# Important Instructions:

1) **Open this MS-Word document and start writing answers below each respective question given on page 2.**
2) **Answers the question in the same sequence in which they appear.**
3) **Provide to the point and concrete answers. Some of the questions are open ended and therefore must be answered using your own opinion and thoughts but backed with logical reasons.**
4) **First read the questions and understand what is required of you before writing the answer.**
5) **Attempt the paper yourself and do not copy from your friends or the Internet. Students with exactly similar answers or copy paste from the Internet will not get any marks for their assignment.**
6) **You can contact me for help if you have any doubt in the above instructions or the assignment questions.**
7) **All questions must be attempted.**
8) **Do not forget to write your name, university ID, class and section information.**
9) **Rename you answer file with your university ID# before uploading to SIC.**
10) **When you are finished with writing your answers and are ready to submit your answer, convert it to PDF and upload it to SIC unzipped, before the deadline mentioned on SIC.**

# Spring Semester 2020 Final Exam
# Course: - Distributed Computing

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

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

---

**Student Name: _AWAID ULLAH_ Student ID#:_12714_**

**Class and Section:_MS(CS)_**

---

## Section: Remote Invocation

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

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

## Section: Indirect Communication

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

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

## Section: OS Support

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

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

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

## Section: Distributed Objects and Components

**Q7. Write in your own words the issues with Object (distributed) oriented middlewares. (13)**

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

**ANS:** Request-reply protocols: Represents the model above message transmission and supports two-way exchange of messages encountered in client-server computing. Provides low-level support for the execution of remote operations, and also provides direct support for RPC and RMI. The conventional procedure call model is extended to RPC in distributed systems, allowing client programs to transparently call procedures in separate processes and on computers different from the client. In the 1990s, object-based on has been extended to allow objects from different processes to communicate with each other through remote method invocation (RMI). RMI is an extension of local method invocation, which allows objects that exist in one process to call methods of active objects in another process. Designed to support message exchange in typical client-server interactions. Normally, in request-response communication, the client process will block until the response arrives (synchronizes) from the server. Since the server's response is actually a confirmation of the receipt sent to the client, it is also reliable. Asynchronous request-response communication is an alternative method, which is useful if the client can afford to retrieve the response later. The request-response protocol we describe here is based on triple communication primitives, perform operations, get requests and send responses. The request-response protocol matches the request with the response.

• It can be designed to provide certain delivery guarantees.

• If UDP datagrams are used, delivery guarantee must be provided by the request-response protocol, which can use server response messages as confirmation of client request messages.

Figure outlines the three communication primitives.

public byte[] do Operation (Remote Ref s, int operation Id, 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[] get Request ();

acquires a client request via the server port.

public void send Reply (byte[] reply, I net Address client Host, int client Port);

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

• The client uses the "Execute Operation" method to invoke the remote operation. Its parameters specify the remote server and the operation to be invoked, as well as other information (parameters) required for the operation. The result is a byte array containing the response. The client that calls do Operation collects the parameters into a byte array and unmasks the results of the returned byte array. The first parameter of do Operation is an instance of the Remote Ref class, which represents the reference of the remote server. This class provides a method for obtaining the Internet address and port of the associated server. The do Operation method sends a request message to the server whose Internet address and port are specified in the remote reference given in the parameter. After sending the request message, Operation calls to receive the response message, extract the result from it and return it to the caller. The caller of do Operation is blocked before the server performs the requested operation and sends the response message to the client process. The server process uses "Get Request" to get the service request. After the server invokes the specified operation, it will use the send response to send the response message to the client. When the client receives the response message, it will perform the original do operation and continue to execute the client program. The message input responds to the request or message. Request ID, which contains the message identifier. The do operation in the client generates a request ID for each request message, and then the server copies these IDs to the corresponding response message. This allows Operation to verify that the response message is the result of the current request, rather than a delayed previous call. The third field is remote reference. The fourth field is the identifier of the operation to be invoked.

Message identifiers

• Any message management solution requires that each message has a unique message identifier, through which it can be referenced.

• The message identifier consists of two parts:

1. request Id, which is extracted from the increasing integer sequence by the sending process; (unique for the sender)
2. The identifier of the sender process, such as its port and Internet address (unique in a distributed system)

• The second part can be obtained from the received message.

• When the value of request Id reaches the maximum value, it will be reset to zero.

• But the lifetime of the message identifier must be much shorter than the time required to exhaust the values in the sequence of integers.

Failure model of the request-reply protocol

• If these three primitives perform operations, get requests and send responses on UDP datagrams, they will suffer the same communication failure. this is:

• They suffered failures of negligence.

• There is no guarantee that the messages are sent in the order of the sender.

• In addition, the protocol may encounter process failures

• We assume that the process has failed. In other words, when they stop, they will stop-

• In order to solve the situation where the server fails or the request or response message is abandoned, the "operation" will use a timeout while waiting for the response message from the server.

If the deadline is exceeded, the action to be taken depends on the delivery guarantee provided.

Timeouts

• After timeout, the operation can perform various options.

• The simplest option is to immediately return the failure of the do operation.

• This is not the usual method-the delay may be caused by the loss of the request or response message, in the latter case, the operation will already be performed.

• In order to compensate for the possibility of missing messages, Operation repeatedly sends request messages until it obtains a response or reasonably determines that the delay is due to a lack of response from the server and not due to the delay. Lost message.

• Finally, when the operation returns, it will notify the customer that no result has been received.

Discarding duplicate request messages

• In the case of resending the request message, the server can receive it multiple times.

• For example, the server may receive the first request message, but it takes longer than the client's expiration time to complete the command and return a response.

• This may cause the server to perform operations on the same request multiple times.

• To avoid this, the protocol aims to identify consecutive messages (from the same client) with the same request identifier and filter out duplicates.

• If the server has not yet sent a response, you do not need to take any specific action-the server will forward the response after completing the operation.

Lost reply messages

• If the server has sent a response when it receives a repeated request, it must perform the operation again to obtain the result unless the result of the original execution is stored.

• Some servers can run their operations multiple times and get the same results each time.

• Idempotent operations are operations that can be performed repeatedly, and the effect is the same as performing them only once.

• Servers where all operations are idempotent do not require special measures to avoid performing their operations multiple times.

**Q2. Explain the technical difference between RPC and RMI?**

**ANS:** Remote procedure call (RPC) is a network communication protocol with server and client architecture. The idea of RPC is to call remotely implemented code, just like we are just calling functions. The only difference between RMI and RPC is the case of calling RPC functions through proxy functions, while for RMI, we call methods through proxy objects.

RMI is a Java solution for RPC that can be connected to existing systems using native methods. RMI can use natural, direct and fully optimized methods to provide enterprise distributed computing technology, which enables us to add Java functionality to the entire system. In order to obtain the cross-platform portability provided by Java, RPC requires more overhead than RMI. RPC must convert parameters between architectures so that each computer can use its native data type.

Java-RMI is closely related to the Java language. RPC is not specific to a language; we can use other languages to implement RPC.

Because RMI can be implemented using Java, it has all advantages, such as object-oriented computing, design model, easy to write and reuse, security and safety, writing unique and executing anytime, anywhere. But for RPC, to get these benefits, you need to write the implementation code.

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

**ANS:** Direct communication is a word that conveys a clear message or clearly indicates action. Direct communication is often used in the workplace to ensure clarity about who has the authority to issue orders and what they are. When there is no room for discussion or compromise, direct communication can be used. This style usually does not allow listeners to respond with opinions or opinions. For example, your supervisor may tell you: "You must go to work on time every day. You don't have to be late again. This is simple, right? There is little room to misunderstand what your supervisor means.

Please consider that if the supervisor chooses to communicate this information more indirectly, and this method of communication does not convey absolutely clear information, or instructs you to perform the actions specified explicitly, the information may be different. Suppose your supervisor tells you: "Please consider your daily arrival time." You may know that you are sometimes late for work 15 minutes or more, and the supervisor may tell you that they do not like being late. However, she started to convey the information to you through indirect communication in a vague way or in a way involving only ideas.

You may well explain your supervisor's statement that you should "understand the daily arrival time" as a suggestion that you know exactly the time of arrival, and may make up for your lost time. For example, later in the day jobs. The indirect and less clear information of the supervisor may be more pleasant, but it does not convey specific or obvious ideas or instructions.

Indirect communication is defined as communication between entities of a distributed system through an intermediary, without direct coupling between the sender and receiver.

Strong coupling-interacting through a stable interface-hard-coded API calls and loose coupling-flexible relationships between two or more systems or organizations with a certain exchange relationship-each end of the transaction is clearly required, for example. As an interface description, and without making any assumptions on the other end decoupling-using (event) messages (for example, through message-oriented middleware (MoM), decoupling in space and time, publish and subscribe))-usually no Asynchronous indirect communication of state (for example, publish-subscribe or complex event processing system)

Key properties

Spatial separation: The sender does not know or need to know the identity of the receiver, and vice versa.

Time decoupling: The sender and receiver can have independent lifetimes. Indirect communication is often used in distributed systems where changes are expected.

example

• Mobile environment where users can quickly connect and disconnect from the network

• Manage the flow of events in the financial system

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

**ANS:** In a computer network, multicast is a type of group communication in which data transmission is addressed to a group of target computers simultaneously. Multicast can be one-to-many or many-to-many distribution. Multicast should not be confused with point-to-multipoint communication at the physical layer. Group communication is very important, because the group can make decisions through information, manage conflicts and establish the necessary relationships to put the group in a difficult situation. The exchange of information determines what the team will be and what the team can accomplish. For example, the way families exchange messages about pending choices shapes important characteristics, such as mutual understanding among members, whether they will respect each other, and whether they are motivated to make decisions. In many applications, the common operation is to send the same message to multiple recipients. Your CDN reader site is a good example. After adding a new book to the site, it must be distributed to all mirror sites. One method is to require the main server to send new content to each mirror-multiple unicasts. This method puts a great burden on the server. IP multicast and overlay multicast provide a mechanism to deliver the same message to multiple recipients more efficiently.

More generally, group communication refers to the idea of allowing a process to communicate with a group of processes, usually without knowing which processes belong to the group. Group communication usually involves reliable delivery of messages and the order in which messages are received in each process.
The term conversion here means that certain data (packet stream) is transmitted to the client receiver through the communication channel, which helps them communicate. Let us look at some of the "dominant" concepts that are common in the field of computer networks. IP multicast is a one-to-many and many-pair real-time communication technology on the IP infrastructure in the network. It does not require the identity of the a priori receptor or the number of a priori receptors, so it can adapt to a larger group of receptors. Multicast effectively utilizes the network infrastructure by requiring the source to send only one packet, even if it must be delivered to a large number of receivers. Network nodes (usually network switches and routers) are responsible for copying packets to reach multiple recipients, so that messages are sent only once through each network link.
Multicast IP routing protocol is used to distribute data (for example, audio/video streaming broadcast) to multiple recipients. Using multicast, the source can send a single copy of the data to a single multicast address and then distribute it to the entire recipient group. It is a specific form of IP multicast used for multimedia streaming and other network applications. It uses multicast address blocks specifically reserved in IPv4 and IPv6. Many devices use multicast/broadcast traffic to advertise and discover services on the network. Protocols such as Bonjour and MDNS help stream to Apple TV or connect to a printer

1. Unicast-

This type of information transmission is useful when a single sender and a single receiver are involved. So, in short, you can describe it as a one-to-one transmission. For example, if a device with an IP address of 10.1.2.0 in one network wants to send traffic (data packets) to a device with an IP address of 20.12.4.2 in another network, unicast becomes a picture. It is the most common form of data transmission over the network.

2. Multicast-

In multicast, one or more senders and one or more receivers participate in data transmission traffic. In this method, traffic is skewed between unicast (one-to-one) and broadcast (one-to-one) limits. Multicasting allows the server to directly copy the data stream and then emulate and route it to the host requesting it. IP multicast requires the support of some other protocols, such as IGMP (Internet Group Management Protocol) for multicast routing to work. Also in class IP addressing, class D is reserved for multicast groups.

**Q5. Differentiate a between a network OS and distributed OS.**

**ANS:** Network operating systems belong to the category of distributed architecture, in which a large number of computer systems are connected to each other using a network. Although the implementation of the network operating system is simpler than the distributed operating system. The difference between a network operating system and a distributed operating system lies in the functions they have. For example, each system of a network operating system runs its own operating system, while a distributed operating system runs an operating system. The overall system-wide operating system.
Definition of Network operating system

The network operating system is a platform for running system software on the server and allowing the server to manage users, data, groups, security, applications, and other networking functions. It is considered to be the main form of operating system for distributed architecture. The idea of a network operating system is to allow resources to be shared between two or more computers running its own operating system. The following diagram illustrates the operation of the network operating system.
This is the network OS layer between the local OS kernel and user processes. Essentially, the process interacts with the OS layer of the network, not the kernel of the local operating system. When the process requests a non-local resource, the OS layer of the network will communicate with the OS layer of the network containing the node of the

resource and use it to access the resource. On the other hand, if the process requests local resources, the OS layer of the network will send the request to the local OS kernel.
Unlike distributed operating systems, network operating systems cannot work together. The local operating system that resides on each specific computer retains its logo, which is also visible to users and behaves like a separate operating system. In some implementations, there are remote connections that allow remote operating systems to access resources. The network operating system cannot control resource usage, which results in resource allocation errors. There are no fault tolerance provisions in the network operating system.
Definition of Distributed operating system

A distributed operating system manages a group of independent computers and makes them look like ordinary centralized operating systems. This is achieved by allowing good communication between different computers connected to each other. The main goal of a distributed operating system is transparency, which hides the use of multiple hardware resources from users. Distributed operating systems have less autonomy than network operating systems because the system has full control in this environment. It dynamically assigns processes to random CPUs, and file storage is also managed by the operating system, which means users will not know what hardware is used to process their calculations and store their files. .
As mentioned above, a distributed operating system allows resource sharing, where applications can use resources located in any computer system. It provides availability (continuity of service) rather than failure. The distributed operating system manages the operations of all nodes in the system in an integrated manner, because each node has its own core to perform control functions on its behalf. By executing the calculation components in different computer systems, the calculation speed can also be increased.
Key Differences Between Network operating system and Distributed operating system
1. The main goal of the network operating system is to provide local services to remote clients. On the other hand, the goal of distributed operating systems is to provide hardware resource management.
2. The network operating system is intended to be a loosely coupled system and used in heterogeneous computers. On the other hand, distributed operating systems are considered to be tightly coupled systems, mainly used in multiprocessors or similar computers.
3. The network operating system has a two-tier client/server architecture, while the distributed operating system uses an n-tier architecture.
4. The transparency of the network operating system is low. In contrast, distributed operating systems exhibit great transparency and mask the use of resources.
5. In a distributed operating system, communication between computers (nodes) is achieved by sharing memory or sending messages. Instead, the network operating system sends files to communicate with other nodes.
6. The network operating system manages the resources on each node, while the resources in the distributed operating system are managed globally, whether centralized or distributed.
7. Compared with the distributed operating system, the network operating system is easy to implement.
8. The scalability of the network operating system is greater than that of the distributed operating system, and it is also open to users.
9. In the network operating system, the operating system installed on the computer may be different, but not in the distributed operating system.
10. The network operating system is more autonomous than the distributed operating system. However, distributed operating systems have higher fault tolerance.

**Q6. Describe briefly how the OS supports middleware in a distributed system by providing and managing.**
**ANS:** The middleware in the context of distributed applications is a type of software that provides services beyond those provided by the operating system to allow various components of the distributed system to communicate and manage data. Middleware supports and simplifies complex distributed applications. It includes a web server, application server, email, and similar tools that support application development and delivery. Middleware is an integral part of modern information technology based on XML, SOAP, Web services and service-oriented architecture.

Middleware usually allows interoperability between applications running on different operating systems, thereby providing services so that applications can exchange data in a standardized manner. Middleware is somewhere between application software that can run on different operating systems. It is similar to the middle layer of a single three-tier system architecture, except that it is distributed across multiple systems or applications. Examples include EAI software, telecommunications software, transaction monitors, and messaging and queuing software. The distinction between operating system and middleware functions is somewhat arbitrary. Although kernel functions can only be provided by the operating system itself, some functions previously provided by separately sold middleware are now integrated into the operating system. A typical example is the TCP/IP stack used for telecommunications, which is currently included in almost all operating systems.

**1: Process and threads**

An application contains one or more processes. Simply put, a process is a running program. One or more threads are running in the context of the process. A thread is the basic unit to which the operating system allocates processor time. A thread can execute any part of the process code, including the part executed by another thread. Work objects allow you to manage process groups as a unit. Work objects are named, protectable, and shareable objects that control the attributes of the processes associated with them. The operations performed on the work object affect all processes associated with the work object. A thread pool is a collection of worker threads that effectively execute asynchronous callbacks on behalf of the application. The thread pool is mainly used to reduce the number of application threads and manage worker threads. Fiber is a thread that must be manually scheduled by the application. The fiber runs in the context of the thread that schedules it.
Process:
The process indicates that any program is running. The process control block controls the operation of any process. The process control block contains information about the process, such as: process priority, process identifier, process status, CPU, registers, etc. A process can create other processes called child processes. This process takes longer to complete and is isolated, which means it will not share memory with any other process.
Thread:
A thread is part of a process, meaning that a process can have multiple threads, and these multiple threads are contained in a process. The thread has three states: running, ready and blocked. Compared to this process, this thread takes less time and does not isolate threads from similar processes.

**2: System Virtualization**

Compared with the normal operating system, operating system virtualization includes a modified form, so different users can use their different end applications. The entire process must be performed on one computer at a time. In operating system virtualization, the Virtual Eyes environment accepts commands from any user who uses it, and performs different tasks on the same computer by running different applications.

In operating system virtualization, even if the applications are running on the same computer, the applications will not interfere with each other. The operating system kernel allows multiple isolated user space instances. These instances are called software containers, and they are virtualization engines.

**Q7. Write in your own words the issues with Object (distributed) oriented middlewares.**
**ANS:** Of course, middleware is crucial to this kind of discussion about the pain of integration. In many cases, middleware has helped companies solve difficult IT problems, but in other cases, middleware has only caused problems and made the pain more serious, so the company just wants to avoid this situation. Come. So how do we understand middleware in a service-oriented context? Is this part of the solution or problem?
Unfortunately, the term "middleware" already means many different things. In the most general sense, middleware is software that transfers functions or data between two originally independent and independent systems. In most cases, middleware does not provide actual applications used by consumers or a recording system that stores data and

content information. In contrast, middleware is essentially a glue that combines different systems that provide the user interface, application logic, and data storage needed by distributed applications to meet business needs. If you have to connect the two systems together to work, but apply too much glue or glue in the wrong place, it will get stuck soon!

Thanks to mobile phones and tablets, today's business world has become an endless connection environment. This means that millions of people are constantly using voice, text, work and work today, and they use a large number of applications. All these applications cannot handle the workload by themselves. This must fall between middleware that bridges the gap between applications and operating systems.

The same is true of desktop computing and business processes. Middleware applications are the bridge that enables your computers to complete the work you want them to work with business systems. Without it, the database, client POS, internal messaging or any other process will be useless.

Managing this middleware is a difficult task. However, this is critical to the vitality of SOA and customer satisfaction. From mobile devices to offices, today's applications are carefully designed to connect to distributed servers, even to databases, and then connect to legacy applications running on mainframes. Very complicated, but very necessary. This brings challenges to the management of the middleware. These are three of these challenges.

1. Distribution of Applications and Infrastructure

The applications and infrastructure of the middleware architecture are highly dispersed. This means that when managing or testing, you need to do it in a holistic manner. You must adopt the entire system rather than the components to see the full integration.

2. Monitoring of Architecture

For managers and architects, another challenge is the continuous monitoring of middleware from the front-end application to the back-end. This requires an in-depth understanding of how the entire infrastructure works, as well as an understanding of the possible problems in this environment.

3. Middleware is Transparent to Applications

The middleware system is a landing platform for messages relayed between the operating system and the application itself. These messages are put in the queue. The data is then retrieved by another data server or part of the application, and the process is repeated from there. Middleware does not actually participate in the function of the application, but enables middleware to flow from one data point to another. Testing is an important part of any middleware management. Understanding the challenges you may face in these tests is one way to identify possible problems. Although these challenges are few, they can reveal how your process works.