IQRA National University Peshawar

Name: Noor rahman

Reg ID # 14232

Course: - Distributed Computing

Course:- MS Computer Science

Spring Semester 2020 Final Exam

Department of Computer Science

Q1. Describe briefly the purpose of the three communication primitives in request-reply protocols.

Answer:

Request-reply protocols

The Definition of the Request-reply protocols is this form of communication is designed to support the roles and message exchanges in typical client-server interactions. In the normal case, request-reply communication is synchronous because the client process blocks until the reply arrives from the server.
Here protocol we describe here is based on a trio of Communication primitives, do Operation, get Request and send Reply

Three communication primitives.

The Following is the Main three communication primitives.
The process of communication of,

public byte[] doOperation (RemoteRef s, int operationId, byte[] arguments)

sends a request message to the remote server and returns the reply.
The arguments specify the remote server, the operation to be invoked and the
arguments of that operation.

public byte[] getRequest ();

acquires a client request via the server port.

public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);

sends the reply message reply to the client at its Internet address and port.

As Above process show all the communication preemptive and we explain that.

- Its result is a byte array containing the reply.
- This class provides methods for getting the Internet address and port of the associated server.
- After sending the request message, *doOperation* invokes receive to get a reply message, from which it extracts the result and returns it to the caller.
- The client calling *doOperation* marshals the arguments into an array of bytes and marshals the results from the array of bytes that is returned.
- The doOperation method sends a request message to the server whose Internet address and port are specified in the remote reference given as an argument.
- The caller of doOperation is blocked until the server performs the requested operation and transmits a reply message to the client process.

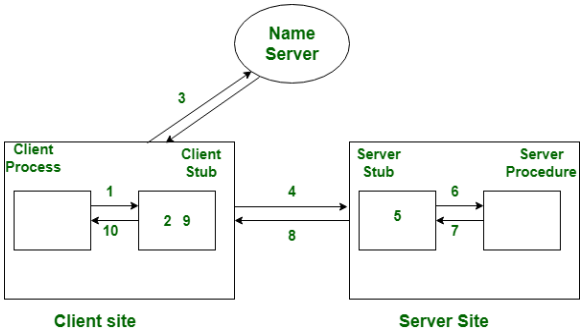
- Request message or a reply message is shown in Figure 5.4
- Message Type a Request or a Reply message.
- request ID, contains a message identifier.
- A *doOperation* in the client generates a request ID for each request message, and the server copies these IDs into the corresponding reply messages.
- This enables *doOperation* to check that a reply message is the result of the current request, not a delayed earlier call.
- The third field is a remote reference.
- The fourth field is an identifier for the operation to be invoked.

……

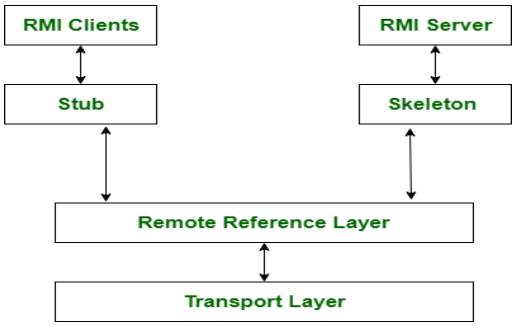
Q2. Explain the technical difference between RPC and RMI?

The following is main difference between RPC and RMI

RPC stands for **Remote Procedure Call** which supports procedural programming. it's almost like IPC mechanism wherever the software permits the processes to manage shared information Associated with an environment wherever completely different processes area unit death penalty on separate systems and essentially need message-based communication. **RPC** stands for **Remote Procedure Call** which supports procedural programming. Tt's almost like IPC mechanism wherever the software permits the processes to manage shared information Associated with an environment wherever completely different processes area unit death penalty on separate systems and essentially need message-based communication.



RMI stands for **Remote Method Invocation**, is a similar to PRC but it supports object-oriented programming which is the java's feature. A thread is allowable to decision the strategy on a foreign object. In RMI, objects are passed as a parameter rather than ordinary data.



RPC and RMI both are similar but the basic difference between RPC and RMI is that RPC supports procedural programming, on the other hand, RMI supports object-oriented programming.

S.NO	RPC	RMI
1.	RPC is a library and OS dependent platform.	Whereas it is a java platform.
2.	RPC supports procedural programming.	RMI supports object oriented programming.
3.	RPC is less efficient in comparison of RMI.	While RMI is more efficient than RPC.
4.	RPC creates more overhead.	While it creates less overhead than RPC.
5.	The parameters which are passed in RPC are ordinary or normal data.	While in RMI, objects are passed as parameter.
6.	RPC is the older version of RMI.	While it is the successor version of RPC.
7.	There is high Provision of ease of programming in RPC.	While there is low Provision of ease of programming in RMI.
8.	RPC does not provide any security.	While it provides client level security.
9.	It's development cost is huge.	While it's development cost is fair or reasonable.
10.	There is a huge problem of versioning in RPC.	While there is possible versioning using RDMI.
11.	There is multiple codes are needed for simple application in RPC.	While there is multiple codes are not needed for simple application in RMI.

…….

Q:3 In contrast to Direct Communication, which two important properties are present in Indirect Communication?

Answer:

Indirect communication

The definition od In Indirect communication is acting out rather than directly saying what a person is thinking or feeling using facial expressions, tone of voice, and/or gestures that is called the indirect communication.

1. Space uncoupling, in which the sender does not know or need to know the identity of the receiver(s), and vice versa. Because of this space uncoupling, the system developer has many degrees of freedom in dealing with change: participants (senders or receivers) can be replaced, updated, replicated or migrated.

2. Time uncoupling, in which the sender and receiver(s) can have independent lifetimes. In other words, the sender and receiver(s) do not need to exist at the same time to communicate. This has important benefits, for example, in more volatile environments where senders and receivers may come and go.

3. Indirect communication is defined as communication between entities in a distributed system through an intermediary with no direct coupling between the sender and the receiver(s). The precise nature of the intermediary varies from approach to approach, as will be seen in the rest of this chapter. In addition, the precise nature of coupling varies significantly between systems, and again this will be brought out in the text that follows. Note the optional plural associated with the receiver; this signifies that many indirect communication paradigms explicitly support one-to-many communication.

Q: 4 provide three reasons as why group communication (single multicast operation) is more efficient than individual unicast operation?

Answer:

In my view the following reason group communication is more efficient then individual unicast operation.

- The reliable dissemination of information to potentially large numbers of clients, including in the financial industry, where institutions require accurate and up-to date access to a wide variety of information sources.
- support for collaborative applications, where again events must be disseminated to multiple users to preserve a common user view.
- support for system monitoring and management, including for example load balancing strategies
- Support for a range of fault-tolerance strategies, including the consistent update of replicated data (as discussed in detail in Chapter 18) or the implementation of highly available (replicated) server

Q:5 Differentiate a between a network OS and distributed OS.

In this topic we shall see the difference between Network Operating System and Distributed Operating System. The main difference between these two operating systems (Network Operating System and Distributed Operating System) is that in network operating system each node or system can have its own operating system on the other hand in distribute operating system each node or system have same operating system which is opposite to the network operating system.

The difference Between Network Operating System and Distributed Operating System are given below:

S.NO	Network Operating System	Distributed Operating System
1.	Network Operating System's main objective is to provide the local services to remote client.	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.	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.	Distributed Operating System is less scalable than Network Operating System.
4.	In Network Operating System, fault tolerance is less.	While in Distributed Operating System, fault tolerance is high.
5.	Rate of autonomy in Network Operating System is high.	While The rate of autonomy in Distributed Operating System is less.
6.	Ease of implementation in Network Operating System is also high.	While in Distributed Operating System Ease of implementation is less.
7.	In Network Operating System, All nodes can have different operating system.	While in Distributed Operating System, All nodes have same operating system.

Q6. Describe briefly how the OS supports middleware in a distributed system by providing and managing (6)

a) **Process and threads**

b) **System Virtualization**

OS supports middleware in a distributed system by providing and managing.

Middleware is basically the software that connects software components or enterprise applications. It is the software layer that lies between the operating system and the applications on each side of a distributed computer network. Middleware in the context of distributed applications is software that provides services beyond those provided by the operating system to enable the various components of a distributed system to communicate and manage data. Middleware supports and simplifies complex distributed applications. In distributed systems it hides the distributed nature of the application. It keeps collection of

interconnected parts that are operational and running in distributed locations, out of view making things easier and simpler to manage.

a) Process and threads

Process means any program is in execution. Process control block controls the operation of any process. Process control block contains the information about processes for example: Process priority, process id, process state, CPU, register etc. A process can creates other processes which are known as Child Processes. Process takes more time to terminate and it is isolated means it does not share memory with any other process. The solution reached was to enhance the notion of a process so that it could be associated with multiple activities. Nowadays, a process consists of an execution environment together with one or more threads. A thread is the operating system abstraction of an activity (the term derives from the phrase 'thread of execution'). An execution environment is the unit of resource management: a collection of local kernelmanaged resources to which its threads have access. An execution environment primarily consists of:

• an address space;
• thread synchronization and communication resources such as semaphores and communication interfaces (for example, sockets);
• higher-level resources such as open files and windows.

A process is an active program i.e. a program that is under execution. It is more than the program code as it includes the program counter, process stack, registers, program code etc. Compared to this, the program code is only the text section.

A thread is a lightweight process that can be managed independently by a scheduler. It improves the application performance using parallelism. A thread shares information like data segment, code segment, files etc. with its peer threads while it contains its own registers, stack, counter etc. hread is the segment of a process means a process can have multiple threads and these multiple threads are contained within a process. A thread have 3 states: running, ready, and blocked.

Threads can be created and destroyed dynamically, as needed. The central aim of having multiple threads of execution is to maximize the degree of concurrent execution between operations, thus enabling the overlap of computation with input and output, and enabling concurrent processing on multiprocessors. This can be particularly helpful within servers, where concurrent processing of clients' requests can reduce the tendency for servers to become bottlenecks. For example, one thread can process a client's request while a second thread servicing another request waits for a disk access to complete. An execution environment provides protection from threads outside it, so that the data and other resources contained in it are by default inaccessible to threads residing in

Thread takes less time to terminate as compared to process and like process threads do not isolate.

b) System Virtualization

Virtualization creates an environment wherein it emulates and imitates various hardware components like CPU, OS, software, I/O devices and storage devices to numerous virtual machines (VM). Each node in the distributed system is a virtual machine running independently.

Virtualization is an important concept in distributed systems. We have already seen one application of virtualization in the context of networking, in the form of overlay networks (see Section 4.5) offering support for particular classes of distributed application. Virtualization is also applied in the context of operating systems; indeed, it is in this context that virtualization has had the most impact. In this section, we examine what it means to apply virtualization at the operating system level (system virtualization) and also present a case study of Xen, a leading example of system-level The technological trends have always been aiming towards development of such systems that can run many processes simultaneously, can share the available resources across various users, groups and organizations and that can attain maximum performance from the available infrastructure. Distributed computing is a complex process when comes to its development and deployment because of its transparency in sharing resources and parallel execution of multiple processes. Most data centers today have a three-or four-tier hierarchical networking structure. Three-tier networking architectures were designed around client-server applications and single-purpose application servers. This type of network architecture, however, is becoming problematic for the data center. Now-a-days application environments are more distributed, often with multiple tiers, and oriented toward service delivery [1]. Main concern of distributed computing is to improve energy efficiency and resource utilization. The technological improvements in the form of distributed computing, grid computing and cloud computing in WAN has made it possible to aggregate distributed resources.For improvising the system's maintenance and management, rapid advancement in system virtualization at various levels is taking place. There is need to employ a performance enhancer mechanism to maximize the reliability, scalability and fault tolerance of the distributed systems.

Q7. Write in your own words the issues with Object (distributed) oriented middleware's.

Answer:

Issues with Object (distributed) oriented middleware's.

Object-oriented middleware provides reusable service/protocol component and framework software that functionally bridges the gap between **Object-oriented middleware** provides capabilities whose qualities are critical to help simplify and coordinate how networked applications are connected and how they interoperate.

Middleware based on distributed objects is designed to provide a programming model based on object-oriented principles and therefore to bring the benefits of the object oriented approach to distributed programming. Emmerich [2000] sees such distributed objects as a natural evolution from three strands of activity:

Middleware based on distributed objects is designed to provide a programming model based on object-

oriented principles and therefore to bring the benefits of the object oriented approach to distributed programming. Emmerich [2000] sees such distributed objects as a natural evolution from three strands of activity:

• In distributed systems, earlier middleware was based on the client-server model and there was a desire for more sophisticated programming abstractions.

• In programming languages, earlier work in object-oriented languages such as Simula-67 and Smalltalk led to the emergence of more mainstream and heavily used programming languages such as Java and C++ (languages used extensively in distributed systems).

• In software engineering, significant progress was made in the development of object-oriented design methods, leading to the emergence of the Unified

Modelling Language (UML) as an industrial-standard notation for specifying (potentially distributed) object-oriented software systems. In other words, through adopting an object-oriented approach, distributed systems developers are not only provided with richer programming abstractions (using familiar programming languages such as C++ and Java) but are also able to use object-oriented design principles, tools and techniques (including UML) in the development of distributed systems software. This represents a major step forward in an area where, previously, such design techniques were not available. It is interesting to note that the OMG, the organization that developed CORBA, also manages the standardization of UML. Distributed object middleware offers a programming abstraction based on object oriented principles. Leading examples of distributed object middleware include Java RMI (discussed in Section 5.5) and CORBA (examined in depth in Section 8.3 below). While Java RMI and CORBA share a lot in common, there is one important difference: the use of Java RMI is restricted to Java-based development, whereas CORBA is a multi-language solution allowing objects written in a variety of languages to interoperate. (Bindings exist for C++, Java, Python and several others.)

…….