Spring Semester 2020 Final Exam
Course: - Distributed Computing

Deadline: - Mentioned on SIC **Marks: - 50**

Program: - MS (CS) **Dated: 24 June 2020**

Student Name: AQEEL KHAN **Student ID#: 5953**

Class and Section: MS CS – 4th Semester

Section: Remote Invocation

Q1. Describe briefly the purpose of the three communication primitives in request-reply protocols. (6)

Answer: Communication in distributed systems is always based on low-level message passing as offered by the underlying network. Expressing communication through message passing is harder than using primitives based on shared memory, as available for no distributed platforms. Modern distributed systems often consist of thousands or even millions of processes scattered across a network with unreliable communication such as the Internet There are three widely-used models for communication protocol: **Remote Procedure Call (RPC), Message-Oriented Middleware (MOM)**, and **data streaming**.

RPC: An RPC aims at hiding most of the intricacies of message passing, and is ideal for client-server applications.

MOM: is a subject important enough to warrant a section of its own.

Data Streaming: lacking support for communication of continuous media, such as audio and video.

Q2. Explain the technical difference between RPC and RMI? (4)

Answer: **RPC**: R**emote 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 and also procedure call model to RPC for distributed systems allows client programs to call procedures transparently in in separate processes and in different computers from the client.

RMI: (Remote Method Invocation) is a way that a programmer, using the Java programming language and development environment, can write object-oriented programming in which objects on different computers can interact in a distributed network and its invocation that allows an object living in one process to invoke the methods of an object living in another process.

Main Technical difference of RPC & RMI

Main Technical difference between the two is that the parameters passed to remote procedures call consist of ordinary data structures. On the other hand, the parameters passed to remote method consist of objects.

Section: Indirect Communication

Q:3 In contrast to Direct Communication, which two important properties are present in Indirect Communication? (6)

Answer: **Direct Communication:** In the Direct Communication, each process that wants to communicate must explicitly name the recipient or sender of the communication. In this scheme, the send and receive primitives are defined as follows:

Send (P, message) - Send a message to process P.
Receive (Q, message) - Receive a message from process Q.

Indirect communication 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.

Two important properties are present in Indirect Communication

There are two important properties are present in direct communication which are given below:

Space uncoupling: The sender does not know or need to know the identity of the receiver(s), and vice versa.

Time uncoupling: The sender and the receiver(s) can have independent lifetimes.

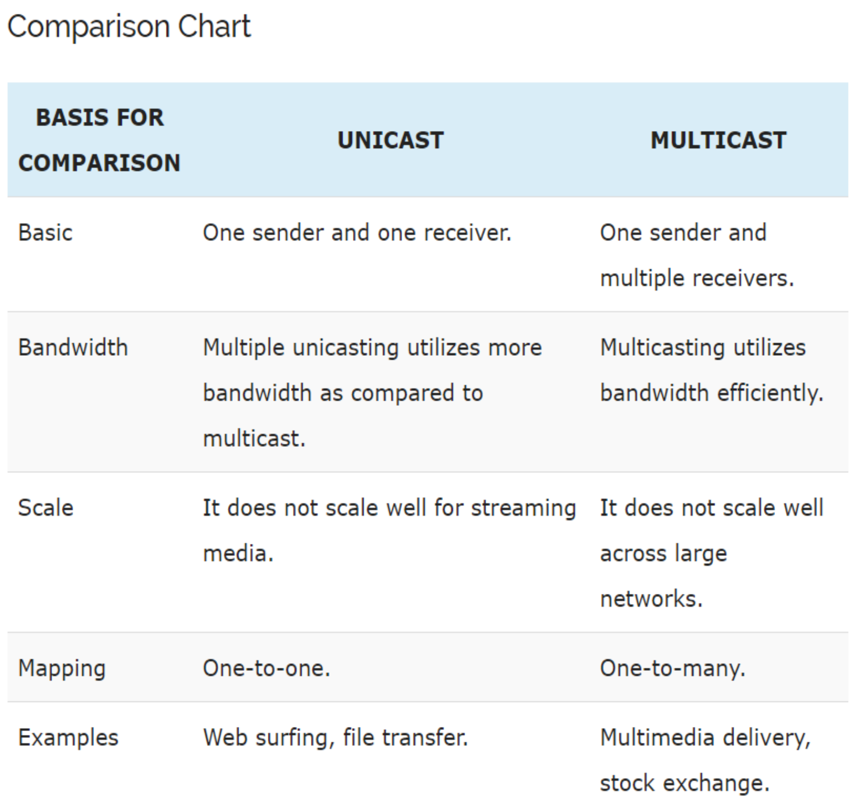
Q:4 Provide three reasons as why group communication (single multicast operation) is more efficient than individual unicast operation? (9)

Answer: **Single Multicast Operation**: is the term used to describe communication where a piece of information is sent from one or more points to a set of other points. In this case there is may be one or more senders, and the information is distributed to a set of receivers (there may be no receivers, or any other number of receivers).

Unicast Operation: is the term used to describe communication where a piece of information is sent from one point to another point. In this case there is just one sender, and one receiver and also a packet is sent from a single source to a specified destination.

Reasons:

Both unicast and multicast serve very distinct purposes, and there are requirements that you'll need to support for each of them. Broadcasters and corporate video streamers may find a need for one or the other, and may also find themselves requiring a hybrid of both in some cases. Unicast is typically used in Over-the-Top (OTT) streaming applications OTT is a term used to describe streaming over the internet. Multicast is more secure than OTT transmission because you're operating within a private IP network, as opposed to the public Internet, where your streams could be much more vulnerable.



Comparison Chart

BASIS FOR COMPARISON	UNICAST	MULTICAST
Basic	One sender and one receiver.	One sender and multiple receivers.
Bandwidth	Multiple unicasting utilizes more bandwidth as compared to multicast.	Multicasting utilizes bandwidth efficiently.
Scale	It does not scale well for streaming media.	It does not scale well across large networks.
Mapping	One-to-one.	One-to-many.
Examples	Web surfing, file transfer.	Multimedia delivery, stock exchange.

Section: OS Support

Q5. Differentiate a between a network OS and distributed OS. (6)

Answer: **Network Operating System**: Network Operating System falls under the category of Distributed architectures where a large number of computer systems are connected with each other with the help of a network. Although the implementation of the network operating system is simpler than the distributed operating system.

Distribute operating system: distribute operating system each node or system have same operating system which is opposite to the network operating system and Each individual node holds a specific software subset of the global aggregate operating system.

The Main Difference Between Network Operating System and Distributed Operating System are given below:

- Network Operating System's main objective is to provide the local services to remote client.
- In Network Operating System, Communication takes place on the basis of files.
- Distributed Operating System's main objective is to manage the hardware resources.
- In Distributed Operating System, Communication takes place on the basis of messages and shared memory

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

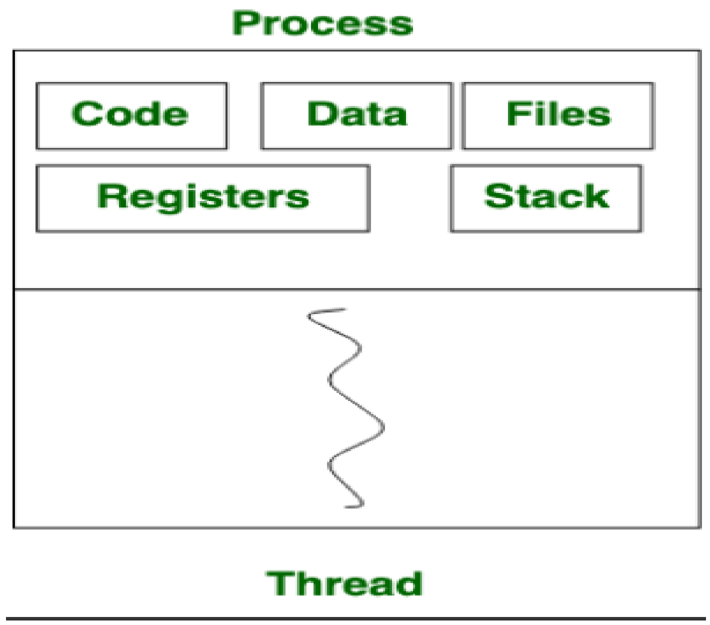
a) **Process and threads**
b) **System Virtualization**

Answer: Middleware is a software layer situated between applications and operating systems. Middleware is typically used in distributed systems where it simplifies software development by doing the following:

- Hides the intricacies of distributed applications
- Hides the heterogeneity of hardware, operating systems and protocols
- Provides uniform and high-level interfaces used to make interoperable, reusable and portable applications
- Provides a set of common services that minimizes duplication of efforts and enhances collaboration between applications

Process:
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 create other processes which are known as Child Processes.
Thread:
Thread is the segment of a process means a process can have multiple threads and these multiple threads are contained within a process. A thread has 3 states: running, ready, and blocked and also Thread takes less time to terminate as compared to process and like process threads do not isolate.

Figure are given below



System Virtualization is the most effective way to reduce IT expenses and boost efficiency large to mid-size and small organizations. Virtualization lets you run multiple operating systems and applications on a single server, consolidate hardware to get vastly higher productivity from fewer servers and simplify the management, maintenance, and the deployment of new applications.

Problems related to performance overhead, such as:

- The host operating system employs CPU, memory, and other hardware IT resources.
- Hardware-related calls from guest operating systems need to navigate numerous layers to and from the hardware, which shrinkage overall performance.
- Licenses are frequently essential for host operating systems, in addition to individual licenses for each of their guest operating systems.

Section: Distributed Objects and Components

Q7. Write in your own words the issues with Object (distributed) oriented middleware's. (13)

Answer: Middleware

Consists of a layer of services added between those of a regular network OS and the actual applications. These services facilitate the implementation of distributed applications and attempt to hide the heterogeneity of the underlying system architectures (both hardware and software). Middleware is usually based on a particular paradigm, or model, for describing distribution and communication. Providing such a paradigm automatically provides an abstraction for programmers to follow, and provides direction for how to design and set up the distributed applications.

Why Middleware?

Distributed system construction directly on top of a transport layer is rather difficult. The communication will often need to send complex parameters. Different encodings of data types in memory. Parameters and return values might refer to other distributed system components. Developers and administrators would have to implement component activation. Type safety is an important concern. Achieving type safety manually is quite error prone. Implementing synchronization is rather tedious and error prone.

Challenges

The main challenge in designing a distributed-object based middleware system is to provide transparency. The model naturally provides location, migration, and replication transparency. A greater challenge is to provide failure transparency. As discussed previously, remote method invocation can fail in ways that local method invocation cannot. Moreover, trying to mask the possibility of these kinds of failures is not always a good idea. A distributed object system must, however, deal with these kinds of failures. In particular, the system must deal with partial failures, that is failures of some components of the system (e.g., network, a server, etc.) that make it unclear where the fault occurred (for example, a remote server failure may be indistinguishable from a network failure: did the client not receive a response because the server failed, because the request never arrived at the server, or because the response never arrived at the client.

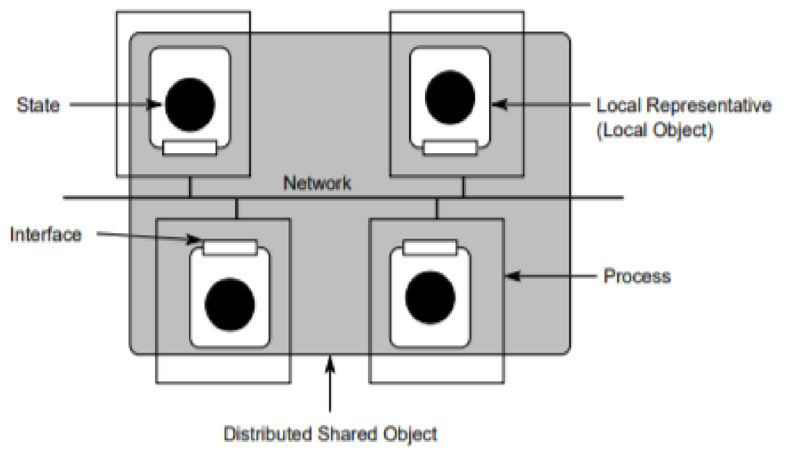
Distributed Shared Objects

So far the description of distributed-object systems has assumed a remote-object model. In this model the object is hosted on a single object server and clients invoke methods on the object by sending invocation requests to that server. A key property of this model is that the object state is stored only at the object server, there are no copies of the state at any other servers or clients. A different approach to distributed objects is the distributed shared object (DSO) model. In this model an object's state can be replicated or partitioned over multiple locations. Furthermore, state can be stored at both clients and object servers. Methods may be executed at any of the locations where the object's state is stored. By adding the concept of replication to the distributed-object model, the DSO model provides the benefits of greater scalability (through decentralization and replication), but also introduces many of the problems involved with replication and consistency.

Distributed Shared Object Model

In this DSO there are four processes bound to the object. Each process contains an instance of the object's local representative (LR) in its address space, and each LR exports the object interface. In this example, each of these LRs contains a copy of the object state. This example does not distinguish between clients and object servers. In the DSO model the main difference between the two is that clients invoke methods directly on the LR, while in an object server method invocations are only performed as a result of request messages received over the network.

Diagram given below:



Replication and Consistency

DSO model allows state to be replicated the issues discussed in the previous lecture on replication and consistency play an important role in DSO systems. In particular, a DSO system must provide mechanisms for keeping replicated state consistent. This means that the system (or possibly individual objects) must implement a consistency policy and a consistency protocol. There is also the issue of state migration: can LRs be migrated to different servers, and can state be migrated between different LRs? The various LRs may also take on different roles in the system. For example, some LRs can take on the roles of permanent replicas, while others take on the roles of server-initiated and client-initiated replicas. A final issue with regards to replication and DSOs is that of replicated invocations. This is a particularly relevant issue if replicated DSOs can hold and use references to other replicated DSOs.

Types of Middleware

1. Transaction-Oriented Middleware
2. Message-Oriented Middleware

- **Transaction-Oriented Middleware** is often used with distributed database applications. Object-oriented middleware has transaction-oriented middleware capabilities.

- **Message-Oriented Middleware** is used when reliable, asynchronous communication is the dominant form of distributed system interaction.

Object-Oriented Middleware

Object-Oriented Middleware evolved more or less directly from the idea of remote procedure calls. The first of these systems was OMG's Common Object Request Broker Architecture (CORBA). Microsoft added distribution capabilities to its Component Object Model (COM). Sun provided a mechanism for Remote Method Invocation (RMI). The idea here is to make object-oriented principles available for the development of distributed systems.