**TOPIC THREE: Inter-process Communication Techniques (IPC)**

The communication between two processes takes place via buffer. The alternative way of Communication is the process of the inter-process communication. The simple mechanism of this is synchronizing their action and without sharing the same addresses space. This play an important role in the distribute processing environment. The Structured Query Language (SQL) Server uses Named Pipes as the communications protocol.

Named **Pipes** is a inter-process communication (IPC) protocol common to multitasking operating systems such as UNIX and OS/2, and it is used in SQL Server to manage both client and server communications

across the network.

While The more redefined method of inter process communication are message queues, semaphores and shared memory. There are four types of mechanisms, involved for such a communications:-

1. Message passing.
2. Direct communication.
3. Indirect communication.
4. Remote procedures call.
5. **Message passing:** This mechanism allows process to communicate without restoring the shared data, for example in micro kernel, message is passed to communicate in which services acts as an ordinary user where these services act outside the kernel. At least, there are two processes involved in an IPC. See the Figure given below:

**As per drawings from lectures class**

Figure : Message Passing

* Sending process for sending the message.
* Receiving process for receiving the message.

1. **Direct communication:** In this mechanism of communication processes have to specify the name of sender and recipient process name. This type of communication has the following features:

* A link is established in between the sender and receiver along with full known information of their names and addresses.
* One link must be established in between the processes.
* There is symmetry in between the communication of the processes.

1. **Indirect communication:** In indirect communication, messages are sending to the mail box and then they are retrieved from mailbox, see the Figure given below:

The role of the mailbox is quite similar to the role of the postman. The indirect communication can also communicate with other processes via one or more mailbox. The following features are associated with indirect communication:

* A link is established between a pair of process, if they share a mailbox.
* A link is established between more than one processes.
* Different number of links can be established in between the two communicating processes.

Communication between the processes takes place by executing calls to the send andreceive primitive. Now there is several different ways to implement these primitives, they can be “blocking” and “non-blocking”. The different possible combinations are:

* *Blocking send*: Sending the process is blocked until the message is received.
* *Non-blocking send*: In it process sends the message and then it resumes the operation.
* *Blocking receive:* Receiver is blocked until the message is available.
* *Non-blocking receive:* The receiver receives either a valid message or a null.

1. **Remote procedures call:** RPC is a powerful technique for constructing distributed client-server based applications. The essence of the technology is to allow programs on different machines to interact using simple procedure call or return semantics, just as if the two programs were on the same machine. It is based on extending the notion of conventional or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them. That is, the procedure call is used for access to remote services.

In client-server based applications a binding is formed when two applications have made a logical connection and are prepared to exchange commands and data. This client server binding specifies how the relationship between a remote procedure and the calling program will be established.

**How RPC Works**:

When an RPC is made, the calling arguments are passed to the remote procedure and the caller waits for a response to be returned from the remote procedure. The general architecture of remote procedure call mechanism that takes place during an RPC call between two networked systems. The client makes a procedure call that sends a request to the server and waits. The thread is blocked from processing until either a reply is received, or it times out. When the request arrives, the server calls a dispatch routine that performs the requested service, and sends the reply to the client. After the RPC call is completed, the client program continues. RPC specifically supports network applications. RPCs also manage the network interface and handle security and directory services. Tools of RPC comprises of:

* A language and a compiler to produce portable source code.
* Some run time facilities to make the system architecture and network protocols transparent to the application procedure.

The mechanism of RPC can be considered as a refinement of reliable, blocking message passing. Figure given below, illustrates the general architecture understanding.

**As per drawings from lectures class**

**Figure:** RPC Mechanism

Tools of RPC comprises of:

• A language and a compiler to produce portable source code.

• Some run time facilities to make the system architecture and network protocols transparent to the application procedure.

***Limitation of RPC:*** There are number of limitations associated with RPC given below.

1. RPC requires synchronous connections. If an application uses an RPC to link to a server that is busy that time then application will have to wait for the data rather than switching to other task.
2. Local procedure call fails under the circumstances where RPC can be duplicated under and executed more than one, which is due to unreliable communication.
3. The communication in between the client and server is done with help of the standard procedure calls; therefore some binding must take place during the link load and execution, such that the process is replaced by the address. The RPC binds the same thing to the client and server. A general problem that exists is that there is no shared memory in between them so how they can come to know about the address of the other system.
4. The binding information may be predetermined in the form of the port address, at the compile time, a RPC call, that has a fix port number is associated with it. Once a program is compiled, it then cannot change its port number.
5. Binding can be done dynamically by rendezvous mechanism. Typically an operating system provides rendezvous demon requesting the port address of RPC, it needed to execute. The port address is then returned and the RPC call may be sent to the port until the process terminates.

**Client-Server Communication**

Several other strategies for communication in client-server systems:

* ***SOCKETS***

***LESSON FOUR : Sockets***

***Socket-based Client Server Systems***

Sockets are the endpoints of two-way communication between programs running in a network. They are software objects that connect applications to network protocols, so they become intelligible.

Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O actions are done by writing or reading to a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else.

To a programmer a socket looks and behaves much like a low level file descriptor. This is because commands such as read() and write() work with sockets in the same way they do with files and pipes. The differences between sockets and normal file descriptors occurs in the creation of a socket and through a variety of special operations to control a socket.

**Where is Socket used?**

Socket is used in a client server application frameworks. A server is a process which does some function on request from a client. Most of the application level protocols like FTP, SMTP and POP3 make use of Sockets to establish connection between client and server and then for exchanging data.

# Socket Types:

There are four types of sockets available to the users. The first two are most commenly used and last two are rarely used.

Processes are presumed to communicate only between sockets of the same type but there is no restriction that prevents communication between sockets of different types.

* **Stream Sockets:** Delivery in a networked environment is guaranteed. If you send through the stream socket three items "A,B,C", they will arrive in the same order - "A,B,C". These sockets use TCP (Transmission Control Protocol) for data transmission. If delivery is impossible, the sender receives an error indicator. Data records do no have any boundaries.
* **Datagram Sockets:** Delivery in a networked environment is not guaranteed. They're connectionless because you don't need to have an open connection as in Stream Sockets - you build a packet with the destination information and send it out. They use UDP (User Datagram Protocol).
* **Raw Sockets:** provides users access to the underlying communication protocols which support socket abstractions. These sockets are normally datagram oriented, though their exact characteristics are dependent on the interface provided by the protocol. Raw sockets are not intended for the general user; they have been provided mainly for those interested in developing new communication protocols, or for gaining access to some of the more esoteric facilities of an existing protocol.
* **Sequenced Packet Sockets:** They are similar to a stream socket, with the exception that record boundaries are preserved. This interface is provided only as part of the Network Systems (NS) socket abstraction, and is very important in most serious NS applications. Sequenced-packet sockets allow the user to manipulate the Sequence Packet Protocol (SPP) or Internet Datagram Protocol (IDP) headers on a packet or a group of packets either by writing a prototype header along with whatever data is to be sent, or by specifying a default header to be used with all outgoing data, and allows the user to receive the headers on incoming packets.
* A socket is identified by an IP address
* Concatenated with port number
  + Sockets use client-server architecture
  + The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**
  + Communication consists between a pair of sockets



**Server Side** **Client Side**

1. socket(); 1. socket();

2. bind(); 2. connect();

3. listen(); 3. send()/recv()

4. accept();

5. send()/recv()

**Remote Procedure Calls**

Remote procedure call (RPC) abstracts procedure calls between processes on networked systems

**Stubs** → Client-side proxy for the actual procedure on the server

* **Client-side stub** locates server and *marshalls* the parameters
* Server-side stub receives this message, unpacks the *marshalled* parameters, and performs the procedure on server

Data Representation client and server machines

**Semantics**

* *at most once*, *exactly once*

**Remote Method Invocation**

Remote Method Invocation (RMI) is a Java mechanism similar to RPCs

*RMI = RPC + Object-Orientation*

RMI allows a Java program on one machine to invoke a method on a remote object



**Client/server application architecture**

There are variety of ways to divide the processing between client and server. But the exact

distribution of data and application programming depends on the nature of the database,

the type of application supported, the availability of interoperable vendor equipment, and

the usage patterns within an organization. Depending on the database applications various

classes of Client/Server Application has been characterized

1. Host-based processing.
2. Server-based processing.
3. Client-based processing.
4. Cooperative processing.

**Host-based processing:** Virtually all the processing is done on a central host, often user interface is via a dumb terminal. It is mainly mainframe environment, not true Client/Server computing. In such a processing’s workstations have very limited role as shown in Fig. 5.1 given below:

**Server-based processing:** All the processing is done on the server, and server is responsible for providing graphical user interface. A considerable fraction of the load is on the server, so this is also called *fat server* model shown in Fig. 5.2 given below:

**Client-based processing:** Virtually all application processing may be done at the client, with the exception of data validation routines and other database logic functions that are best performed at the server. Some of the sophisticated database logic functions residing on the client side. This architecture is the most common Client/Server approach in current use. It enables the user to employ applications tailored to local need shown in Fig 5.3 given below:

**CLIENT SERVICES**

Any workstation that is used by a single user is a client. Some of the main services that client performs (role of client) are listed below:

* Responsible for managing the user interface.
* Provides presentation services.
* Accepts and checks the syntax of user inputs. User input and final output, if any, are presented at the client workstation.
* Acts as a consumer of services provided by one or more server processors.
* Processes application logic.
* The role of the client process can be further extended at the client by adding logic that is not implemented in the host server application. Local editing, automatic data entry, help capabilities, and other logic processes can be added in front of the existing host server application.
* Generates database request and transmits to server.
* Passes response back to server.

Other important services that are directly or indirectly attached with the client services are given below:

1. Inter process communication.
2. Remote services.
3. Window services.
4. Dynamic data exchange.
5. Object linking and embedding.

**Remote Services**

In client server model applications can be invoked directly from the client to execute remotely on a server. The workstation is responsible to provide various remote services. Among them some services like remote login, remote command execution, remote backup services, remote tape drive access and remote boot services, and remote data access are important. Software available with Network Operating System is responsible to run on the client workstation to initiate all these remote services. Then partial operating system will be able to load the remote software’s that provides the remaining services and applications functions to the client workstation. This is known as remote boot service provided by client workstation and X-terminals.

**Window Services**

In client server application, operating system at the client workstation provides some windows services, these services are capable of to move, view, activate, hide, or size a particular window. This is very helpful in implementation of several applications because a client workstation may have several windows open on-screen at a time. The NOS provides some software’s on the client workstation which is able to manage the creation of pop-up windows that display alerts generated from remote servers.

**Dynamic Data Exchange (DDE)**

DDE is usually described as a conversation between two applications, a client application and a server application. As we know that the client program is on that requests (receives) the information, and the server is the one that response (supplies) it. A DDE link may be automatic or manual. An automatic link is refreshed whenever the source data changes, provided both the client and server applications are running. A manual link is refreshed only when you issue a command in the client application.

**Object Linking and Embedding (OLE)**

Object Linking and Embedding two services collectively called as a single one, carried out with simple edit menu procedures. OLE is a software package accesses data created from another through the use of a *viewer* or *launcher*. These viewers and launchers must be custom built for every application. With the viewer, users can see data from one software package while they are running another package. Launchers invoke the software package that created the data and thus provide the full functionality of the launched software. To link with OLE copy data from OLE supporting program to the Clipboard.

**SERVER SERVICES**

The server is responsible for controlling and providing shared access to available server resources. Remote workgroups have needed to share these resources when they are connected with server station through a well-managed network. The applications on a server must be isolated from each other so that an error in one application cannot damage another application.

The server is responsible for managing the server requester interface so that an individual client request response is synchronized and directed back only to the client requester. This implies both security when authorizing access to a service and integrity of the response to the request.

the server is responsible for managing the server requester interface so that an individual client request response is synchronized and directed back only to the client requester. This implies both security when authorizing access to a service and integrity of the response to the request tasks from providing service

There must be a means of defining the relative priority of the tasks on the server. These are specific requirements to the Client/Server implementation. One of the prime server characteristic is that it must support for multiple simultaneous client request for service. So that, the server must provide shared memory services and multitasking support The main operations that server perform are

1. Accepts and processes database requests from client.
2. Checks authorization.
3. Ensure that integrity constraints are not violated.
4. Performs query/update processing and transmits response to client.
5. Maintains system catalog.
6. Provide concurrent database access.
7. Provides recovery control.