**Assignment No**

**TITLE**: FTP using socket programming for TCP

**AIM:** Write FTP program using socket programming for TCP using C++

**OBJECTIVE:**

1. To study and implement FTP Protocol
2. To understand file Exchange between FTP client and server

**THEORY:**

FTP (File Transfer Protocol) is a protocol for transferring a file from one host to another host. The protocol dates back to 1971 (when the Internet was still an experiment), but remains enormously popular. The primary general file transfer protocol in the TCP/IP suite shows its “generality” directly through its unqualified name: the File Transfer Protocol (FTP). FTP is one of the most widely used application protocols in the world. It was designed to allow the efficient transfer of files between any two devices on a TCP/IP internetwork. It automatically takes care of the details of how files are moved, provides rich command syntax to allow various supporting file operations to be performed (such as navigating the directory structure and deleting files) and operates using the TCP transport service for reliability.

**FTP is used for:**

1. Uploading web pages to web servers for publishing on the Internet
2. Browsing and downloading files from public software sites
3. Transferring large files among two parties that are too large for email attachments
4. Downloading and uploading content like university’s assignments via an FTP server
5. Distributing the latest revisions of programs by software developers.

**Essential FTP Terms**

Below describes some of the essential FTP terms.

**Anonymous FTP:** Various public servers allow anonymous login. Users can log in to servers without an account to download files. Uploading is not allowed for anonymous login. Take note that your IP address is tracked even though it is an anonymous session.

**Get:** Also called “Download”. Copy files from the FTP site to the FTP client’s system.

**Put:** Also called “Upload”. Copy files from the client’s system to the FTP site. Uploading is restricted to authorized users only.

**FTP Site:** A hosting server that contains files for download and upload. To access the FTP site, you need to type in the address, which begins with ftp:// (instead of http://).

**FTP Control Connection and Data Connection**

A critical concept in understanding FTP is that while it uses TCP like many other applications, it does not use just one TCP connection for all communication the way most protocols do. The FTP model is designed around two logical channels of communication between the server and user FTP processes:

**Control Connection:** This is the main logical TCP connection that is created when an FTP session is established. It is maintained throughout the FTP session and is used only for passing control information, such as FTP commands and replies. It is not used to send files.

**Data Connection:** Each time that data is sent from the server to the client or vice-versa, a distinct TCP data connection is established between them. Data is transferred over this connection. When the file transfer is complete, the connection is terminated.

The reason for having these separate channels is that it provides flexibility in how the protocol is used, as we will see later in this section. It does, however, add complexity to FTP.

**FTP Command Groups**

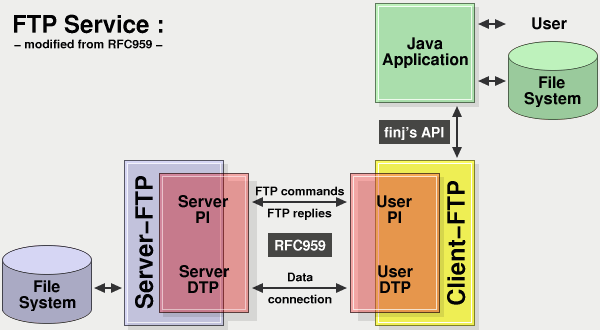
Each command is identified by a short three-letter or four-letter command code for convenience, and performs a specific task in the overall functionality of FTP. There are several dozen of these protocol commands, and to help organize them, the FTP standard categorizes them into three groups, based on overall function type:

**Access Control Commands:** Commands that are part of the user login and authentication process, are used for resource access, or are part of general session control.

**Transfer Parameter Commands:** Commands that specify parameters for how data transfers should occur. For example, commands in this group specify the data type of a file to be sent, indicate whether passive or active data connections will be used, and so forth.

**FTP Service Commands:** This is the largest group, containing all the commands that actually perform file operations, such as sending and receiving files. Commands to implement support functions, such as deleting or renaming files, are also here.

**FTP Block Diagram:**



**Client / Server Communication:**

Internet and WWW have emerged as global ubiquitous media for communication and changed the way we conduct science, engineering, and commerce. The key element for developing Internet enabled applications are a good understanding of the issues involved in implementing distributed applications and sound knowledge of the fundamental network programming models.

A computer running a program that makes request for service is called client machine. A computer running a program that offers requested services from one or more clients is called server machine. The media for communication can be wire or wireless network. All this involve networking services provided by the transport TCP/IP stack.

The transport layer comprises two types of protocols:

* TCP (Transmission Control Protocol)
* UDP (User Datagram Protocol)

The most widely used programming interfaces for these protocols are sockets.

TCP is a connection oriented protocol that provides a reliable flow of data between two computers. Ex. HTTP, FTP, Telnet.

UDP is a protocol that sends independent packets of data, called datagrams with no guarantee of delivery at destination and may be out of order. Ex. applications like Clock server and Ping.

Socket based communication is independent of a programming language used for implementing it. Sockets are a protocol independent method of creating a connection between processes. Socket is the channel through which application can connect & communicate with each other. Two processes can communicate with each other using sockets. It returns the socket descriptor, and user communicate through it using the specialized send() and recv() socket calls.

Socket is described by 5 things:

* Family:IPv4, IPv6 or Unix Domain Sockets
* Type: Stream or datagram or raw sockets
* Protocol: 0 for TCP and UDP
* Local Socket Address: IP + Port Address
* Remote Socket Address: IP + Port Address

**Types of Sockets:**

* Stream Sockets (SOCK\_STREAM) : Connection Oriented Sockets, Reliable, Example: TELNET, HTTP and Uses TCP
* Datagram Sockets (SOCK\_DGRAM) : Connectionless Sockets, Non Reliable , Example: TFTP, BOOTP and Uses UDP
* Raw Sockets: Example ICMP or OSPF

**Socket structure:** Basically two structures are used in sockets.

structsockaddr{

unsigned short sa\_family;

char sa\_data[14];

}

structsockaddr\_in {

shortintsin\_family;

unsigned short intsin\_port;

structin\_addrsin\_addr;

unsigned char sin\_zero[8];

};

structin\_addr{unsigned long s\_addr; };

**Socket Methods**

**1) Socket**

# include <sys/types.h>

# include <sys/socket.h>

**int socket (int domain, int type, int protocol);**

**domain**should be set to PF\_INET/AF\_INET. Other domains are AF\_INET6, AF\_UNIX, AF\_TELEPHONY etc. **type**tells the kernel what kind of socket this is: SOCK\_STREAM or SOCK\_DGRAM. **protocol**to be set to “0” as type itself defines the type of socket. socket() simply returns a socket descriptor that can be used in later system calls, or -1 on error.

AF\_INET **A**ddress **F**amily sockets can be either connection-oriented or connectionless AF\_INET sockets can also use a type of SOCK\_RAW. If this type is set, the application connects directly to the IP layer and does not use either the TCP or UDP transports.AF\_INET6 sockets provide support for IP6.

**2) bind**

**int bind(intsockfd, structsockaddr \*my\_addr, intaddrlen);**

sockfdis the socket file descriptor returned by socket(). my\_addris a pointer to a structsockaddr. addrlencan be set to sizeof (structsockaddr).

**3) listen**

**int listen(intsockfd, int backlog);**

backlogis the number of connections allowed on the incoming queue. All Incoming connections are going to wait in this queue until you accept() them and this is the limit on how many can queue up. Most systems limit this number to about 20.

**4) connect**

**int connect(intsockfd, structsockaddr \*serv\_addr, intaddrlen);**

serv\_addris a structsockaddrcontaining the destination port and IP address. addrlencan be set to sizeof(structsockaddr).

**5) accept**

**int accept(intsockfd, structsockaddr \*addr, socklen\_t \*addrlen);**

It’ll return to you a new socket file descriptor to use for thissingle connection. You have two socket file descriptors for the price of one! The original one is still listening on your port and the newly created one is finally ready to send() and recv().

**6) send**

**int send (intsockfd, const void \*msg, intlen, int flags);**

msg is a pointer to the data you want to send. lenis the length of that data in bytes. Just set flags to 0.

char \*msg = “Hello AIT";

intlen, bytes\_sent;

len = strlen(msg);

bytes\_sent = send(sockfd, msg, len, 0);

send() returns the number of bytes actually sent out – this might be less than the number.

**7) recv**

**intrecv(intsockfd, void \*buf, intlen, unsigned int flags);**

bufis the buffer to read the information into. len is the maximum length of the buffer. flags can again be set to 0. recv() returns the number of bytes actually read into the buffer, or -1 on error. recv() can return 0. This can mean - the remote side has closed the connection on you.

**8) close**

**close(intsockfd)**

This will prevent any more reads and writes to the socket. Anyone attempting to read or write the socket on the remote end will receive an error.

**9) shutdown**

**int shutdown(intsockfd, int how);**

It allows you to cut off communication in a certain direction, or both ways. how is one of the following: 0– receives are disallowed, 1 – sends are disallowed and 2 – sends and receives are disallowed. It’s important to note that shutdown() doesn’t actually close the file descriptor – it just changes Its usability. To free a socket descriptor, you need to use close().

**API for managing names and IP**

* hostent structure: describes IP, hostname pairs
* gethostbyname:hostent of a specified machine
* htons, htonl, ntohs, ntohl: byte ordering
* inet\_pton, inet\_ntop: conversion of IP numbers between presentation and strings

**hostenet structure**

#include <netdb.h>

structhostent \*gethostbyname(char \*name);

structhostent {

char \*h\_name;

char \*\*h\_aliases;

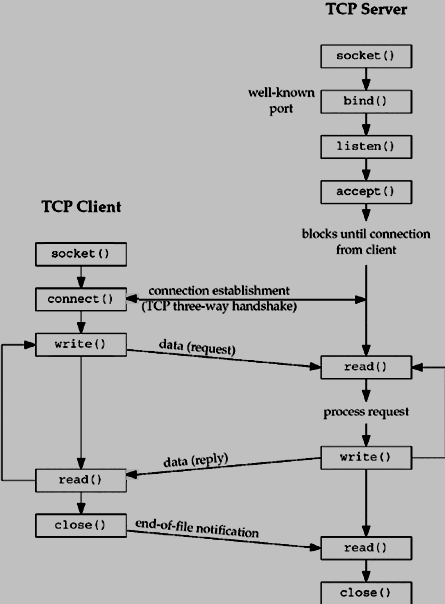
inth\_addrtype;

inth\_length;

char \*\*h\_addr\_list;

};

**TCP Sockets flow:**



**TCP Socket Flow Description on Server :**

1. The socket() function returns a socket descriptor representing an endpoint. The statement also identifies that the INET (Internet Protocol) address family with the TCP transport (SOCK\_STREAM) will be used for this socket.
2. After the socket descriptor is created, the bind() function gets a unique name for the socket.
3. The listen() allows the server to accept incoming client connections. In this example, the backlog is set to 10. This means that the system will queue 10 incoming connection requests before the system starts rejecting the incoming requests.
4. The server uses the accept() function to accept an incoming connection request. The accept() call will block indefinitely waiting for the incoming connection to arrive.
5. The select() function allows the process to wait for an event to occur and to wake up the process when the event occurs.
6. The recv() function receives data from the client application.
7. The send() function echoes the data back to the client.
8. The close() function closes any open socket descriptors.

**TCP Socket Flow Description on Client:**

1. The socket() function returns a socket descriptor representing an endpoint.
2. In the client example program, if the server string that was passed into the inet\_addr() function was not a dotted decimal IP address, then it is assumed to be the hostname of the server. In that case, use the gethostbyname() function to retrieve the IP address of the server.
3. After the socket descriptor is received, the connect() function is used to establish a connection to the server.
4. The send() function sends bytes of data to the server.
5. The recv() function waits for the server to echo the bytes of data back.
6. The close() function closes any open socket descriptors.

**CONCLUSION:** Thus, we studied and implement the FTP using socket programming.

**FAQ:**

1. Differentiate between FTP and HTTP.
2. Enlist and describe FTP Commands?
3. What are the Difference between FTP and TFTP?
4. What is socket inheritance? Explain.