Pool de hilos

Un pool de hilos (o pool de threads) es un patrón de diseño que gestiona un conjunto de hilos reutilizables para ejecutar tareas concurrentes. En lugar de crear y destruir hilos para cada tarea, un pool mantiene un número fijo de hilos disponibles, lo que ayuda a mejorar la eficiencia y el rendimiento de una aplicación, especialmente en sistemas donde la creación y destrucción de hilos puede ser costosa.

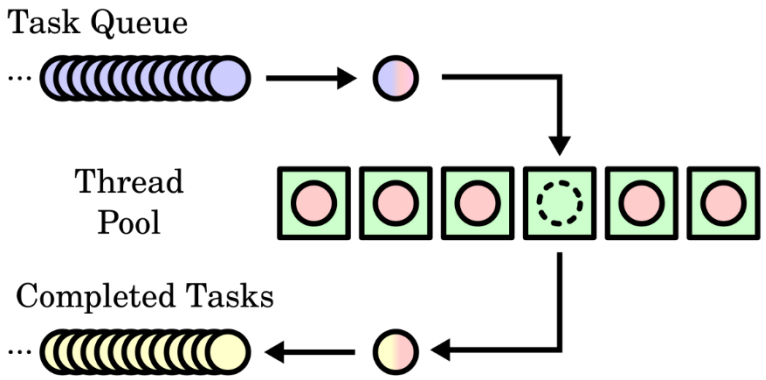
¿Para qué sirve un pool de hilos?

Eficiencia: Reduce el overhead de la creación y destrucción de hilos al reutilizar hilos existentes.

Control de recursos: Limita el número de hilos activos al mismo tiempo, evitando la sobrecarga del sistema.

Manejo de tareas concurrentes: Permite ejecutar múltiples tareas en paralelo, mejorando el rendimiento en aplicaciones que requieren procesamiento simultáneo.

Facilidad de uso: Simplifica la programación concurrente al manejar la sincronización y el ciclo de vida de los hilos.



Un pool de hilos se compone de varios componentes clave que trabajan juntos para gestionar y ejecutar tareas concurrentes. A continuación se detallan los componentes, el orden de creación y las instrucciones más importantes para usar un pool de hilos en Java.

Componentes de un Pool de Hilos

- **Hilos (Threads):** Son las unidades de trabajo que se ejecutan en paralelo. El pool contiene un número fijo o variable de hilos que pueden ser reutilizados para ejecutar tareas.
- **Cola de tareas (Task Queue):** Es donde se almacenan las tareas que esperan ser ejecutadas. Si todos los hilos del pool están ocupados, las nuevas tareas se colocan en esta cola hasta que un hilo esté disponible.
- **Administrador de hilos (Thread Manager):** Es responsable de la creación, asignación y gestión de los hilos. Supervisa el estado de los hilos y las tareas en la cola.

- **Tareas (Tasks):** Son las unidades de trabajo que se envían al pool. En Java, generalmente se implementan como objetos que implementan la interfaz Runnable o Callable.

Para crear un pool de hilos en Java se puede seguir el siguiente orden que más adelante se va a abordar:

1. **Importar las Clases Necesarias:** Importar las clases del paquete java.util.concurrent.
2. **Crear un Pool de Hilos:** Utilizar la clase Executors para crear el pool, especificando el tipo de pool deseado (fijo, variable, etc.).
3. **Enviar Tareas al Pool:** Crear tareas (implementando Runnable o Callable) y enviarlas al pool usando el método submit() o execute().
4. **Cerrar el Pool:** Después de que todas las tareas han sido enviadas, se debe cerrar el pool utilizando shutdown().

Desarrollo:

Para la presente práctica se va a realizar un programa que transfiera archivos utilizando el protocolo FTP, para esta práctica se va a desarrollar usando el lenguaje de programación Java, para ello se va a tener un cliente y un servidor, un servidor se encontrara ejecutándose en un sistema operativo de Ubuntu y el cliente se va a encontrar en ejecución en un sistema operativo de Windows.

Para ello primero se va a comenzar creando una clase para el cliente teniendo en cuenta el tema a tratar donde se esta manejando un pool de hilos que va a permitir establecer diversas conexiones al servidor.

Para el cliente se tiene la clase FTPCliente.java teniendo los siguientes fragmentos de código:

```
private JTextField filePathField;
private JButton uploadButton;

public FTPCliente() {
    setTitle("Cliente FTP");
    setSize(400, 200);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(new FlowLayout());

    filePathField = new JTextField(20);
    uploadButton = new JButton("Seleccionar y Subir Archivo");

    uploadButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            selectFile();
        }
    });

    add(filePathField);
    add(uploadButton);
}
```

se configura la interfaz gráfica con una ventana (JFrame) titulada "Cliente FTP". Se añaden dos componentes: un campo de texto (JTextField) donde se muestra la ruta del archivo y un botón (JButton) que

permite al usuario seleccionar y subir el archivo. El botón está vinculado a un ActionListener que ejecuta el método selectFile() cuando es presionado.

```
private void selectFile() {
    JFileChooser fileChooser = new JFileChooser();
    int returnValue = fileChooser.showOpenDialog(this);

    if (returnValue == JFileChooser.APPROVE_OPTION) {
        File selectedFile = fileChooser.getSelectedFile();
        filePathField.setText(selectedFile.getAbsolutePath());
        uploadFile(selectedFile.getAbsolutePath());
    }
}
```

Este método abre un cuadro de diálogo de selección de archivos (JFileChooser). Si el usuario selecciona un archivo y confirma la acción (APPROVE_OPTION), la ruta del archivo se muestra en el campo de texto y se llama al método uploadFile() para iniciar la subida del archivo seleccionado.

```
public void uploadFile(String filePath) {
    try (Socket socket = new Socket(SERVER_ADDRESS, PORT);
         OutputStream output = socket.getOutputStream();
         InputStream input = socket.getInputStream();
         PrintWriter writer = new PrintWriter(output, true);
         BufferedReader reader = new BufferedReader(new
InputStreamReader(input))) {

        // Enviar comando PUT con el nombre del archivo
        File file = new File(filePath);
        writer.println("PUT " + file.getName());

        // Confirmar que el servidor está listo
        String serverResponse = reader.readLine();
        if ("READY".equals(serverResponse)) {
            System.out.println("Servidor listo para recibir el archivo.");

            // Enviar el archivo
            sendFile(file, output);

            // Confirmar finalización
            writer.println("END");

            // Respuesta final del servidor
            String finalResponse = reader.readLine();
            System.out.println("Respuesta final del servidor: " + finalResponse);
        } else {
            System.out.println("Error: El servidor no está listo. Respuesta: " +
serverResponse);
        }

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

El cliente se conecta al servidor FTP en la dirección y puerto especificados. Se establece una conexión usando Socket y se envía un comando PUT con el nombre del archivo a transferir. Luego, el cliente espera la confirmación del servidor con la respuesta "READY". Si la respuesta es válida, el archivo es enviado con el método sendFile().

```
private void sendFile(File file, OutputStream output) throws IOException {
    try (FileInputStream fileInput = new FileInputStream(file)) {
        byte[] buffer = new byte[4096];
        int bytesRead;

        System.out.println("Enviando archivo: " + file.getName());

        while ((bytesRead = fileInput.read(buffer)) != -1) {
            output.write(buffer, 0, bytesRead);
        }

        output.flush();
        System.out.println("Archivo " + file.getName() + " enviado con éxito.");
    }
}
```

Este método se encarga de enviar el contenido del archivo al servidor. Usa un FileInputStream para leer el archivo en bloques de 4096 bytes. Cada bloque de datos se escribe en el OutputStream que envía los datos al servidor. Cuando todo el archivo ha sido transferido, el flujo se vacía (flush) para asegurar que los datos han sido completamente enviados.

```
public class FTPCliente extends JFrame {
    private static final String SERVER_ADDRESS = "IP_DEL_SERVIDOR"; // IP del
servidor FTP
    private static final int PORT = 21;
    private JTextField filePathField;
    private JButton uploadButton;
```

Ahora se debe de agregar la lógica del servidor que se va a encargar de fungir como receptor de los archivos que sean transferidos por el cliente:

```
private static final int PORT = 21; // Puerto FTP estándar
private static final int MAX_THREADS = 10;

public static void main(String[] args) throws IOException {
    ServerSocket serverSocket = new ServerSocket(PORT);
    ExecutorService threadPool = Executors.newFixedThreadPool(MAX_THREADS);

    System.out.println("Servidor FTP iniciado en el puerto " + PORT);

    while (true) {
        Socket clientSocket = serverSocket.accept();
        threadPool.execute(new FTPHandler(clientSocket));
    }
}
```

El servidor FTP escucha conexiones en el puerto 21, utilizando un ServerSocket. Para gestionar múltiples conexiones de clientes simultáneamente, se usa un pool de hilos con un máximo de 10 hilos (MAX_THREADS). Cada vez que un cliente se conecta, se asigna un nuevo hilo del pool para manejar la conexión, ejecutando una instancia de la clase FTPHandler.

```
class FTPHandler implements Runnable {
    private Socket clientSocket;

    public FTPHandler(Socket socket) {
        this.clientSocket = socket;
    }

    @Override
    public void run() {
        try {
            InputStream input = clientSocket.getInputStream();
            OutputStream output = clientSocket.getOutputStream();
            BufferedReader reader = new BufferedReader(new
InputStreamReader(input));
            PrintWriter writer = new PrintWriter(output, true);

            String command = reader.readLine();
            System.out.println("Comando recibido: " + command);

            if (command.startsWith("PUT")) {
                String fileName = command.split(" ")[1];

                // Confirmación de que el servidor está listo para recibir el
archivo
                writer.println("LISTO");

                // Recibir el archivo
                receiveFile(fileName, input, reader, writer);
            }

            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

La clase FTPHandler es la responsable de gestionar cada conexión con un cliente. Implementa la interfaz Runnable, lo que permite que cada instancia se ejecute en un hilo separado. El servidor espera un comando del cliente, y si el comando recibido es PUT, que indica la subida de un archivo, el servidor responde con LISTO, señalando que está listo para recibir el archivo.

```
private void receiveFile(String fileName, InputStream input, BufferedReader
reader, PrintWriter writer) {
    try {
        File file = new File("servidor_" + fileName);
        FileOutputStream fileOutput = new FileOutputStream(file);
```

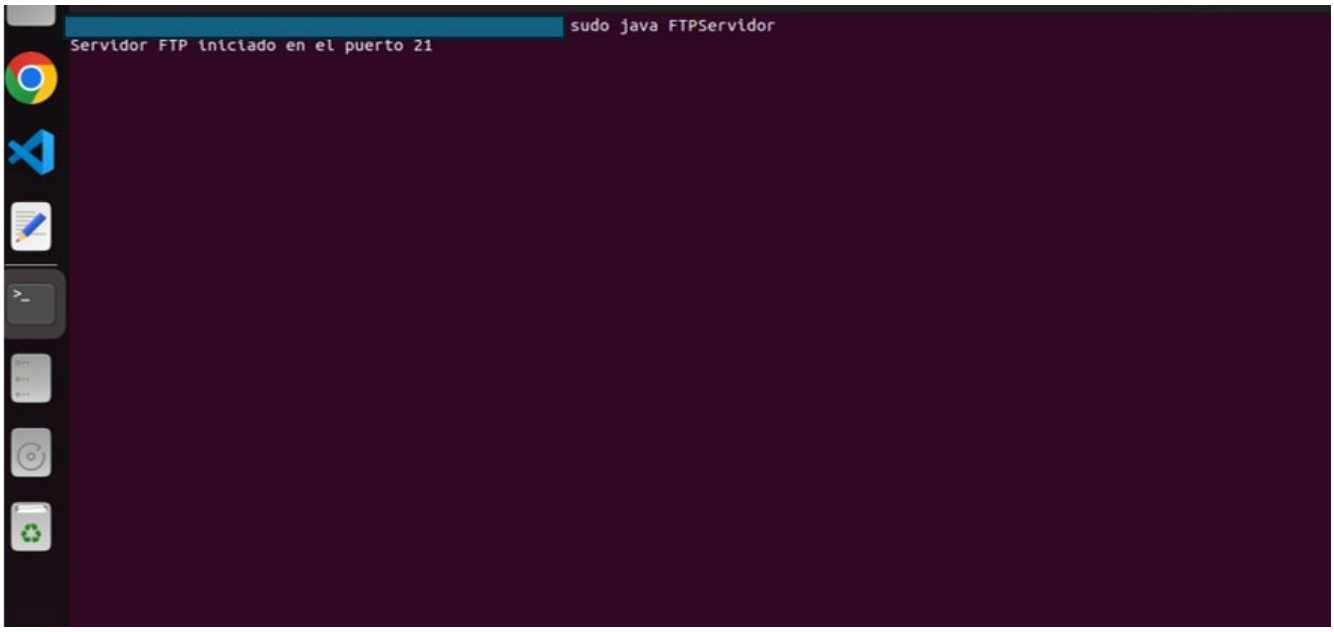
```
        byte[] buffer = new byte[4096];
        int bytesRead;

        // Recibir archivo hasta que se detecte el comando "END"
        while ((bytesRead = input.read(buffer)) != -1) {
            fileOutput.write(buffer, 0, bytesRead);
            // Verificar si el cliente envió "END"
            if (reader.ready() && "END".equals(reader.readLine())) {
                break;
            }
        }

        fileOutput.close();
        System.out.println("Archivo " + fileName + " recibido con éxito.");

        // Enviar respuesta al cliente
        writer.println("Archivo " + fileName + " subido correctamente.");
    } catch (IOException e) {
        e.printStackTrace();
        writer.println("Error al recibir el archivo " + fileName);
    }
}
```

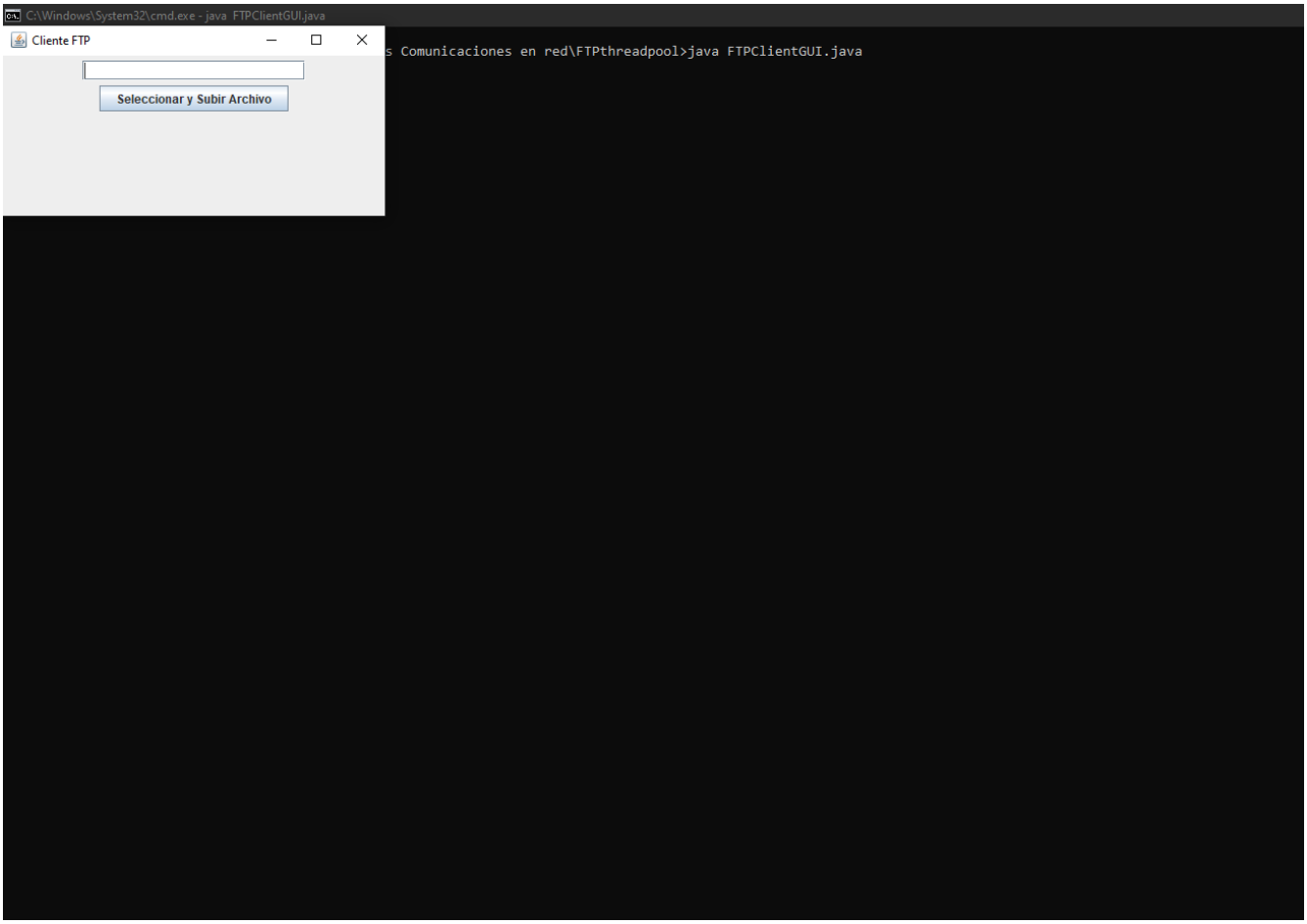
Este método gestiona la recepción del archivo desde el cliente. El servidor crea un archivo en el sistema con el nombre recibido y luego usa un FileOutputStream para escribir el contenido del archivo en bloques de 4096 bytes. El servidor sigue leyendo datos hasta que detecta el comando END enviado por el cliente, lo que indica que la transferencia ha finalizado.

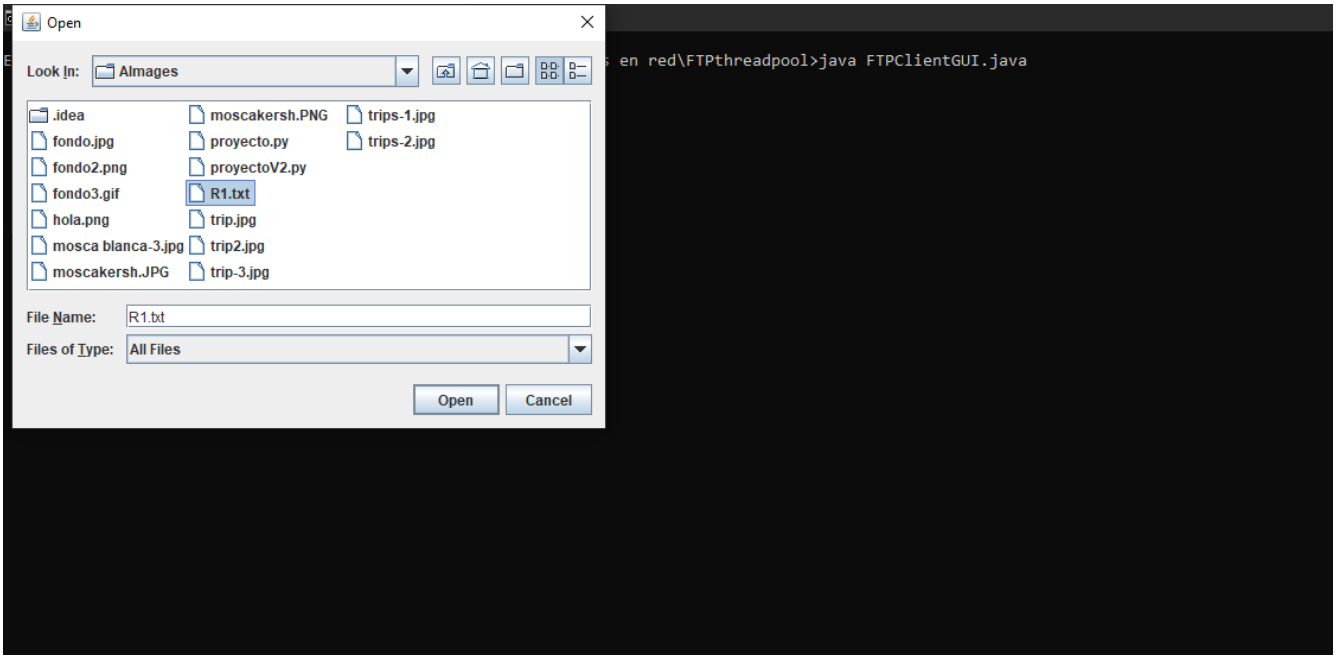
Ahora en la ejecución se realiza la compilación de ambas clases java con el siguiente comando:

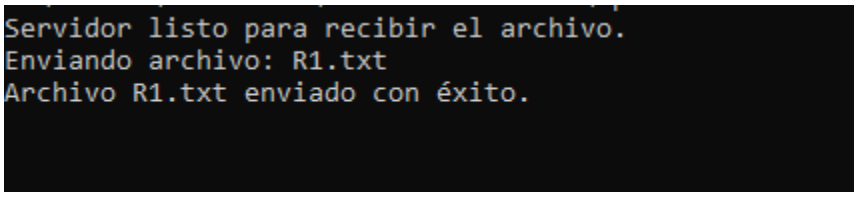
```
Javac *.java
```

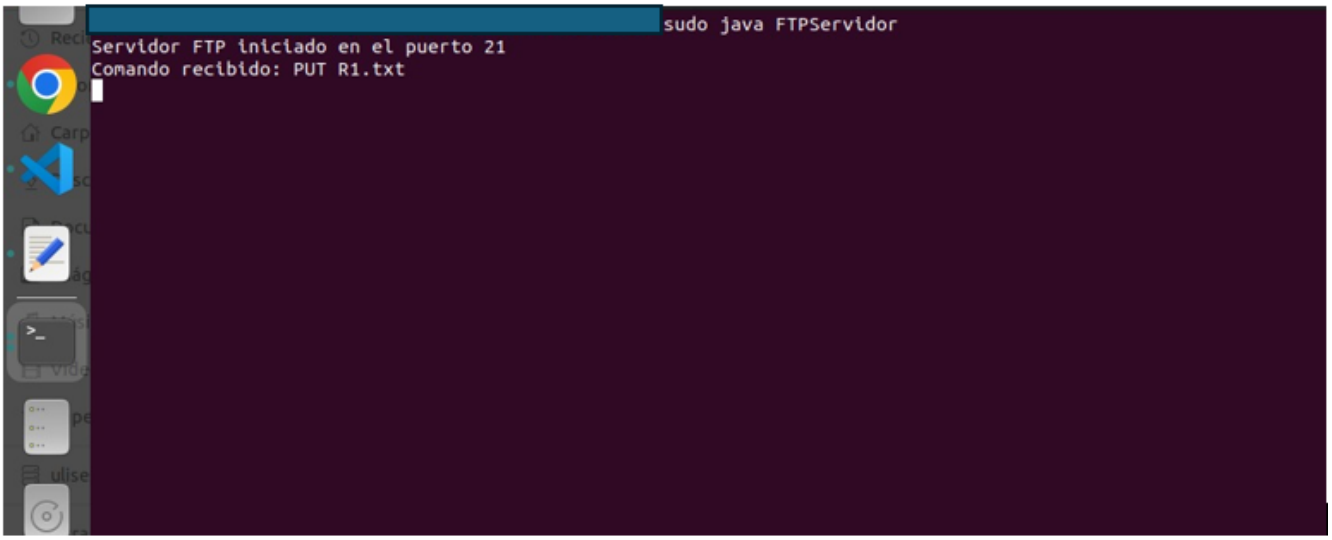
Después se ejecuta el servidor para que se encuentre escuchando:






Open
Look In:
Almages
.idea
fondo.jpg
fondo2.png
fondo3.gif
hola.png
mosca blanca-3.jpg
moscakersh.JPG
moscakersh.PNG
proyecto.py
proyectoV2.py
R1.txt
trip.jpg
trip2.jpg
trip-3.jpg
trips-1.jpg
trips-2.jpg
File Name:
R1.txt
Files of Type:
All Files
Open
Cancel
en red\FTPthreadpool>java FTPClientGUI.java


Servidor listo para recibir el archivo.
Enviando archivo: R1.txt
Archivo R1.txt enviado con éxito.


sudo java FTPServidor
Servidor FTP iniciado en el puerto 21
Comando recibido: PUT R1.txt


FTPHandler
.class
FTPServido
r.class
FTPServido
r.java
servidor_
R1.txt

Al final al abrir el archivo podemos ver que su contenido se encuentra ahí: