

PROPUESTA DE ENTREGA DE PRÁCTICO, EN VEZ DE IR AL EXAMEN DE RECUPERACIÓN

YA QUE TE QUEDA SÓLO UNA PARTE DE LA ASIGNATURA, TE PLANTEO UN PRÁCTICO A ENTREGAR HASTA EL DÍA 08/03, EN VEZ DEL EXÁMEN PRESENCIAL: (LO TRABAJARÍAS EN CASA)

"IMPLEMENTAR LA VERSIÓN COMPLETA DEL SERVIDOR TFTP QUE OS PLANTEÉ AL FINAL DE LA PRESENTACIÓN DEL TEMA3 (VERSIÓN UDP). CON LA OPCIÓN DE LOGUEO CON UN USUARIO O ANONIMOUS". ESTÁ EXPUESTO POR PARTES AMPLIADAS.

- A PARTE DE QUE DEBE FUNCIONAR, EN LA PARTE DE PROCESOS QUE ES LA QUE TE QUEDA, CUANDO EL USUARIO LOGUEADO EJECUTE EL COMANDO `list`. EN EL SERVIDOR DEBERÁS LANZAR UN PROCESO QUE EJECUTE UN COMANDO DEL TIPO `ls -l /raiz/carpetas_usuario`. DEBERÁ RECOGER LOS DATOS DEL LISTADO Y DEVOLVERSELOS AL CLIENTE.

PIDO:

- CLIENTE/SERVIDOR MULTITHREAD CON DATAGRAMA (UDP).
.- PDF CON LA EJECUCIÓN DE LA APLICACIÓN UTILIZANDO DIFERENTES VENTANAS DE CLIENTES. (DEBERÁS PROBAR TODOS LOS COMANDOS `list`, `get`, `remove`, `put`...

Clase Servidor:

```

/**
 * Clase para el servidor, donde gestionamos los comandos.
 * @author Andrea Castilla Cocera
 * acascoc098@g.educaand.es
 */
public class Servidor {
    private static int serverPort = 3000;
    private static int nextPort = 5000;
    private static ConcurrentHashMap<Integer, Integer> clientPorts = new ConcurrentHashMap<>();
    private static String rootDirectory = "/Propuesta_Recuperacion";//Pongo este directorio como
ejemplo

    public static void main(String[] args) throws IOException {
        try (DatagramSocket serverSocket = new DatagramSocket(serverPort)) {
            byte[] buffer = new byte[500];

            while (true) {
                DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
                serverSocket.receive(packet);

                String command = new String(packet.getData(), 0, packet.getLength());
                InetAddress clientIP = packet.getAddress();
                int clientPort = packet.getPort();

                if (command.startsWith("connect")) {
                    clientPorts.put(clientPort, nextPort);
                    DatagramPacket responsePacket = new
DatagramPacket(String.valueOf(nextPort).getBytes(), String.valueOf(nextPort).length(), clientIP,
clientPort);
                    serverSocket.send(responsePacket);
                    nextPort++;
                } else {
                    Runnable handler = new CommandHandler(serverSocket, packet, command);
                    new Thread(handler).start();
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static class CommandHandler implements Runnable {
        private DatagramSocket serverSocket;

```

```
private DatagramPacket packet;
private String command;

public CommandHandler(DatagramSocket serverSocket, DatagramPacket packet, String
command) {
    this.serverSocket = serverSocket;
    this.packet = packet;
    this.command = command;
}

@Override
public void run() {
    String[] parts = command.split("\\s+");
    String response = "";
    switch (parts[0]) {
        case "list":
            response = executeListCommand();
            break;
        case "get":
            response = executeGetCommand(parts);
            break;
        case "remove":
            response = executeRemoveCommand(parts);
            break;
        case "disconnect":
            executeDisconnectCommand(parts, response);
            break;
        default:
            response = "Comando no reconocido";
            break;
    }
}

private String executeListCommand() {
    //comando list
    StringBuilder fileList = new StringBuilder();
    try {
        Process process = Runtime.getRuntime().exec("ls -l " + rootDirectory);
        BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));
        String line;
        while ((line = reader.readLine()) != null) {
            fileList.append(line).append("\n");
        }
        reader.close();
        process.waitFor();
    } catch (IOException | InterruptedException e) {
        e.printStackTrace();
    }
}
```

```
    }  
    return "Error al listar archivos en " + rootDirectory;  
    }  
    return fileList.toString();  
}
```

```
private String executeGetCommand(String[] parts) {
    //comando get
    String fileName = parts[1];
    File file = new File(rootDirectory, fileName);
    if (!file.exists() || !file.isFile()) {
        return "El archivo " + fileName + " no existe en " + rootDirectory;
    }

    return "Enviando archivo: " + fileName;
}

private String executeRemoveCommand(String[] parts) {
    //comando remove
    String fileName = parts[1];
    File file = new File(rootDirectory, fileName);
    if (!file.exists() || !file.isFile()) {
        return "El archivo " + fileName + " no existe en " + rootDirectory;
    }
    if (file.delete()) {
        return "Archivo " + fileName + " eliminado correctamente";
    } else {
        return "Error al eliminar el archivo " + fileName;
    }
}

private void executeDisconnectCommand(String[] parts, String response) {
    //comando disconnect
    try {
        if (!parts[0].equals("disconnect")) {
            DatagramPacket responsePacket = new DatagramPacket(response.getBytes(),
response.length(), packet.getAddress(), packet.getPort());
            serverSocket.send(responsePacket);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Clase Cliente:

```
import java.io.*;
import java.net.*;

/**
 * Clase para el cliente, donde tenemos los comandos
 * @author Andrea Castilla Cocera
 * acasoc098@g.educand.es
 */
public class Cliente {
    public static void main(String[] args) throws IOException {
        String serverAddress = "127.0.0.1";
        int serverPort = 3000;
        String archivo = "listado.txt";

        try (DatagramSocket clientSocket = new DatagramSocket()) {
            InetAddress serverIP = InetAddress.getByName(serverAddress);
            byte[] buffer = new byte[500];

            String connectCommand = "connect " + serverAddress;
            DatagramPacket connectPacket = new DatagramPacket(connectCommand.getBytes(),
connectCommand.length(), serverIP, serverPort);
            clientSocket.send(connectPacket);

            DatagramPacket responsePacket = new DatagramPacket(buffer, buffer.length);
            clientSocket.receive(responsePacket);
            String response = new String(responsePacket.getData(), 0, responsePacket.getLength());
            int newPort = Integer.parseInt(response);

            String[] commands = {"list", "get archivo.txt", "remove " + archivo, "disconnect"};
            for (String command : commands) {
                DatagramPacket commandPacket = new DatagramPacket(command.getBytes(),
command.length(), serverIP, newPort);
                clientSocket.send(commandPacket);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```