Actividad HTTP

proxy_server_co.py

Este archivo contiene el código principal para ejecutar el proxy, esto es, creación de los sockets para establecer las conexiones entre cliente-proxy y proxy-servidor. Para ejecutar este archivo se debe entregar el nombre y ruta del archivo en donde se encuentra el archivo JSON que contiene las palabras y recursos prohibidos de acceder:

```
$ python proxy_server_co.py json_actividad_http.json .
```

Con respecto al código presente en este archivo, primero se obtienen los inputs del usuario para el nombre y dirección del archivo en el cual se encuentra tanto las palabras como los recursos prohibidos, posteriormente se define el tamaño del buffer destinado a recibir los mensajes HTTP. Luego se establece la dirección en la cual el proxy se mantendrá esperando a recibir peticiones de clientes como se observa en la Figura 1 que muestra el flujo en una comunicación orientada a conexión entre cliente y servidor.

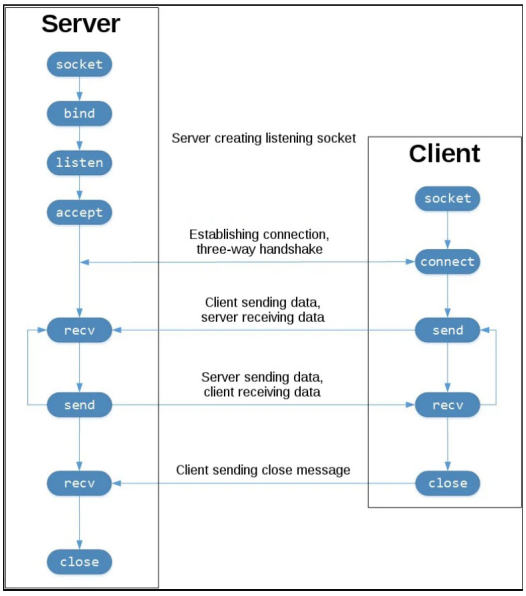

Figure 1: Comunicación orientada a conexión.

Dado que el proxy debe actuar como servidor y cliente, al momento de esperar las conexiones de clientes este debe establecer una conexión con el servidor al cual el cliente desea conectarse.

A continuación se presentan las preguntas planteadas en la Actividad en el punto 5 de la parte 2, relacionadas a la creación de una función que reciba mensajes HTTP dado un tamaño de buffer determinado:

- **¿Cómo sé si llegó el mensaje completo?**:

  Para verificar que un mensaje HTTP fue enviado por completo es necesario recordar la estructura de estos mensajes, estos se componen por un HEAD, que contiene el start-line y headers necesarios, separados por la secuencia \r\n, donde el último header de esta termina con la secuencia de caracteres \r\n\r\n. Opcionalmente, los mensajes HTTP pueden componerse de un BODY

que contiene recursos requeridos por el cliente y enviados por el servidor, sin embargo en tal caso de existir este componente es obligatorio declarar el header 'Content-Length' con el respectivo tamaño del recurso en bytes.

Dada esta estructura que siguen los mensajes HTTP es posible observar que para verificar que un mensaje llegó por completo es necesario observar si el mensaje contiene la secuencia \r\n\r\n, sin embargo esto no es suficiente pues además se debe realizar la búsqueda del header 'Content-Length' para consultar si el mensaje contiene un BODY, en caso de contenerlo simplemente se obtiene el valor del header 'Content-Length' y se llama *recv*() con este tamaño de buffer.

- **¿Qué pasa si los headers no caben en mi buffer?**:

  En caso de que los headers, así como un posible BODY, no puedan caber en el buffer se debe llamar nuevamente *recv*() las veces necesarias verificando cada vez si el mensaje contiene la secuencia \r\n\r\n y posiblemente un BODY.

- **¿Cómo sé que el HEAD llegó completo?¿Y el BODY?**:

  Nuevamente, dada la estructura de los mensajes HTTP, el HEAD se verifica que llegó por completo observando si el mensaje contiene la secuencia \r\n\r\n que demarca el final de este componente. Por otra parte, el BODY se verifica que llego por completo observando la cantidad de bytes que llegaron luego de la secuencia \r\n\r\n y comparandolos con la cantidad de bytes que posee el BODY dado el header 'Content-Length'.

utils.py

Este archivo posee funciones que son de utilidad para recibir, manejar y parsear los mensajes HTTP. Dado que solamente posee funciones, las cuales se encuentran documentadas, no es necesario explayarse en detalle en la descripción de cada una de estas.