Distributed Computing Assignment - Unit 1

Q1. Write a short note on the Client Server Model.

A1. The client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests. Examples of computer applications that use the client–server model are Email, network printing, and the World Wide Web.

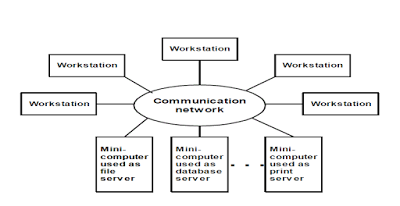
The client-server characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services.Servers are classified by the services they provide. For example, a web server serves web pages and a file server serves computer files. A shared resource may be any of the server computer's software and electronic components, from programs and data to processors and storage devices. The sharing of resources of a server constitutes a service.Whether a computer is a client, a server, or both, is determined by the nature of the application that requires the service functions. For example, a single computer can run web server and file server software at the same time to serve different data to clients making different kinds of requests. Client software can also communicate with server software within the same computer.Communication between servers, such as to synchronize data, is sometimes called inter-server or server-to-server communication.In general, a service is an abstraction of computer resources and a client does not have to be concerned with how the server performs while fulfilling the request and delivering the response. The client only has to understand the response based on the well-known application protocol, i.e. the content and the formatting of the data for the requested service.

Clients and servers exchange messages in a request–response messaging pattern. The client sends a request, and the server returns a response. This exchange of messages is an example of inter-process communication. To communicate, the computers must have a common language, and they must follow rules so that both the client and the server know what to expect. The language and rules of communication are defined in a communications protocol. All client-server protocols operate in the application layer. The application layer protocol defines the basic patterns of the dialogue. To formalize the data exchange even further, the server may implement an application programming interface (API).The API is an abstraction layer for accessing a service. By restricting communication to a specific content format, it facilitates parsing. By abstracting access, it facilitates cross-platform data exchange.A server may receive requests from many distinct clients in a short period of time. A computer can only perform a limited number of tasks at any moment, and relies on a scheduling system to prioritize incoming requests from clients to accommodate them. To prevent abuse and maximize availability, server software may limit the availability to clients. Denial of service attacks are designed to exploit a server's obligation to process requests by overloading it with excessive request rates.

Q2. What is Workstation Server Model?

A2. The workstation model is a network of personal workstations, each with its own disk and a local file system. A workstation with its own local disk is usually called a diskful workstation and a workstation without a local disk is called a diskless workstation. With the proliferation of high-speed networks, diskless workstations have become more popular in network environments than diskful workstations, making the workstation-server model more popular than the workstation model for building distributed computing systems.A distributed computing system based on the workstation-server model consists of a few minicomputers and several workstations (most of which are diskless, but a few of which may be diskful) interconnected by a communication network.Note that when diskless workstations are used on a network, the file system to be used by these workstations must be implemented either by a diskful 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.

Therefore in the workstation-server model, in addition to the workstations, there are specialized machines (may be specialized workstations) for running server processes (called servers) for managing and providing access to shared resources. 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 network, for managing the files of all the users in the system. Due to this reason, a distinction is often made between the services that are provided to clients and the servers that provide them. That is, a service is an abstract entity that is provided by one or more servers. For example, one or more file servers may be used in a distributed computing system to provide file service to the users.



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 migrated to the server machines for getting the work done by those machines.

Q3. Write a short note on TCP/IP and UDP/IP communications and programming sockets in Java.

A3. There are two communication protocols that one can use for socket programming: datagram communication and stream communication.

Datagram communication: The datagram communication protocol, known as UDP (user datagram protocol), is a connectionless protocol, meaning that each time you send datagrams, you also need to send the local socket descriptor and the receiving socket's address. As you can tell, additional data must be sent each time a communication is made.

Stream communication: The stream communication protocol is known as TCP (transfer control protocol). Unlike UDP, TCP is a connection-oriented protocol. In order to do communication over the TCP protocol, a connection must first be established between the pair of sockets. While one of the sockets listens for a connection request (server), the other asks for a connection (client). Once two sockets have been connected, they can be used to transmit data in both (or either one of the) directions.

## Programming sockets in Java:

1. Open a socket:

Where Machine name is the machine you are trying to open a connection to, and PortNumber is the port (a number) on which the server you are trying to connect to is running. When selecting a port number, you should note that port numbers between 0 and 1,023 are reserved for privileged users (that is, super user or root). These port numbers are reserved for standard services, such as email, FTP, and HTTP. When selecting a port number for your server, select one that is greater than 1,023!

Socket MyClient;

try {

MyClient = new Socket("Machine name", PortNumber);

}

catch (IOException e) {

System.out.println(e);

}

If you are programming a server, then this is how you open a socket:

ServerSocket MyService;

try {

MyServerice = new ServerSocket(PortNumber);

}

catch (IOException e) {

System.out.println(e);

}

When implementing a server you also need to create a socket object from the ServerSocket in order to listen for and accept connections from clients.

Socket clientSocket = null;

try {

serviceSocket = MyService.accept();

}

catch (IOException e) {

System.out.println(e);}

2. Create an input stream:

On the client side, you can use the DataInputStream class to create an input stream to receive response from the server:

DataInputStream input;

try {

input = new DataInputStream(MyClient.getInputStream());

}

catch (IOException e) {

System.out.println(e);

}

The class DataInputStream allows you to read lines of text and Java primitive data types in a portable way. It has methods such as read, readChar, readInt, readDouble, and readLine,. Use whichever function you think suits your needs depending on the type of data that you receive from the server.

On the server side, you can use DataInputStream to receive input from the client:

DataInputStream input;

try {

input = new DataInputStream(serviceSocket.getInputStream());

}

catch (IOException e) {

System.out.println(e);

}

3. Create an output stream:

On the client side, you can create an output stream to send information to the server socket using the class PrintStream or DataOutputStream of java.io:

PrintStream output;

try {

output = new PrintStream(MyClient.getOutputStream());

}

catch (IOException e) {

System.out.println(e);

}

The class PrintStream has methods for displaying textual representation of Java primitive data types. Its Write and println methods are important here. Also, you may want to use the DataOutputStream:

DataOutputStream output;

try {

output = new DataOutputStream(MyClient.getOutputStream());

}

catch (IOException e) {

System.out.println(e);

}

The class DataOutputStream allows you to write Java primitive data types; many of its methods write a single Java primitive type to the output stream. The method writeBytes is a useful one.

On the server side, you can use the class PrintStream to send information to the client.

PrintStream output;

try {

output = new PrintStream(serviceSocket.getOutputStream());

}

catch (IOException e) {

System.out.println(e);

}

Note: You can use the class DataOutputStream as mentioned above.

4. Close sockets:

On the client side:

try {

output.close();

input.close();

MyClient.close();

}

catch (IOException e) {

System.out.println(e);

}

On the server side:

try {

output.close();

input.close();

serviceSocket.close();

MyService.close();

}

catch (IOException e) {

System.out.println(e);

}

Q4. What is IPC Synchronization?

A4. Central issue in communication structure is synchronization imposed on the communicating processes by the communication primitives.Two semantics Blocking and non-blocking can be used.A primitive is said to havenon-blocking semantics if its invocation does not block the execution of its invoker.If execution of invoker is blocked, it isblocking semantics.These semantics are primarily used for send and receiveprimitives. Incase of blocking send primitive, sending process is blocked after execution of send until it receives an ACK from receiver that themessage is received.Incase of non-blocking send, process proceeds with its execution assoon as the message get copied to buffer ( transferred if Null-buffer isused).

Incase of block-receive, receiving process is blocked until itreceives a message (ACK).

Incase of non-blocking receive, process proceeds with its execution as soon as receive primitive is executed.

Non-blocking receive knows that message is arrived in buffer by:

1.Polling :a test primitive is provided to allow receiver to check the buffer status. A periodic execution of test is carried out called as polling.

2.Interrupt :when buffer get filled and becomes ready to beused by receiving process a software interrupt notifies thisto receiver. Saves repeated unsuccessful check of polling.

A variant to non-blocking receive primitive is conditional receive primitive. This returns control to invoking process immediately, either with a message or with an indicator of no-message.In blocking-send primitive, sending process could get blocked forever if receiver crashes or if message loss due to other reasons. To avoid this blocking primitives uses time-outvalue ( time-stamp, waiting-time) specifying interval of timeafter which the operation of blocking-primitive ( blocking-send) is terminated with an error status.Time-out value is either default ( system calculated ) or user defined ( human-time) with respect to communication criteria.

Q5. Explain Message Buffering Strategies in IPC.

A5. The different strategies are:

1.Null Buffer :

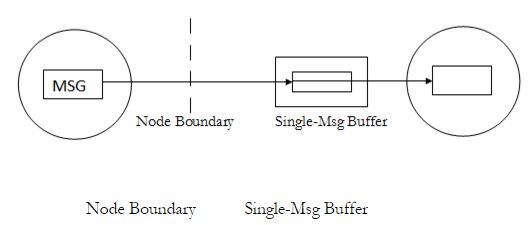
* No place to temporarily store message.
* One of following implementation strategy is used.
* Message remains in senders address space and execution of send is delayed until receiver finishes current receive.
* When receiver finishes, act is sent to sender kernel, then sender is unblocked and send is executed.
* Message is discarded & timeout mechanism is used to resend message after timeout period.



Fig. Logical Path of MSG Transfer is directly from sender’s address space to Receiver Address Space.

**2] Single-Message Buffer:**

Synchronous communication use single msg buffer because atmost, one msg may be outstanding at a time.



**3] Unbounded Capacity Buffer:**

In the asynchronous mode of communication since sender does not wait for receiver to ready, there may be several pending msgs, so unbounded capacity buffer is used.

**4] Finite Bound Buffer (Multiple Msg):**

Unbounded Capacity of buffer is practically impossible so in practice systems using asynchronous mode used finite bound buffer also known multiples msg buffer.

* With Finite bounds, buffer overflow is the problem.
* Thus can be deal with one of the following problems.

**Unsuccessful Communication :**

Msg Transfer simply fails when there is no more buffer space, send return error message to sending process.

**Flow Controlled Communication :**

You use flow control the sender is blocked until receiver accepts some messages, thus creating space in buffer for new msgs.



Submitted by:

Rashi Tyagi (1MS14IS085)

message.Such messages needto be stored somewhere, usually in the buffer of receiver,for later reception and it is slower than asynchronous communication.ommunicationbetween two processes useblocking semantics, thecommunication is said to besynchronous; otherwise it issaid to be asynchronous.Synchronous communicationcomparatively easy toimplement than asynchronouscommunicationen both send and receiveprimitives of a communicationbetween two processes useblocking semantics, thecommunication is said to besynchronous; otherwise it issaid to be asynchronous.Synchronous communicationcomparatively easy toimplement than asynchronouscommunicationWhen both send and receiveprimitives of a communicationbetween two processes useblocking semantics, thecommunication is said to besynronous; otherwise it issaid to be asynchronous.Synchronous communicationcomparatively easy toimplement than asynchronouscommunication