3. Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present. Implement the above program using as message queues or FIFOs as IPC channels

```
//TCP SERVER
import java.io.*;
import java.net.*;
import java.nio.charset.StandardCharsets;

public class FileTransferServer {
    // Corresponds to #define SERVER_PORT
1238
    private static final int SERVER_PORT =
1238;
    // Corresponds to the buffer size char
buf[2000] and the read size.
    private static final int BUFFER_SIZE =
2000;

    public static void main(String[] args) {
        // ServerSocket handles socket(),
bind(), listen()
        try (ServerSocket serverSocket = new
ServerSocket(SERVER_PORT)) {
```

```
            System.out.println("TCP Server
started and listening on port " +
SERVER_PORT + "...");

            // Corresponds to the while(1)
loop for continuous client handling
            while (true) {
                // accept() blocks until a
client connects
                Socket clientSocket =
serverSocket.accept();
                System.out.println("\nClient
connected from: " +
clientSocket.getInetAddress().getHostAddress
());

                // Handle the client
connection in a separate method
                handleClient(clientSocket);
            }
        } catch (IOException e) {
            // Catches errors like BIND
FAILED or listen failed
            System.err.println("Could not
start server: " + e.getMessage());
```

```
        }
    }

    private static void handleClient(Socket
clientSocket) {
        // Use try-with-resources to ensure
streams and socket are closed (like
close(sa))
        try (
            // OutputStream for sending raw
file data (bytes)
            OutputStream socketOut =
clientSocket.getOutputStream();
            // BufferedReader to read the
client's initial message (the filename) as a
line of text
            BufferedReader socketIn = new
BufferedReader(
                    new
InputStreamReader(clientSocket.getInputStrea
m(), StandardCharsets.UTF_8));
        ) {
            // Corresponds to read(sa, buf,
2000) to get the filename
```

```
            String fileName =
socketIn.readLine();
            System.out.println("Requested
file: " + fileName);

            if (fileName == null) {
                System.out.println("Client
disconnected before sending filename.");
                return;
            }

            File file = new
File(fileName.trim()); // The trim() handles
potential extra newline/padding from C
client's write

            // Corresponds to fd=open(buf,
O_RDONLY) and checking if(fd<0)
            if (!file.exists() ||
file.isDirectory()) {
                // Corresponds to 'file open
failed' and sending the error message
                String errorMessage = "file
Does not exits";
```

```
System.out.println(errorMessage);
                // Send the error message
back to the client

socketOut.write(errorMessage.getBytes(Standa
rdCharsets.UTF_8));
            } else {
                // File found, now read and
send its contents
                System.out.println("File
found. Sending contents...");

                // FileInputStream
corresponds to the opened file descriptor
(fd)
                try (FileInputStream fileIn
= new FileInputStream(file)) {
                    byte[] buffer = new
byte[BUFFER_SIZE];
                    int bytesRead;

                    // Corresponds to the
while(1) loop with read(fd, buf, 2000)
```

```
                    while ((bytesRead =
fileIn.read(buffer)) != -1) {
                        // Corresponds to
write(sa, buf, bytes)

socketOut.write(buffer, 0, bytesRead);
                    }
                    // Flushes any buffered
output to ensure all data is sent
                    socketOut.flush();
                } catch (IOException e) {

System.err.println("Error reading or sending
file: " + e.getMessage());
                }
            }

        } catch (IOException e) {
            System.err.println("Error
processing client request: " +
e.getMessage());
        } finally {
            // The try-with-resources
statement automatically closes clientSocket
```

```
            System.out.println("Client
connection closed.");
        }
    }
}
```