# IMPLEMENT - FILE TRANSFER PROTOCOL USING TCP SOCKETS

**AIM:**

To write a java program to implement the file transfer protocol.

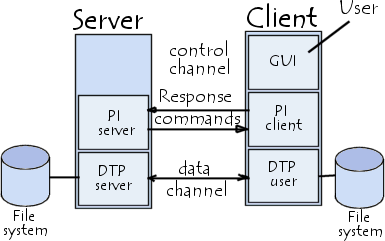
# THEORY:

FTP protocol (*File Transfer Protocol*) is, as its name indicates a [protocol](http://en.kioskea.net/contents/protocol.php3) for transferring files. FTP protocol defines the way in which data must be transferred over a [TCP/IP](http://en.kioskea.net/contents/tcpip.php3) network. The aim of FTP protocol is to:

* allow file sharing between remote machines
* allow independence between client and server machine system files
* enable efficient data transfer

FTP protocol falls within a client-server model, i.e. one machine sends orders (the client) and the other awaits requests to carry out actions (the server). During an FTP connection, two transmission channels are open:

* A channel for commands (control channel)
* A channel for data



So, both the client and server have two processes allowing these two types of information to be managed:

* **DTP** (*Data Transfer Process*) is the process in charge of establishing the connection and managing the data channel. The server side DTP is called *SERVER-DTP*, the client side DTP is called *USER-DTP*
* **PI** (*Protocol Interpreter*) interprets the protocol allowing the DTP to be controlled using commands received over the control channel. It is different on the client and the server:
  + The SERVER-PI is responsible for listening to the commands coming from a USER-PI over the control channel on a [data port](http://en.kioskea.net/contents/port.php3), establishing the connection for the control channel, receiving FTP commands from the USER-PI over this, responding to them and running the SERVER-DTP.
  + The USER-PI is responsible for establishing the connection with the FTP server, sending FTP commands, receiving responses from the SERVER-PI and controlling the USER-DTP if needed.

When an FTP client is connected to a FTP server, the USER-PI initiates the connection to the server according to the Telnet protocol. The client sends FTP commands to the server, the server interprets them, runs its DTP, then sends a standard response. Once the connection is established, the server-PI gives the port on which data will be sent to the Client DTP. The client DTP then listens on the specified port for data coming from the server. It is important to note that since the control and data ports are separate channels, it is possible to send commands from one machine and receive data on another. So, for example it is possible to transfer data between FTP servers by passing through a client to send control instructions and by transferring information between two server processes connected on the right port. In this configuration, the protocol imposes that the control channels remain open throughout the data transfer. So a server can stop a transmission if the control channel is broken during transmission.

# ALGORITHM :

CLIENT:

1. Start the program.
2. Create the client packet.
3. After transferring the packet statementis displayed.
4. Stop the program.

SERVER:

1. Start the program.
2. Create the server socket.
3. Call the I/O stream.
4. Print the file has been sent.
5. Send the intimation to the client.
6. Stop the program

# PROGRAM:

SERVER :

import java.net.\*; import java.io.\*; import java.util.\*; class FileTransfer

{

Socket soc; ServerSocket ss; DataInputStream din; DataOutputStream dout; String fname;

File recfile; String filecon;

FileTransfer() throws Exception

{

ss=new ServerSocket(2788); soc=ss.accept();

din=new DataInputStream(soc.getInputStream()); dout=new DataOutputStream(soc.getOutputStream());

}

void sendfile() throws Exception

{

Scanner in=new Scanner(System.in);

//dout.writeUTF("R");

//System.out.println("File Name :"); fname=din.readUTF(); System.out.println("File name :"+fname); recfile=new File(fname); if(recfile.exists())

{

dout.writeUTF("Y"); if((din.readUTF()).equalsIgnoreCase("Y")) startsend();

else

{

}

}

else

{

System.out.println(" Not Found"); dout.writeUTF("N");

}

}

void startsend() throws Exception

{

String filecon;

FileInputStream fos=new FileInputStream(recfile); int ch;

System.out.println("Transferring File ");

while(true)

{

ch=fos.read();

//ch=Integer.parseInt(filecon); if(ch!=-1)

{

dout.writeUTF(String.valueOf(ch));

}

else break;

}

System.out.println("File Transferring Completed .");

}

}

public class FTPServer

{

public static void main(String[] args) throws Exception

{

FileTransfer fs=new FileTransfer(); fs.sendfile(); System.out.println("Disconnected .");

}

} CLIENT:

import java.net.\*; import java.io.\*; import java.util.\*; class FileTransfer

{

Socket soc;

DataInputStream din; DataOutputStream dout; String fname;

File recfile;

String filecon;

FileTransfer() throws Exception

{

soc=new Socket("127.0.0.1",2788);

din=new DataInputStream(soc.getInputStream()); dout=new DataOutputStream(soc.getOutputStream());

}

void rec\_file() throws Exception

{

Scanner in=new Scanner(System.in);

//dout.writeUTF("R"); System.out.println("File Name :"); fname=in.nextLine(); dout.writeUTF(fname);

String server\_msg=din.readUTF(); if(server\_msg.equalsIgnoreCase("Y"))

{

recfile=new File(fname); if(recfile.exists())

{

System.out.println("File Already exists.Want to replace?(Y/N)"); if((in.nextLine()).equalsIgnoreCase("Y"))

{

dout.writeUTF("Y"); startrec();

}

else

{

System.out.println("Want to give a new File name or exit ?(Y/N)"); if((in.nextLine()).equalsIgnoreCase("Y"))

{

System.out.println("Enter :"); String newname=in.nextLine(); recfile=new File(newname); dout.writeUTF("Y");

startrec();

}

else

{

dout.writeUTF("N");

}

}

}

else

{

System.out.println(" Transferring File..."); startrec();

}

}

else

{

System.out.println("File not Found in Server");

}

}

void startrec()

{

try

{

String filecon;

FileOutputStream fos=new FileOutputStream(recfile); int ch;

System.out.println("Transferring File ");

while(true)

{

filecon=din.readUTF(); ch=Integer.parseInt(filecon); if(ch!=-1)

{

fos.write(ch);

}

else break;

}

System.out.println("File is Received .");

}catch(Exception x)

{

}

}

}

public class FTPClient

{

public static void main(String[] args) throws Exception

{

FileTransfer fs=new FileTransfer(); fs.rec\_file();

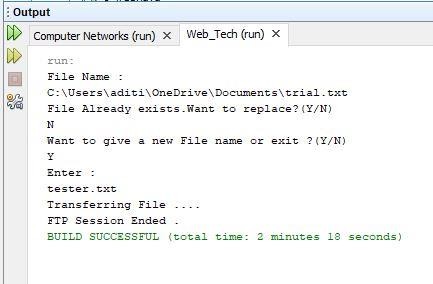
System.out.println("FTP Session Ended .");

}

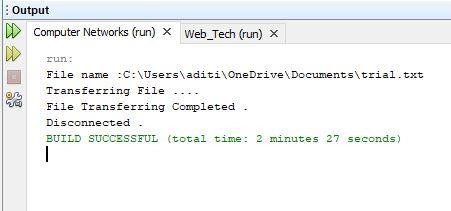
}

# OUTPUT:

FTP CLIENT :



FTP SERVER :



# RESULT:

Thus the implementation of FTP client and server has been executed and output has been verified successfully.

# 2) FILE TRANSFER PROTOCOL

**AIM :**

To write a java program to implement the file transfer protocol.

# PROGRAM :

**FileTransferServer.java**

import java.io.BufferedInputStream; import java.io.File;

import java.io.FileInputStream; import java.io.OutputStream; import java.net.InetAddress; import java.net.ServerSocket; import java.net.Socket;

public class FileTransferServer {

public static void main(String[] args) throws Exception { ServerSocket ssock = new ServerSocket(5000);

Socket socket = ssock.accept();

InetAddress IA = InetAddress.getByName("localhost"); File file = new File("F:\\some.java");

FileInputStream fis = new FileInputStream(file); BufferedInputStream bis = new BufferedInputStream(fis); OutputStream os = socket.getOutputStream();

byte[] contents;

long fileLength = file.length(); long current = 0;

long start = System.nanoTime(); while(current!=fileLength){

int size = 10000;

if(fileLength - current >= size) current += size;

else{

size = (int)(fileLength - current); current = fileLength;

}

contents = new byte[size]; bis.read(contents, 0, size); os.write(contents);

System.out.print("Sending file ... "+(current\*100)/fileLength+"% complete!");} os.flush();

socket.close(); ssock.close();

System.out.println("File sent succesfully!");

}}

# FileTransferClient.java

import java.io.BufferedOutputStream; import java.io.FileOutputStream;

import java.io.InputStream; import java.net.InetAddress; import java.net.Socket;

public class FileTransferClient {

public static void main(String[] args) throws Exception{

Socket socket = new Socket(InetAddress.getByName("localhost"), 5000); byte[] contents = new byte[10000];

FileOutputStream fos = new FileOutputStream("d:\\data.java"); BufferedOutputStream bos = new BufferedOutputStream(fos); InputStream is = socket.getInputStream();

int bytesRead = 0; while((bytesRead=is.read(contents))!=-1) bos.write(contents, 0, bytesRead); bos.flush();

socket.close();

System.out.println("File saved successfully!");

}}

# OUTPUT :

SERVER :



CLIENT :



.