Daniel Verdes Martínez

Programación Servicios y Procesos  2º DAM

PSP02P01 - SOCKETS

Contenido

[Estructura 2](#_Toc159173024)

[Server 3](#_Toc159173025)

[HiloServidor 4](#_Toc159173026)

[Run() 4](#_Toc159173027)

[Mensaje “HOLA” 5](#_Toc159173028)

[MENSAJE “TRANSFERIR” 6](#_Toc159173029)

[MENSAJE “DESCONECTAR” 8](#_Toc159173030)

[Cliente 9](#_Toc159173031)

[MENSAJE “HOLA” 10](#_Toc159173032)

[INSTRUCCIÓN 10](#_Toc159173033)

[MENSAJE “TRANSFERIR” 11](#_Toc159173034)

[enviarArchivo() 12](#_Toc159173035)

[MENSAJE “DESCONECTAR” 14](#_Toc159173036)

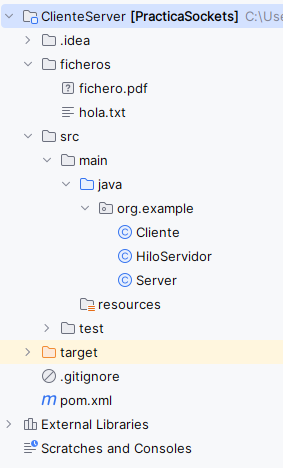
[Pruebas de funcionamiento 15](#_Toc159173037)

[Transferir .txt 15](#_Toc159173038)

[Transferir .pdf 17](#_Toc159173039)

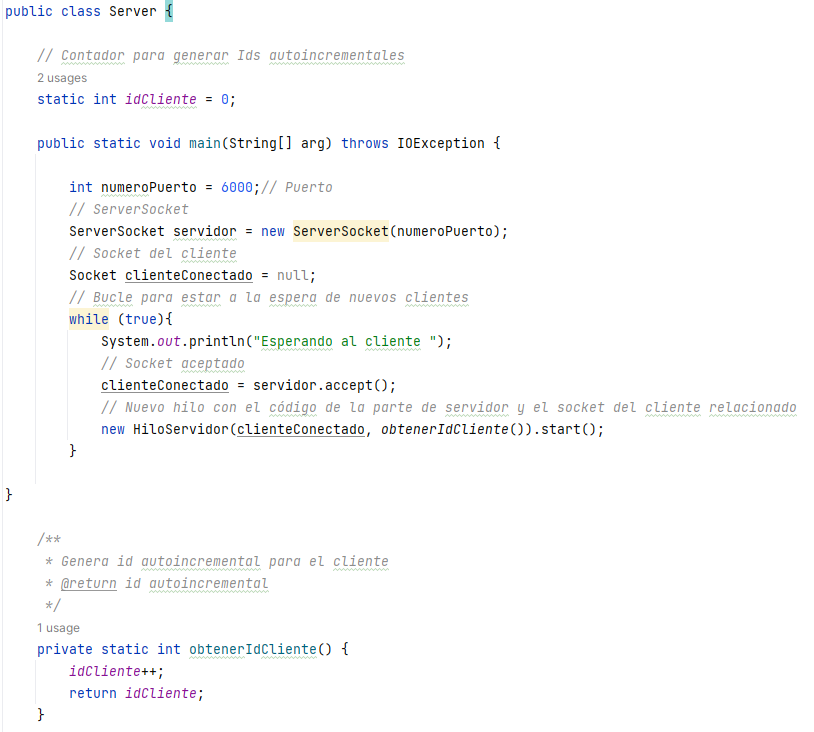
[Desconectar 20](#_Toc159173040)

# Estructura



* **Cliente**: clase con el código de comunicación de la parte cliente. Se ha activado la ejecución paralela para poder conectar varios clientes al servidor de manera concurrente.
* **Server**: clase con el código de comunicación de la parte servidor. Crea objetos HiloServidor por cada Cliente conectado.
* **HiloServidor**: subclase de Thread que contiene el núcleo de la comunicación por parte del servidor para cada cliente.

# Server

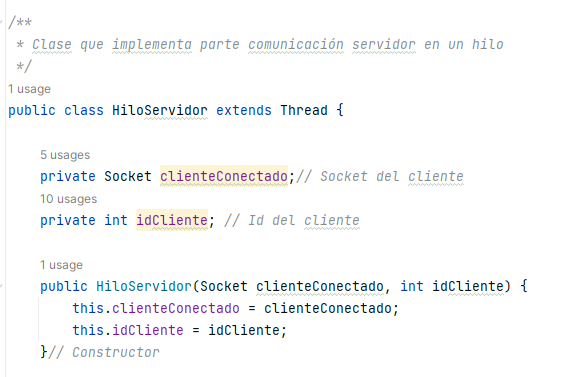


Genera un objeto ServerSocket.

Espera a conexión de cliente para crear un objeto Socket para dicho cliente, sobre dicho socket crea un objeto HiloServidor y lo lanza.

Asigna además una id autoincremental a los clientes que se van conectando.

# HiloServidor

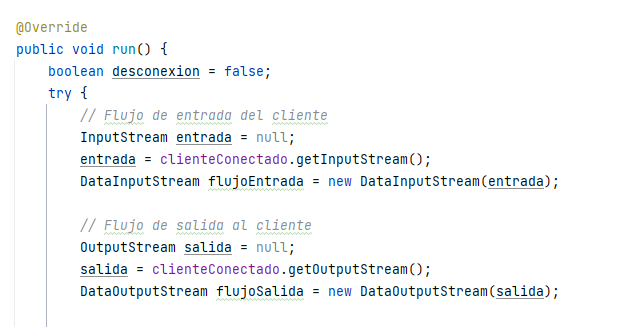


Extiende la clase Thread (tiene un método run()).

Atributos:

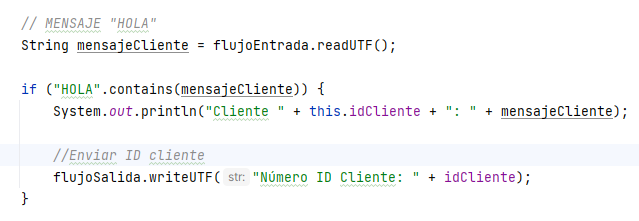
* **clienteConectado**: Socket del cliente.
* **idCliente**: int con id autoincremental del cliente.

## Run()



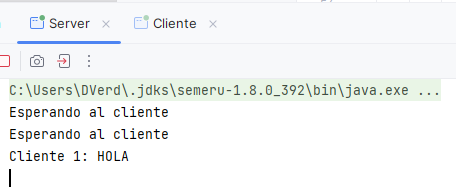
Crea los flujos de entrada y de salida necesarios para comunicarse con el cliente a raíz del objeto Socket.

### Mensaje “HOLA”

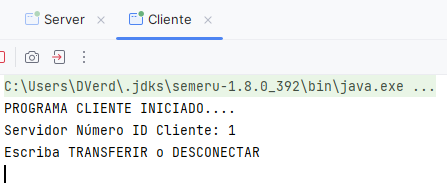


Lee saludo del cliente y en caso de coincidir con el String “HOLA” le devuelve la idCliente respectiva a dicho cliente.

Consola servidor:



Consola cliente:



### MENSAJE “TRANSFERIR”

Realiza las comprobaciones pertinente con cliente con ACKs + nombre de archivo.



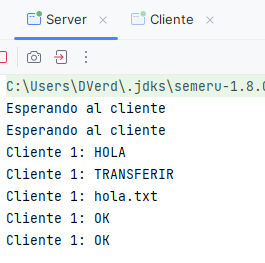


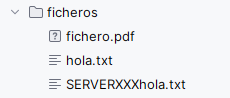
#### recibirArchivo()

En la función recibirArchivo procesa el objeto tipo File troceado por la parte cliente y lo escribe en un nuevo fichero iterando a partir de la longitud del archivo original:

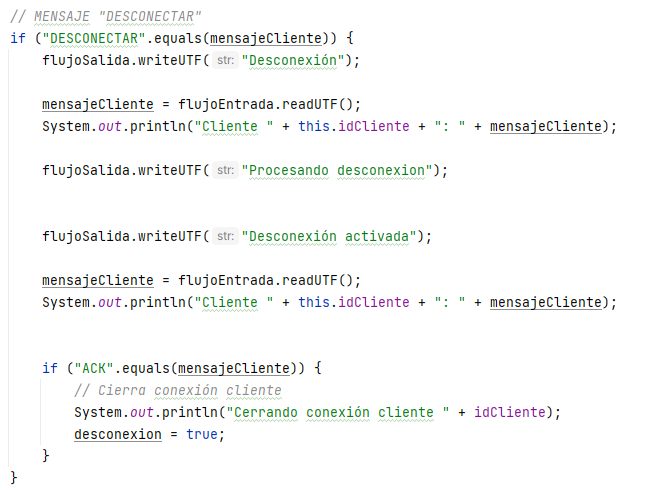


Detalle de mensaje TRANSFERIR en consola parte Server:



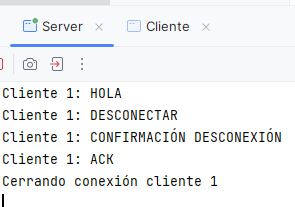


### MENSAJE “DESCONECTAR”

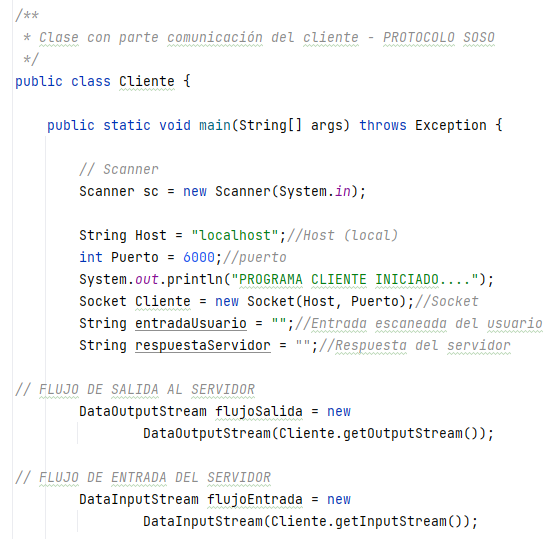


Después de las comprobaciones e intercambio de mensajes con el cliente cambia el valor del booleano *desconexion*, provocando el cierre de recursos y la finalización del método run() de la instancia de **HiloServidor** que desencadena el fin de la comunicación del cliente actual con el servidor.

Detalle de consola Server:



# Cliente



Permite ejecución paralela.

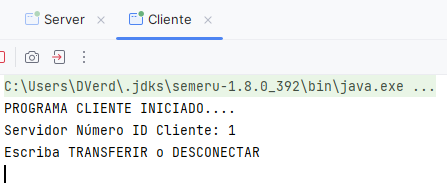
Crea objeto Socket con el puerto correspondiente al ServerSocket de la parte Server.

Crea flujos de salida y entrada al servidor.

## MENSAJE “HOLA”

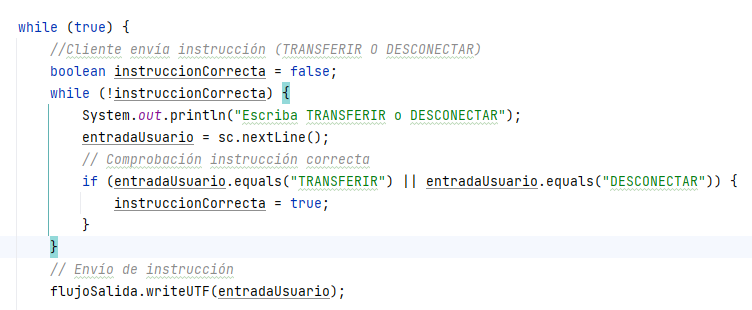


Envía “HOLA” al extremo del servidor, recibe ID otorgada por el servidor.

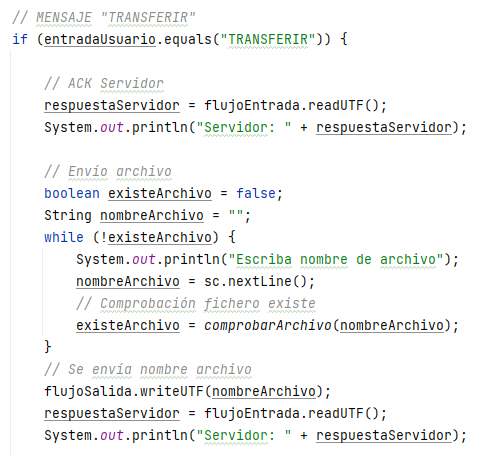


## INSTRUCCIÓN

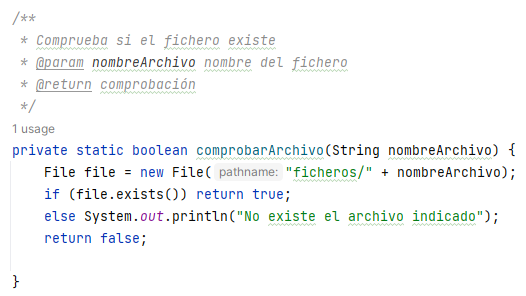
Válida que la instrucción introducida por el usuario sea correcta:



## MENSAJE “TRANSFERIR”



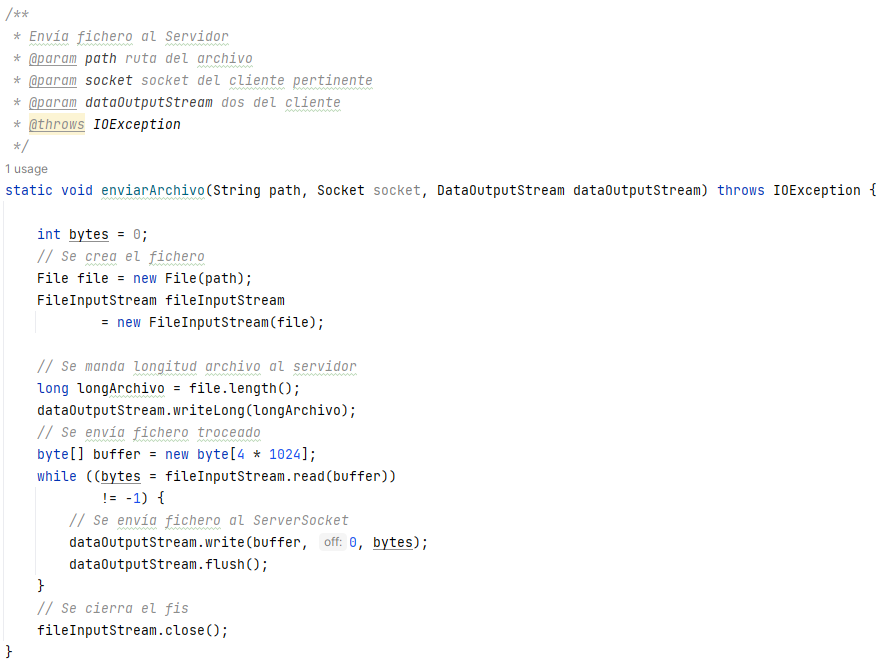
Recibe ACK del servidor y comprueba que el nombre del archivo exista mediante el método *comprobarArchivo(String nombreArchivo)* y lo envía:





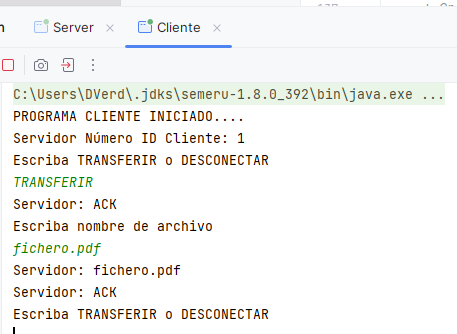
Tras sendas comprobaciones, se invoca al método enviarArchivo para proceder al envío del fichero.

### enviarArchivo()



Este método crea un objeto File, le asigna la ruta consecuente al nombre del archivo y envía la longitud de dicho fichero al servidor.

Después trocea el contenido del fichero mediante el uso de un Array de tipos byte y lo envía al servidor.



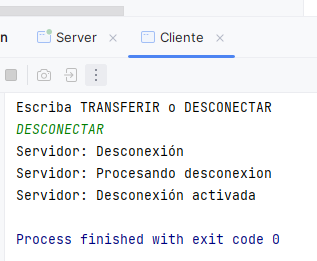
Arriba consola del Cliente tras instrucción “TRANSFERIR”.

## MENSAJE “DESCONECTAR”



Intercambia mensajes pertinentes con la parte de Servidor y mediante un break sale del bucle de instrucciones, provocando el cierre de recursos y el fin de la ejecución del cliente.

Detalle de consola de parte Cliente tras instrucción DESCONECTAR:



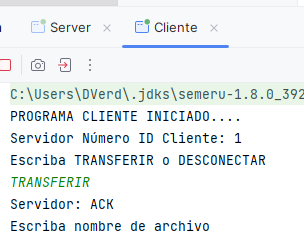
# Pruebas de funcionamiento

El uso y salida del mensaje HOLA ya se ha capturado y demostrado en la descripción de las clases del aplicativo.

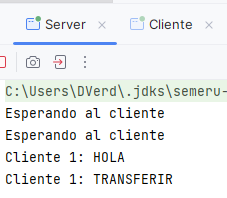
## Transferir .txt

Iniciados el Server y un cliente escribimos la instrucción “TRANSFERIR”.

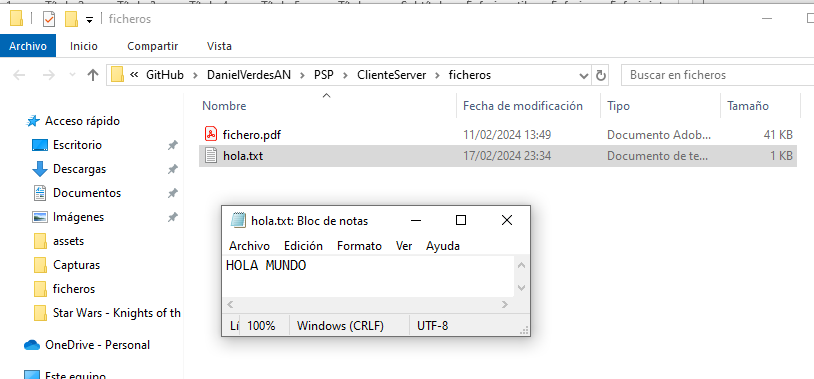
**Cliente**:



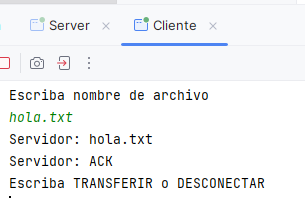
**Servidor**:



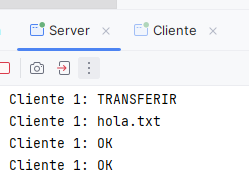
A continuación pasamos como nombre de archivo el fichero hola.txt con el siguiente contenido:



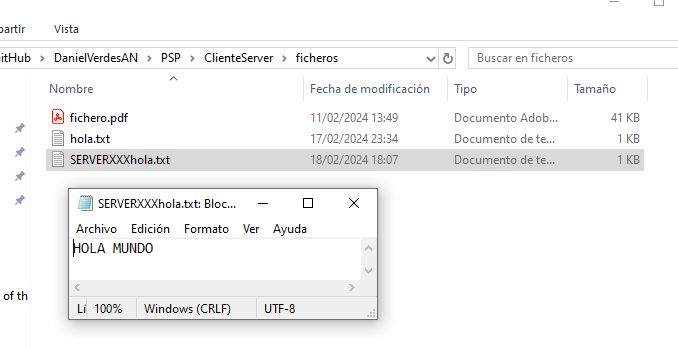
**Cliente:**

****

**Servidor:**

****

**Directorio:**

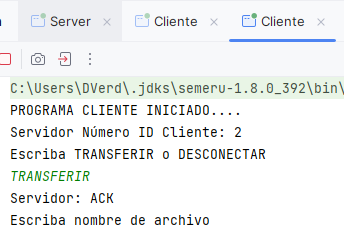
****

Se puede apreciar que se ha transferido el archivo exitosamente, guardándolo con el prefijo SERVERXXX por el servidor.

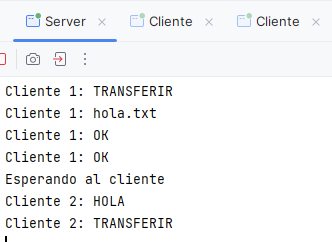
## Transferir .pdf

Sin parar las ejecuciones actuales, vamos a ejecutar otro Cliente y le vamos a pasar también la instrucción TRANSFERIR:

**Cliente:**

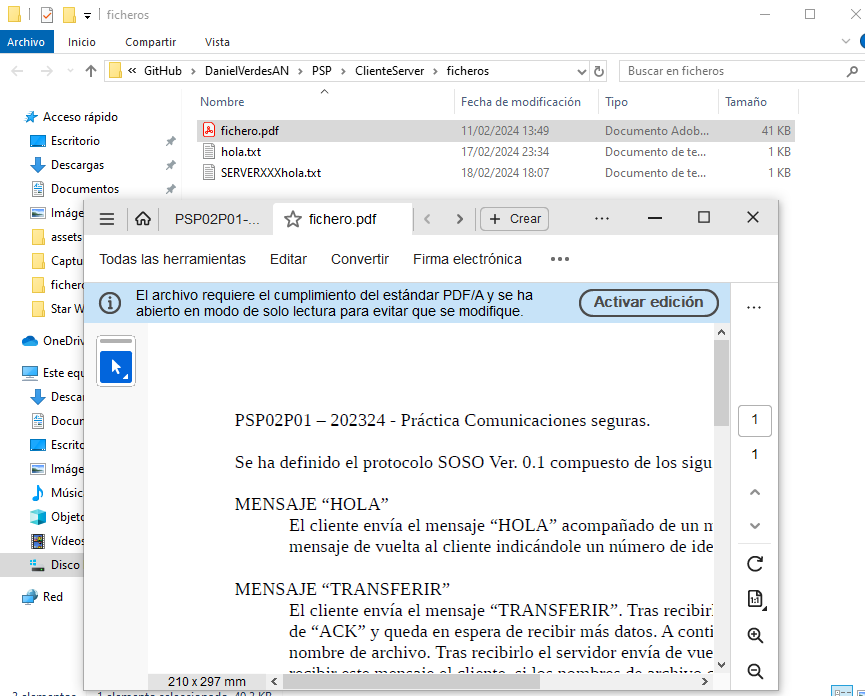
****

**Server:**

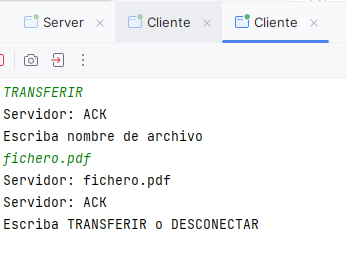
****

Nótese que a este nuevo cliente se le ha asignado una Id diferente al anterior y ambos están conectados simultáneamente al servidor.

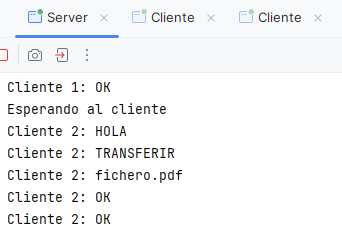
A continuación vamos a escribir el nombre de archivo fichero.pdf (que es el .pdf correspondiente a las instrucciones de la práctica actual):



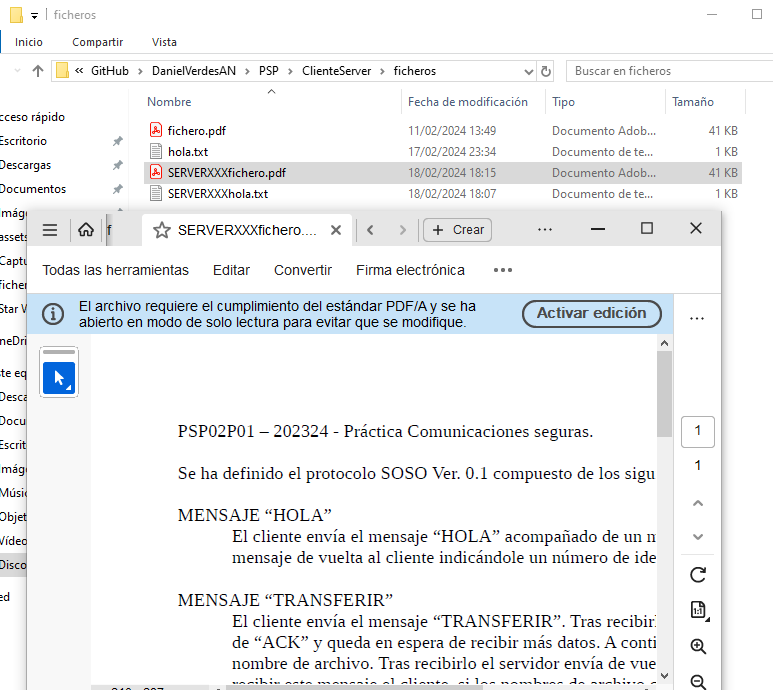
**Cliente:**

****

**Servidor:**

****

**Directorio:**

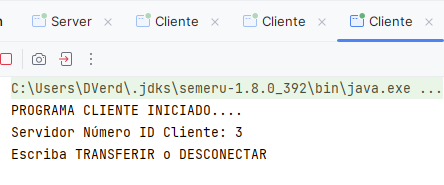
****

Se demuestra que la instrucción TRANSFERIR ha vuelto a funcionar según lo indicado, manejando otra extensión de archivo y en otro cliente en ejecución.

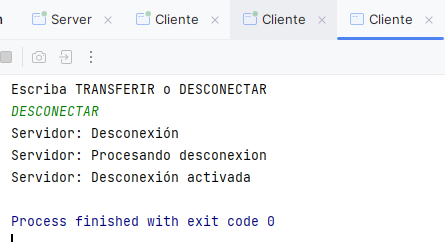
## Desconectar

Para probar la instrucción DESCONECTAR, iniciamos un tercer cliente concurrente e introducimos dicha instrucción:

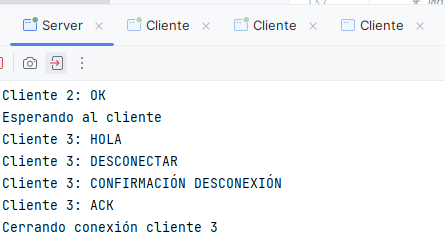




**Cliente:**

****

**Servidor:**

****

De esta manera se demuestra tanto el funcionamiento de la instrucción DESCONECTAR como la posibilidad de que varios clientes se conecten de forma concurrente al servidor.