MINI-PROJECT

TO IMPLEMENT E-MAIL USING SOCKET PROGRAMMING

Introduction

Sockets provide an interface for programming networks at the transport layer. Network communication using Sockets is very much like performing file I/O. In fact, socket handle is treated like file handle.

The streams used in file I/O operation are also applicable to socket-based I/O. Socket-based communication is independent of a programming language used for implementing it. That means, a socket program written in Java language can communicate to a program written in non-Java (say C or C++) socket program.

A server (program) runs on a specific computer and has a socket that is bound to a specific port. The server listens to the socket for a client to make a connection request. If everything goes well, the server accepts the connection.

Upon acceptance, the server gets a new socket bound to a different port. It needs a new socket (consequently a different port number) so that it can continue to listen to the original socket for connection requests while serving the connected client.

Types of sockets:

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

- **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 not 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** − These provide users access to the underlying communication protocols. 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 cryptic 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 a part of the Network Systems (NS) socket abstraction. 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.

Socket function

```
#include <sys/socket.h>
int socket int family, int type, int protocol);
```

The family specifies the protocol family

Family	Description
AF_INET	IPV4 protocol
AF_INET6	IPV6 protocol
AF_LOCAL	unix domain protocol
AF_ROUTE	routing sockets
AF_KEY	key socket

Type	Description
SOCK_STREAM	Stream description
SOCK_DGRAM	Datagram socket
SOCK_RAW	Raw socket

The protocol argument to the socket function is set to zero except for raw sockets.

Connect function: The connect function is used by a TCP client to establish a connection with a TCP Server.

```
int connect(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);
```

Bind function: The bind function assigns a local protocol address to a socket.

```
int bind(int sockfd, const struct sockaddr *myaddr, s ocklen_t addrlen);
```

Bzero: It sets the specified number of bytes to 0(zero) in the destination. We often use this function to initialize a socket address structure to 0(zero).

```
#include<strings.h>
void bzer(void *dest,size_t nbytes);
```

Memset: It sets the specified number of bytes to the value c in the destination

#include<string.h> void *memset(void *dest, int c, size_t len);

Close function: The normal UNIX close function is also used to close a socket and terminate a TCP connection.

#include<unistd.h>int close(int sockfd);

Return 0 if ok, -1 on error.

Listen function: The second argument to this function specifies the maximum number of connection that the kernel should queue for this socket.

int listen(int sockfd, int backlog);

Accept function: The cliaddr and addrlen argument are used to ret urn the protocol address of the connected peer processes (client)

int accept(int sockfd, struct sockaddr *cliaaddr, socklen_t addrlen);

IPv4 Socket Address Structure: An IPv4 socket address structure, commonly called an "Internet socket address structure, " is named sockaddr_in and defined by including the **<netinet/in.h>** header**.**

```
struct in_addr
{
 in_addr_t s_addr;          /* network byte ordered */
};
struct sockaddr_in
{
uint8_t  sin_len;       /* length of structure(16) */
sa_family_t  sin_family;  /* AF_INET */
in_port_t  sin_port;        /* 16-bit TCP or UDP port number*/
                          /* network byte ordered */
struct in_addr  sin_addr;     /* 32-bit IPv4 address */
                             /*newtork byte ordered */
char sin_zero[8];                  /* unused */
};
```

Address Conversion functions

```
#include<netinet/in.h>
Uint16_t  htons( uint16_t   host16bitvalue);
Uint32_t  htonl( uint32_t   host32bitvalue);
Uint16_t  ntohs( uint16_t   net16bitvalue);
Uint32_t  ntohl( uint32_t    net32bitvalue);
```