ITBA

Trabajo Práctico Especial:

Diseño e Implementación de “socks5d” un proxy SOCKS versión 5 con esteroides.

[72.07] - Protocolos de Comunicación

Grupo 01

2022 1Q

Fecha de entrega: 21/06/2022

Docentes:

- Garberoglio, Marcelo Fabio (Titular)
- Codagnone, Juan Francisco (J.T.P.)
- Kulesz, Sebastian (ATP)

Integrantes:

- Bartellini Huapalla, Mateo F. (61438).
- Brave, Jerónimo (61053).
- García Matwieiszyn, Juan I (61441).
- Colonnello, Joaquin (59574).

Índice

Descripción de los protocolos y aplicaciones desarrolladas.

Se diseñó y desarrolló un protocolo aplicado a un servidor no bloqueante que atiende IPv4 y IPv6 de manera multiplexada.

El mismo es una versión “con esteroides” de socks5 [RFC 1929] que permite autenticar por usuario y contraseña. Los datos se transmiten por medio del proxy de manera transparente, pero internamente se pueden sniffear y obtener datos, por ejemplo, usuario y contraseña de conexiones con servidores pop3.

También se pueden consultar datos históricos, como cantidad de conexiones, bytes transferidos y más. Para esto se puede utilizar el protocolo de monitoreo. El mismo se encuentra en el repositorio git, bajo el nombre Protocolo de monitoreo. Ahí esta explicado el mismo mucho más en detalle.

Problemas encontrados durante el diseño y la implementación.

Los problemas encontrados fueron varios, como por ejemplo comprender el protocolo y las herramientas provistas (selector.c y stm.c) y dieñar el protocolo de monitoreo y cómo parsearlo, ya que el mismo es de texto. Para esto se decidió utilizar re2c. Muchos otros problemas surgieron de implementaciones en principio erróneas que nos llevaron a muchas horas de debug para poder comprender los problemas que surgían.

Limitaciones de la aplicación y Posibles extensiones.

La aplicación cuenta con algunas limitaciones. La primera y más clara es que debido al uso de la syscall pselect(), sólo se pueden monitorear como máximo 1024 file descriptors, con lo que se tendrán, como máximo 510 clientes (sin contar clientes del protocolo de monitoreo).

Otra limitación a considerar es que debido a nuestra forma de parsear los request de socks, necesitamos como mínimo un buffer de 513 bytes, y por ende no está permitido setear un tamaño de buffer inferior al mismo. También decidimos limitar el tamaño máximo de buffer a 10240 bytes, para no sobrecargar la memoria del sistema (y porque las diferencias de tiempos a partir de el tamaño de buffer estándar de 2048 bytes dejaron de ser sustanciales).

Conclusiones.

El trabajo resultó ser un desafío extraordinario que puso a prueba nuestro entendimiento de las implementaciones de los diversos protocolos de comunicación. En fin nos encontramos con un proxy que puede ser utilizado para una variedad de implementaciones y es lo suficientemente flexible para modificar sus parámetros sobre la marcha y asegurar transparencia para el usuario.

Ejemplos de prueba.

Las primeras pruebas que se llevaron a cabo fueron pruebas básicas de funcionamiento, es decir curls con y sin resolución de nombres y ncat. Una vez que eso anduvo de forma correcta se realizaron 3 tipos de pruebas. La primera consistió en pasar con netcat y utilizando el proxy un archivo grande (aproximadamente 3,5GB). Para hacer esta prueba se fue alterando también el tamaño del buffer de socks.

Sin proxy:

Buffer de 513 bytes:

Buffer de 1024 bytes:

```
Downloads > time cat ubuntu-22.04-desktop-amd64.iso|ncat --proxy-type socks5 --proxy localhost localhost 9999
cat ubuntu-22.04-desktop-amd64.iso  0,03s user 1,83s system 16% cpu 11,624 total
ncat --proxy-type socks5 --proxy localhost localhost 9999  1,08s user 2,17s system 27% cpu 11,639 total
Downloads > cat ubuntu-22.04-desktop-amd64.iso|sha256sum
b85286d9855f549ed9895763519f6a295a7698fb9c5c5345811b3eefadfb6f07  -
Downloads > _

Downloads > ncat -l localhost 9999 > copy.iso
Downloads > cat copy.iso|sha256sum
b85286d9855f549ed9895763519f6a295a7698fb9c5c5345811b3eefadfb6f07  -
Downloads > _
```

Buffer de 2048 bytes (tamaño default):

```
Downloads > time cat ubuntu-22.04-desktop-amd64.iso|ncat --proxy-type socks5 --proxy localhost localhost 9999
cat ubuntu-22.04-desktop-amd64.iso  0,03s user 1,86s system 25% cpu 7,312 total
ncat --proxy-type socks5 --proxy localhost localhost 9999  1,15s user 2,24s system 46% cpu 7,327 total
Downloads > cat ubuntu-22.04-desktop-amd64.iso|sha256sum
b85286d9855f549ed9895763519f6a295a7698fb9c5c5345811b3eefadfb6f07  -
Downloads > _

Downloads > ncat -l localhost 9999 > copy.iso
Downloads > cat copy.iso|sha256sum
b85286d9855f549ed9895763519f6a295a7698fb9c5c5345811b3eefadfb6f07  -
Downloads > _
```

Buffer de 4096:

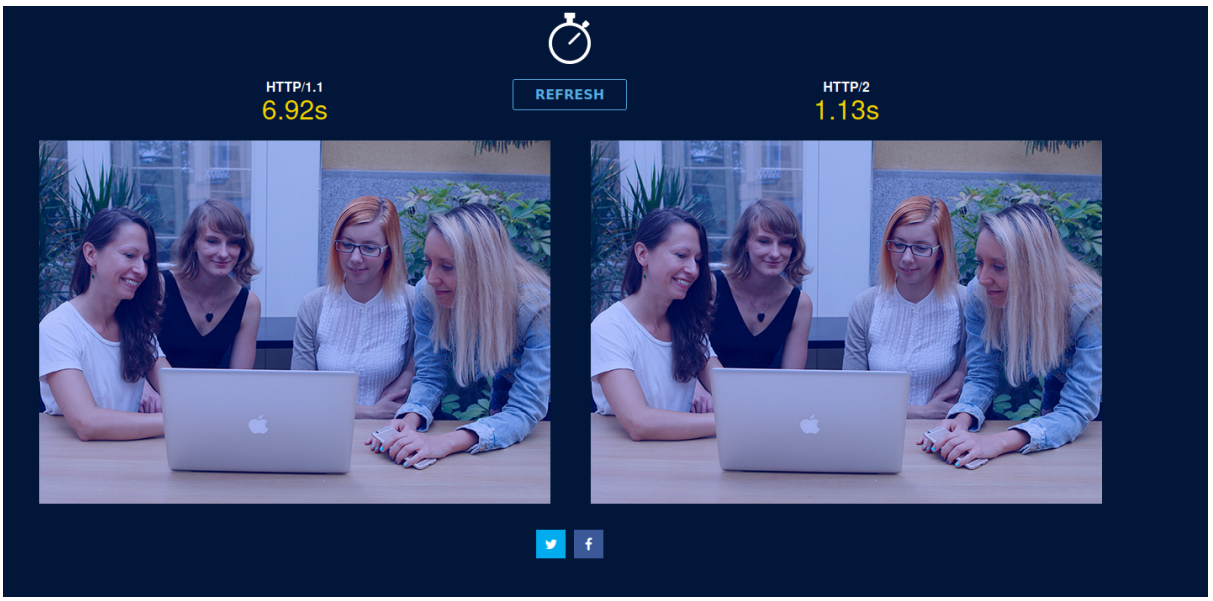
```
Downloads > time cat ubuntu-22.04-desktop-amd64.iso|ncat --proxy-type socks5 --proxy localhost localhost 9999
cat ubuntu-22.04-desktop-amd64.iso  0,02s user 1,64s system 26% cpu 6,185 total
ncat --proxy-type socks5 --proxy localhost localhost 9999  0,99s user 1,96s system 47% cpu 6,194 total
Downloads > cat unbuntu-22.04-desktop-amd64.iso|sha256sum
cat: unbuntu-22.04-desktop-amd64.iso: No such file or directory
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855  -
Downloads > cat ubuntu-22.04-desktop-amd64.iso|sha256sum
b85286d9855f549ed9895763519f6a295a7698fb9c5c5345811b3eefadfb6f07  -
Downloads > _

Downloads > ncat -l localhost 9999 > copy.iso
Downloads > cat copy.iso|sha256sum
b85286d9855f549ed9895763519f6a295a7698fb9c5c5345811b3eefadfb6f07  -
Downloads > _
```

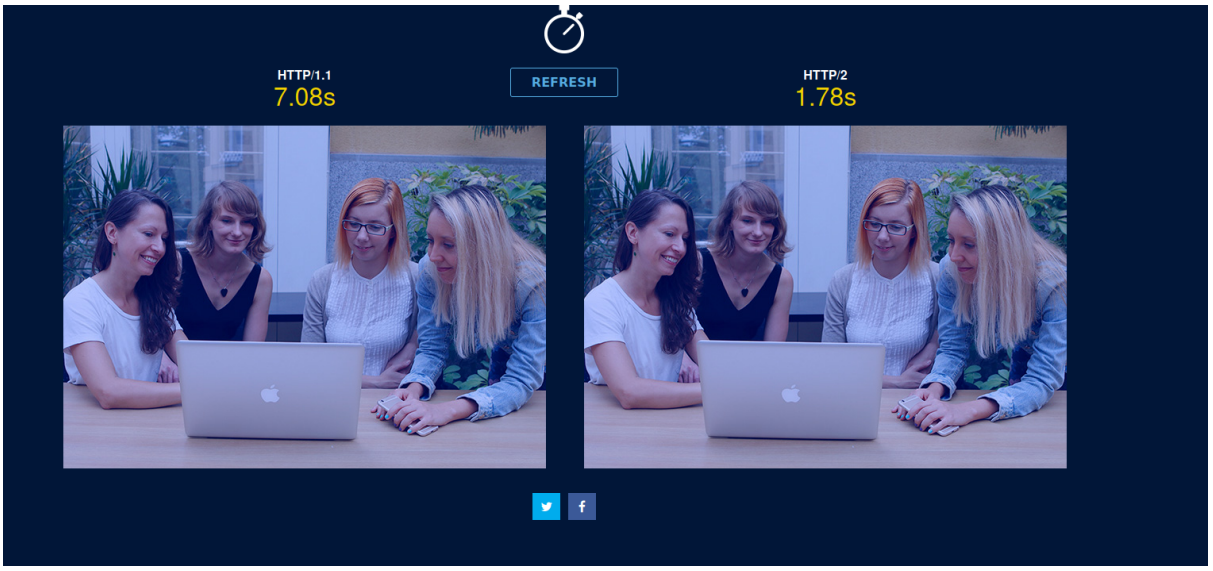
Como vemos, el tiempo utilizando el proxy fue siempre superior comparado a cuando no se lo utilizó, y siempre fue más rápido a menor buffer. Cabe aclarar que igualmente de ese tiempo dependen más cosas además del tamaño del buffer, como por ejemplo los tiempos de acceso a disco. Cabe destacar también que en todos los casos se comparó que el archivo se haya copiado bien con el hash.

Otra prueba que se realizó fue utilizar el server como proxy socks del browser, en particular Firefox. Una de las pruebas que se realizó fue la siguiente, por medio de la página http://http2demo.io:

Sin proxy:



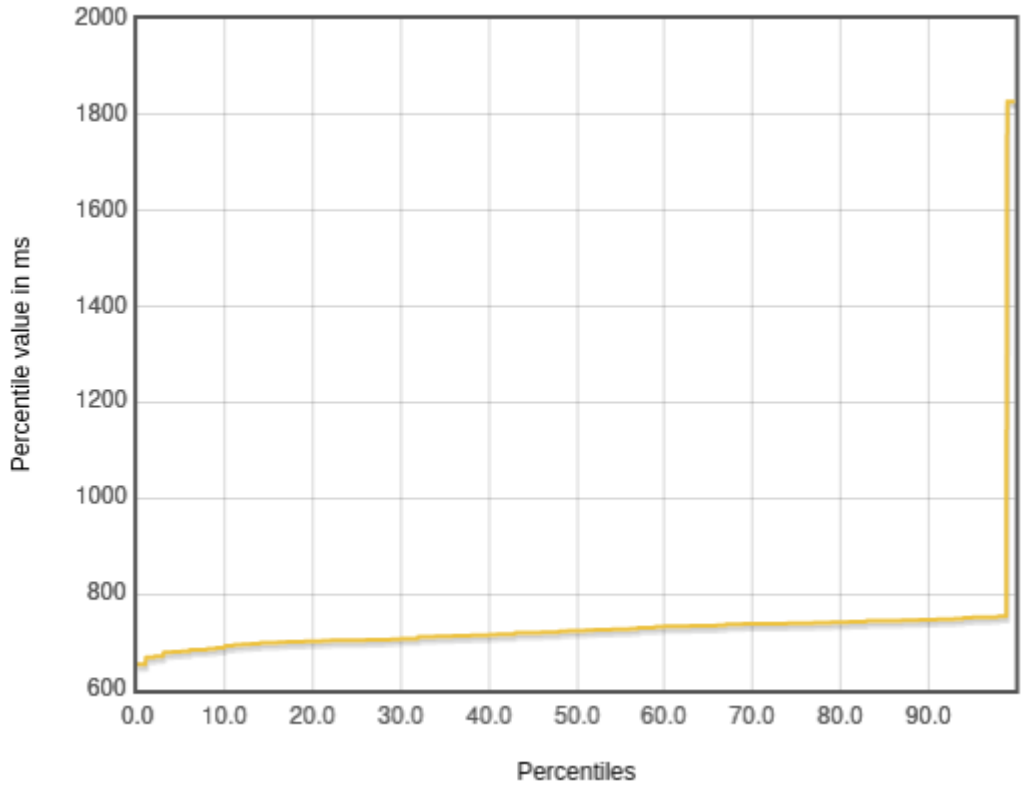
Con proxy:

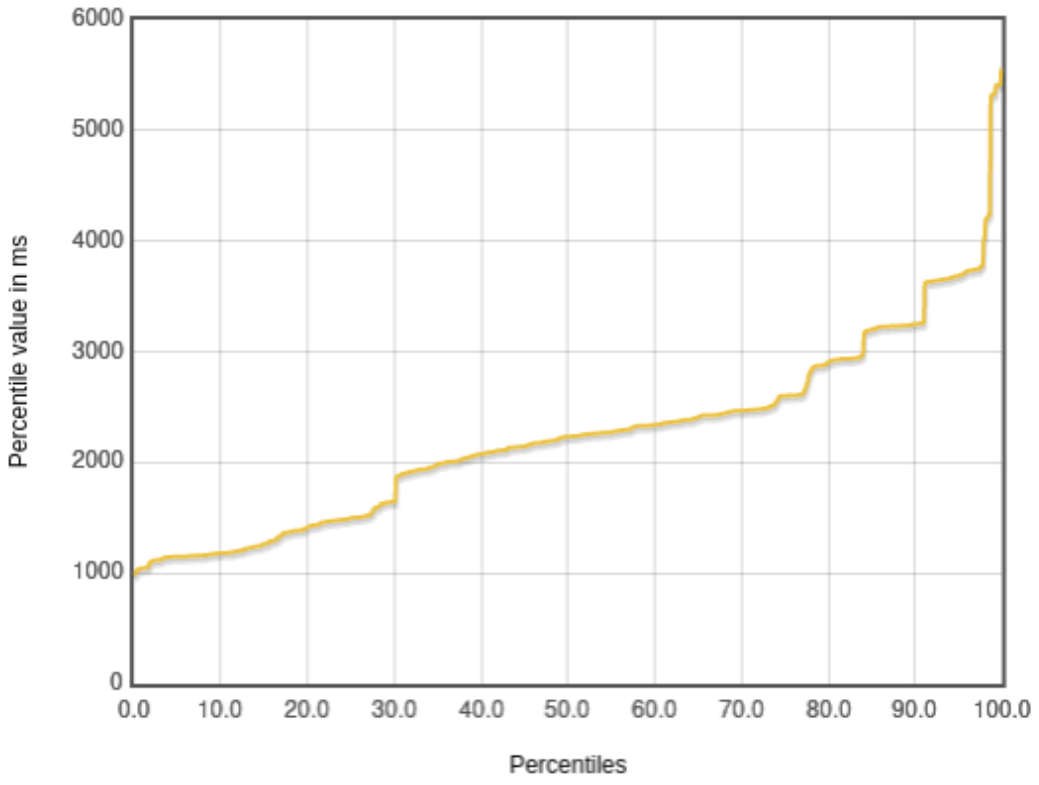


Vemos que los tiempos de las mismas son muy similares pero con un leve aumento al usar el proxy (cabe destacar que las pruebas fueron realizadas múltiples veces para asegurar los resultados). También se realizaron pruebas utilizando scripts para generar muchas conexiones a través del proxy simultáneas, las cuales resultaron exitosas.

Por último se realizaron dos pruebas utilizando Apache Jmeter, con 100 y 500 hilos respectivamente, haciendo pedidos a http://foo.leak.com.ar. Las mismas arrojaron los siguientes resultados (la primer imagen corresponde a la de 100 hilos, la segunda la de 500):

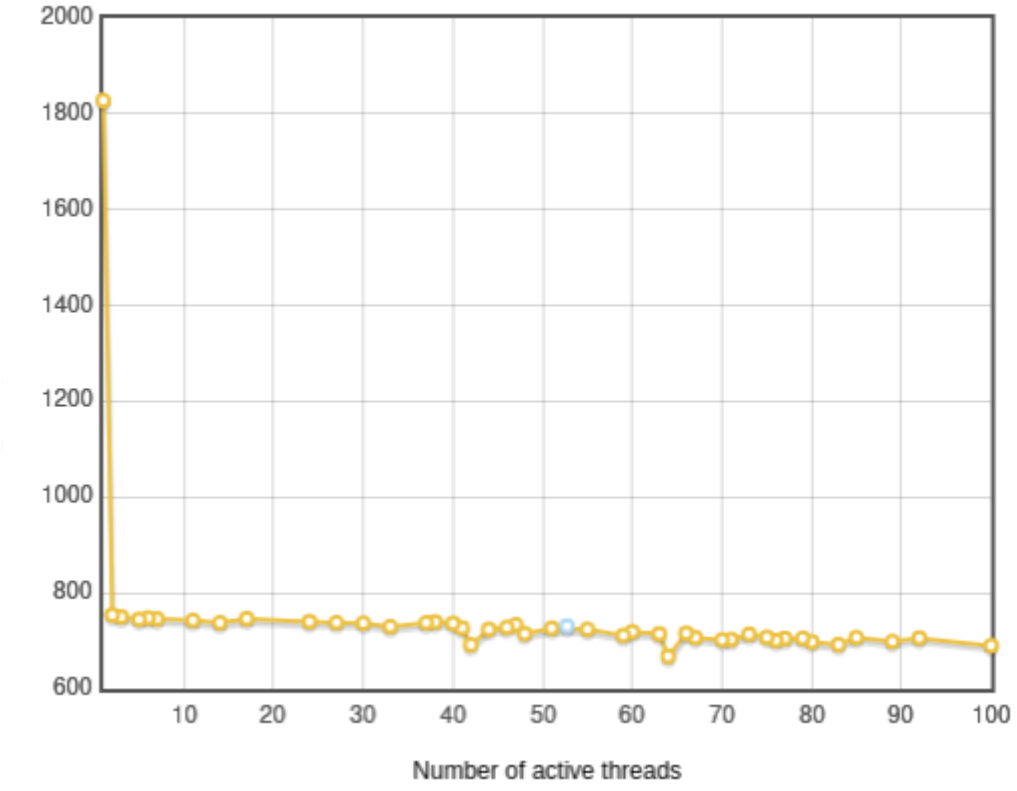
Percentiles de tiempos de respuestas:

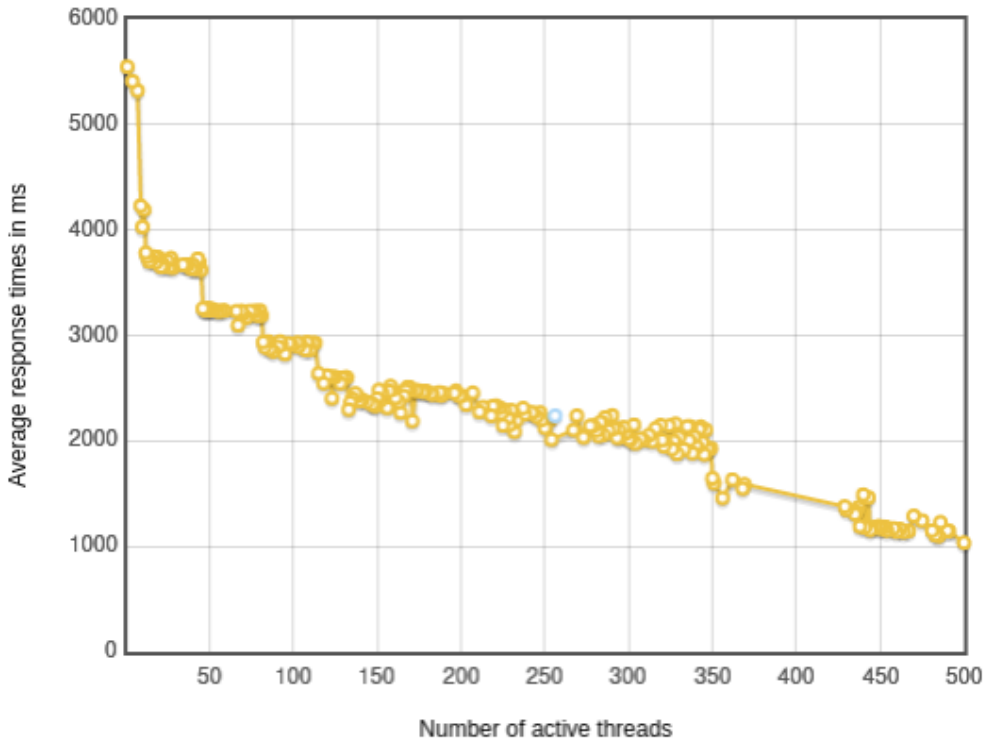




Vemos que al incrementar la cantidad de hilos y por consecuente de pedidos, en el primer caso casi todas las respuestas tienen un tiempo menor a 800ms, mientras que en el segundo caso la situación empeora considerablemente.

Tiempo de respuesta vs hilos:





Podemos apreciar una situación similar a la anterior. Al principio cuando se generan más hilos, los handshakes y pedidos de conexión hacen que al principio lo tiempos de respuesta sean elevados. A medida que los hilos terminan el pedido, el tiempo de respuesta de los mismos mejora considerablemente. Esta situación no es tan así en el primer caso, en el que los tiempos se estabilizan bastante rápido.

Guía de instalación.

Primero y principal se procede a listar los requerimientos de compilación:

- GCC (>=9.4.0)
- Make (>=4.2.1)

luego basta con pararse en el root del proyecto y correr:

```
>$ make
```

Esto generará dos carpetas nuevas build/ y client/build que contienen los archivos `socks5d` y `client` respectivamente.

De todas maneras, el repositorio cuenta con un README.md que cuenta también con la guía de instalación.

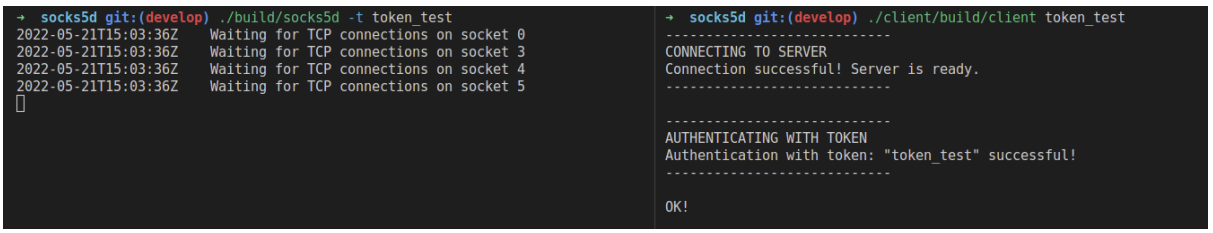
Además de los targets de compilación de archivos .c, hay un target lexer que permite recompilar el archivo .re para crear el parser. Para correrlo es necesario tener instalado re2c. **No es necesario correr el target lexer para crear los ejecutables ya que el mismo fue ejecutado previamente y se subieron los respectivos archivos generados.**

Instrucciones para la configuración.

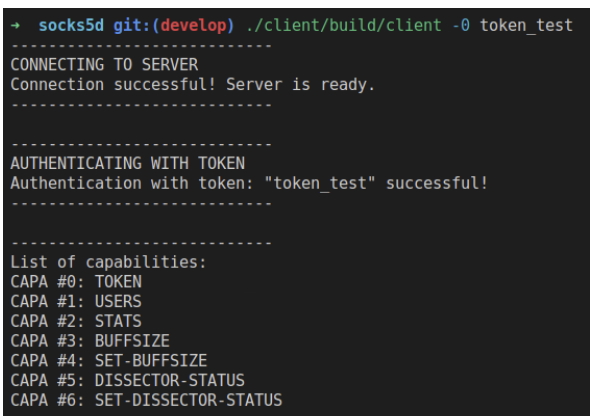
Las instrucciones para la configuración se encuentran en el README.md del proyecto.

Ejemplos de configuración y monitoreo.

En principio podemos ver como con los comandos previamente mencionados podemos monitorear la conexión al servidor.



Luego, podemos pedir las capacidades del servidor.



Podemos ver las estadísticas,

```
→  socks5d git:(develop) ./client/build/client -1 token_test
----------------------------
CONNECTING TO SERVER
Connection successful! Server is ready.
----------------------------

----------------------------
AUTHENTICATING WITH TOKEN
Authentication with token: "token_test" successful!
----------------------------

----------------------------
List of statistics:
Bytes transferred: 0
Historical connections: 6
Concurrent connections: 1
----------------------------

OK!
```

Y ver el tamaño del buffer y la lista de usuarios que tiene el servidor,

```
→  socks5d git:(develop) ./client/build/client -2 -3 token_test
----------------------------
CONNECTING TO SERVER
Connection successful! Server is ready.
----------------------------

----------------------------
AUTHENTICATING WITH TOKEN
Authentication with token: "token_test" successful!
----------------------------

----------------------------
List of users:
----------------------------

----------------------------
Current size of the buffer: 2048
----------------------------

OK!
```

También, registrar un usuario y ver luego como no solo lo podemos ver en la lista de usuarios sino que no se nos permite volver a crearlo, pues ya existe.

```
→  socks5d git:(develop) ./client/build/client -2 -3 -7 jero:jero token_test
----------------------------
CONNECTING TO SERVER
Connection successful! Server is ready.
----------------------------

----------------------------
AUTHENTICATING WITH TOKEN
Authentication with token: "token_test" successful!
----------------------------

----------------------------
List of users:
----------------------------

----------------------------
Current size of the buffer: 2048
----------------------------

----------------------------
Successfully added user to the list of users.
----------------------------

OK!
```

```
→  socks5d git:(develop) ./client/build/client -2 -3 -7 jero:jero token_test
----------------------------
CONNECTING TO SERVER
Connection successful! Server is ready.
----------------------------

----------------------------
AUTHENTICATING WITH TOKEN
Authentication with token: "token_test" successful!
----------------------------

----------------------------
List of users:
USER #0: jero
----------------------------

----------------------------
Current size of the buffer: 2048
----------------------------

Error running ADD-USER: -ERR Error adding user jero

OK!
```

Por último, podemos ver que se le puede especificar el puerto a conectar, el default es el `8080` con Host en `localhost`.

Documento de diseño del proyecto.

El servidor socks consiste de un solo hilo de ejecución (salvo para la resolución de nombres), que ejecuta la syscall pselect() para ver que file descriptors están disponibles para operaciones de entrada y salida. A estos fds se les asocia la información del cliente al que corresponden. Como el servidor es concurrente, cada cliente puede estar en un estadío distinto de ejecución, por lo que es fundamental mantener su estado.

Para hacer esto más prolijo, se utilizó la librería de máquina de estados provista. La misma sabe que se debe ejecutar en cada momento, lo cual permite modularizar mucho más las funciones.

El cliente también consiste únicamente en un hilo de ejecución que parsea los argumentos para identificar qué puertos y dirección se desea usar, luego por cada argumento numérico, se hace el pedido al servidor y se lee hasta que se reciba un CRLF . CRLF o únicamente CRLF dependiendo el caso. Estos datos se guardan en un buffer para luego pasarlos individualmente y así proyectar una versión más Human readable de los resultados del servidor. Esto podría ser modificado para interpretar los datos y presentarlos de la manera que se desee.