Aplicación Servidor

Protocolo HFTP

Llamaremos *Home-made File Transfer Protocol* (HFTP) a un protocolo de transferencia de archivos casero, creado por nosotros específicamente para este laboratorio.

HFTP es un protocolo de capa de aplicación que usa TCP como protocolo de transporte. TCP garantiza una entrega segura, libre de errores y en orden de todas las transacciones hechas con HFTP. Un servidor de HFTP escucha pedidos en el puerto TCP 19500.

Comandos y Respuestas

El cliente HFTP inicia el intercambio de mensajes mediante pedidos o **comandos** al servidor. El servidor envía una **respuesta** a cada uno antes de procesar el siguiente hasta que el cliente envía un comando de fin de conexión. En caso de que el cliente envíe varios pedidos consecutivos, el servidor HFTP los responde en el orden en que se enviaron. El protocolo HFTP es un protocolo ASCII, no binario, por lo que todo lo enviado (incluso archivos binarios) será legible por humanos como strings.

- Comandos**:** consisten en una cadena de caracteres compuesta por elementos separados por un único espacio y terminadas con un fin de línea estilo DOS (`\r\n`). El primer elemento del comando define el tipo de acción esperada por el comando y los elementos que siguen son argumentos necesarios para realizar la acción.
- Respuestas**:** comienzan con una cadena terminada en `\r\n`, y pueden tener una continuación dependiendo el comando que las origina. La cadena inicial comienza con una secuencia de dígitos (código de respuesta), seguida de un espacio, seguido de un texto describiendo el resultado de la operación. Por ejemplo, una cadena indicando un resultado exitoso tiene código `0` y con su texto descriptivo podría ser `0 OK`.

Comandos	Descripción y Respuesta

`get_file_listing`	Este comando no recibe argumentos y busca obtener la lista de archivos que están actualmente disponibles. El servidor responde con una secuencia de líneas terminadas en `\r\n`, cada una con el nombre de uno de los archivos disponibles. Una línea sin texto indica el fin de la lista.
	
	`Comando: get_file_listing`
	`Respuesta: 0 OK\r\n`
	`archivo1.txt\r\n`
	`archivo2.jpg\r\n`
	`\r\n`

`get_metadata FILENAME`	Este comando recibe un argumento `FILENAME` especificando un nombre de archivo del cual se pretende averiguar el tamaño. El servidor responde con una cadena indicando su valor en bytes.
	
	`Comando: get_metadata archivo.txt`
	`Respuesta: 0 OK\r\n`
	`3199\r\n`

`get_slice FILENAME OFFSET SIZE`	Este comando recibe en el argumento `FILENAME` el nombre de archivo del que se pretende obtener un *slice* o parte. La parte se especifica con un `OFFSET` (byte de inicio) y un `SIZE` (tamaño de la parte esperada, en bytes), ambos no negativos. El servidor responde con el fragmento de archivo pedido codificado en base64 y un `\r\n`.
	
	`Byte: 0 5 10 15 20 25 30 35 40`
	`v v v v v v v v v`
	`Archivo: !Que calor que hace hoy, pinta una birra!`
	`Comando: get_slice archivo.txt 5 20`
	`Respuesta: 0 OK\r\n`
	`Y2Fsb3IgcXVlIGhhY2UgaG95LCA=\r\n`

`quit`	Este comando no recibe argumentos y busca terminar la conexión. El servidor responde con un resultado exitoso (`0 OK`) y luego cierra la conexión.

Manejo de Errores

En caso de algún error, el servidor responderá con códigos de respuestas diferentes a `0`, más algún texto descriptivo a definir por el implementador. En particular:

- `0` La operación se realizó con éxito.
- `100` Se encontró un carácter `\n` fuera de un terminador de pedido `\r\n`.
- `101` Alguna malformación del pedido impidió procesarlo.
- `199` El servidor tuvo algún fallo interno al intentar procesar el pedido.
- `200` El comando no está en la lista de comandos aceptados.
- `201` La cantidad de argumentos no corresponde o no tienen la forma correcta.
- `202` El pedido se refiere a un archivo inexistente.
- `203` El pedido se refiere a una posición inexistente en un archivo.

Los errores con código iniciado en `1` son considerados fatales y derivan en el cierre de la conexión una vez reportados por el servidor. Los errores que inician con `2` permiten continuar con la conexión y recibir pedidos posteriores.

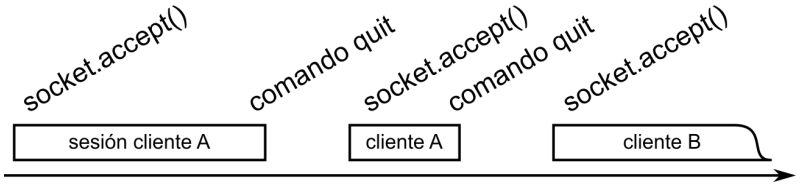
Tarea

Se debe diseñar e implementar un servidor de archivos en Python 3 que soporte **completamente** un protocolo de transferencia de archivos HFTP. El servidor debe ser robusto y tolerar comandos intencional o maliciosamente incorrectos.

1. El kickstarter provee una estructura para el servidor que se deberá completar (`server.py` y `connection.py`), un archivo con las constantes a utilizar (`constants.py`), el cliente HFTP funcionando y un archivo de testeo server-tets.py para correr junto con el servidor.
2. Armar un entorno virtual de python con python 3.6 según esta nota
3. Ejecutar el laboratorio como está
4. Modificar el archivo server para que acepte conexiones y con esa conexion cree un objeto connection, testearlo con telnet
5. Implementar los distintos comandos empezando por el quit, después testear cada comando con telnet. (usar el archivo client.py para sacar ideas de cómo manejar las conexiones)

6. Una vez implementados los comandos , probar el funcionamiento con el cliente que se le entrega en el kickstarter (client.py)
7. Una vez que funcione el cliente ejecutar el test para probar los casos “no felices”
8. Implementar múltiples clientes utilizando hilos
9. (Punto estrella) Implementar múltiples clientes con poll (https://stackoverflow.com/questions/27494629/how-can-i-use-poll-to-accept-multiple-clients-tcp-server-c https://betterprogramming.pub/how-to-poll-sockets-using-python-3e1af3b047)

El cliente y el servidor a desarrollar podrán estar corriendo en máquinas distintas (sobre la misma red) y el servidor será capaz de manejar varias conexiones a la vez.



A continuación se muestra un ejemplo de ejecución del servidor atendiendo a un único cliente.

```
caro@victoria:~/Ayudantia/Redes$ python server.py
Running File Server on port 19500.
Connected by: ('127.0.0.1', 44639)
Request: get_file_listing
Request: get_metadata client.py
Request: get_slice client.py 0 1868
Closing connection...
```

El servidor debe aceptar en la línea de comandos las opciones:

- `-d` directory para indicarle donde están los archivos que va a publicar.
- `-p` port para indicarle en que puerto escuchar. Si se omite usará el valor por defecto.
- Deben utilizar el comando `telnet <dir IP> <num Port>` para enviar comandos mal formados o mal intencionados y probar la robustez del servidor.