Memoria Practica X Tema

Que

Se desea realizar un sistema de login básico.
El programa cliente al iniciarse solicitará un usuario y contraseña. Este usuario y contraseña se enviará al servidor.

El programa servidor al recibir el usuario y la contraseña el servidor comprueba si el usuario está activo. Si lo está, responderá al usuario con el mensaje, “el usuario se ha desconectado”.

En el caso de que el usuario no esté activo, comprobará el usuario y contraseña en el fichero: usuarios.txt que se encontrará en C:/sistemalogin. Este fichero tiene el formato:

```
ID#usuario#contraseña
1#pocoyo#1234
```

Si existe el usuario y contraseña, el usuario pasará al estado activo y contestará al cliente con el mensaje: “el usuario se ha conectado correctamente”.

En el caso de que el usuario y/o contraseña sean incorrectos, el servidor le enviará el mensaje “usuario y/o contraseña incorrecto”.

Para qué

Este ejercicio ha resultado útil para aprender y practicar el funcionamiento de los Sockets. La actividad sirve para tener una idea básica del funcionamiento de los Sockets con Java o C, dependiendo del lenguaje que queramos usar. Yo he decidido utilizar Java, ya que estoy más familiarizado con él.

Pseudocodigo

Servidor

```
Inicio del programa Servidor:
    Definir PUERTO como 5432
    Crear HashMap activos para almacenar el estado de los usuarios

    Intentar:
        Crear ServerSocket con PUERTO
        Imprimir "Servidor esperando conexiones..."

        Mientras (true):
            Aceptar conexión del cliente usando accept()
```

```
            Imprimir "Cliente accedió desde: " + dirección IP del cliente

            Iniciar un nuevo hilo para manejar la conexión del cliente
usando manejarConexion()

    Capturar IOException:
        Imprimir traza de pila del error

Fin del programa Servidor

Función manejarConexion(socketCliente):
    Intentar:
        Crear BufferedReader para entrada desde el socket
        Crear PrintWriter para salida hacia el socket

        Leer usuario desde entrada
        Leer contraseña desde entrada

        Si usuario está activo:
            Enviar "El usuario se ha desconectado" al cliente
            Marcar usuario como inactivo en el HashMap

        Sino:
            Si verificarCredenciales(usuario, contraseña):
                Enviar "El usuario se ha conectado correctamente" al
cliente
                Marcar usuario como activo en el HashMap
            Sino:
                Enviar "Usuario y/o contraseña incorrecto" al cliente

        Cerrar socketCliente

    Capturar IOException:
        Imprimir traza de pila del error

Función usuarioEstaActivo(usuario):
    Devolver el valor asociado a usuario en el HashMap activos (o false si
no existe)

Función verificarCredenciales(usuario, contraseña):
    Definir rutaArchivo como "res/usuarios.txt"

    Intentar:
        Crear BufferedReader para leer desde rutaArchivo

        Mientras haya líneas en el archivo:
            Leer línea
            Dividir la línea en campos usando "#"

            Si hay 3 campos y el segundo campo es igual a usuario y el
tercer campo es igual a contraseña:
                Devolver true

    Capturar IOException:
```

```
        Imprimir traza de pila del error

    Devolver false
```

Cliente

```
Inicio del programa Cliente:
    Definir SERVIDOR_IP como "localhost"
    Definir PUERTO como 5432

    Intentar:
        Crear Socket con SERVIDOR_IP y PUERTO
        Crear BufferedReader para entrada desde el socket
        Crear PrintWriter para salida hacia el socket
        Crear BufferedReader para entrada desde el teclado

        Imprimir "Ingrese usuario: "
        Leer usuario desde teclado
        Imprimir "Ingrese contraseña: "
        Leer contraseña desde teclado

        Enviar usuario al servidor usando PrintWriter
        Enviar contraseña al servidor usando PrintWriter

        Leer respuesta del servidor desde BufferedReader
        Imprimir "Respuesta del servidor: " + respuesta del servidor

    Capturar IOException:
        Imprimir traza de pila del error

Fin del programa Cliente
```

Cómo

Clase Cliente

Esta clase solo contiene la función principal. En ella, he definido dos variables de configuración: el puerto y la IP, donde se conecta el cliente con el socket. Luego, creo el socket con esas dos variables para establecer una conexión con el servidor a través de la IP y el puerto. Luego, creo un `BufferedReader` con un `InputStreamReader` para convertir el flujo de bytes del socket a caracteres, pasándole como parámetro una función del socket, `getInputStream`, que es el canal por el cual los datos son enviados desde el servidor. También creo un `PrintWriter` que hace lo mismo en el `BufferedReader`, pero en vez de recibir, envía los datos al servidor. Y creo un último `BufferedReader` para pedir los datos al usuario. Después, envío los datos al servidor utilizando el `PrintWriter`, recibo la respuesta del servidor y la muestro por pantalla.

Clase Servidor

En esta clase, hay un `HashMap` donde se guarda el estado de los usuarios y tres funciones excepcionales a la función principal.

- `verificarCredenciales()`: Esta función crea un `BufferedReader` del archivo de loggings y con un bucle while comprueba si los datos del `SocketCliente`, el usuario y la contraseña que son los parámetros de entrada de la función, coinciden con alguno de los datos del archivo. Si es así, devuelve true; si no, false.
- `usuarioEstaActivo()`: Esta función retorna el estado del usuario en el `HashMap`, si es la primera vez que el usuario se conecta, lo establece como false.
- `manejarConexion()`: Esta función crea un `BufferedReader` que lee los datos del cliente. Este objeto está conectado al flujo de entrada del socket del cliente y crea un objeto `PrintWriter` para enviar datos al cliente, que también está conectado al flujo de salida del cliente. Luego, recibe los datos que ha enviado el cliente. Luego entra en un if comprobando si el `usuarioEstaActivo`. Si lo está, cambiará su estado a false y le enviará al cliente que está desconectado. Si no, comprobará las credenciales e informará si se ha conectado correctamente o la contraseña era incorrecta. Posteriormente, cerrará el `SocketCliente` y, por consecuencia, el programa Cliente acabará.

En la función principal, se creará un `ServerSocket` que estará a la espera de conexiones y luego entra en un bucle infinito para esperar nuevas conexiones. Cuando un cliente se conecta, se acepta la conexión con la función `accept()`, que devuelve un socket que representa la conexión entre el servidor y el cliente. Finalmente, crea un hilo que llama a la función `manejarConexion()`. Sé que realmente no hace falta ya que el ejercicio solo pide que se pueda conectar un cliente a la vez, pero he decidido implementarlo porque me ha parecido lo más conveniente. Además, ya que ahora sé cómo funcionan los hilos, ¿por qué no utilizarlos?

Conclusión

En conclusión, considero que esta actividad ha sido interesante y ha contribuido significativamente a mejorar mis habilidades de programación con Sockets. También he aprendido mejor cómo usar el `BufferedReader` y el `PrintWriter`, ya que nunca los había utilizado, y es otra forma de hacer lo mismo que un `Scanner`. Creo que está bien saber hacer lo mismo pero con varias formas distintas.
Además, me intriga conocer las diferentes aproximaciones que mis compañeros han tomado para abordar esta actividad, ya que soy consciente de que hay varias formas de implementarla. Consultar a mis amigos y obtener explicaciones sobre sus enfoques podría proporcionarme valiosas perspectivas y aprender nuevas técnicas. También ver cómo se hace esta actividad en C para poder comprender un poco más cómo funciona C.

En general, me ha parecido muy interesante el hecho de que dos programas diferentes puedan compartir información de esa manera. Considero que la dificultad de esta práctica no ha sido elevada, pero tampoco ha sido fácil.