FTP Pot

Ejercicio N°3

Objetivos	● Diseño y construcción de sistemas con acceso distribuido ● Encapsulación de Threads y Sockets en Clases ● Definición de protocolos de comunicación ● Protección de los recursos compartidos ● Uso de buenas prácticas de programación en C++
Instancias de Entrega	**Entrega 1:** clase 7 (1/10/2019). **Entrega 2:** clase 9 (15/10/2019).
Temas de Repaso	● Definición de clases en C++ ● Contenedores de STL ● Excepciones / RAII ● Move Semantics ● Sockets ● Threads
Criterios de Evaluación	● Criterios de ejercicios anteriores ● Eficiencia del protocolo de comunicaciones definido ● Control de paquetes completos en el envío y recepción por Sockets ● Atención de varios clientes de forma simultánea ● Eliminación de clientes desconectados de forma controlada

Índice

FTP Pot

Introducción

Un "honeypot" es una aplicación que simula ser un servidor de otra aplicación mayor, para que cuando un usuario malicioso se conecte, ataque este servidor falso, permitiendo tomar registro de las técnicas de ataque que se utilizan en la red.

En este trabajo práctico prototiparemos un honeypot de un servidor FTP.

Protocolo FTP

El protocolo FTP es un **protocolo de texto**, formado por mensajes delimitados por un **salto de línea**.

Los comandos son palabras en su mayoría de 4 letras, seguidos de un espacio que los separa de sus argumentos. El servidor responde con un código numérico y un texto descriptivo.

Cliente FTP

Los comandos que puede ejecutar el cliente son los siguientes

- `USER <username>`: Envía un nombre de usuario para realizar login
- `PASS <password>`: Envía un password para el usuario
- `SYST`: Consulta información del sistema
- `LIST`: Consulta los archivos contenidos en el directorio actual
- `HELP`: Consulta los comandos disponibles
- `PWD`: Consulta el directorio actual
- `MKD`: Crea un directorio

Para simplificar el trabajo práctico, la transferencia de datos se realizará respondiendo en el mismo puerto por el cual se conectan inicialmente los clientes.

Esto implica que los comandos `PASV` y `PORT` no serán necesarios para transmitir información (por ejemplo con el comando `LIST`)

Servidor FTP

El servidor FTP responde con los siguientes mensajes:

- Cuando un cliente recién se conecta: `220 <newClient>`
- Cuando un cliente quiere operar sin haber hecho login: `530 <clientNotLogged>`
- Luego de que un cliente envía un comando `USER`: `331 <passRequired>`
- Si el usuario envía un comando que no es `PASS` luego de un comando `USER`: `530 <clientNotLogged>`
- Si el usuario realiza un login válido (el usuario enviado con `USER` y la contraseña enviada con `PASS` son válidas, es decir, concuerdan con las configuradas): `230 <loginSuccess>`
- Si el usuario realiza un login inválido: `530 <loginFailed>`

Una vez que el cliente realizó un login exitoso, se habilitan varios comandos al cliente:

- Si el usuario envía el comando `SYST`: `215 <systemInfo>`
- Si el usuario envía el comando `HELP`: `214 <commands>`
- Si el usuario envía el comando `LIST`, se realizará una respuesta en 3 partes:

La primer linea será `150 <listBegin>`

Luego enviará una línea por cada directorio cargado, con el siguiente formato:

```
drwxrwxrwx 0 1000 1000 4096 Sep 24 12:34 <nombre directorio>
```

Los directorios serán listados en orden alfabético.

Luego de enviar las líneas, enviará `226 <listEnd>`.

- Si el usuario envía el comando `PWD`: `257 <currentDirectoryMsg>`
- Si el usuario envía el comando `MKD <nombreDir>`, intenta agregar "`<nombreDir>`" a la lista de directorios existentes responde:

`257 "<nombreDir> <mkdSuccess>"` en caso de que el directorio no existía.

`550 <mkdFailed>` si ya existía

- Si el usuario envía el comando `RMD <nombreDir>` intenta remover "`<nombreDir>`" de la lista de directorios y responde:

250 `<rmdSuccess>` si el directorio existe.

550 `<rmdFailed>` si no existía.

- Finalmente, el usuario puede pedirle al servidor que termine la conexión enviando la palabra QUIT. En tal caso el servidor responde con 221 `<quitSuccess>` y ambos cierran sus conexiones ordenadamente.

Cabe destacar que la entrada de comandos puede no terminar con QUIT, debiendo el cliente cerrar su conexión al llegar al final del stream y el servidor liberar sus recursos adecuadamente.

Configuración de mensajes

El servidor FTP lee un archivo de configuración las siguientes variables

- `user`: Usuario para realizar login
- `password`: Password para realizar login
- `newClient`: Mensaje enviado a un cliente recién conectado
- `clientNotLogged`: Mensaje enviado a un cliente que quiere operar sin haber hecho login
- `passRequired`: Mensaje de solicitud de password
- `loginSuccess`: Mensaje de password aceptado
- `loginFailed`: Mensaje de password rechazado
- `systemInfo`: Mensaje de información del sistema
- `unknownCommand`: Mensaje de comando inválido
- `quit`: Mensaje de desconexión del usuario

El archivo de configuración posee un formato "`clave=valor`", y debe ser cargado al iniciar la aplicación. No se validará que se encuentren todas las claves necesarias, en caso de faltar una clave necesaria, es a discreción del desarrollador como contemplar este caso.

Formato de Línea de Comandos

Servidor

```
./server <puerto/servicio> <configuracion>
```

Donde `<puerto/servicio>` es el puerto TCP (o servicio) en donde estará escuchando la conexiones entrantes, `<configuracion>` el archivo con las variables del servidor.

Cliente

El cliente se ejecuta utilizando el siguiente formato de línea de comandos

```
./client <ip/hostname> <puerto/servicio>
```

El cliente se conectará al servidor corriendo en la máquina con dirección IP `<ip>` (o `<hostname>`), en el puerto (o servicio) TCP `<puerto/servicio>`.

Códigos de Retorno

El servidor y el cliente devolverán siempre 0

Entrada y Salida Estándar

Servidor

El servidor no imprimirá nada por salida estándar.
En cambio, esperará el caracter `'q'` por entrada estándar. Cuando lo reciba, el servidor deberá finalizar todas las conexiones y cerrarse.

Cliente

El cliente leerá los comandos por entrada estándar e imprimirá las respuestas recibidas por el cliente en salida estándar

Ejemplo de Ejecución

Dado el archivo de configuración `taller.cfg` con el siguiente contenido:

```
newClient=TallerFTP
user=admin
password=tomenague
passRequired=Please specify the password.
loginFailed=Login incorrect.
loginSuccess=Login successful.
clientNotLogged=Please login with USER and PASS.
currentDirectoryMsg=”/home/admin/ftp” is the current directory
listBegin=Here comes the directory listing.
listEnd=Directory send OK.
mkdSuccess=created
mkdFailed=Create directory operation failed
rmdSuccess=Remove directory operation successful.
```

```
rmdFailed=Remove directory operation failed.
quitSuccess=Goodbye
```

Ejecutamos el servidor

```
./server 8021 taller.cfg
```

Y un cliente

```
./client localhost 8021
```

El cliente recibe el mensaje
`230 TallerFTP`

El cliente decide iniciar un proceso de login:
`USER admin`

El servidor le solicita la contraseña
`331 Please specify the password.`

El cliente le envía un password inválido
`PASS alvlgeddl`

Dado que la contraseña no coincide con la almacenada en el archivo de configuración, devuelvo contraseña invalida
`530 Login incorrect.`

Intento ejecutar un comando sin haber hecho login
`PWD`

El servidor me responde que no inicié sesión
`530 Please login with USER and PASS.`

Intento nuevamente hacer login
`USER admin`

El servidor le solicita nuevamente la contraseña
`331 Please specify the password.`

El cliente le envía un password válido
`PASS tomenague`

El servidor le responde con éxito
```
230 Login successful.
```

El cliente solicita ver la ruta del directorio actual
```
PWD
```

El servidor le responde con éxito
```
257 "/home/admin/ftp" is the current directory.
```

El cliente envia un comando invalido
```
INVALIDCOMMAND
```

El servidor le responde con un error
```
500 Unknown command.
```

El cliente solicita listar los directorios disponibles
```
LIST
```

El servidor le envía una lista vacía de directorios
```
150 Here comes the directory listing.
226 Directory send OK.
```

El cliente solicita crear un directorio "foo"
```
MKD foo
```

El servidor le responde con éxito (la carpeta no se crea físicamente, sólo encolamos su nombre en memoria)
```
257 "foo" created
```

El cliente solicita crear un directorio "bar"
```
MKD bar
```

El servidor le responde con éxito
```
257 "bar" created
```

El cliente solicita crear nuevamente a "foo"
```
MKD foo
```

El servidor le responde con un error
```
550 Create directory operation failed.
```

El cliente solicita listar los directorios disponibles

```
LIST
```

El servidor le envía una lista de los directorios ordenados alfabéticamente
```
150 Here comes the directory listing.
drwxrwxrwx 0 1000 1000 4096 Sep 24 12:34 bar
drwxrwxrwx 0 1000 1000 4096 Sep 24 12:34 foo
226 Directory send OK.
```

El cliente intenta eliminar a bar
```
RMD bar
```

El servidor le responde con éxito
```
250 Remove directory operation successful.
```

El cliente intenta eliminar nuevamente a bar
```
RMD bar
```

El servidor le responde con un error
```
550 Remove directory operation failed.
```

El cliente solicita listar los directorios disponibles
```
LIST
```

El servidor le envía una lista de los directorios que quedaron
```
150 Here comes the directory listing.
drwxrwxrwx 0 1000 1000 4096 Sep 24 12:34 foo
226 Directory send OK.
```

El cliente solicita desconectarse
```
QUIT
```

El servidor se despide agraciadamente
```
221 Goodbye.
```

Restricciones

La siguiente es una lista de restricciones técnicas exigidas por el cliente:

1. El sistema debe desarrollarse en ISO C++11.
2. Está prohibido el uso de variables globales.
3. La búsqueda que el servidor hace para buscar directorios se debe realizar en un tiempo menor que O(N).

4. Debe haber uso de polimorfismo al ejecutar comandos. Se recomienda el uso del patrón “Factory method”
5. No se puede asumir que los mensajes tienen un largo máximo.

Referencias

[1]: http://www.cplusplus.com/reference/set/set/
[2]: https://sourcemaking.com/design_patterns/factory_method/cpp/1