**Assignment No. 1**

1. **SOCKET PROGRAMMING USING THREAD**

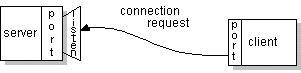
Implement using Socket Programming (TCP/UDP/IP) in C / C++ / JAVA.

* 1. Addition of digits of a given number. (Ex 12345 = 15).
  2. Find the Factorial of a Number.
  3. Perform String Operations. (Length, Lower to Upper)

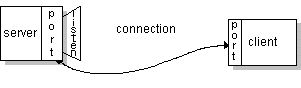
**What is Socket ?**

Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.



If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.

The client and server can now communicate by writing to or reading from their sockets.

**Definition:**  A *socket* is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent.

An endpoint is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints. That way you can have multiple connections between your host and the server

**3.2 Definition of Socket**

A socket is a communication endpoint — an object through which a Windows Sockets application sends or receives packets of data across a network. A socket has a type and is associated with a running process, and it may have a name. Currently, sockets generally exchange data only with other sockets in the same “communication domain,”

which uses the Internet Protocol Suite. Both kinds of sockets are bi-directional: they are data flows that can be communicated in both directions simultaneously (full-duplex).

Two socket types are available:

• Stream sockets: Stream sockets provide for a data flow without record boundaries — a stream of bytes. Streams are guaranteed to be delivered and to be correctly sequenced and unduplicated.

• Datagram sockets: Datagram sockets support a record-oriented data flow that is not guaranteed to be delivered and may not be sequenced as sent or unduplicated

**Uses for Sockets**

Sockets are highly useful in at least three communications contexts:

• Client/Server models

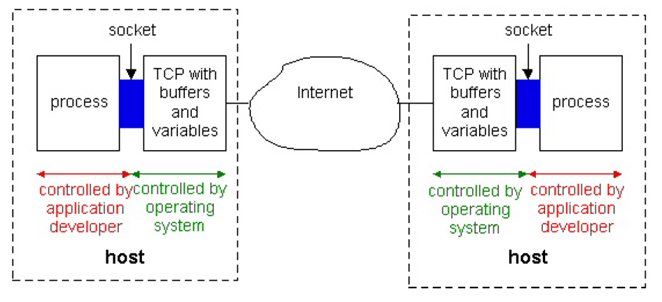
• Peer-to-peer scenarios, such as chat applications

• Making remote procedure calls (RPC) by having the receiving application interpret a message as a function call

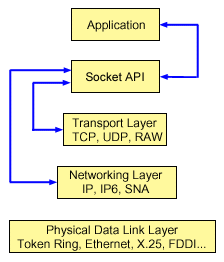
Berkeley's 4.2 BSD system10 introduced sockets (communication connection points) that exist in domains (naming spaces). The design is powerful enough to provide most of the needed facilities.

**How socket works as interface?**



****

**Interface with Communication layer**

****

**Uses of Sockets in UNIX**

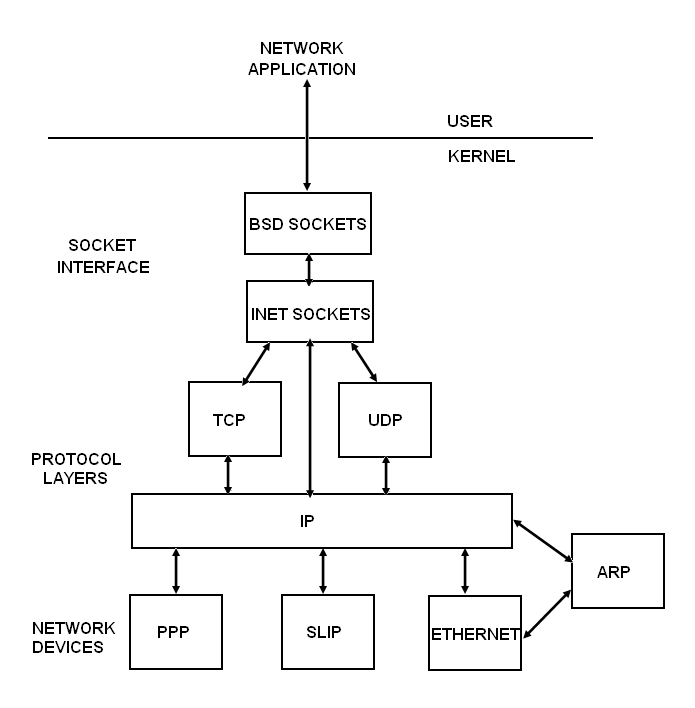
The use of a socket for communication often follows the client/server model. One method of communication between server and client processes is to design the server following these steps:

1. Create the socket.

2. Assign a name to the socket.

3. Attach a connection to the socket.

4. Transfer data via the socket.

5. Clean up the socket after use. The connection to a socket also uses a socket. The connecting socket used for a client follows similar steps, except that the assignment of a name is not always necessary.****

**Types of sockets**

* Connectionless – UDP

- Datagram Sockets (SOCK\_DGRAM)

- Rely on UDP

- Connection is unreliable

* Connection oriented – TCP

- Stream Sockets (SOCK\_STREAM)

- Connection oriented

- Rely on TCP to provide reliable two-way connected communication

**Internet Addressing Data Structures:**

struct sockaddr\_in: A parallel structure that makes it easy to reference elements of the socket address

struct sockaddr\_in {

short int sin\_family; // set to AF\_INET

unsigned short int sin\_port; // Port number

struct in\_addr sin\_addr; // Internet address

unsigned char sin\_zero[8]; //set to all zeros

}

**Socket Data Structure:**

* Family :
  + Protocol Group: IPv4, IPv6, Unix domain
  + IPv4 : IF\_INET, IPv6 : IF\_INRT6
* Type :
  + Socket Type : SOCK-STREM, SOCK\_DGRAM, SOCK\_ SEQPACKET, SOCK\_RAW (ip)
* Protocol : It Users the Interface (Set 0)
* Local Socket address : Local IP + Port no
* Remote socket address Remote IP + port No

**Socket System Calls**

1. **socket() 2. bind() 3. listen() 4. accept() 5. connect() 6. send()**
2. **recv() 8. sendto() 9. recvfrom() 10. close()**

**Connection Oriented Protocol-TCP :**

**Server: Client:**

1. Create socket 1. Create socket

2. Initialization of information 2. Initialization of server information

3. Bind it with port 3. Connect with server

4. Put socket in listen 3.1) Send data

5. Accept connections 3.2) Receive data

5.1. Send data 4. Close socket

5.2. Receive data

****

**Connection less Protocol-UDP:**

**Server: Client:**

1. Create socket Create socket

2. Initialization of information 2. Initialization of server information

3. Bind it with port Send to server

4. Receive from Receive from server

5. Send to Close socket



**Conclusion:**