**Assignment 9 : Group B (Unit III & IV )**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **W (4)** | **C (4)** | **D (4)** | **V(4)** | **T (4)** | **Total** | **Sign** |
|  |  |  |  |  |  |  |

**Date of Performance \_\_\_\_\_\_\_\_\_\_\_\_**

**Date of Completion** :\_\_\_\_\_\_\_\_\_\_\_\_\_

**Problem Definition:**

**Problem Definition: Write a program using TCP socket for wired network for following a. Say Hello to Each other b. File transfer c. Calculator**

**1.1Prerequisite:**

a) Socket Header b) Network Programming c) Ports

**Learning Objectives**:

1. To understand Work of Socket
2. Different methods associated with Client & Server Socket

**New Concepts:**

1. Client Server Communication
2. Port Address
   1. **Theory:**
      1. **Introduction**

Theory: Socket Programming: The Berkeley socket interface, an API, allows communications between hosts or between processes on one computer, using the concept of a socket. It can work with many different I/O devices and drivers, although support for these depends on the operating system implementation. This interface implementation is implicit for TCP/IP, and it is therefore one of the fundamental technologies underlying the Internet. It was first developed at the University of California, Berkeley for use on Unix systems. All modern operating systems now have some implementation of the Berkeley socket interface, as it has become the standard interface for connecting to the Internet. Programmers can make the socket interfaces accessible at three different levels, most powerfully and fundamentally at the RAW socket level. Very few applications need the degree of control over outgoing communications that this provides, so RAW sockets support was intended to be available only on computers used for developing Internet related technologies. TCP TCP provides the concept of a connection. A process creates a TCP socket by calling the socket() function with the parameters PF\_INET or PF\_INET6 and SOCK\_STREAM. Server Setting up a simple TCP server involves the following steps: Creating a TCP socket, with a call to socket(). Computer Networks Lab 2 Binding the socket to the listen port, with a call to bind(). Before calling bind(), aprogrammer must declare a sockaddr\_in structure, clear it (with bzero() ormemset()), and the sin\_family (AF\_INET or AF\_INET6), and fill its sin\_port (the listening port, in network byte order) fields. Converting a short int to networkbyte order can be done by calling the function htons() (host to network short). Preparing the socket to listen for connections (making it a listening socket), with acall to listen(). Accepting incoming connections, via a call to accept(). This blocks until anincoming connection is received, and then returns a socket descriptor for theaccepted connection. The initial descriptor remains a listening descriptor, andaccept() can be called again at any time with this socket, until it is closed. Communicating with the remote host, which can be done through send() andrecv(). Eventually closing each socket that was opened, once it is no longer needed, using close(). Note that if there were any calls to fork(), each process must close the sockets it knew about (the kernel keeps track of how many processes have a descriptor open), and two processes should not use the same socket at once. Client: Setting up a TCP client involves the following steps: 1. Creating a TCP socket, with a call to socket(). 2. Connecting to the server with the use of connect, passing a sockaddr\_in structurewith the sin\_family set to AF\_INET or AF\_INET6, sin\_port set to the port theendpoint is listening (in network byte order), and sin\_addr set to the IPv4 or IPv6address of the listening server (also in network byte order.) 1. Communicating with the server by send()ing and recv()ing.Terminating the connection and cleaning up with a call to close(). Again, if therewere any calls to fork(), each process must close() the socket. Functions: 1. socket(): socket() creates an endpoint for communication and returns a descriptor. socket() takes three arguments: domain, which specifies the protocol family of the created socket. For example: PF\_INET for network protocol IPv4 or PF\_INET6 for IPv6). type, one of: Computer Networks Lab 3 SOCK\_STREAM (reliable stream-oriented service) SOCK\_DGRAM (datagram service) SOCK\_SEQPACKET (reliable sequenced packet service), or SOCK\_RAW (raw protocols atop the network layer).protocol usually set to 0 to represent the default transport protocol for the specified domain and type values (TCP for PF\_INET or PF\_INET6 andSOCK\_STREAM, UDP for those PF\_ values and SOCK\_DGRAM), but whichcan also explicitly specify a protocol. The function returns -1 if an error occurred. Otherwise, it returns an integer representing the newly-assigned descriptor. Prototype: int socket(int domain, int type, int protocol); connect(): connect() returns an integer representing the error code: 0 represents success, while -1 represents an error. Certain types of sockets are connectionless, most commonly user datagram protocol sockets. For these sockets, connect takes on a special meaning: the default target for sending and receiving data gets set to the given address, allowing the use of functions such as send() and recv() on connectionless sockets. Prototype: int connect(intsockfd, conststructsockaddr \*serv\_addr, socklen\_taddrlen); bind(): bind() assigns a socket an address. When a socket is created using socket(), it is given an address family, but not assigned an address. Before a socket may accept incoming connections, it must be bound. bind() takes three arguments: sockfd, a descriptor representing the socket to perform the bind onmy\_addr, a pointer to a sockaddr structure representing the address to bind to. addrlen, a socklen\_t field representing the length of the sockaddr structure. It returns 0 on success and -1 if an error occurs. Prototype: int bind(intsockfd, structsockaddr \*my\_addr, socklen\_taddrlen); listen() listen() prepares a bound socket to accept incoming connections. This function is only applicable to the SOCK\_STREAM and SOCK\_SEQPACKET socket types. It takes two arguments: sockfd, a valid socket descriptor. Computer Networks Lab 4 backlog, an integer representing the number of pending connections that can be queued up at any one time. The operating system usually places a cap on this value. Once a connection is accepted, it is dequeued. On success, 0 is returned. If an error occurs, -1 is returned. Prototype: int listen(intsockfd, int backlog); accept() Programmers use accept() to accept a connection request from a remote host. It takes the following arguments: sockfd, the descriptor of the listening socket to accept the connection from. cliaddr, a pointer to the sockaddr structure that accept() should put the client'saddress information into. addrlen, a pointer to the socklen\_t integer that will indicate to accept() how largethe sockaddr structure pointed to by cliaddr is. When accept() returns, the ocklen\_t integer then indicates how many bytes of the cliaddr structure wereactually used. The function returns a socket corresponding to the accepted connection, or -1 if an error occurs. Prototype: int accept(intsockfd, structsockaddr \*cliaddr, socklen\_t \*addrlen); Blocking vs. nonblocking Berkeley sockets can operate in one of two modes: blocking or non-blocking. A blocking socket will not "return" until it has sent (or received) all the data specified for the operation. This may cause problems if a socket continues to listen: a program may hang as the socket waits for data that may never arrive. A socket is typically set to blocking or nonblocking mode using the fcntl() or ioctl() functions. Cleaning up The system will not release the resources allocated by the socket() call until a close() call occurs. This is especially important if the connect() call fails and may be retried. Each call to socket() must have a matching call to close() in all possible execution paths.

Algorithm: Server Program

1. Open the Server Socket: ServerSocket server = new ServerSocket( PORT );

2. Wait for the Client Request: Socket client = server.accept();

3. Create I/O streams for communicating to the client DataInputStream is = new DataInputStream(client.getInputStream()); DataOutputStreamos = new DataOutputStream(client.getOutputStream());

4. Perform communication with client Receive from client: String line = is.readLine(); Send to client: os.writeBytes(“Hello\n”)

5. Close socket: client.close(); Client Program

1) Create a Socket Object: Socket client = new Socket(server, port\_id);

2) Create I/O streams for communicating with the server. is = new DataInputStream(client.getInputStream()); os = new DataOutputStream(client.getOutputStream());

3) Perform I/O or communication with the server: Receive data from the server: String line = is.readLine(); Send data to the server: os.writeBytes(“Hello\n”);

4) Close the socket when done:

5) client.close();

**TYPES OF SOCKETS**

## Socket Types

There are four types of sockets available to the users. The first two are most commonly used and the 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 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, 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 cryptic facilities of an existing protocol.

Here is the description of the parameters −

* **socket\_family −** This is either AF\_UNIX or AF\_INET, as explained earlier.
* **socket\_type −** This is either SOCK\_STREAM or SOCK\_DGRAM.
* **protocol −** This is usually left out, defaulting to 0.

Once you have socket object, then you can use required functions to create your client or server program. Following is the list of functions required –

## SERVER SOCKET METHODS

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **s.bind()**This method binds address (hostname, port number pair) to socket. |
| 2 | **s.listen()**This method sets up and start TCP listener. |
| 3 | **s.accept()**This passively accept TCP client connection, waiting until connection arrives (blocking). |

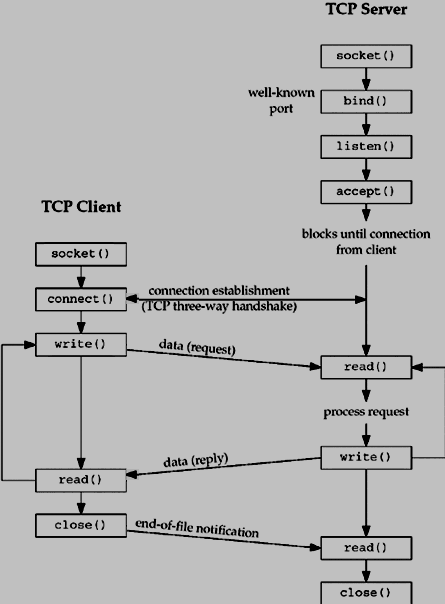
## CLIENT SOCKET METHODS

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **s.connect()**This method actively initiates TCP server connection. |

## GENERAL SOCKET METHODS

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **s.recv()**This method receives TCP message |
| 2 | **s.send()**This method transmits TCP message |
| 3 | **s.recvfrom()**This method receives UDP message |
| 4 | **s.sendto()**This method transmits UDP message |
| 5 | **s.close()**This method closes socket |
| 6 | **socket.gethostname()**Returns the hostname. |

**Methods Associated with Socket**:The following diagram shows the complete Client and Server interaction −



**Questions:**

1. What is a socket? Explain different types of Sockets

2. What is the difference between a connection-less and a connection-oriented communication system? How are they implemented?

3. Why is the port number required?

4. How is the socket programming in linux different from that in windows?

5.What are the Methods Associate with Server Socket ?

6. What are the methods associated with Client Socket ?

CONCLUSION Thus we have successfully implemented the socket programming for TCP