Informe Trabajo Práctico Especial

Integrantes:

- Tomás Dorado
- Tomás Dallas
- Joaquín Battilana
- Guido Princ

Grupo: 6

Fecha: 12/11/2019

Índice

Descripción de protocolos y aplicaciones

Protocolo POP3

El protocolo POP3 está diseñado para recibir mails. Utiliza el protocolo de transporte TCP para realizar sus conexiones y es un protocolo de texto, orientado a la conexión.

Al recibir una conexión nueva primero envia un *greeting* al cliente con un mensaje positivo (+OK) indicando que puede enviarle mensajes.

Algunos mensajes posibles son:

- USER <usuario> Envía el usuario para ingresar.
- PASS <contraseña> Envía la clave después de enviar un usuario.
- CAPA Solicita el envío de un mensaje con los comandos posibles.
- STAT Da el número de mensajes no borrados en el buzón y su longitud total.
- LIST Muestra todos los mensajes no borrados con su longitud.
- RETR <número> Solicita el envío del mensaje especificando el número (no se borra del buzón).
- DELE <número> Borra el mensaje especificando el número.
- QUIT Cierra la conexión.

Todo mensaje de respuesta de un servidor con este protocolo tiene siempre su primer línea un mensaje positivo si el pedido fue correcto (+OK) o un mensaje negativo si fue incorrecto (-ERR).

Cada vez que se solicita un mensaje, este viene finalizado con los siguientes caracteres \r\n.\r\n entonces si el mensaje anterior se desea tener un punto después de un \n , se realiza un byte-stuffing, que agrega un punto más al lado en donde se encuentren puntos después de un \n, así evitando que se envíe el mensaje de finalización antes.

Servidor Proxy

El servidor proxy es un servidor que se conecta con un servidor POP3 y actúa de intermediario para las conexiones que deseen filtrar el contenido de los mails pedidos al mismo. Este realiza las transformaciones del contenido con dos posibles métodos: por un comando de consola o filtrando determinados tipos de media types. Ambos tipos de transformación se pueden setear al encender el servidor o mientras su ejecución, mediante el protocolo de monitoreo, con su respectiva aplicación cliente que más adelante se detalla.

Para realizar la aplicación, primero se hizo una versión básica con sockets bloqueantes para realizar la conexión. En este primer modelo se creaba un nuevo thread en para manejar cada nueva conexión entrante. Esto lo que puede generar es que si se tiene muchas conexiones cada una tendrá muy poco tiempo de procesador por lo tanto no es una buena solución para hacer este tipo de servidor.

La segunda versión utiliza sockets no bloqueantes, entonces para esto el servidor tiene un ciclo en el cual verifica que conexiones o sockets tienen algún contenido para recibir o enviar, y resuelve todas esos mensajes hasta volver a recibir otras. En este ciclo se verifican todas las posibles conexiones, tanto de clientes POP3 como del monitoreo de la aplicación.

Para las conexiones de monitoreo, como hay un solo administrador en teoría, se manejan en un thread separado, el cual se crea cuando hay una conexión nueva de monitoreo y se cierra cuando esta termina. Como no habrá más de un administrador, esto no va a generar problemas de tiempo de procesamiento.

Para realizar los sockets de conexión de cada cliente con el servidor de origen, como esto puede bloquear el procesamiento, la solución dada es la creación de un nuevo thread, que solamente crea ese socket y cuando termina envía una señal al ciclo principal de que ya está disponible el socket para que el cliente le envíe mensajes.

El servidor soporta conexiones tanto en IPv6 como IPv4, siendo este último mapeado a una IPv6, entonces simplificando qué tipo de conexiones estar esperando en IPv6.

La aplicación utiliza diferentes estructuras para el manejo de los datos a continuación se detallaran las más importantes:

- Configuración: Es una estructura principal que tiene la información que utilizan todos los clientes. Aquí se encuentran las configuraciones de transformaciones externas, puertos, direcciones. Esto es lo que modifica el administrador mediante el protocolo de monitoreo.
- Métricas: Esta estructura tiene 3 elementos que puede ver solamente el administrador al monitorear, las cuales son bytes transferidos en total, conexiones totales y conexiones actuales.
- Lista de Clientes: Esta es usada para el ciclo principal el cual va chequeando que cliente tiene que realizar algo.
- Cliente : Cada cliente tiene una estructura en la cual se mantiene el estado de su conexión (ya sea que está mandando o esperando), los sockets que posee y tres buffers, una para los mensajes que recibe del cliente, otra para lo que recibe del servidor origen y otra para lo que recibe después de realizar la transformación externa.
- Estructura de parseo de transformaciones: Como el envío de mensajes se realiza por buffers fue necesario crear esta estructura en los clientes, para saber en qué estado estaba al enviar el buffer anterior y así saber cómo seguir.

Las transformaciones de mensajes se realizan en procesos separados, que reciben por entrada estándar el mensaje y devuelve por salida estándar el mensaje transformado. Entonces para enviar los mensajes se crea un pipe que se conecta con el stdin de ese proceso hijo, y otro que se conecta con el stdout del mismo. Así se le puede ir enviando el mensaje mientras se va recibiendo. Al momento de enviar por el pipe, primero se realiza el byte-unstuffing de los puntos que tiene el mensaje y al momento de recibir por el otro pipe el mensaje transformado, se realiza el byte-stuffing de los puntos que lo requieran de nuevo, antes de enviarlo al cliente.

Entonces, el ciclo que hace una conexión nueva de un cliente, que ingrese y pida un mail sería el siguiente, suponiendo que alguna transformación está activa:

- Cliente se conecta al puerto y dirección del proxy.
- Proxy acepta la conexión y crea configuración inicial de cliente.
- Espera a que se cree el socket al servidor origen, hasta que se crea y manda señal al thread principal.
- Se lee el mensaje de greeting del socket.
- Se envía mensaje de greeting al cliente.
- Cliente envía user y pass.
- Proxy los recibe y envía al servidor origen.
- Proxy recibe la respuesta del servidor origen y envía al cliente.
- Cliente recibe las respuestas y envía un RETR de un mensaje.
- Proxy recibe el pedido y lo envía al servidor origen.
- Proxy recibe la respuesta y la envía a través de un pipe, realizando el byte-unstuffing, al programa que realice la transformación del mensaje.
- Proxy recibe la transformación del menaje mediante otro pipe, realiza el byte-stuffing, y envía al cliente.
- Cliente recibe el mensaje transformado.

Protocolo de monitoreo

El protocolo de monitoreo tiene como principales acciones: activar o desactivar las transformaciones externas; obtener métricas como conexiones concurrentes, conexiones totales y bytes transferidos; setear o sacar media types a ser filtrados; o cambiar el comando a ejecutarse si es así la transformación externa.

El mismo está realizado sobre el protocolo de transporte SCTP, y el protocolo es orientado a la conexión y binario. Cada posible comando tiene un identificador propio, que es el primer número del request que se haga, y cada respuesta puede ser satisfactoria o negativa.

Para conectarse se utiliza un token de 8 bytes que solamente puede tener el administrador y así poder logear, mediante el protocolo, al proxy.

Los request que tiene son(junto con su número):

- 0x00 = LOGIN
- 0x01 = LOGOUT
- 0x02 = GET CC (concurrent connections)
- 0x03 = GET TC (total connections)
- 0x04 = GET BT (bytes transfered)
- 0x05 = GET MTYPES
- 0x06 = GET CMD
- 0x07 = SET CMD
- 0x08 = SET MTYPES
- 0x09 = ENABLE MTYPE TRANSFORMATIONS
- 0x0A = ENABLE CMD TRANSFORAMATION
- 0x0B = DISABLE TRANSFORMATION

De los cuales los que tienen parámetros que se envía, se realiza así en cada caso:

- SET MTYPES <mtype1,mtype2,mtype3,...>
  Se traduce a un byte con un 08 y luego vienen todos los media-types que se quieran filtrar separados con una , entre cada uno. Tiene un \0 al final. Este comando pisa los media types que estuviesen antes activos.
- SET CMD <cmd>
  Se comporta igual solo que despues del 08 se ubica el comando nuevo a setear, finalizando con un \0.
- LOGIN <token>

Todos los request son respondidos por un mensaje de respuesta, y un mensaje que devuelve lo que se puso correctamente en caso de SET CMD y SET MTYPES, que puede ser:

- 0x00 = OK
- 0x01 = ERR

Por ejemplo la respuesta correcta de SET CMD es:

- OK <cmd> con el código de OK junto al cmd y un \0 al final.

Y en el caso de SET MTYPES devuelve:

- OK <mtype1,mtype2,mtype3…> con el mismo formato que cuando se envia

Y además los siguientes comandos vienen con información adicional en su respuestas:

- GET CC = OK <conexiones concurrentes>
- GET TC = OK <conexiones totales>
- GET BC = OK <bytes transferidos>
  Que devolverán el número correspondiente en formato de string, con un \0 al final.

- GET MTYPES = OK <mtype1,mtype2,mtype3,...>
- GET CMD = OK <cmd actual>

Ambos tienen como mensaje un string finalizado en un \0

Cliente de monitoreo

El cliente de monitoreo utiliza el protocolo mencionado arriba para conectar con el proxy. Este simplemente traduce los mensajes que salen o entran por el socket a texto humano.

Se le agregaron 2 opciones al ejecutar: una para setear la dirección en la que está escuchando el servidor de monitoreo y la otra el puerto en donde está escuchando. Si alguno de estos no es seteado, se toma los valores por defecto, los cuales son 127.0.0.1 y el puerto 9090.

Para poder realizar cualquier comando primero hay que loguearse enviando un token de 8 bytes. El cliente que se realizó ya ingresa el token desde el código que tiene adentro, para hacer más simple la aplicación. El token es: “ZXN0YUFw”

Filtro de Media-Types

Para el filtro de media types lo que se trató fue de tener una mejor performance, tanto espacial como de tiempo, por lo que en muy pocos casos se guardó una variable con lo que se iba leyendo de STDIN y muchas pocas veces se volvió a releer. Particularmente se uso una pila para ir apilando los media type que íbamos utilizando y así poder manejarse dentro del “árbol” de media types, donde si se encuentra un multipart se abre un nodo con hijos y si hay otro tipo nos encontramos con una hoja.

En el caso de tener que censurar algún media type, el flujo que se tomó fue cambiar solo el contenido al texto de censura, los demás headers se mantuvieron por un tema de performance y no tener que guardar todo hasta que encontremos Content-Type.

Problemas encontrados y limitaciones

Servidor Proxy

- **Cantidad de conexiones:** El servidor utiliza pselect para la espera de cambios en los sockets o recibir una señal. Esto es un problema ya que está limitado por el límite de fd_set. Lo ideal sería que utilice ppoll, pero por cuestiones de tiempo no se realizó.
- **PIPELINING:** En el caso de que el servidor de origen lo soporte, y estén las transformaciones activas, cada vez que llega un RETR el servidor tiene que esperar la respuesta de ese RETR y enviarla al proceso transformante. El problema es que no se pudo encontrar manera de que podamos enviar muchos mensajes y luego encontrar en las respuestas, que pueden llegarle al proxy todas juntas, las respuestas de cada determinado RETR. Por esta razón, cuando el servidor tiene pipelining, pero están las transformaciones activadas, el proxy va a recibir todos los mensajes juntos que mande el cliente, pero los pedidos los hará uno por uno para

así poder saber cuando se pide un RETR y poder enviar su respuesta a la transformación externa. En caso de que no esten activas las transformaciones, se enviará todo tal cual le llegue al proxy, si el servidor origen tiene pipelining, y si no lo tiene se enviará mensaje a mensaje.

Cliente de monitoreo

El buffer tiene tamaño fijo lo cual limita un poco el uso de la cliente.

Al principio pensamos el protocolo de una manera compleja, diferenciando entre tipo de comando (get, set, rm) y tipo de operadores (bytes, concurrent, etc), que representaban 1 byte identificatorio cada uno, y armabamos la respuesta utilizando la misma lógica, pero eso llevó a una complejidad alta en el parseo y la comunicación entre el proxy y el protocolo. Decidimos simplificarlo e identificar cada comando con 1 byte, y enviarlo al proxy.

Ejemplo representativo:

Antes:

-GET BYTES estaba representado por 0x01 (GET) 0x01 (BYTES)

-GET MTYPES estaba representado por 0x01 (GET) y 0x03 (MTYPES)

Ahora:

-GET BYTES es representado por 0x04

-GET MTYPES es representado por un 0x05

Otro problema que tuvimos fue en la manera que habiamos pensado el SET y RM, el comando era : ***SET/RM MTYPES mtypesQty <mtype1,...,mtypeN>*** y cada mtype estaba separado por un '\0', pero esto complejizó nuevamente mucho el parseo de datos y comunicación con el proxy, por lo que decidimos cambiar esto, simplificándolo, y ahora el formato del comando es: ***SET MTYPES <mtype1,...mtypeN>***, cada mtype está separado por una ',' y el string está terminado en '\0', y siempre se pisan todos los mtypes.

Al cambiar todo el protocolo, tuvimos que borrar todos los tests que teníamos hecho sobre el protocolo anterior.

También tuvimos problemas con la conexión SCTP, del lado del proxy y del lado del cliente. No sabíamos si la conexión se estaba realizando, aunque parecía que sí, porque estábamos enviando un comando, el proxy lo recibía, pero quedaba bloqueado para próximas interacciones. Ahí nos dimos cuenta que el ***sctp_recvmsg*** es bloqueante, por lo que decidimos armar un thread para el socket sctp, pero igualmente seguíamos teniendo problemas. Al hacer un netstat, no veíamos la conexión con el socket sctp en todas las computadoras del equipo. En una con netstat versión 1,42 no aparecia, pero en otra versión 2,2 si. Luego de investigar un poco, encontramos que la conexión sctp se puede ver en ***/proc/sctp/eps*** en las computadoras que no tengan la version mas nueva de netstat, y ahí confirmamos que no era un problema de la lógica dentro de la aplicación y que la conexión existía.

Filtro de Media-Types

Nos encontramos con varios problemas, el más predominante fue que era algo tedioso para hacer ya que había muchos casos y no teníamos el contenido guardado en una variable en memoria, si no que teníamos que leer de a un carácter y queríamos no tener que guardarlo para mejorar la performance. Un problema grande fue cuando empezamos a probar con los casos de prueba y el caso del email que estaba contenido dentro del mensaje rompía y no nos dábamos cuenta porque, esto era debido a que el Content-Type: message/… tiene un formato especial, el cual su body es el comienzo del header de otro email y no lo estábamos contemplando.

Hay varias limitaciones que pusimos para que sea más simple el problema:

La cantidad de media types que puede haber dentro de la variable de entorno MEDIA_TYPES, es limitado a 5, es fácil de escalar pero no lo tuvimos en cuenta.

El contenido máximo de un media type, contando el “/” es 127, esto es debido a que si bien en el nuevo RFC de MIME se declara que no hay un límite, para retrocompatibilidad recomiendan usar el máximo de 127 que es el que estaba declarado antes. Lo usamos porque nos facilito tema de manejo de memoria.

El contenido máximo de un boundary son 70 caracteres, esto es debido a que está definido así.

Posibles extensiones

Servidor Proxy

Filtrar correos validando contra una lista negra, ya sea local o externa.

Cliente de monitoreo

Agregar pipelining. Ofrecer autenticación para varios usuarios y manejar varias conexiones a la vez.

Conclusiones

Al ser un trabajo práctico evolutivo y donde hubo que investigar muchos temas e implementaciones, hubo un aprendizaje profundo sobre varios temas. La constante refactorización hizo que pudiéramos pensar e implementar varias soluciones para un mismo problema, analizarlas desde distintos puntos de vista, ejecutar prueba sobre ellas, y elegir siempre la que nos pareció la más ideal y que mejor performaba. Tuvimos, además, que leer mucha documentación y documentos RFC, lo que hizo que ampliáramos los conocimientos sobre la materia, en particular de POP3, mimeTypes, SCTP y demás conocimientos aplicados al proxy.

Si bien la idea de tener un proxy pop3 es una gran solución para filtrar contenidos no deseados en una casilla de correo, hoy en día esta solución es casi obsoleta debido a que casi todo servicio de mail tiene implementado su propio filtro de contenido no deseado, y hoy pop3 no es un protocolo muy utilizado.

Guía de instalación

- Clonar el repositorio
- Ir al root del proyecto
- make --> esto genera 3 ejecutables: pop3filter, pop3clt, stripmime
- Para ejecutar el pop3filter:
    - *./pop3filter [POSIX Style Options] <origin-server-address>*
- Para ejecutar el cliente para el administrador:
    - *./adminclient -a <management-address> -p <management-port>*

- Para ver todas las opciones a ejecutar del pop3filter:
  ./pop3filter -h
- Para ver todas las opciones a ejecutar del cliente para el administrador:
    1. Ejecutar la aplicación: **./pop3clt**
    2. Entrar el comando **help**

Instrucciones para la configuración

USAGE: ./pop3filter [POSIX style options] <origin-server-address>

- **<origin-server-address>** : Direccion del servidor origen POP3

POSIX style options:

- **-h** : Imprime los comandos posibles y termina.
- **-v** : Imprime la versión del Proxy POP3 Filtro y termina.
- **-P <origin-server-port>** : Especifica el puerto TCP en donde el servidor origen POP3 está ubicado. Por defecto es 110.
- **-l <listen-address>** : Especifica la dirección en donde el proxy va a escuchar. Por defecto escucha en todas..
- **-p <listen-port>** : Especifica el puerto TCP en donde el proxy escuchará a las conexiones POP3 que vengan. Por defecto es 1110.
- **-L <management-address>** : Especifica la dirección en donde el protocolo de administrador va a estar escuchando. Por defecto es loopback.
- **-o <management-port>** : Especifica el puerto SCTP en donde el escuchara el protocolo de administrador. Por defecto es 9090.
- **-M <filtered-media-type>** : Especifica un media-type que va a ser censurado. Por defecto no aplica censuras.
- **-m <replacement-message>** : Especifica el mensaje por el cual se transformaran los espacios censurados. Por defecto es “Parte Reemplazada.”..
- **-t <filtered-command>** : Comando usado para transformaciones externas. Por defecto no aplica transformaciones.

- **-e <error-file>** : Especifica el archivo en el cual se va a redirigir el stderr. Por defecto es “/dev/null”.

Ejemplos de configuración y monitoreo

Ciente de monitoreo

./pop3clt

```
+OK Connection established
login
+OK Logged in.
get bytes
Invalid request, write help for options.
get bt
+OK 0
help
Commands:
   login
   logout
   get cc(concurrent connections)
   get tb(total connections
   get bt(bytes transferidos)
   get mtypes
   get cmd
   set cmd <cmd>
   set mtypes <mtype,mtype,mtype,mtype...>
   enable mtype transformations
   enable cmd transformations
   disable transformations
set mtypes hola/mtype1,hola/mtype2
+OK media types added: hola/mtype1,hola/mtype2
get mtypes
+OK hola/mtype1,hola/mtype2
get cmd
+OK cat
get bt
+OK 0
get mypes
Invalid request, write help for options.
get mtypes
+OK hola/mtype1,hola/mtype2
get mypes
Invalid request, write help for options.
get mtypes
+OK mtype
```

Servidor Proxy

./pop3filter 0.0.0.0 -e error

[2019-11-12 14:36:00] Creating proxy server socket.
[2019-11-12 14:36:00] Proxy server socket created successfully.
[2019-11-12 14:36:03] Successfully sent data to admin
[2019-11-12 14:36:16] Closing admin connection
[2019-11-12 14:36:36] Successfully sent data to admin
[2019-11-12 14:36:48] Successfully received data from admin
[2019-11-12 14:36:48] Successfully sent data to admin
[2019-11-12 14:36:57] Successfully received data from admin
[2019-11-12 14:36:57] Successfully sent data to admin
[2019-11-12 14:37:33] Successfully received data from admin
s = hola/mtype1,hola/mtype2
[2019-11-12 14:37:33] Successfully sent data to admin
[2019-11-12 14:37:40] Successfully received data from admin
[2019-11-12 14:37:40] Successfully sent data to admin
[2019-11-12 14:37:44] Successfully received data from admin

Documento de diseño del proyecto