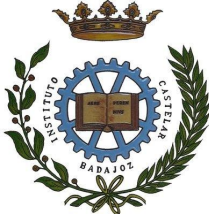




PROTOCOLO TFTP

Por Pablo Luis Andérica Torrado y Alejandro Julián Matías
Gutiérrez

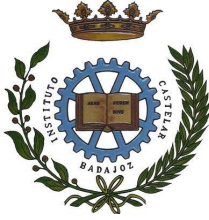


El protocolo TFTP

- Se utiliza para la transferencia de archivos
- Más rápido que el FTP, pero con menos funcionalidades
- Utiliza el protocolo UDP en el puerto 69.

Protocolo Simple de Transferencia de Ficheros

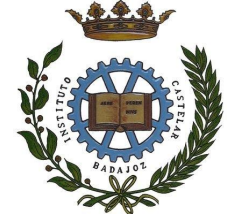




El protocolo TFTP vs FTP

- Menos seguro, sin log in ni verificación de errores
- Prioriza la velocidad a la seguridad
- No permite el transporte de ficheros pesados ni en binario.





Clases fundamentales TFTP

El protocolo TFTP utiliza cinco tipos de paquetes, cada uno de los cuales empieza con un campo de **código de operación** de 16 bits (Operations Code) con el valor correspondiente:

Código de operación	Tipo de paquete	Descripción
1	RRQ (Read request)	Solicitud de lectura
2	WRQ (Write request)	Solicitud de escritura
3	DATA (Data)	Paquete de datos
4	ACK (Acknowledgment)	Paquete de confirmación
5	ERROR (Error)	Paquete de error



Servidor TFTP

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

public class TFTPServer {

    public static void main(String args[]) {
        try {
            // Usar puerto 6973
            DatagramSocket socket = new DatagramSocket(6973);
            System.out.println("Servidor listo. Puerto: " + socket.getLocalPort());

            // Escuchar solicitudes
            while (true) {
                TFTPpacket entrada = TFTPpacket.receive(socket);
                // Recibir solicitud de lectura
                if (entrada instanceof TFTPRead) {
                    System.out.println("Solicitud de lectura desde " + entrada.getAddress());
                    TFTPserverRRQ r = new TFTPserverRRQ(((TFTPRead) entrada));
                }
                // Recibir solicitud de escritura
                else if (entrada instanceof TFTPWrite) {
                    System.out.println("Solicitud de escritura desde " + entrada.getAddress());
                    TFTPserverWRQ w = new TFTPserverWRQ(((TFTPWrite) entrada));
                }
            }
        } catch (SocketException e) {
            System.out.println("Servidor terminado (SocketException): " + e.getMessage());
        } catch (TftpException e) {
            System.out.println("Servidor terminado (TftpException): " + e.getMessage());
        } catch (IOException e) {
            System.out.println("Servidor terminado (IOException): " + e.getMessage());
        }
    }
}
```



Servidor TFTP-Clase packet

```
public class TFTPpacket {

    // Constantes TFTP
    public static int Puerto = 69;
    public static int LongitudMaximaDePaquete = 516;
    public static int DatosMaximosTFTP = 512;

    // Códigos de operación TFTP
    protected static final short tftpRRQ = 1;
    protected static final short tftpWRQ = 2;
    protected static final short tftpDATA = 3;
    protected static final short tftpACK = 4;
    protected static final short tftpERROR = 5;

    // Desplazamientos en el paquete
    protected static final int opOffset = 0;
    protected static final int fileOffset = 2;
    protected static final int blkOffset = 2;
    protected static final int dataOffset = 4;
    protected static final int numOffset = 2;
    protected static final int msgOffset = 4;

    // El paquete real para la transferencia UDP
    protected byte[] message;
    protected int length;

    // Información de dirección (necesaria para respuestas)
    protected InetAddress host;
    protected int port;

    // Constructor
    public TFTPpacket() {
        message = new byte[LongitudMaximaDePaquete];
        length = LongitudMaximaDePaquete;
    }
}
```

```
// Métodos para recibir el paquete y convertirlo al tipo correcto (datos/ACK/lectura,...)
public static TFTPpacket receive(DatagramSocket Socket) throws IOException {
    TFTPpacket Entrada = new TFTPpacket(), PaqueteDevuelto = new TFTPpacket();
    // Recibir datos y ponerlos en in.message
    DatagramPacket PaqueteEntrada = new DatagramPacket(Entrada.message, Entrada.length);
    Socket.receive(PaqueteEntrada);

    switch (Entrada.get(0)) {
        case tftpRRQ:
            PaqueteDevuelto = new TFTPread();
            break;
        case tftpWRQ:
            PaqueteDevuelto = new TFTPwrite();
            break;
        case tftpDATA:
            PaqueteDevuelto = new TFTPdata();
            break;
        case tftpACK:
            PaqueteDevuelto = new TFTPack();
            break;
        case tftpERROR:
            PaqueteDevuelto = new TFTPerror();
            break;
    }
    PaqueteDevuelto.message = Entrada.message;
    PaqueteDevuelto.length = PaqueteEntrada.getLength();
    PaqueteDevuelto.host = PaqueteEntrada.getAddress();
    PaqueteDevuelto.port = PaqueteEntrada.getPort();

    return PaqueteDevuelto;
}
```




Servidor TFTP-Clase packet

```
✓ final class TFTPdata extends TFTPpacket {  
  
    protected TFTPdata() {}  
  
    public TFTPdata(int numeroBloque, FileInputStream in) throws IOException {  
        this.message = new byte[LongitudMaximaDePaquete];  
        this.put(opOffset, tftpDATA);  
        this.put(blkOffset, (short) numeroBloque);  
  
        length = in.read(message, dataOffset, DatosMaximosTFTP) + 4;  
    }  
  
    public int numeroBloque() {  
        return this.get(blkOffset);  
    }  
  
    public int write(FileOutputStream out) throws IOException {  
        out.write(message, dataOffset, length - 4);  
        return (length - 4);  
    }  
}
```

```
final class TFTPread extends TFTPpacket {  
  
    protected TFTPread() {}  
  
    public TFTPread(String nombreArchivo, String modoDatos) {  
        length = 2 + nombreArchivo.length() + 1 + modoDatos.length() + 1;  
        message = new byte[length];  
        put(opOffset, tftpRRQ);  
        put(fileOffset, nombreArchivo, (byte) 0);  
        put(fileOffset + nombreArchivo.length() + 1, modoDatos, (byte) 0);  
    }  
  
    public String nombreArchivo() {  
        return this.get(fileOffset, (byte) 0);  
    }  
  
    public String tipoSolicitud() {  
        String nombre = nombreArchivo();  
        return this.get(fileOffset + nombre.length() + 1, (byte) 0);  
    }  
}  
  
final class TFTPwrite extends TFTPpacket {  
    // Constructores  
    protected TFTPwrite() {}  
  
    public TFTPwrite(String nombreArchivo, String modoDatos) {  
        length = 2 + nombreArchivo.length() + 1 + modoDatos.length() + 1;  
        message = new byte[length];  
        put(opOffset, tftpWRQ);  
        put(fileOffset, nombreArchivo, (byte) 0);  
        put(fileOffset + nombreArchivo.length() + 1, modoDatos, (byte) 0);  
    }  
  
    public String nombreArchivo() {  
        return this.get(fileOffset, (byte) 0);  
    }  
  
    public String tipoSolicitud() {  
        String nombre = nombreArchivo();  
        return this.get(fileOffset + nombre.length() + 1, (byte) 0);  
    }  
}
```



Servidor TFTP-Clase RRQ y WRQ

```
public TFTPserverRRQ(TFTPread solicitud) throws TftpException {
    try {
        req = solicitud;
        // Abrir nuevo socket con número de puerto aleatorio para transferencia
        Socket = new DatagramSocket();
        Socket.setSoTimeout(1000);
        nombreArchivo = solicitud.nombreArchivo();

        host = solicitud.getAddress();
        port = solicitud.getPort();

        File archivoFuente = new File("../" + nombreArchivo);

        if (archivoFuente.exists() && archivoFuente.isFile() && archivoFuente.canRead()) {
            source = new FileInputStream(archivoFuente);
            this.start();
        } else
            throw new TftpException("violación de acceso");

    } catch (Exception e) {
        TFTPError ePak = new TFTPError(1, e.getMessage());
        try {
            ePak.send(host, port, Socket);
        } catch (Exception f) {
        }

        System.out.println("Fallo de inicio del cliente: " + e.getMessage());
    }
}
```

```
public TFTPserverWRQ(TFTPwrite solicitud) throws TftpException {
    try {
        req = solicitud;
        Socket = new DatagramSocket();
        Socket.setSoTimeout(1000);

        host = solicitud.getAddress();
        puerto = solicitud.getPort();
        fileName = solicitud.nombreArchivo();

        saveFile = new File("../" + fileName);

        if (!saveFile.exists()) {
            outFile = new FileOutputStream(saveFile);
            TFTPack a = new TFTPack(0);
            a.send(host, puerto, Socket);
            this.start();
        } else
            throw new TftpException("El archivo ya existe");

    } catch (Exception e) {
        TFTPError ePak = new TFTPError(1, e.getMessage());
        try {
            ePak.send(host, puerto, Socket);
        } catch (Exception f) {
        }

        System.out.println("Inicio fallido del cliente: " + e.getMessage());
    }
}
```




Servidor TFTP-Clase RRQ y WRQ

```
public void run() {
    int bytesRead = TFTPpacket.LongitudMaximaDePaquete;

    if (req instanceof TFTPread) {
        try {
            for (int numBloque = 1; bytesRead == TFTPpacket.LongitudMaximaDePaquete; numBloque++) {
                TFTPdata outPak = new TFTPdata(numBloque, source);
                bytesRead = outPak.getLength();
                outPak.send(host, port, Socket);

                while (timeoutlimit != 0) {
                    try {
                        TFTPpacket ack = TFTPpacket.receive(Socket);

                        if (!(ack instanceof TFTPack)) {
                            throw new Exception("Fallo del cliente");
                        }
                        TFTPack a = (TFTPack) ack;

                        if (a.numeroBloque() != numBloque) {
                            throw new SocketTimeoutException("Paquete perdido, reenviar");
                        }
                        break;
                    } catch (SocketTimeoutException t) {
                        System.out.println("Reenviar bloque " + numBloque);
                        timeoutlimit--;
                        outPak.send(host, port, Socket);
                    }
                }
                if (timeoutlimit == 0) {
                    throw new Exception("falla de conexión");
                }
            }
            System.out.println("Transferencia completada. (Cliente " + host + ")");
        } catch (Exception e) {
            TFTPError ePak = new TFTPError(1, e.getMessage());

            try {
                ePak.send(host, port, Socket);
            } catch (Exception f) {
            }

            System.out.println("Fallo del cliente: " + e.getMessage());
        }
    }
}
```

```
public void run() {

    if (req instanceof TFTPwrite) {
        try {
            for (int numBloque = 1; bytesOut == 512; bytesOut += 512; numBloque++) {
                while (timeoutlimit != 0) {
                    try {
                        TFTPpacket inPak = TFTPpacket.receive(Socket);

                        if (inPak instanceof TFTPError) {
                            TFTPError p = (TFTPError) inPak;
                            throw new TftpException(p.mensaje());
                        } else if (inPak instanceof TFTPdata) {
                            TFTPdata p = (TFTPdata) inPak;

                            if (p.numeroBloque() != numBloque) {
                                throw new SocketTimeoutException();
                            }
                            bytesOut = p.write(outFile);
                            TFTPack a = new TFTPack(numBloque);
                            a.send(host, puerto, Socket);
                            break;
                        }
                    } catch (SocketTimeoutException t2) {
                        System.out.println("Tiempo de espera agotado, reenviar ACK");
                        TFTPack a = new TFTPack(numBloque - 1);
                        a.send(host, puerto, Socket);
                        timeoutlimit--;
                    }
                }
                if (timeoutlimit == 0) {
                    throw new Exception("Fallo de conexión");
                }
            }
            System.out.println("Transferencia completada. (Cliente " + host + ")");
        } catch (Exception e) {
            TFTPError ePak = new TFTPError(1, e.getMessage());
            try {
                ePak.send(host, puerto, Socket);
            } catch (Exception f) {
            }

            System.out.println("Fallo del cliente: " + e.getMessage());
            saveFile.delete();
        }
    }
}
```

Cliente TFTP



```
1
2
3 import java.net.InetAddress;
4 import java.net.UnknownHostException;
5
6
7 public class ClienteTFTP {
8     public static void main(String[] args){
9         String host = "";
10        String nombreArchivo = "";
11        String modo = "octeto"; //8 bits = 1 byte
12        String tipo = "";
13        try {
14
15            if (args.length == 0)
16                System.err.println("Uso: ClienteTFTP [host] [Tipo(L/E)] [nombreArchivo]");
17
18            if (args.length == 3) {
19                host = args[0];
20                tipo = args[args.length - 2];
21                nombreArchivo = args[args.length - 1];
22            }
23
24            else if (args.length == 4) {
25                host = args[0];
26                modo = args[args.length - 1];
27                tipo = args[args.length - 3];
28                nombreArchivo = args[args.length - 2];
29            } else
30                System.err.println("comando incorrecto.Uso modo: ClienteTFTP [host] [Tipo(L/E)] [nombreArchivo]");
31
32            InetAddress servidor = InetAddress.getByName(host);
33
34
35            if (tipo.matches("L")) {
36                TFTPClienteRRQ r = new TFTPClienteRRQ(servidor, nombreArchivo, modo);
37            }
38
39            else if (tipo.matches("E")) {
40                TFTPClienteWRQ w = new TFTPClienteWRQ(servidor, nombreArchivo, modo);
41            } else {
42                System.err.println("comando incorrecto. Uso: ClienteTFTP [host] Tipo(L/E) [nombreArchivo]");
43            }
44
45        } catch (UnknownHostException e) {
46            System.out.println("Host desconocido " + host);
47        } catch (Exception e) {
48            System.out.println(e.getMessage());
49        }
50    }
51 }
```



Cliente TFTP-Clase Packet

```
public class PaqueteTFTP {

    // Constantes TFTP
    public static int puertoTftp = 69;
    public static int longitudMaximaPaqueteTftp = 516;
    public static int datosMaximosTftp = 512;

    // Códigos de operación TFTP
    protected static final short tftpRRQ = 1;
    protected static final short tftpWRQ = 2;
    protected static final short tftpDATA = 3;
    protected static final short tftpACK = 4;
    protected static final short tftpERROR = 5;

    // Desplazamientos en el paquete
    protected static final int desplazamientoOperacion = 0;
    protected static final int desplazamientoNumero = 2;
    protected static final int desplazamientoDatos = 4;
    protected static final int desplazamientoMensaje = 4;
    protected static final int desplazamientoArchivo = 2;
    protected static final int desplazamientoBloque = 2;

    protected byte[] mensaje;
    protected int longitud;

    protected InetAddress host;
    protected int puerto;

    public PaqueteTFTP() {
        mensaje = new byte[longitudMaximaPaqueteTftp];
        longitud = longitudMaximaPaqueteTftp;
    }
}
```

```
// Métodos para recibir el paquete y convertirlo datos,ack,lectura
+ Pablo +
public static PaqueteTFTP recibir(DatagramSocket sock) throws IOException {
    // Se crean instancias de PaqueteTFTP para almacenar la información de entrada y salida
    PaqueteTFTP in = new PaqueteTFTP(), retPak = new PaqueteTFTP();

    // Se crea un DatagramPacket para recibir el paquete de entrada
    DatagramPacket inPak = new DatagramPacket(in.mensaje, in.longitud);

    // Se bloquea la ejecución
    sock.receive(inPak);

    // Se determina el tipo de operación según el primer byte del paquete recibido
    switch (in.get(0)) {
        case tftpRRQ:
            retPak = new LecturaTFTP();
            break;
        case tftpWRQ:
            retPak = new EscrituraTFTP();
            break;
        case tftpDATA:
            retPak = new DatosTFTP();
            break;
        case tftpACK:
            retPak = new ACKTFTP();
            break;
        case tftpERROR:
            retPak = new ErrorTFTP();
            break;
    }

    // Se copian los datos de entrada a la salida
    retPak.mensaje = in.mensaje;
    retPak.longitud = inPak.getLength();
    retPak.host = inPak.getAddress();
    retPak.puerto = inPak.getPort();
    return retPak;
}
```



Cliente TFTP-Clase Packet

```
/* Pablo */
final class DatosTFTP extends PaqueteTFTP {
    /* Pablo */
    protected DatosTFTP() {
    }
    /* Pablo */
    public DatosTFTP(int numeroBloque, FileInputStream in) throws IOException {
        // Se inicializa el array de bytes con la longitud máxima de un paquete TFTP = 516
        this.mensaje = new byte[longitudMaximaPaqueteTftp];

        // Se coloca el código de operación data
        this.colocar(desplazamientoOperacion, tftpDATA);

        // Se coloca el número de bloque en el paquete
        this.colocar(desplazamientoBloque, (short) numeroBloque);

        // Se lee desde el FileInputStream y se guarda en el array de bytes 'mensaje'
        // Se suma 4 (código de operación y número de bloque)
        longitud = in.read(mensaje, desplazamientoDatos, datosMaximosTftp) + 4;
    }
    // Método que devuelve el número de bloque
    /* Pablo */
    public int numeroBloque() {
        return this.get(desplazamientoBloque);
    }
    // Método para escribir los datos en un FileOutputStream
    /* Pablo */
    public int escribir(FileOutputStream out) throws IOException {
        // Se escribe en el FileOutputStream los datos del paquete
        out.write(mensaje, desplazamientoDatos, longitud - 4);
        return (longitud - 4);
    }
}
```

```
/* Paquete ACK: coloca el código de operación correcto */
final class ACKTFTP extends PaqueteTFTP {

    protected ACKTFTP() {
    }
    public ACKTFTP(int numeroBloque) {
        longitud = 4;
        this.mensaje = new byte[longitud];
        colocar(desplazamientoOperacion, tftpACK);
        colocar(desplazamientoBloque, (short) numeroBloque);
    }

    public int numeroBloque() {
        return this.get(desplazamientoBloque);
    }
}
```

```
/* Paquete LECTURA: coloca el código de operación y el nombre de archivo */
/* Pablo */
final class LecturaTFTP extends PaqueteTFTP {
    /* Pablo */
    protected LecturaTFTP() {
    }
    /* Pablo */
    public LecturaTFTP(String nombreArchivo, String modoDatos) {
        // Calcula la longitud del paquete
        longitud = 2 + nombreArchivo.length() + 1 + modoDatos.length() + 1;

        mensaje = new byte[longitud];

        // Coloca el código de operación RRQ en el paquete
        colocar(desplazamientoOperacion, tftpRRQ);

        // Coloca el nombre del archivo en el paquete
        colocar(desplazamientoArchivo, nombreArchivo, (byte) 0);

        // Coloca el modo de datos en el paquete
        colocar(desplazamientoArchivo + nombreArchivo.length() + 1, modoDatos, (byte) 0);
    }
    /* Pablo */
    public String nombreArchivo() {
        return this.get(desplazamientoArchivo, (byte) 0);
    }
    /* Pablo */
    public String tipoSolicitud() {
        // Obtiene el nombre del archivo
        String nombreArchivo = nombreArchivo();

        // Devuelve el modo de datos almacenado
        return this.get(desplazamientoArchivo + nombreArchivo.length() + 1, (byte) 0);
    }
}
```



Cliente TFTP-Clase RRQ

```
try {  
    DatagramSocket socket = new DatagramSocket();  
    socket.setSoTimeout(2000);  
  
    FileOutputStream archivoSalida = new FileOutputStream("../" + nombreArchivo);  
  
    LecturaTFTP paqueteSolicitud = new LecturaTFTP(nombreArchivo, modoDatos);  
    paqueteSolicitud.enviar(servidor, 6973, socket);  
  
    ACKTFTP ack = null;  
    InetAddress nuevaIP = servidor;  
    int nuevoPuerto = 0;  
    int limiteTiempo = 5;  
    int pruebaPerdida = 1;
```

```
    catch (SocketTimeoutException t) {  
  
        if (numBloque == 1) {  
            System.out.println("No se pudo llegar al servidor");  
            paqueteSolicitud.enviar(servidor, 6973, socket);  
            limiteTiempo--;  
        }  
  
        else {  
            System.out.println("Tiempo de conexión agotado, reenviar último ack. límite de tiempo restante=" + limiteTiempo);  
            ack = new ACKTFTP(numBloque - 1);  
            ack.enviar(nuevaIP, nuevoPuerto, socket);  
            limiteTiempo--;  
        }  
    }  
    if (limiteTiempo == 0) {  
        System.err.println("ERROR de conexión");  
        System.exit(1);  
    }  
    System.out.println("\nDescarga finalizada.\nNombre de archivo: " + nombreArchivo);
```

```
//bucle para recibir y procesar los paquetes de datos del servidor  
for (int numBloque = 1, bytesSalida = 512; bytesSalida == 512; numBloque++) {  
    while (limiteTiempo != 0) {  
        try {  
            PaqueteTFTP paqueteEntrada = PaqueteTFTP.recibir(socket);  
            //se verifica el tipo de paquete recibido  
            if (paqueteEntrada instanceof ErrorTFTP) {  
                ErrorTFTP p = (ErrorTFTP) paqueteEntrada;  
                System.err.println("ERROR");  
                System.exit(1);  
            } else if (paqueteEntrada instanceof DatosTFTP) {  
                DatosTFTP p = (DatosTFTP) paqueteEntrada;  
  
                //se muestra el progreso  
                if (numBloque % 500 == 0) {  
                    System.out.print("\b.>");  
                }  
                if (numBloque % 15000 == 0) {  
                    System.out.println("\b.");  
                }  
  
                nuevaIP = p.obtenerDireccion();  
  
                if (nuevoPuerto != 0 && nuevoPuerto != p.obtenerPuerto()) {  
                    continue;  
                }  
                nuevoPuerto = p.obtenerPuerto();  
                //verifica  
  
                if (numBloque != p.numeroBloque()) {  
                    throw new SocketTimeoutException();  
                }  
  
                bytesSalida = p.escribir(archivoSalida);  
  
                ack = new ACKTFTP(numBloque);
```



Cliente TFTP-Clase WRQ

```
public TFTPCliente(InetAddress ip, String nombre, String modo) {
    servidor = ip;
    nombreArchivo = nombre;
    modoDatos = modo;

    try {

        DatagramSocket socket = new DatagramSocket();
        socket.setSoTimeout(2000);
        int limiteTiempo = 5;

        FileInputStream fuente = new FileInputStream("../" + nombreArchivo);

        EscrituraTFTP paqueteSolicitud = new EscrituraTFTP(nombreArchivo, modoDatos);
        paqueteSolicitud.enviar(servidor, 6973, socket);

        PaqueteTFTP respuestaEnvio = PaqueteTFTP.recibir(socket);

        int puerto = respuestaEnvio.obtenerPuerto();

        if (respuestaEnvio instanceof PaqueteTFTP) {
            PaqueteTFTP respuesta = (PaqueteTFTP) respuestaEnvio;
            System.out.println("--Servidor listo--\nSubiendo");
        } else if (respuestaEnvio instanceof ErrorTFTP) {
            ErrorTFTP respuesta = (ErrorTFTP) respuestaEnvio;
            fuente.close();
            System.err.println("ERROR");
            System.exit(1);
        }

        int bytesLeidos = PaqueteTFTP.longitudMaximaPaqueteTftp;

        for (int numBloque = 1; bytesLeidos == PaqueteTFTP.longitudMaximaPaqueteTftp; numBloque++) {
            DatosTFTP paqueteSalida = new DatosTFTP(numBloque, fuente);
            bytesLeidos = paqueteSalida.obtenerLongitud();
            paqueteSalida.enviar(servidor, puerto, socket);

            if (numBloque % 500 == 0) {
                System.out.print("\b.>");
            }
            if (numBloque % 15000 == 0) {
                System.out.println("\b.");
            }
        }
    }
}
```

```
while (limiteTiempo != 0) {
    try {
        PaqueteTFTP ack = PaqueteTFTP.recibir(socket);
        if (!(ack instanceof PaqueteTFTP)) {
            break;
        }

        PaqueteTFTP a = (PaqueteTFTP) ack;

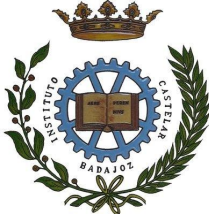
        if (puerto != a.obtenerPuerto()) {
            continue;
        }

        break;
    } catch (SocketTimeoutException t0) {
        System.out.println("Reenviar blk " + numBloque);
        paqueteSalida.enviar(servidor, puerto, socket);
        limiteTiempo--;
    }
}

if (limiteTiempo == 0) {
    System.err.println("ERROR de conexion");
    System.exit(1);
}

fuente.close();
socket.close();

System.out.println("\nSubida finalizada!\nNombre de archivo: " + nombreArchivo);
```

Prueba de ejecución

```
pablo@MSIdePablo:/mnt/c/users/pablo/IdeaProjects/TFTP/TFTPServer/src$ java TFTPServer
Server Ready. Port: 6973
```

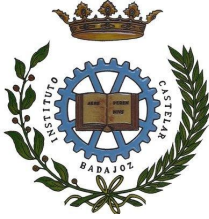
```

├── TFTPClient
│   └── src
│       ├── ClienteTFTP.java
│       ├── PaqueteTFTP.java
│       ├── TFTPClienteRRQ.java
│       └── TFTPClienteWRQ.java
├── client2.txt
├── TFTPServer
│   └── src
│       ├── TFTPpacket.java
│       ├── TFTPServer.java
│       ├── TFTPserverRRQ.java
│       └── TFTPserverWRQ.java
└── server2.txt
```



Vemos que archivos txt hay

```
pablo@MSIdePablo:/mnt/c/users/pablo/IdeaProjects/TFTP/TFTPClient/src$ java ClienteTFTP localhost L server2.txt
```



Prueba de ejecución

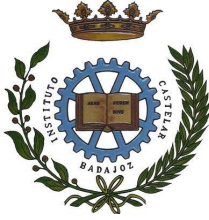
Descargando

Descarga finalizada.

Nombre de archivo: server2.txt

```
pablo@MSidePablo:/mnt/c/users/pablo/IdeaProjects/UT4-TFTPserver/UT4/TRABAJO/TFTPClient/src$ java ClientTFTP localhost E client2.txt
--Servidor listo--
Subiendo
¡Subida finalizada!
Nombre de archivo: client2.txt
```

```
pablo@MSidePablo:/mnt/c/users/pablo/IdeaProjects/UT4-TFTPserver/UT4/TRABAJO/TFTPServer/src$ java TFTPServer
Servidor listo. Puerto: 6973
Solicitud de lectura desde /127.0.0.1
Transferencia completada. (Cliente /127.0.0.1)
Solicitud de escritura desde /127.0.0.1
Transferencia completada. (Cliente /127.0.0.1)
```



Resultado Ejecución

