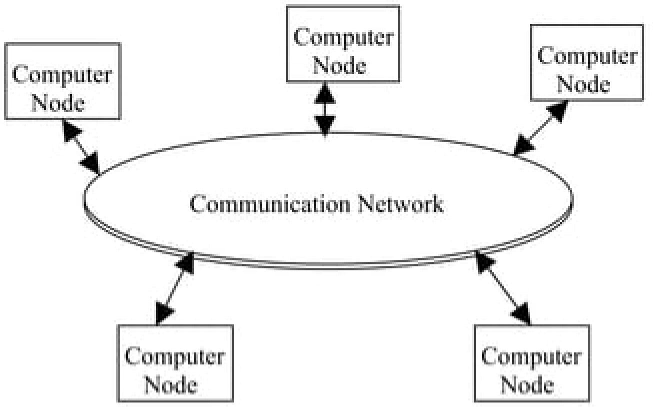
Architecture of Distributed Systems

Introduction

A Distributed System (DS) is one in which

- Hardware and software components, located at remote networked computers, coordinate and communicate their actions only by passing messages. Any distance may separate computers in the network.
- Sharing of resources is the main motivation of distributed systems. Resources may be managed by servers and accessed by clients, or they may be encapsulated as objects and accessed by client objects.

A distributed operating system runs on multiple independent computers, connected through communication network, but appears to its users as a single virtual machine and runs its own OS.



Architecture of a Distributed System

Each computer node has its own memory. Examples of Distributed Systems are: Internet, Intranet, Mobile and ubiquitous computing. As a consequence of this definition, the characteristics of distributed systems or networked-computers are:

- **Concurrency:** How to handle the sharing of resources between clients? Execution of concurrent programs share resources: e.g. web pages, files, etc.

- **No global clock:** In a distributed system, computers are connected through network and have their own clocks. Communication between programs is only through messages and their coordination depends on time. Every client's (computer) perception of time is different. Accurate time synchronization is not possible in DS. How to synchronize activities?

- **Independent Failure:** Distributed systems should be planned for the consequences of possible failures of its components. How to handle a failure in the network or in a particular client? Other clients might not be immediately aware of a failure. Each component of the distributed system can fail independently leaving others still running. Faults in the network results in isolation of the failed component only, but system continue running.

Characteristics of Distributed Systems

A Distributed System has the following characteristics:

- It consists of several independent computers connected through communication network,
- The computers communicate with each other by exchanging message over a communication network.
- Each computer has its own memory, clock and runs its own operating system.
- Each computer has its own resources, called local resources
- Remote resources are accessed through the network

We will discuss issues that arise in the design of a distributed operating system and how communication takes place between the programs running on the connected computers.

Motivation:

The prime motivation of distributed systems is to share *resources*. A resource is an entity that can be usefully shared among users. Any hardware or software entity is a resource. We use shared resources all the time. Resources are managed by a *service*. A *service* is managed by one or more *servers*, which provide access to a set of resources to *clients* via a set of well-defined operations (an *interface*).

The motivation behind the development of Distributed Systems was:

- Users desire to have computational power at low cost.
- Need of the people working in a group to communicate with each other
- Sharing of information (data)
- Sharing of expensive computer resources.

Designing such systems became possible with the availability of cheap and powerful microprocessors and advances in communication technology. When a few powerful workstations are interconnected and can communicate with each other, the total computing power available in such a system can be enormous.

Advantages of Distributed Systems are:

- **Resource Sharing**: Due to communication between connected computers resources can be shared among computers.

- **Enhance Performance**: This is due to the fact that many tasks can be executed concurrently at different computers. Load distribution among computers can further improve response time.

- **Improved reliability and availability:** Increased reliability is due to the fact that if few computers fail others are available and hence the system continues.

- **Modular expandability**: New hardware and software resources can be added without replacing the existing resources.

Inherent Limitations of Distributed Systems

The lack of *common memory* and system wide *common clock* is an inherent problem in distributed systems.

- Without a shared memory, up-to-date information about the state of the system is not available to every process via a simple memory lookup. The state information must therefore be collected through communication.

- In the absence of global time, it becomes difficult to talk about temporal order of events. The combination of unpredictable communication delays and the lack of global time in a distributed system make it difficult to know how up-to-date collected state information really is.

System Architecture Types

Distributed systems can be modeled into several types. Various models are used for building distributed computing systems. These models can be broadly classified into five categories, and they are described below:

1. Mini Computer Model,
2. Workstation Model,
3. Workstation Server Model,
4. Processor Pool Model, and
5. Hybrid Model.

1. Mini Computer Model

In this model, the distributed system consists of several minicomputers. Each computer supports multiple users and provides access to remote resources. The ratio of processors to users is normally less than one.

Minicomputer model is a simple extension of the centralized time-sharing system. As shown in Figure 1, a distributed computing system based on this model consists of a few minicomputers. They may be large supercomputers as well interconnected by a communication network. Each minicomputer usually has multiple users simultaneously logged on to it.

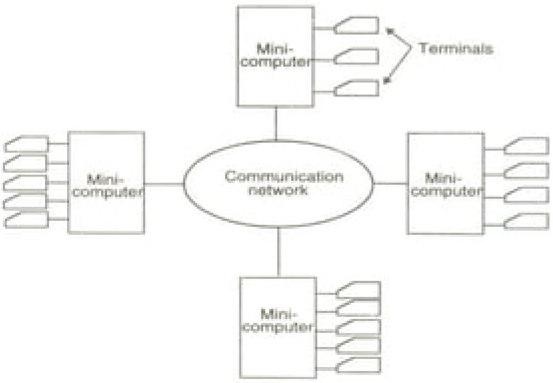


Figure 1: The distributed system based on minicomputer model

Several interactive terminals are connected to each minicomputer. Each user is logged on to one specific minicomputer, with remote access to other minicomputers. The network allows a user to access |remote resources that are available on some machine other that the one on to which the user is currently logged.

The minicomputer model may be used when resource sharing (such as sharing of information databases of different types, with each type of database located on a different machine) with remote users is desired. The early ARpAnet is an example of a distributed computing system based on the minicomputer model.

2. Workstation Model

In this model, the distributed system consists of several workstations; every user has a workstation where user's work is performed. With the help of distributed file system, a user can access data regardless of the location of the data. The ratio of processors to users is normally one. The workstations are independent computers with memory, hard disks, keyboard and console. Workstations are connected with each other through communication network.

As shown in Figure 2, a distributed computing system based on the workstation model consists of several workstations interconnected by a communication network. A company's office or a university department may have several workstations scattered throughout a building or compass each workstation equipped with its own disk and serving as a single-user computer.

It has been often found that in such an environment at any one time (especially at night), a significant proportion of the workstations are idle, resulting in the waste of large amounts of CPU time. Therefore, the idea of the workstation model is to interconnect all these workstations by a high-speed LAN so that idle workstations may be used to process jobs of users who are logged onto other workstations and do not have sufficient processing power at their own workstations to get their jobs processed efficiently.

In this model a user logs onto one of the workstations called his or her *home workstation* and submits jobs for execution. When the system finds that the user's workstation does not have sufficient processing power for executing the processes of the submitted jobs efficiently, it transfers one or more of the processes from the user's workstation to some other workstation that is currently idle and gets the process executed there, and finally the result of execution is returned to the user's, in. workstation.

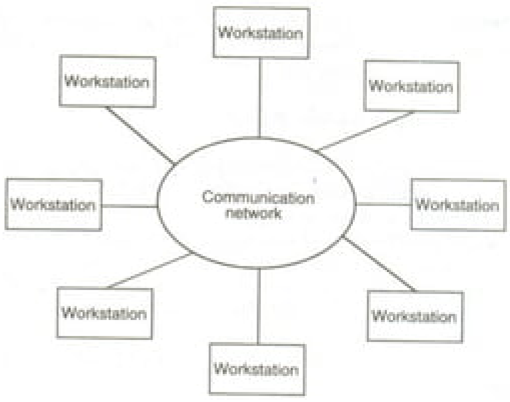


Figure 2: A distributed system based on the workstation model

This model is not so simple to implement because several issues must be resolved. These issues are as follows:

- How does the system find an idle workstation?
- How is a process transferred from one workstation to get it executed on another workstation?
- What happens to a remote process if a user logs onto a workstation that was idle until now and was being used to execute a process of another workstation?

Three commonly used approaches for handling the third issue are as follows:

1. The first approach is to allow the remote process share the resources of the workstation along win its own logged -on .user's processes. This method is easy to implement, but it defeats the main idea of workstations serving as personal computers, because if remote processes are allowed to execute simultaneously with the logged-on user's own processes, the logged-on user does not get his or her guaranteed response.
2. The second approach is to kill the remote process. The main drawbacks of this method are that all processing done by the remote process gets lost and tie file system may be left in an inconsistent state; making this method unattractive.
3. The third approach is, migrate the remote process back to its home workstation, so that its execution can be continued there. This method is difficult to implement because it requires the system to support preemptive process migration facility.

The Sprite system developed at Xerox is an examples of distributed computing systems based on the workstation model.

3. Workstation-Server Model

A workstation with its own local disk is usually called a *diskfull* workstation and a workstation without a local disk is called a *diskless* workstation. With high-speed networks, diskless workstations have become more popular than diskfull workstations, making the workstation-server model more popular than the workstation model for building distributed computing systems.

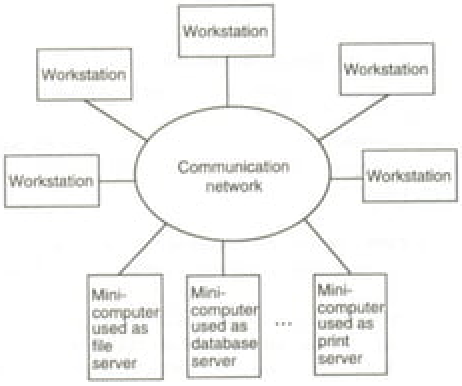

Workstation
Workstation
Workstation
Workstation
Communication network
Workstation
Mini-computer used as file server
Mini-computer used as database server
...
Mini-computer used as print server

Fig 3: A Distributed System based on the workstation-server model

As shown in Figure 3, a distributed computing system based on the workstation-server model consists of a few minicomputers and several workstations interconnected by a communication network. Most of the workstation may be diskless, but a few of may be disk full.

When diskless workstations are used on a network, the file system to be used by these workstations must be implemented either by a diskfull workstation or by a minicomputer equipped with a disk for file storage.

One or more of the minicomputers are used for implementing the file system. Other minicomputers may be used for providing other types of services, such as database service and print service. Therefore, each minicomputer is used as a server machine to provide one or more types of services. For a number of reasons, such as higher reliability, and better scalability, multiple servers are often used for managing the resources of a particular type in a distributed computing system.

For example, there may be multiple file servers, each running on a separate minicomputer and cooperating via the networks for managing the files of all the users in file system.

In this model, a user logs onto a workstation called his or her home workstation. Normal computation activities required by the user's processes are performed at the user's home workstation, but requests for services provided by special servers (such as a file server or a database server) are sent to a server providing that type of service that performs the user's requested activity and returns the result of request processing to the user's workstation. Therefore, in this model, the user's processes need not be migrated to the server machine for getting the work done by those machines.

For better overall system performance the local disk of a dishful workstation is normally used for such purposes as storage of temporary files, storage of unshared files, storage of shared files that are rarely changed, paging activity in virtual-memory management, and caching of remotely accessed data

As compared to the workstation model, the workstation-server model has several advantages:

1. It is much cheaper to use a few minicomputers equipped with large fast disks that are accessed over the network than a large number of dishful workstations, with each workstation having a

small, slow disk.

2. Diskless workstations are also preferred to dishful workstations from a system maintenance point of view. Software installation, backup and hardware maintenance are easier to perform with a few large disks than win many small disks scattered all over a building or campus.

3. In the workstation-server model, since all files are managed by be file servers, users have the flexibility to use any workstation and access the files in the same manner irrespective of which workstation the user is currently logged on. Note that this is not true win the workstation model, in which each workstation has its local file system, because different mechanisms are needed to access local and remote files.

4. In the workstation-server model, the request-response protocol described above is mainly used to access the services of the server machine. Therefore unlike the workstation model, this model does not need a process migration facility which is difficult to implement.

   The request-response protocol is known as the client-server model of communication. In this model, a client process (which in this case resides on a workstation) sends a request to a serve process (which In this case resides on a computer) for getting some service such as reading a block of a file to the server executes the request and sends back a reply to the cheat that contains the result of request processing.

   The client-server model provides an effective general-purpose approach to the sharing of information and resources in distributed computing systems It is not only meant for use with the workstation-server model but also can be Implemented in a variety of hardware and software environments. The computers used to run the client and server processes need not necessarily be workstations and minicomputers. They can be of many types and there is no need to distinguish between them It is even possible for both the client and server processes to be run on the same computer. Moreover some processes are both client and server processes That is, a server process may use the services of another server, appearing as a client to the latter

5. A user has guaranteed response time because workstations are not used for executing remote processes. However the model does not utilize the processing capability of idle workstations.

4. Processor Pooled Model

Processor-pool model is based on the observation that most of the time a user does not need any computing power but once m a while he or she may need a very large amount of computing power for a short time. Therefore, in the processor-pooled model the processors are pooled together to be shared by the users as needed. The pool of processors consists of a large number of microcomputer and minicomputers attached to the network. Each processor m the pool has its own memory to load and run a system program or an application program of the distributed computing system

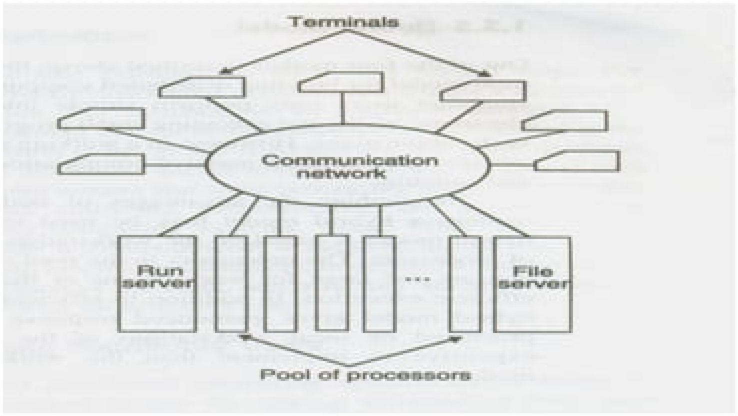


Figure 5: A distributed computing system based on processor-poor model

In the pure processor's model, the processors m the pool have no terminals attached directly to them, and users access the system from terminals that are attached to the network via special devices. These terminals are either small diskless workstations or graphic terminals.

A special server, called a run server, manages and allocates the processors in the pool to different users on a demand bases. When a user submits a Job for computation, an appropriate number of professors are temporarily assigned to the job by the run server.
For Example

- If the user's computation job is the compilation of a program having n segments,
- Each of the segments can be compiled independently to produce separate releasable object files;
- n processors from the pool can be allocated to this job to compile all the segments in parallel.
- When the computation is completed, the processors are returned to the pool for use by other users

In the processor-pool model there is no concept of a home machine. That is, a user does not log onto a particular machine but to the system as a whole This is in contrast to other models in which each user has a home machine (e.g., a workstation or minicomputer) onto which he or she logs and runs most of his or her programs they by default.

As compared to the workstation-server model, the processor-pool model allows better utilization of the available processing power of a distributed computing system. This is because

- In the processor-pool model, the entire processing power of the system is available for use by the currently logged-users, whereas this is not true for the workstation-server model in which several workstations may be idle at a particular time but they cannot be used for processing the jobs of other users.

- Furthermore the processor-pool model provides greater flexibility than the workstation-server model in the sense that the system's services can be easily expanded without the need to install any more computers

- The professors in the pool can be allocated to act as extra servers to carry any additional load arising from an increased user population or to provide new services.

However, the processor-pool model is usually considered to be unsuitable for high-performance interactive applications, especially those using graphics or window systems. This is mainly because of the slow speed of communication between the computer on which the application program of a user is being executed and the terminal via which the user is interacting with to system. The workstation-server model is genially considered to be more suitable for such applications.

Out of the four models describe above, the workstation-server model is the most widely used model for building distributed computing systems. This is because a large number of computer users only perform simple interactive tasks such as editing jobs, sending electronic mails, and executing small programs. The workstation-server model is ideal for such simple usage. However, in a working environment that has groups of users who often perform jobs going massive computation the processor-pool model is more attractive and suitable.

In the processor pool model, the ration of processors to users is greater than one. The model attempts to allocate one or more processors to a user to complete the task. Once the user's task is completed the assigned processors are returned to the pool.

Examples: Amoeba system is combination of workstation and processor pool models. Each user performs quick interactive response type of task on the workstation (such as editing). User can access to pool of processors for executing jobs that need significant numerical computations.

The Hybrid Model

To combine the advantages of both the workstation-server and processor-pool models, a hybrid model may be used to build a distributed computing system. The hybrid model is based on the workstation-server model but with the addition of a pool of processors. The processors in the pool can be allocated dynamically for computations that are too large for workstations or that requires several computers concurrency for efficient execution. In addition to efficient execution of computation-intensive jobs, the hybrid model gives guaranteed response to interactive jobs by allowing them to be processed on local workstations of the users. However, the hybrid model is more expensive to implement than the workstation-server model or the processor-pool model.

Issues in Distributed Operating Systems

A distributed operating system is a program that manages the resources of a computer system and provides users an easy and friendly interface to operate the system. The typical characteristics of the distributed operating systems are:

- System appears to its users as a centralized operating system, but it runs on multiple independent computers.
- Each computer may have the same or different operating system, but not visible to the users.
- User views the system as a virtual uniprocessor, and not a collection of distinct machines.
- User does not know on what computers the job was executed, on what computers the required files are stored and how the system communicates and synchronies among different computers.

Some important issues that arise in the design of a distributed operating system are:

- Unavailability of up to date Global knowledge
- Naming
- Scalability
- Compatibility
  - Binary level,
  - Execution level
  - Protocol level
- Process Synchronization

- Resource management
    - Data migration
    - Computational migration
    - Distributed scheduling
- Security
- Structuring
    - *Monolithic kernel*
    - *Collective kernel structure*
    - *Object-oriented operating system*
- Client Server Computing Model

1. **Global Knowledge**:

Due to the unavailability of a global memory, a global clock and the unpredictability of message delays, it is practically impossible for a computer to collect up-to-date information about the global state of the distributed system.

This leads to the two basic problems in designing Distributed Operating Systems

- How to determine efficient techniques to implement *decentralized system control*, where the system does not know the current and complete up to date status of the global state.

- How to order all the events that occur at different times at different computers in the system, due to the absence of a global clock

2. **Naming:**

Names are used to refer objects (e.g. computers, printers, services, files, users, etc.). a named service maps a logical name into a physical address by using a table or directory lookup The directory of all the named objects in the system must be maintained to allow proper access.

In distributed system directories may be replicated and stored at different locations to overcome a single point failure and to increase the availability of the named service. The drawbacks of replication are:

- It require more storage capacity and
- Synchronization requirement are needed when directories are updated, as directory at each location need to be updated.

Alternately directories may be partitioned to overcome the drawbacks of replicated directories. The drawback of partitioned directories is

- In finding the partition containing the name and address of interest.

Both schemes of replicated directories and partitioned directories have their strengths and weaknesses.

3. **Scalability**

Distributed system generally grows with time. Any mechanisms or approaches adopted in designing a distributed system must not result in badly degraded performance when the system grows.

For example, broadcast based protocol works well for small distributed systems but not for large distributed systems. Consider a distributed file system that locates files by broadcasting queries. Under this file system, every computer in the distributed system is subjected to the message handling overhead. As number of users increases and distributed system gets larger, the file-request queries will increase and the overhead will grow larger.

4. Compatibility

The interoperability among the resources in a distributed system must be an integral part of the design of a distributed system. Three levels of compatibility exist:

- Binary level compatibility,
- Execution level compatibility and
- Protocol level compatibility.

Binary Level Compatibility

In Binary level compatibility, all processors execute the same binary instruction set, even though the processors may differ in performance and in input-output.

- Advantage of binary level compatibility is that it is easier for system development, as the code for many functions depends on the machine instructions.
- On the other hand such distributed system cannot include computers with different architectures.
- Due to this restriction binary compatibility is rarely used in large systems.

Execution Level Compatibility

Execution level compatibility is said to exist, if the same source code can be compiled and executed properly on any computer in the distributed system.

Protocol Level Compatibility

Protocol level compatibility is the least restrictive form of compatibility. It achieves interoperability by requiring all system components to support a common set of protocols.

Advantage of protocol level compatibility is that individual computers can run different operating systems. This is possible by employing common protocols for essential system services such as file access, naming and authentication.

5. Process Synchronization

Process synchronization is difficult in distributed systems due to the lack of shared memory and a global clock.

A distributed system has to synchronize processes running at different computers when they try to concurrently access a shared resource, such as a file directory. For correctness, it is necessary that the shared resource be accessed by a single process at a time (that is access should be atomic). The problem is known as the *mutual exclusion*.

In distributed systems, processes can request resources (local or remote) and release resources in any order. If sequence of the allocation of resources is not controlled, *deadlocks* may occur. In order to maintain system performance, deadlocks must be detected and resolved as soon as possible.

6. **Resource Management**

Resource management refers to schemes and methods devised to make local and remote resources available to users in an *effective and transparent* manner. The specific location of resources should be hidden from the users. Three general schemes exist:

- Data Migration
- Computational Migration
- Distributed Scheduling

Data Migration

In data migration, data (contents of memory or file) is brought to the location of computation that needs access to it, by the distributed operating system. If a computation updates data, the original location may have to be similarly updated.

In case of physical memory access of another computer, the data request is dealt by distributed *shared memory management*, which provides a virtual address space that is shared by all the computers in a distributed system. The major concern is the maintenance of consistency of the shared data and to minimize the access delay.

In case of a file access the data request is dealt by the distributed file system, which is a component of the distributed operating system. The distributed file system provides the same functional capability to access files regardless of their location. This property of a distributed file system is known as *network transparency*.

Computational Migration

In computational migration, computation migrates to another location. For example,

- In distributed scheduling, one computer may require another computer status. It is more efficient and safe to find this information at the remote computer and
- Send the required information back, rather than to transfer the private data structure of the operating system at the remote computer to the requesting computer.
- The remote procedure call (RPC) mechanism is used widely for computational migration and providing communication between computers.

Distributed Scheduling

In distributed scheduling, processes are transferred from one computer to another by distributed operating system. Process relocation may be desirable if the computer where the process originated is overloaded or it does not have the necessary resources required by the process. This is done to enhance utilization of computer resources in the system.

7. **Security**

The security of the system is the responsibility of its operating system. Two issues are relevant:

- Authentication (verifying claims) and
- Authorization (deciding and authorizing the proper amount of privileges).

Authentication is the process of guaranteeing that an entity is what it claim to be, Authorization is a process of deciding what privileges an entity has and providing only these privileges.

8. Structuring

Structuring defines how various parts of the operating system are organized. Three general organizations exist:

- Monolithic Kernel:
- Collective Kernel structure:
- Object-Oriented operating system (also called process model)

Monolithic Kernel Structure

This is a traditional method of structuring operating system, with one big kernel, consisting of *all the services* to be provided in the system.

In the case of distributed systems that often consist of diskless workstations, workstations with local disk storage, multiprocessor computers suitable for intensive numerical computations, etc. it may be wasteful in a distributed environment to run a huge monolithic operating system.

Collective Kernel Structure

All services are implemented as independent processes.

In collective kernel structuring, the operating system implements the following services as independent processes:

- distributed memory,
- distributed file system,
- name services,
- RPC facility,
- time management,

The nucleus of the operating system (microkernel) also supports the interaction between the processes providing system services.
Micro kernel also provides services that are essential to every computer in a distributed system, such as

- task management (i.e. local scheduling),
- processor managements,

- Virtual memory management, etc.

The microkernel runs on all the computers in the system. The other processes may or may not run at a computer depending on the need and the hardware available.

Object-Oriented Operating System (also called Process Model**):**

Another approach to implement system services is as a collection of objects; each object encapsulates a data structure and defines a set of operations on that data structure. Each object is given a type that designates the properties of the object: process, directory, file, etc. by performing operations defined on an object, the data encapsulated can be accessed and modified.

9. **Client Server Computing Model**

In the client-server model processes are categorized as servers and clients. Servers are the processes that provide service. Processes that need services are called clients.

A client process needing some service sends a message to the server and waits for a reply- message. The server process after performing the requested task sends the result in the form of a message to the client process

Communication Primitives

The communication network provides a means to send raw bit stream of data in distributed systems. The communication primitives are the high level constructs with which programs use the network.

We will study two communication models that provide communication primitives. These two models are widely used to develop distributed systems, and the applications for distributed systems. The two models are:

- Message Passing Model
- Remote Procedure Call (RPC)

The Message Passing Model

This model provides two basic communication primitives:

SEND (destination, message)
RECEIVE (source, storage buffer)

An application of these primitives can be found in the client-server computational model. A client process needing some service (e.g. reading data from a file) sends a message to the server and waits for the reply message. After performing the task, the server process sends the result in the form of a reply message to the client process.

While these two primitives provide the basic communication ability to programs, the semantics of these primitives play an important role in developing the programs. There are two design issues that decide the semantics of the two primitives.

- Blocking vs. None-blocking Primitives
- Synchronous vs. Asynchronous Primitives

Blocking vs. None-Blocking Primitives

In the message passing, the message are copied three times

- From the user buffer to kernel buffer
- From the kernel buffer on sending computer to the kernel buffer on receiving computer
- From the receiving kernel buffer to the user buffer

This is known as the *buffered option.*

The non-blocking primitives, the SEND primitive return control to the user process as soon as the message is copied from the user buffer to the kernel buffer. The non-blocking RECEIVE primitive signals its intention to receive a message and provides a buffer to copy the message. The receiving process may either periodically check the arrival of a message or be signaled by the kernel upon arrival of a message.

The primary advantage of nonblocking primitives is that programs have maximum flexibility to perform computation and communication in any order.

The disadvantage is that programming becomes difficult. Program may become time-dependent, thus making the programs very difficult to debug,
In the *unbuffered option*, data is directly copied from one user buffer to another user buffer. In this case, a program issuing a SEND should avoid reusing the user buffer until the message has been transmitted.

With *blocking primitives*, the SEND primitive does not return control to the user program until the message has been sent (an unreliable blocking primitive) or until an acknowledgment has been received (a reliable blocking primitive) the corresponding RECEIVE primitive does not return control until a message is copied to the user buffer. A reliable RECEIVE primitive automatically sends an acknowledgement.

The primary advantage of using blocking primitives is that the behavior of the programs is predictable, thus making the programming easy. The disadvantage is the lack of flexibility in programming and the absence of concurrency between computation and communication.

Synchronous vs. Asynchronous Primitives

With synchronous primitives, a SEND primitive is blocked until a corresponding RECEIVES primitive is executed at the receiving computer. A blocking synchronous primitive can be extended to an unblocking-synchronous primitive by first copying the message to a buffer at the sending side, and then allowing the process to perform other computational activity except another SEND.

With asynchronous primitives, the message is buffered. A SEND primitive does not block even if there is no corresponding execution of a RECEIVE primitive. The corresponding RECEIVE primitive can either be a blocking or a nonblocking primitive. One disadvantage of buffering message is that it is more complex, as it involves creation, managing, and destroying buffers. Another issue is what to do with the messages that are meant for processes that have already died.

Remote Procedure Call (RPC)

While message passing communication model provides highly flexible communication ability, programmers using such model must handle the following details.

- Pairing of responses with request messages.
- Data representation, when computers of different architectures or programs written in different programming languages are communicating.
- Knowing the details of the remote machine or server
- Taking care of communication or system failures.

Handling all these details make the program development difficult. In addition these programs can be time-dependent, thus difficult to debug.

RPC mechanism hides all the above difficulties from the programmers. The RPC mechanism extends the uniprocessor call mechanism to transfer control and data across a communication network. A remote procedure call can be viewed as an interaction between a client and a server. On invoking a remote procedure, the following operations take place:

- The calling process (client) is suspended,
- Parameters, if any, are passed to the remote mechanism,
- The called procedure (server) is executed.
- On completion of the procedure, the results are passed back to the client.
- The client resumes execution.

Design Issues in RPC

1. Structure
2. Binding
3. Parameter and result passing
4. Error handling,
5. Semantics
6. Correctness
7. Other issues

1. Structure:

A widely used RPC mechanism is based on the concept of *sub procedures*. When a client makes a remote procedure call P (x, y), the following action takes place:

- A local dummy procedure (a client sub procedure) called stub corresponding to procedure P is called
- The stub constructs a message containing the identity of the remote procedure and parameters if any to be passed.
- It then sends the message to the remote server machine.
- A stub procedure at the remote machine (a server procedure) receives the message,
- The remote stub procedure make a local call to the procedure specified in the message and pass the parameters
- When the remote procedure completes execution, the control returns to the server-sub procedure
- The server sub-procedure passes the result back to the client-sub procedure
- The client-sub procedure returns the result to client.

The stub procedures can be generated at compile time or can be linked at run time.

2. Binding:

Binding is a process that determines:

- The remote procedure and the machine on which it will be executed, upon a remote procedure invocation.
- It also checks the compatibility of the parameters passed and type of the procedure.
- What results are expected from the remote procedure?

There are two implementation approaches for binding in the client server model

- Make use of a binding server. The servers register the services they provide with the binding server. The binding server stores the server machine address along with the services they provide.
- When a client make a remote procedure call, the client-stub procedure obtains the address of the service machine by querying the binding server.

- Another approach is that the client specifies the machine and the service required and the binding server returns the port number required for communicating with the service requested.

3. Parameter and Result Passing

To pass parameters or results to a remote procedure, a stub procedure has to convert the parameters and results into an appropriate representation (a representation that is understood by the remote machine) first and then pack them into a buffer in the form suitable for transmission.
After the message is received, the message must be unpacked. See figure below. Converting data into an appropriate representation becomes expensive if it has to be done on every call.

First Approach to avoid conversions is to send the parameters along with a code identifying the format used so that the receiver can do the conversion. *The drawback* of this approach is that the each machine has to know how to convert all formats that can possibly be used. This approach has poor portability because whenever a new representation is introduced into the system, existing software needs to be updated.

Second approach to avoid conversion is that each data type may have a standard format in the message. In this technique, the sender will covert the data to the standard format and the receiver will convert from the standard format to its local representation. With this approach a machine does not need to know how to convert all formats that can possibly be used.

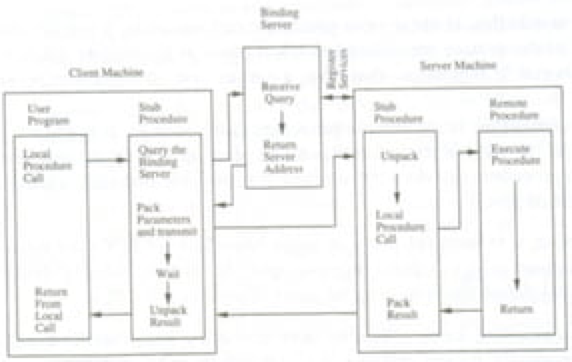


Figure: Remote Procedure Call Mechanism
The method, however, is wasteful if both the sender and the receiver use the same internal representation.

Another issue is to deal with passing parameter *by value and by reference*. Passing parameters by valued is simple, as the stub procedure simply copies the parameters into the message. However, passing parameters by reference is exceedingly more difficult, for example passing file pointer is not sufficient because privileges associated with the calling processes also need to be passed. Therefore, passing parameter by reference is seldom used.

4. Error Handling

Handling failure in distributed systems is difficult. A remote procedure call can fail for two reasons: Computer failure, and Communication failure. Examples are:

Message is lost: Remote server is slow; the program that invokes the remote procedure may invoke the procedure more than once, suspecting a message lost. This may result more than one execution of the procedure.

Client machine crashes: Client machine crashes after invoking a remote procedure call. The remote procedure is executed, but there is no machine to receive the result. On the other end, if the client machine recovers quickly and reissues the remote procedure call, the procedure is executed twice.

5. RPC Semantics:

In view of the above problems of distributed system failures, RPC semantics plays an important role in the program development.

RPC Semantics Classification:

- **At least once semantics**: If the remote procedure call succeeds, it implies that at least once execution of the remote procedure has taken place at the remote machine. If the call does not succeed, it is possible that zero, a partial one, or more execution have taken place.
- **Exactly once semantics**: If the remote procedure call succeeds; it implies that exactly one execution of the remote procedure has taken place at the remote machine. However, if the remote procedure call does not succeed, it is possible that zero, a partial, or one execution has taken place.
- **At most once semantics**: Same as exactly once semantics, but in addition, calls that do not terminate normally do not produce any side effects. These semantics are also refereed to as Zero-or-one semantics

5. **Condition for Correctness**

A simple correctness condition for RPC is defined below:

Let, C_i call made by a machine
W_i represents the corresponding computation at remote machine
Let, $C1 \rightarrow C2$ denotes C2 happen after C1.
Let W1 and W2 share the same data such that W1 and/or W2 modify the shared data.

To be correct, in the presence of failure, a RPC implementation should satisfy the following correctness criteria:

$$C1 \rightarrow C2 \text{ implies } W1 \rightarrow W2$$

Other Issues

RPC mechanism made use of communication facility to pass messages to remote machines.
Issues to be resolved is how to built RPC mechanism

On top of a flow-controlled and error-controlled virtual-circuit mechanism, or
Directly on top of an unreliable, connectionless (datagram) service

In datagram service, a machine simply sends a message in the form od packets to the destination and there is no two-way communication such as automatic acknowledgment.

RPC Classifications

An RPC mechanism is widely accepted method for communication in distributed systems. There are two different categories of applications, using RPC facility

- Low-latency calls: These RPC calls require minimum round-trip delay and are made in case of infrequent calls from the communication facility, such as to mail-server.
- High throughput calls: This type of calls is made when bulk data transfer is required, such as in the case of calls to file servers.

ASTRA RPC mechanism provides option to the users to specify whether low-latency or high-throughput calls are to be made.

ASTRA make use of virtual circuit (TCP) protocol for high-throughput calls. High throughput is achieved by buffering the messages and immediately returning control to users. The user can then make more calls. The buffer is flashed when it is full or convenient. By buffering the message the overhead of communication protocols on every RPC is avoided.

ASTRA makes use of a diagram facility that is more suitable for intermittent exchange due to its simplicity. For low-latency calls, the buffer is flushed immediately.

Concurrency

Invoking a RPC, blocks the calling procedure, which limits the concurrency. Several RPC designs are there to overcome this limitation. One way to achieve parallelism is through multiple processes for each remote procedure call. This scheme allows a process to make multiple- calls to many servers and still execute in parallel with servers.

However,
 Creating processes,
 Switching between processes, and
 Destroying processes,
may not be economical under all circumstances, and is not suitable for large systems.

Parallel Procedure Call (PARPC)

Parallel procedure calls execute a procedure in n different address spaces, that is at n different servers.

- Caller remains blocked, while procedures execute. When result is available
- The caller is unblocked to execute a statement to process the result of the returned call.
- After executing the statement, the caller is again blocked to wait for the next result
- Caller resume execution after all the n-calls have returned, or caller explicitly terminates the PARPC.

Asynchronous RPC Mechanism

In asynchronous RPC calls, calling process does not block after invoking a RPC and allows the client to receive results in any order. The drawback of this mechanism is that programming is difficult.

Short-Comings of Existing RPC Facilities

1. **Incremental Results**: Present RPC facility cannot easily return intermediate results to calling process, while its execution is still in progress.

2. **Protocol flexibility**: In present RPC systems, RPC is not a first-class object. A first-class object is that which can be freely stored in memory. Passed as parameter to both local and remote procedures, and returned as result from both local and remote procedure. This makes the protocol inflexible.
   For example: the following protocol is not-implementable, unless RPC is a first-class object:
   > A process wishes to provide a server with a procedure for use under certain circumstances and the server wishes to pass this procedure on to another server.

Channel Model

Gifford proposed a new communication model, called channel model, in which remote procedures are first-class objects an abstraction (called pipe) permits transportation of bulk data and incremental results.

An abstraction (called channel group) allows sequencing of calls on pipes and procedures.