolerance is high.
5. Rate of autonomy in Network Operating System is high.	5. While The rate of autonomy in Distributed Operating System is less.
6. Ease of implementation in Network Operating System is also high.	6. While in Distributed Operating System Ease of implementation is less.
7. In Network Operating System, All nodes can have different operating system.	7. While in Distributed Operating System, All nodes have same operating system.

Hardware and Software Concepts

All distributed systems consist of multiple CPUs. There are several different ways the hardware can be arranged. The important thing related to hardware is that how they are interconnected and how they communicate with each other. It is important to take a deep look at distributed system hardware, in particular, how the machines are connected together and how they interact.

Many classification schemes for multiple CPU computer systems have been proposed over the years, but none of them have really implemented. Still, the most commonly used taxonomy is Flynn's (1972), but it was in basic stage. In this scheme, Flynn took only two things to consider i.e. the number of instruction streams and the number of data streams.

1. Single Instruction, Single Data Stream (SISD)

A computer with a single instruction stream and a single data stream is called **SISD**. All traditional uni-processor computers (i.e., those having only one CPU) fall under this category, from personal computers to large mainframes. SISD flow concept is given in the figure below.

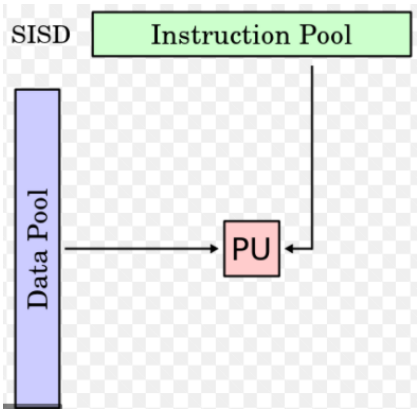


Fig :SISD Flow structure

2. **Single Instruction, Multiple Data Stream (SIMD)**
   It uses an array of processors with only one instruction unit that fetches an instruction, and multiple data units which work in parallel. These machines are used where there need to apply the same instruction for multiple data example, adding up all the elements of 64 independent vectors. Some supercomputers are SIMD. Figure below shows the SIMD flow structure.

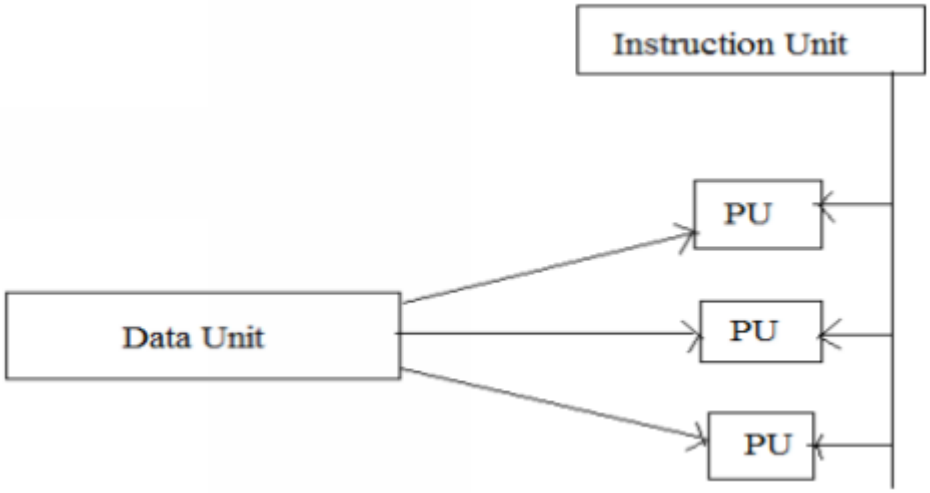


Fig: SIMD Flow structure

3. **Multiple Instructions, Single Data Stream (MISD)**
   This structure was worked when there are multiple different instructions to operate on the same type of data. In general MISD architecture is not use more in practical.

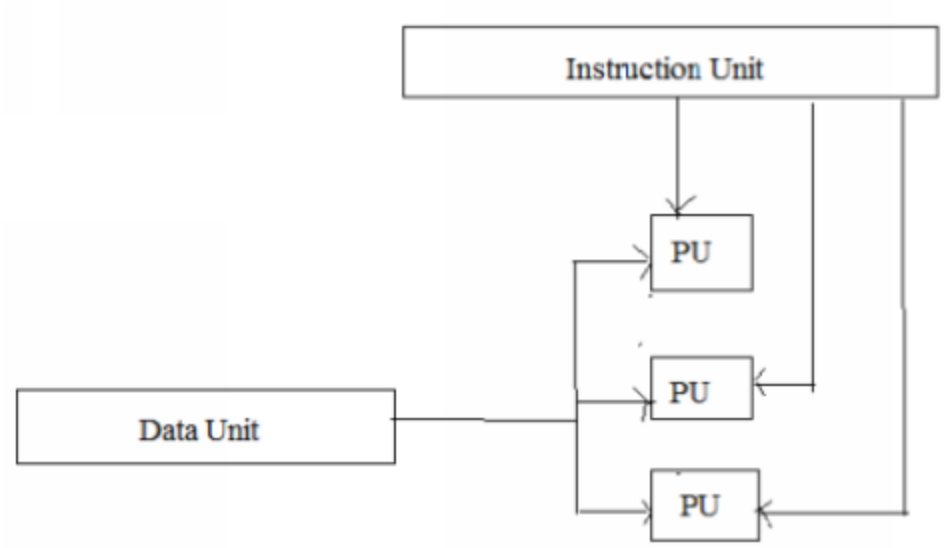


Fig: MISD Flow structure

4. **Multiple Instruction, Multiple Data Stream (MISD)**

The next category is MIMD, which has multiple instructions performances on multiple data units. This means a group of independent computers; each has its own program counter, program, and data. All distributed systems are MIMD, so this classification system is not more useful for simple purposes.

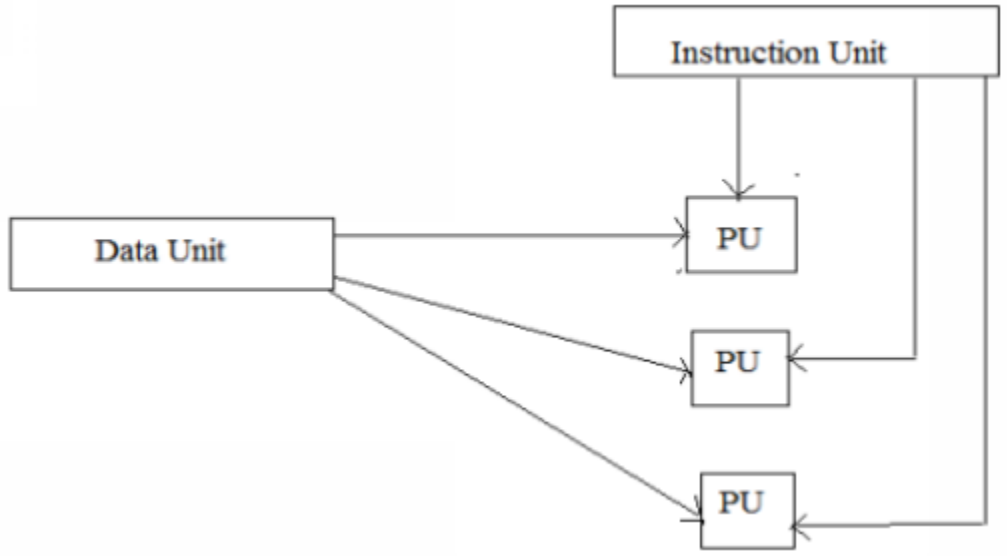


Fig: MIMD Flow structure

Software Concept

Although the hardware is important, the software is even more important. The image that a system presents to its users and how they think about the system is largely determined by the operating system software, not the hardware.

There are basic two types of operating system namely tightly coupled operating system and loosely couple operating system, for multiprocessor and multicomputer. Loosely coupled

software allows machines and users of a distributed system to be fundamentally independent of one another. Consider a group of personal computers, each of which has its own CPU, its own memory, its own hard disk, and its own operating system, but which share some resources, such as laser printers and databases over a LAN. This system is loosely coupled, since the individual machines are clearly distinguishable, each with its own job to do. If the network should go down for some reason, the individual machines can still continue to run to a considerable degree, although some functionality may be lost.

For tightly coupled system consider a multiprocessor dedicated to running a single chess program in parallel. Each CPU is assigned a board to evaluate and it spends its time examining that board and all the boards that can be generated from it. When the evaluation is finished, the CPU reports back the results and is given a new board to work on.

Communication in Distributed Systems

1. Shared memory

Distributed shared memory (DSM) is a form of memory architecture where physically separated 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 that the address space is "shared" (same physical address on two processors refers to the same location in memory).

A distributed-memory system, often called a multicomputer, consists of multiple independent processing nodes with local memory modules which is connected by a general interconnection network. Software DSM systems can be implemented in an operating system, or as a programming library and can be thought of as extensions of the underlying virtual memory architecture. When implemented in the operating system, 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 them differently. However, these systems offer a more portable approach to DSM system implementations. A distributed shared memory system implements the shared-memory model on a physically distributed memory system.

The syntax used for DSM is the same as that of normal centralized memory multiprocessor systems.

read(shared_variable)

write(data, shared_variable)

The read() primitive requires the name of the variable to be read as its argument and the write() primitive requires the data and the name of the variable to which the data is to be written. The operating system locates the variable through its virtual address and, if necessary, moves the portion of memory containing the variable to the machine requiring it.

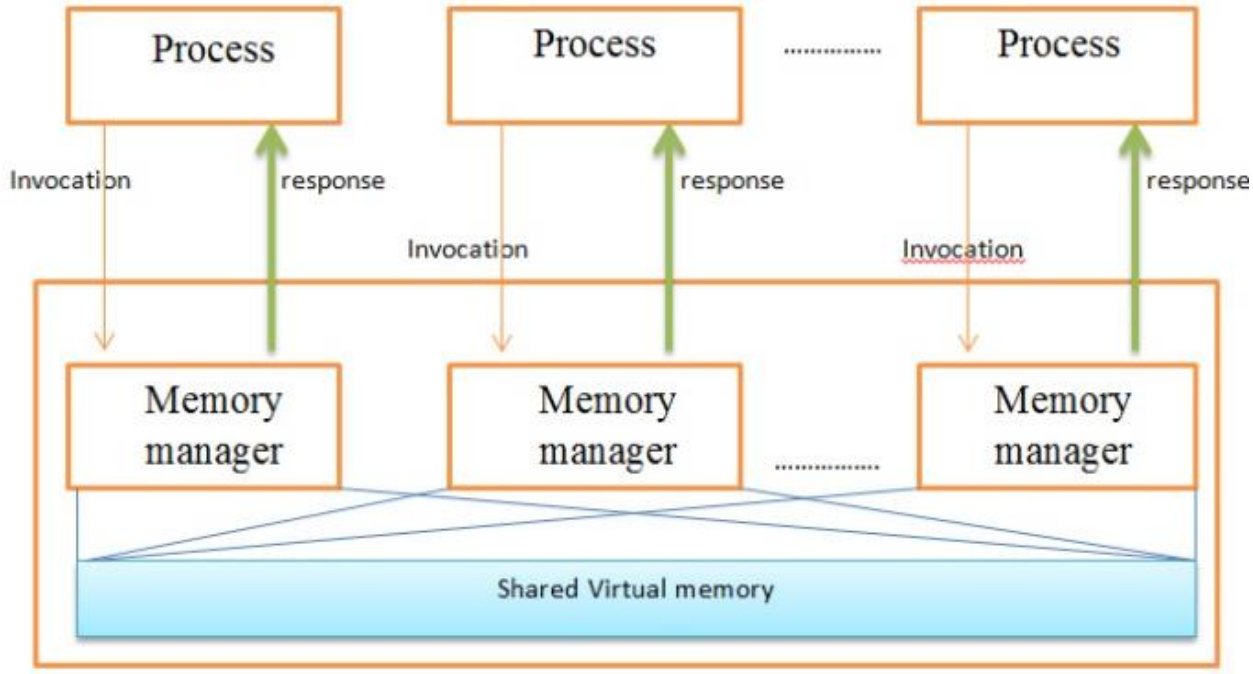


Fig: Distributed shared memory

Advantages

- Scales well with a large number of nodes
- Message passing is hidden
- Can handle complex and large databases without replication or sending the data to processes
- Generally cheaper than using a multiprocessor system
- Provides large virtual memory space
- Programs are more portable due to common programming interfaces
- Shield programmers from sending or receiving primitives

Disadvantages

- Generally slower to access than non-distributed shared memory
- Must provide additional protection against simultaneous accesses to shared data
- May incur a performance penalty
- Little programmer control over actual messages being generated
- Programmers need to understand consistency models, to write correct programs
- DSM implementations use asynchronous message-passing, and hence cannot be more efficient than message-passing implementations

2. Message Passing

Message passing model allows multiple processes to read and write data to the message queue without being connected to each other. Messages are stored on the queue until their recipient retrieves them. Message queues are quite useful for interprocess communication and are used by most operating systems.

Message passing is the basis of most interprocess communication in distributed systems. It is at the lowest level of abstraction and requires the application programmer to be able to identify the destination process, the message, the source process and the data types expected from these processes.

Communication in the message passing paradigm, in its simplest form, is performed using the send() and receive() primitives. The syntax is generally of the form:

send(receiver, message)

receive(sender, message)

The send() primitive requires the name of the destination process and the message data as parameters. The addition of the name of the sender as a parameter for the send() primitive would enable the receiver to acknowledge the message. The receive() primitive requires the name of the anticipated sender and should provide a storage buffer for the message.

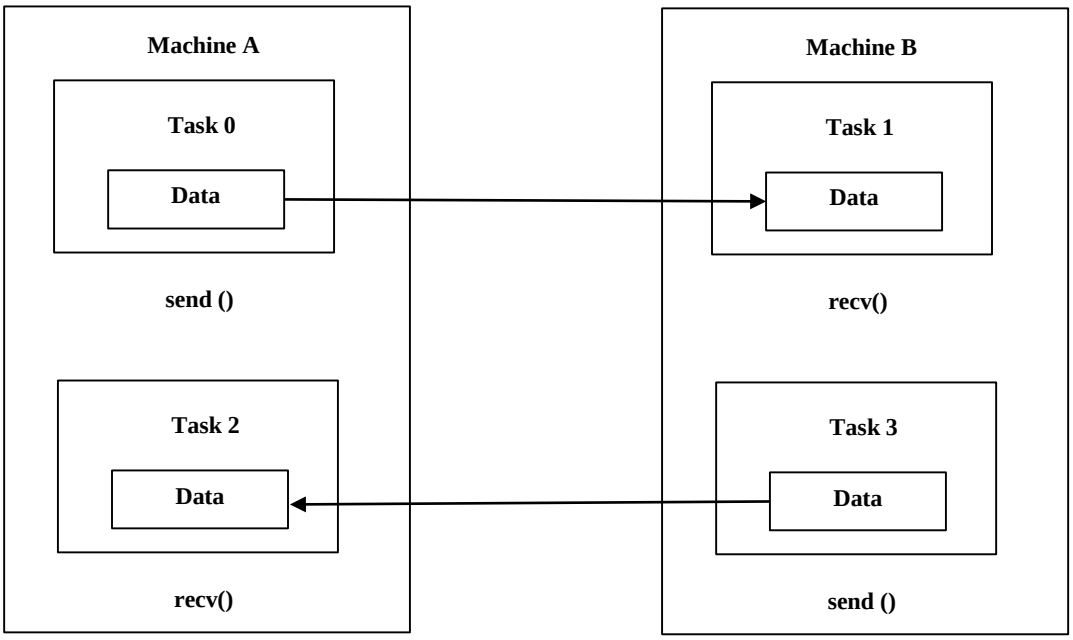
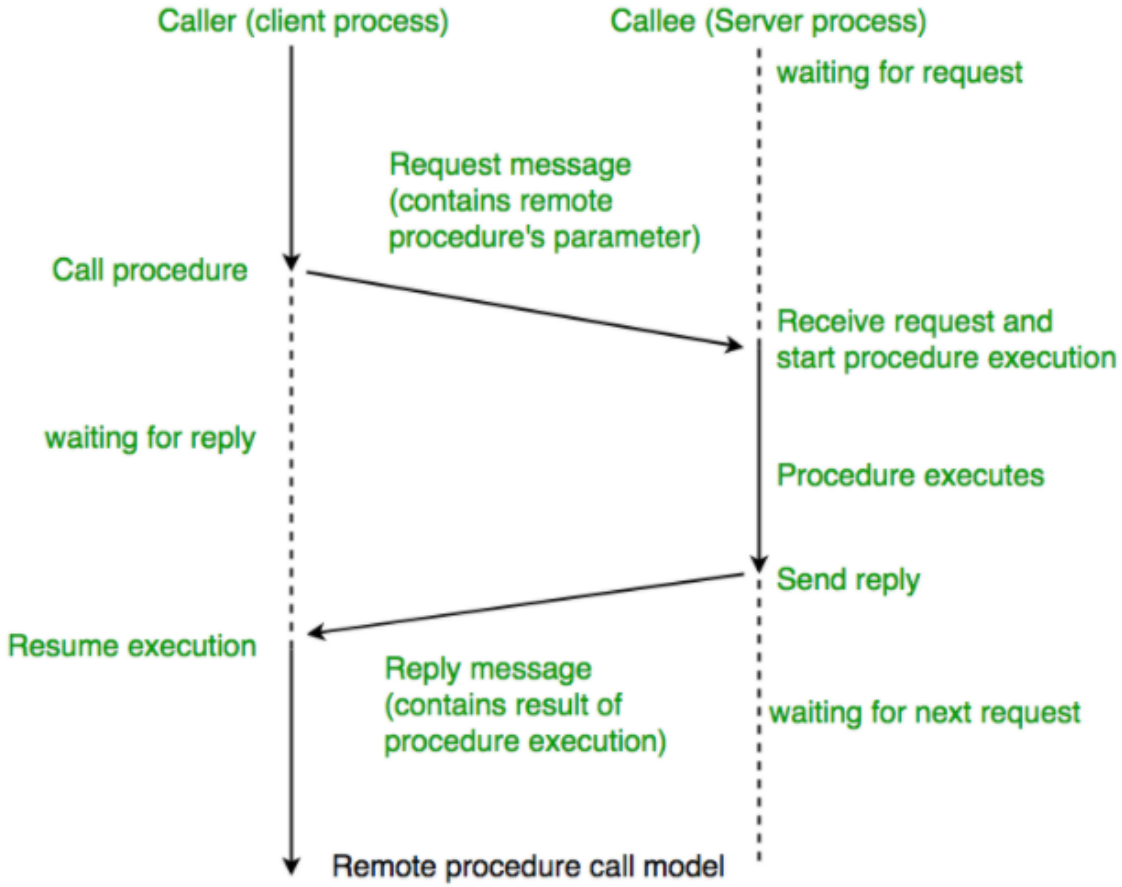


Fig: Message Passing

3. Remote Procedure Call

Message passing leaves the programmer with the burden of the explicit control of the movement of data. Remote procedure calls (RPC) relieve this burden by increasing the level of abstraction and providing semantics similar to a local procedure call.

Remote Procedure Call (RPC) is a powerful technique for constructing distributed, client-server-based applications. It is based on extending the conventional local procedure calling so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them.



The syntax of a remote procedure call is generally of the form:

call procedure_name(value_arguments; result_arguments)

The client process blocks at the call() until the reply is received. The remote procedure is the server processes which has already begun executing on a remote machine. It blocks at the receive() until

it receives a message and parameters from the sender. The server then sends a reply() when it has finished its task. The syntax is as follows:

receive procedure_name(in value_parameters; out result_parameters)

reply(caller, result_parameters)

Clock Synchronization

Distributed System is a collection of computers connected via the high speed communication network. In the distributed system, the hardware and software components communicate and coordinate their actions by message passing. Each node in distributed systems can share their resources with other nodes. So, there is need of proper allocation of resources to preserve the state of resources and help coordinate between the several processes. To resolve such conflicts, synchronization is used. Synchronization in distributed systems is achieved via clocks.

The clock synchronization can be achieved by 2 ways:

External and Internal Clock Synchronization.

1. **External clock synchronization** is the one in which an external reference clock is present. It is used as a reference and the nodes in the system can set and adjust their time accordingly.
2. **Internal clock synchronization** is the one in which each node shares its time with other nodes and all the nodes set and adjust their times accordingly.

There are 2 types of clock synchronization algorithms: Centralized and Distributed.

1. **Centralized** is the one in which a time server is used as a reference. The single time server propagates its time to the nodes and all the nodes adjust the time accordingly. It is dependent on single time server so if that node fails, the whole system will lose synchronization. Examples of centralized are- Berkeley Algorithm, Passive Time Server, Active Time Server etc.
2. **Distributed** is the one in which there is no centralized time server present. Instead, the nodes adjust their time by using their local time and then, taking the average of the differences of time with other nodes. Distributed algorithms overcome the issue of centralized algorithms like the scalability and single point failure. Examples of Distributed algorithms are – Global Averaging Algorithm, Localized Averaging Algorithm, NTP (Network time protocol) etc.

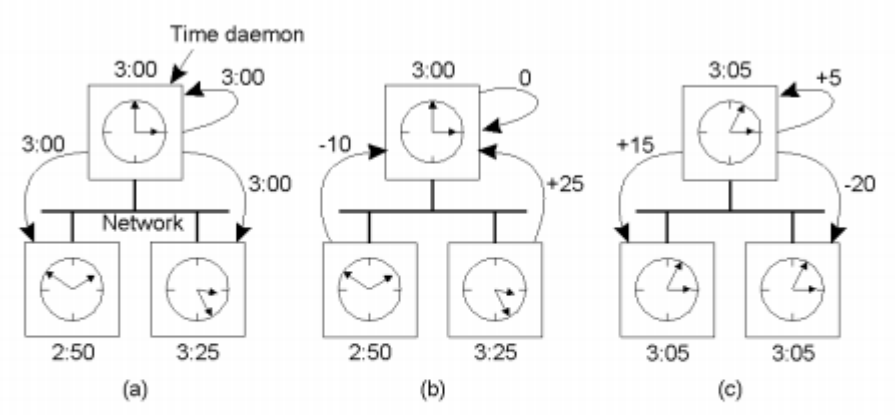
Physical clock synchronization

In distributed systems, this is not the case. Unfortunately, each system has its own timer that drives its clock. These timers are based either on the oscillation of a quartz crytal, or equivalent IC. Although they are reasonably precise, stable, and accurate, they are not perfect. This means that the clocks will drift away from the true time. Each timer has different characteristics --

characteristics that might change with time, temperature, &c. This implies that each systems time will drift away from the true time at a different rate -- and perhaps in a different direction (slow or fast).

Berkeley algorithm

- The server polls each machine periodically, asking it for the time.
- Based on the answer it computes for the average time and tells all the other machine to advance their clocks to the new time or slow their clocks down until some specific reduction has been achieved.
- The time daemon's time must be set manually by operator periodically.



a) The time daemon asks all the other machines for their clock values
b) The machines answer
c) The time daemon tells everyone how to adjust their clock

Logical Clock

Logical Clocks refer to implementing a protocol on all machines within your distributed system, so that the machines are able to maintain consistent ordering of events within some virtual timespan. A logical clock is a mechanism for capturing chronological and causal relationships in a distributed system. Distributed systems may have no physically synchronous global clock, so a logical clock allows global ordering on events from different processes in such systems.

Lamport's algorithm

- Each message carries a timestamp of the sender's clock
- When a message arrives :
  - If the receiver's clock < message timestamp
    set system clock to (message timestamp + 1)

    else do nothing

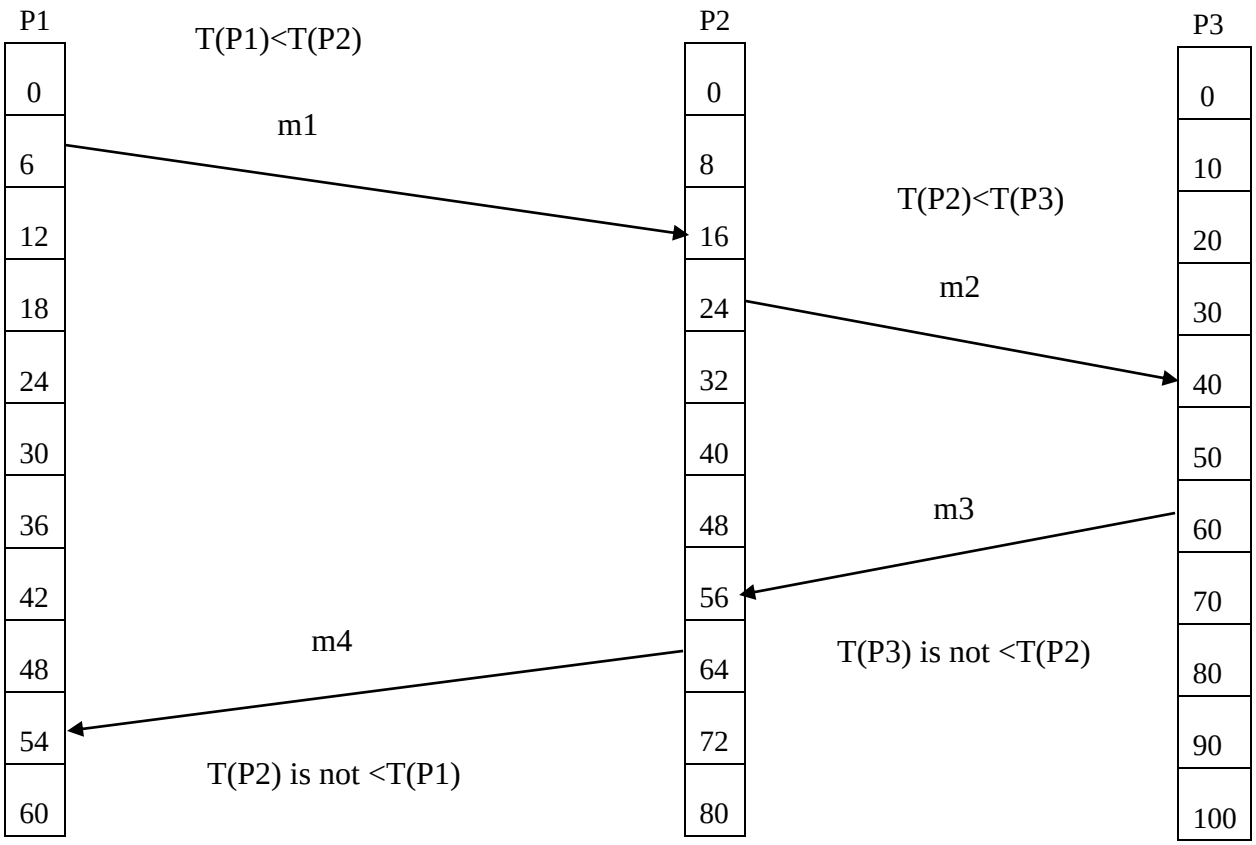
- Clock must be advanced between any two events in the same process.

The algorithm for sending is:

```
# event is known
time = time + 1;
# event happens
send(message, time);
```

The algorithm for receiving a message is:

```
(message, time_stamp) = receive();
time = max(time_stamp, time) + 1;
```



Correcting the clock