**Ex No: 7**

**Date:**

**TCP/IP SOCKET PROGRAMMING IN CLIENT SERVER MODEL FOR FILE TRANSFER APPLICATION**

**Aim:**

To write a Java program to implement the client server model for file transfer application using TCP/IP protocol

**Theory:**

The **File Transfer Protocol (FTP)** is a standard network protocol used for transferring files between a client and a server over a TCP/IP network.

**Key Concepts:**

1. **Client-Server Architecture**:
   * **Client**: Requests files from the server.
   * **Server**: Listens for client requests and sends the requested files.
2. **Process Flow**:
   * **Connection Establishment**: The client establishes a connection to the server using a specific port (usually port 21 for FTP).
   * **Command-Response Protocol**: The client sends commands (such as requests for file transfer), and the server responds with either the requested file or a status message (e.g., file not found, transfer successful).
   * **File Transfer**: Data is transferred from server to client (or vice versa) in binary or text mode. In FTP, this usually happens on a separate data connection.
   * **Connection Termination**: After the transfer, both the client and server close the connection.
3. **Two Modes of FTP**:
   * **Active Mode**: The server actively opens a connection to the client for the data transfer.
   * **Passive Mode**: The client initiates both control and data connections to the server.
4. **Reliability**: FTP ensures that files are transferred reliably using TCP, which handles packet delivery, error correction, and reordering.

**Algorithm:**

# Server

Step 1: **Start the server** on port 8081.

Step 2: **Accept client connection**.

Step 3: **Read the requested file name** from the client.

Step 4: **Check if the file exists**:

* If the file exists:
  + Open the file and **send its contents** to the client using a buffer.
  + **Flush and close** the output stream.
  + **Notify** the server-side that the file was transferred successfully.
* If the file does not exist:
  + **Send an error message** ("ERROR: File not found") to the client.

Step 5: **Close** the client socket and server socket.

# Client

Step 1: **Connect to the server** on port 8081.

Step 2: **Prompt the user** for the file name.

Step 3: **Send the file name** to the server.

Step 4: **Read the server response**:

* If the response starts with "ERROR":
  + **Display the error message** to the user.
* If no error:
  + **Receive the file data** from the server.
  + **Write the received data** into a new file on the client system.
  + **Notify** the client-side that the file was received successfully.

Step 5: **Close** the socket and file streams.

# Run the server code first, then client code.

Make sure the requested filename is present in the directory where the server code runs.

**Program:**

**Server:**

import java.io.\*;  
import java.net.\*;  
  
public class FTSERVER {  
 public static void main(String args[]) throws IOException {  
 ServerSocket ss = new ServerSocket(8081);  
 System.*out*.println("Server started, waiting for connection...");  
  
 Socket cs = ss.accept(); // Wait for client connection  
 System.*out*.println("Client connected.");  
  
 BufferedReader st = new BufferedReader(new InputStreamReader(cs.getInputStream()));  
 String requestedFile = st.readLine();  
 System.*out*.println("The requested file is: " + requestedFile);

File file = new File(requestedFile);  
 PrintWriter put = new PrintWriter(cs.getOutputStream(), true); // Send messages to the client  
  
 if (file.exists()) {  
 // Send file content using a BufferedOutputStream  
 BufferedInputStream fileReader = new BufferedInputStream(new FileInputStream(file));  
 BufferedOutputStream outputStream = new BufferedOutputStream(cs.getOutputStream());  
  
 byte[] buffer = new byte[4096]; // Buffer for reading file content  
 int bytesRead;  
 while ((bytesRead = fileReader.read(buffer)) != -1) {  
 outputStream.write(buffer, 0, bytesRead);  
 }  
  
 fileReader.close();  
 outputStream.flush(); // Ensure all data is sent  
 outputStream.close();  
 System.*out*.println("File transferred successfully.");  
 } else {  
 // Send an error message to the client  
 put.println("ERROR: File not found");  
 System.*out*.println("File does not exist.");  
 }  
  
 put.close();  
 cs.close(); // Close client socket  
 ss.close(); // Close server socket  
 }  
}

# Client:

import java.io.\*;  
import java.net.\*;  
  
public class FTCLIENT {  
 public static void main(String args[]) throws IOException {  
 Socket s = new Socket("localhost", 8081);  
  
 PrintWriter put = new PrintWriter(s.getOutputStream(), true);  
 BufferedReader serverResponse = new BufferedReader(new InputStreamReader(s.getInputStream()));  
 BufferedReader userInputReader = new BufferedReader(new InputStreamReader(System.*in*));  
  
 System.*out*.println("Enter the file name to transfer from the server:");  
 String fileName = userInputReader.readLine(); // Get file name from user  
  
 put.println(fileName); // Send file name to server  
  
 // Check the first line of the server response  
 String response = serverResponse.readLine();  
 if (response != null && response.startsWith("ERROR")) {  
 System.*out*.println("Server error: " + response);  
 } else {  
 // Prepare to receive the file content if no error message is received  
 File receivedFile = new File("received\_" + fileName);  
 FileOutputStream fileOut = new FileOutputStream(receivedFile);  
 BufferedInputStream fileReader = new BufferedInputStream(s.getInputStream());  
  
 byte[] buffer = new byte[4096];  
 int bytesRead;  
 while ((bytesRead = fileReader.read(buffer)) != -1) {  
 fileOut.write(buffer, 0, bytesRead);  
 }  
  
 fileOut.close(); // Close file output  
 fileReader.close(); // Close input stream  
  
 System.*out*.println("File received successfully.");  
 }  
  
 s.close(); // Close socket  
 }  
}

# SCREENSHOTS OF OUTPUT:

# *SERVER CODE DIRECTORY:*

# 

# When requested file is available:

# SERVER:

# 

# CLIENT:

# 

# When requested file is not available:

# SERVER:

# 

# CLIENT:

# 

**Result:**

Thus the echo client server model for file transfer application using TCP/IP protocol, was successfully implemented in Java coding and output obtained.