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 Sept 2020**

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

Class and Section:_ 4th Semester

Section: Remote Invocation

Q1. Explain how a Request-Reply protocol handles the following situation. **(10)**

a) **Server receives same request more than once.**
b) **Server reply message is lost.**

Section: Indirect Communication

Q:3 Differentiate between any three types of indirect communication? **(15)**

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)**

Section: Remote Invocation

Q1. Explain how a Request-Reply protocol handles the following situation.

a) Server receives same request more than once.

b) Server reply message is lost.

ANS:

a) Server receives same request more than once.

A client sends a request to a server, which performs the requested action while the client waits. The server then sends the reply to the client, which receives the reply.
typical client-server interactions – request-reply communication is synchronous because the client process blocks until the reply arrives
The server can receive it multiple times. the server may receive the first request message, but it takes longer than the client timeout to complete the command and return a response. This may cause the server to perform more than one operation on the same request. To avoid this, the protocol aims to identify consecutive messages (from the same client) with the same request ID and filter out duplicates. If the server has not sent a response, it does not need to take any special action-it will forward the response after the operation is completed.

b) Server reply message is lost.

- The operating system starts a timer when the stub is generated and sends a request. If response is not received before timer expires, then a new request is sent.
- Lost message – works fine on retransmission.
- Many requests sent- cannot locate server.
- If request is not lost, we should make sure server knows that its a retransmission.
- Some messages can be retransmitted any number of times without any loss.
- Some retransmissions cause severe loss.
- Solution- client assigns sequence number on requests made by client.
- Drawback- Server maintains administration on each client. How long to maintain.

If the server has already sent a response when it receives a repeated request, it will need to rerun the operation to get the result unless the result of the original execution is stored. Some servers may run their operations multiple times and get the same results each time. Idempotent is an operation that can be performed multiple times, and its effect is the same as if it was performed just once. For example, the operation of adding an element to a set is an idempotent operation, because every time the operation is performed will have the same effect on the set, and the operation of adding an element to a sequence is indeed not an idempotent operation, because it will Run-time expansion sequence. The operations of the server are idempotent, and no special measures are required to avoid performing its operations multiple times.

<u>Section: Indirect Communication</u>

<u>Q:3</u> Differentiate between any three types of indirect communication?

<u>ANS:</u>

Three types of indirect communication

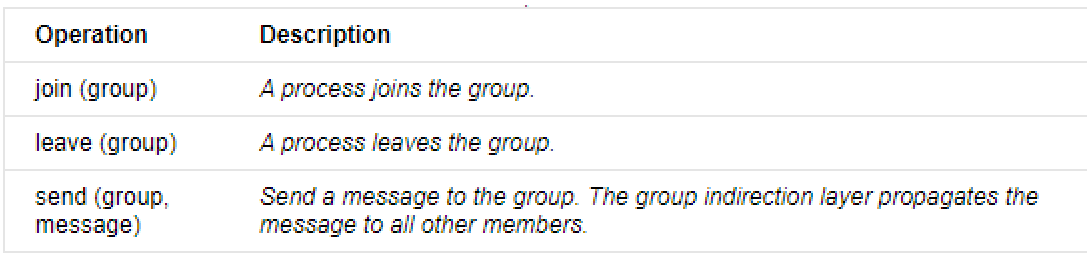
Group communication	**Publish-subscribe system**	**Message queues**
Group communication: is concerned with the delivery of messages to a set of recipients supporting one-to-many communication.	•Publish-subscribe systems: Many systems, such as the financial trading are systems wherein a large number of producers (or publishers) distribute information items of interest (events) to a similarly large number of consumers (or subscribers).	•Message queues: offer a point-to-point service whereby producer processes can send messages to a specified queue and consumer processes can receive messages from the queue or be notified of the arrival of new messages in the queue.
Group is represented in the system by a group identifier.		
Recipients elect to receive messages sent to a group by joining the group.	•It would be complicated and inefficient to employ any of the core communication paradigms discussed above for this purpose	
Senders then send messages to the group via the group identifier, and hence do not need to know the recipients of the message.	•Publish-subscribe systems ensures information generated by producers is routed to consumers who desire this information.	

1) Group communication

Group communication provides a service through which a message is sent to a group, and then the message is delivered to all members of the group. In this operation, the sender does not know the identity of the recipient. Group communication is an abstraction of multicast communication, which can be implemented on IP multicast or equivalent overlay networks, adding considerable additional value in terms of group membership management, fault detection and reliability. And order guarantee. With additional guarantees, group communication is multicast to IP, and TCP refers to point-to-point services in IP.

The operation set for group communication is shown in **table**

Table: Group communication

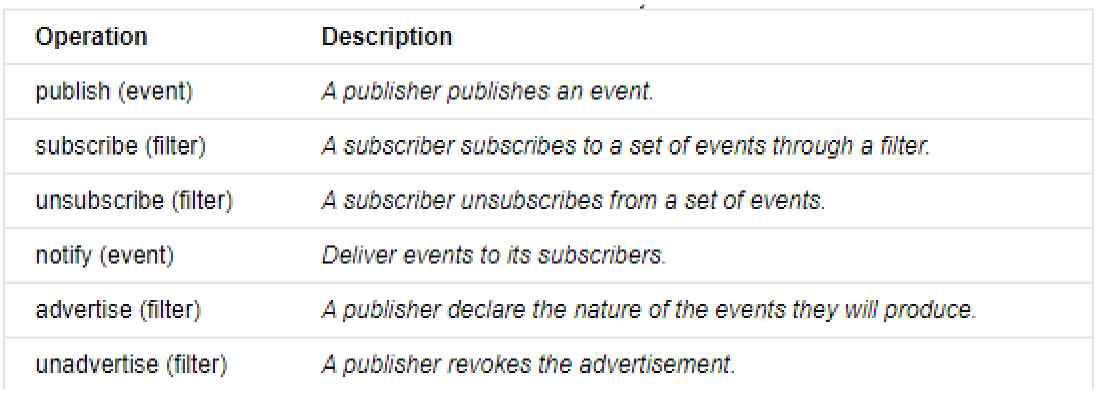


Operation	Description
join (group)	*A process joins the group.*
leave (group)	*A process leaves the group.*
send (group, message)	*Send a message to the group. The group indirection layer propagates the message to all other members.*

2) Publish-subscribe system

A Publish-subscribe system is a system in which a publisher publishes structured events to an event service, and subscribers express interest in specific events through subscriptions. Subscriptions can be any mode of structured events. For example, subscribers may express interest in any event related to this manual, such as whether there is a new version or an update to a related website. The task of the publish-subscribe system is to compare subscriptions with published events and ensure the correct delivery of event notifications. A given event will be broadcast to potentially many subscribers, so publish-subscribe is basically a one-to-many communication paradigm.
The operations of a publish-subscribe system are listed in **Table**

Table: Publish-subscribe system

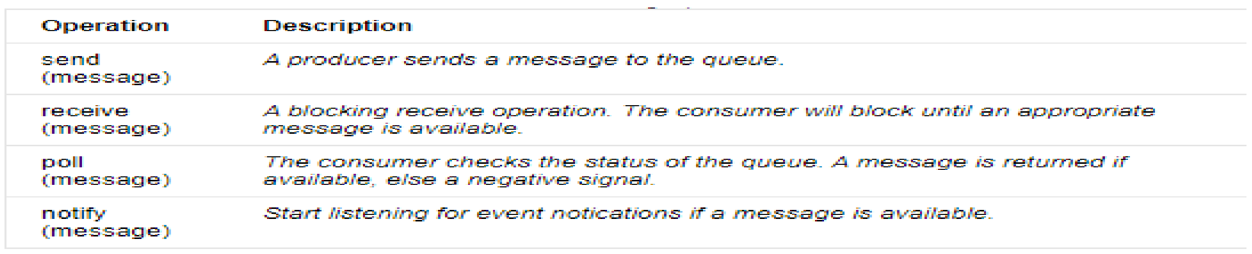


Operation	Description
publish (event)	*A publisher publishes an event.*
subscribe (filter)	*A subscriber subscribes to a set of events through a filter.*
unsubscribe (filter)	*A subscriber unsubscribes from a set of events.*
notify (event)	*Deliver events to its subscribers.*
advertise (filter)	*A publisher declare the nature of the events they will produce.*
unadvertise (filter)	*A publisher revokes the advertisement.*

3) Message queues

Another important category of indirect communication systems is message queues (or more accurately distributed message queues). When groups and publish-subscribe provide one-to-many communication methods, message queues use message queues as an indirect concept to provide point-to-point services, thereby obtaining the desired characteristics of decoupling space and time. They are peer-to-peer, the sender puts the message in the queue, and then a single process deletes it. Message queues are also called message-oriented middleware. It is a major type of commercial middleware with key implementations such as IBM Web Sphere MQ, Microsoft MSMQ and Oracle Streams Advanced Queuing (AQ). The main purpose of these products is to achieve enterprise application integration (EAI), that is, the integration between applications within a given enterprise, which is achieved by the following means: the inherent loose coupling of message queues.

The operations that can be invoked on a message queue are listed in **table**



Operation	Description
send (message)	*A producer sends a message to the queue.*
receive (message)	*A blocking receive operation. The consumer will block until an appropriate message is available.*
poll (message)	*The consumer checks the status of the queue. A message is returned if available, else a negative signal.*
notify (message)	*Start listening for event notications if a message is available.*

Messages are usually added to the queue based on the first-in-first-out (FIFO) policy, but priorities may be used as well. Message queues try to ensure reliable delivery by persisting messages: messages are eventually delivered (time uncoupling). Messages are also only sent once and as received to provide integrity.
The consumers may receive messages by actively checking (polling) if messages are available, or by receiving notifications that messages have become available. Messages may be filtered based on certain properties.

<u>Section: OS Support</u>

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

<u>**ANS:**</u>
The main difference between network OS and distributes OS is that distributed OS are for some specific uses only. They are not used generally. Whereas the network OS are generally used by the end users to run their applications and meet their needs.
Moreover, the network OS allows users to have a degree of autonomy and have interactive responsiveness of their machines where the work of one user is totally in his/her hand and does not affect the other whereas the distributed OS does not lead the end user to be autonomous as the programs of one user can affect the other.
In other words, both the Network Operating System and Distributed Operating System have a common hardware base but the difference lies in software. Some more of the differences are as:
A network OS is made up of software and associated protocols that allow a set of computer network to be used together while a distributed OS is an ordinary centralized operating system but runs on multiple independent CPUs.
In network OS, Environment users are aware of multiplicity of machines whereas in distributed OS, Environment users are not aware of multiplicity of machines.
Control over file placement is done manually by the user in network OS while in distributed OS, the control lover file placement can be done automatically by the system itself.

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

a) Process and threads

b) System Virtualization

ANS:

The task of middleware is to provide a high-level programming abstraction for the development of distributed systems. The OS supports middleware to do so with the following ways.

A) How the OS supports middleware in a distributed system by providing and managing Process and Threads:

The operating system running at a node supports middleware in distributed system by providing and managing process and threads associated with user level services such as communication libraries. Through the management of Process and threads, the OS provides ease to manage local hardware resources for processing, storage and communication. Middleware utilizes a combination of these local resources to implement its mechanism for remote invocations between objects or processes at the nodes. The OS enables the middleware to deliver distributed resources sharing to users. Kernel and server processes are the components that manage the resources and present clients with an interface to the resources by providing and managing process and Threads.

B) How the OS support middleware in a distributed system by providing and managing System Virtualization

They OS support middleware for System Virtualization by providing multiple virtual machines over underlying physical machine architectures. The OS enables the middleware to support potentially large numbers of virtual machines and multiplex resources between them. The virtualization system allocates the physical processor(s) and other resources of a physical machine between all virtual machines that it supports. Hence on server machines, an organization assigns each service it offers to a virtual machine and then optimally allocates the virtual machine to physical servers. This way, the middleware get supported by the OS in a distributed system by providing and managing system virtualization.

Section: Distributed Objects and Components

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

ANS:

As a matter of fact, an increasing number of next-generation applications will be developed as distributed “systems of systems,” which include many interdependent levels, such as network/bus interconnects, local and remote end systems, and multiple layers of common and domain-specific middleware. There are number of issues with Object (distributed) oriented middleware. Some of them are the following as per my own understanding of the topic.

Implicit Dependencies: object interfaces do not describe what the implementation of the objects depend on , making object-based systems difficult to develop and subsequently manage.

Programming Complexity: Programming distributed object middleware leads to a need to master many low-level details associated with middleware implementations.

Lack of Separation of Distribution Concerns: Application developers are obliged to consider details of concerns such as security, failure handling and concurrency, which are largely similar from one application to another.

Moreover, Predictability of Operating Characteristics, which is that it is difficult to predict accurate performance of the working of middlewares.
The other issue I understand is Controllability of Operating Characteristics, which states that it is difficult for the operating system to control all the characteristics of Object oriented middlewares.
Moreover, the adaptability of Operating Characteristics for applications is another key concern. They may not be adaptable with all components.
I believe that all the above issues are associated with respect to some features such as;
Time,
Quantity of information,
Accuracy,
Confidence, and
Synchronization.
All these issues become highly volatile in systems of systems, due to the dynamic interplay of the many interconnected parts. These parts are often constructed in a similar way from smaller parts. Given the complexity of the issues with Object Oriented middlewares, various tools and techniques are needed to configure and reconfigure these issues hierarchically so they can adapt to a wider variety of situations and result in smooth operation.
