PROTOCOLOS DE COMUNICACIÓN

Trabajo Práctico Especial

Proxy POP3

Grupo 5

Integrantes
Leandro Llorca (54859)
Juan Pablo Dantur (54623)
José Ángel Vitali (54197)
Leandro Matias Rivas (51274)

Docentes
Marcelo Fabio Garberoglio
Juan Francisco Codagnone
Juan Martín Sotuyo Dodero

Fecha de entrega
14 de junio de 2016

Índice

I - Protocolos y aplicaciones desarrolladas

i. Proxy Server

El servidor se basó en el patrón Reactor. La lógica que sigue es la siguiente

1. El thread principal se queda escuchando con el selector por sockets que quieren aceptar/conectar/escribir/leer
2. Cuando recibe una lista de keys, por cada una llama a un Worker al que le manda el key

Todo esto lo hace con un thread (razón por esto analizado en *Problemas* y *Diseño del Proyecto*).

El worker es el que hace el trabajo pesado con cada key - se fija qué tipo de llave es, y hace lo que corresponde.

El attachment de las keys son el handler. El handler lo que tiene es toda la lógica que se necesita para manejar la data recibida, y modificarla antes de ser enviada de ser necesario (llamando al Parser). Un ejemplo puede ser como resolvimos el tema de los reads parciales. Como java Nio no nos asegura que en un *read* lea todo lo que tiene que leer, un importante método del handler es *bufferDone()*, que devuelve un booleano que indica si lo que actualmente está en el buffer se debe mandar o si debe esperar un read más. Así, el handler maneja cuando mandar los datos y cuándo recibir más (y sin meterse con el proxy).

Cuando se decide que se debe realizar una escritura, el ByteBuffer donde se encuentran los bytes a enviar se encola en una cola de buffers de escritura. El método handleRead desencola un buffer de la cola, realiza la escritura, y si se escribieron todos los datos del buffer se vuelve a desencolar hasta que la cola quede vacía. En caso de que no se escriban todos los bytes de un buffer, el mismo es ubicado a la cabeza de la cola, de manera que sea el primero en ser desencolado en la próxima operación de escritura.

Si se hacen muchas lecturas consecutivas en el mismo SocketChannel y no se se hacen escrituras en el socketChannel asociado, se deja de leer del primer socket hasta que se hayan realizado escrituras en el otro socket, ya que si no se hiciera esto, la cola de buffers de escritura podría crecer indiscriminadamente.

ii. Parser

Para el proceso de transformación de mails, se tiene una función que recibe el buffer con una porción del mensaje en cuestión y procede a encolar cada línea completa que encuentra (almacenando lo que hay al final para luego unirlo al buffer que reciba en el próximo llamado). Una vez hecho esto, procede a desencolar cada una de las líneas y procesarlas (almacenando convenientemente en que estado del mail se encuentra y si se habilitaron las transformaciones pedidas o no). Para el asunto y el contenido que no se transforma se procesa cada línea por separado debido a que según los RFC pertinentes, si el asunto es multilínea no es necesario tenerlo completo en memoria para procesarlo. Con tener cada línea alcanza. La codificación y decodificación de las mismas se realiza mediante las clases BCodec y QCodec de Apache-commons-codec, y expresiones regulares para obtener el formato y la codificación de cada linea. En dicha función se procede a reemplazar los caracteres del String que pide transformar la consigna antes de ser recodificado. Para cada imágen (una vez vistos los headers y almacenado el formato con una RegEx convenientemente definida) se procede a guardar cada una de sus líneas en otro buffer, ya que para rotarla es necesario tenerla completa. (Se tiene una constante con un tamaño máximo de imágen a transformar para no generar desbordes de memoria, si la imágen supera ese tamaño se limpia el buffer auxiliar y se procesa línea a línea, a fin de dejarla intacta y preservar la integridad del mail). Una vez visto el Boundary se rota lo obtenido y se procede normalmente. Para el proceso de rotación, se utiliza la clase ImageIO de la librería JavaXT y respectivos codificadores y decodificadores Base64. La imagen devuelta en algunos casos tiene unos pocos caracteres más que la imagen recibida. En caso de que el formato de la

imagen no sea soportado por JavaXT (los formatos soportados son png, jpg, gif, bmp, wbmp), esta no será rotada.

iii. Administrador

En cuanto al protocolo para configurar y obtener estadísticas del proxy, se pensó un protocolo muy similar al POP3. Habrá 2 estados. AUTHENTICATION y TRANSACTION.

Por default, el administrador escucha en el puerto 9999.
Los comandos son case-insensitive.
Es orientado a texto de tipo request-response.
Las respuestas multilínea están terminadas en CRLF.CRLF. Los comandos con respuesta multilínea son CAPA y STAT.

El protocolo soporta los siguientes mensajes:

- USER name
    - Parámetros
        - Nombre de usuario administrador.
    - Descripción
        - Disponible en el estado de AUTHENTICATION. Retorna una respuesta positiva.
- PASS string
    - Parámetros
        - Contraseña requerida para acceder como administrador
    - Descripción
        - Una vez ingresado el USER y PASS, se determina si se debe conceder acceso como administrador.
        - Luego del comando y el espacio, todo lo restante formará parte de la contraseña.

    - Posibles Respuestas:
        - +OK Logged in.
        - -ERR USER or PASS incorrect.
- STAT
    - Parámetros
        - No recibe parámetros
    - Descripción
        - Devuelve las estadísticas del proxy POP3
- QUIT
    - Parámetros
        - No recibe parámetros
    - Descripción
        - Cierra la sesion del administrador
- LEET state
    - Parámetros
        - Recibe un parámetro para indicar si se activa o desactiva esta transformación.
    - Descripción
        - Activa o desactiva la rotación
    - Posibles Respuestas
        - +OK
        - -ERR Input not valid.
- ROTATION state
    - Parámetros
        - Recibe un parámetro para indicar si se activa o desactiva esta transformación.
    - Descripción
        - Activa o desactiva la rotación de imágenes en 180 grados.
    - Posibles Respuestas
        - +OK
        - -ERR Input not valid.
- CAPA

    - Parámetros
        - No recibe parámetros
    - Descripción
        - Retorna todos los comandos disponibles

Resumen de Comandos:

USER name	valid in the AUTHORIZATION state
PASS string	
QUIT	
STAT	valid in the TRANSACTION state
LEET state	
ROTATION state	
QUIT	

Respuestas de administrador:

+OK

-ERR

iv. Multiplexing

Una vez que un usuario se conecta al proxy, el mismo solo dispondrá de unos pocos comandos hasta que se loguee exitosamente. Los mismos serán:

- USER String
    - Parámetros
        - Nombre de usuario
    - Descripción
        - Se almacena el nombre de usuario, si ya había uno, el mismo se reemplaza.
        - Luego del comando y el espacio, todo lo restante formará parte del nombre de usuario.

    - Posibles Respuestas:
        - +OK

- PASS String
    - Parámetros
        - Contraseña
    - Descripción
        - Una vez proporcionados el nombre de usuario y la contraseña, se intenta hacer un log in con el servidor correspondiente.
        - Luego del comando y el espacio, todo lo restante formará parte de la contraseña.
    - Posibles Respuestas:
        - +OK
        - -ERR No username given.
        - Lo que responda el servidor al que se intente conectar.
        - -ERR server down.
        - -ERR can’t reach server.

- CAPA
    - Parámetros
        - Ninguno
    - Descripción
        - Se devuelve una lista de las operaciones soportadas por el proxy.
    - Posibles Respuestas:
        - +OK
          USER
          CAPA
          .

Para resolver a qué servidor conectarse, se busca el nombre de usuario ingresado en el archivo de configuración. Si es encontrado, se conectara al host y puerto correspondiente. De no ser encontrado, se conectará con un servidor por

defecto. Si al conectarse ocurre algún error (no hay un servidor en el host y puerto donde se intenta la conexión o el nombre de usuario y/o la contraseña son incorrectos), se imprime el mensaje de error adecuado. Si el log in fue satisfactorio, el proxy deja de comportarse como servidor y simplemente comunica al cliente con el servidor que corresponda, haciendo las alteraciones pertinentes en los mails.

v. Logs

Se implementó un sistema de logs para registrar los movimientos del servidor y el administrador. Uno ahí puede verificar que el servidor se booteo correctamente, y ver posibles errores. Para esto, se utilizó la librería *log4j* que nos facilitó todo lo relacionado con los logs. Los logs se encuentran en el archivo proxyPop3.log

II - Problemas encontrados

Hemos encontrado diversos problemas a lo largo del desarrollo del trabajo práctico. He aquí una mención de algunos de ellos.

Multithreading

El trabajo práctico se planteó para que funcione con muchos threads, basándonos en el patrón *Reactor* (más en *Diseño del Proyecto*). Los problemas relacionados a concurrencia, threads que modifican los keys, y condiciones de carrera fueron resueltos (por lo menos eso creemos). El problema que tuvimos fue una fuerte caída en la velocidad del servidor a la hora de pasar de estar en un thread con un pool de N threads. Para dar un ejemplo, al testear el tiempo que le tomaba al servidor devolver un mail de 100 Mb, con un thread lo hacía en aproximadamente 20 segundos, y con 5 threads tardó más de 5 minutos (muy notable la diferencia). Es un problema muy raro, ya que uno no espera a que la velocidad decaiga tanto, y al ser consultado por la cátedra, no resultó muy notable el causante.

Como consecuencia de este problema, se decidió que la aplicación corra en un único hilo de ejecución, con sockets no bloqueantes. No es lo mejor para lidiar con varios clientes, pero por lo menos nos aseguramos que funciona eficientemente. Igual se mantuvo toda la estructura que lidia con los temas de concurrencia - eso todavía se ve en el código.

Mails grandes

En una primera instancia, el proxy solía almacenar los mails en su totalidad en memoria, lo cual funcionaba con mails pequeños, pero a la hora de recibir mails de un tamaño considerablemente grande, se tenía una excepción porque el servidor agotaba toda la memoria disponible. Para resolver este problema primero se consideró almacenar los mails que se consideraban grandes en archivos, de manera tal que se pueda manejar muchos mails grandes simultáneamente. El

problema de esta solución es que se deben hacer muchos accesos a disco, lo cual tiene un impacto negativo en la performance. Fue por esta razón que se decidió no hacer uso de archivos, y parsear los mails a medida que se los va recibiendo (línea por línea) y solo almacenar en memoria las imágenes si es que la opción de rotación de las mismas se encuentra activada. En caso de una imagen supere un valor máximo soportado (1 Mb) la misma se enviará sin rotar. De esta manera nunca se tiene un mail completo almacenado (ni en memoria ni en disco).

III - Limitaciones de la aplicación

Se instalaron ciertas cotas para garantizar el funcionamiento del proxy

- Tamaño máximo de imagen a convertir: 1Mb
- Tamaño máximo de una línea de mail: 1000 caracteres

-> Cuando las cotas no se cumplen, no es que se descartan. Lo que pasa es que decimos que más que eso no vamos a analizar, entonces le mandamos al cliente la data como la tenemos.

El transformador NO GARANTIZA la preservación de correos mal formados (es decir, que no respeten el RFC). Solo se garantiza que no haya desbordes de memoria.

IV - Posibles extensiones

El código podría evolucionar a ser más genérico tal que se pueda tener un servidor proxy que no sepa exactamente qué protocolo tiene que implementar. Esta fue la idea principal del tp, pero por falta de tiempo terminamos abandonandola (ver *Diseño del Proyecto*). Como un proxy siempre hace lo mismo, lee de un lado para escribir en otro (desde ambos lados), mucho de la lógica es independiente del protocolo - sólo el handler es lo que se debe diferenciar. Así que si uno desea extender este proyecto para que ahora sea un Http proxy server, lo que se cambie sea sólo el parser y unos métodos del handler.

Existen ciertas funcionalidades que nos hubiera gustado implementar, pero por falta de tiempo no lo hicimos. Uno por ejemplo es poder brindar más estadísticas al administrador - como clientes conectados en el momento, mails borrados, etc. Esto sería darle más importancia al rol del administrador y brindarle más herramientas con las cuales trabajar. Otra función sería un manejo de clientes inactivos propio - no depender del servidor al cual nos conectamos para que maneje eso. La idea seria desconectar clientes si llevan mucho tiempo inactivo, como manera de saber que clientes activos tenemos y liberar recursos de los que ya hace un tiempo no interactuan.

V - Conclusiones

Trabajar con Java Nio fue bastante satisfactorio. La habilidad de tener llamadas no bloqueantes, y que un thread principal sea la que maneje los cambios en todos los sockets es algo fantástico. Que no te asegure cuanto es lo que efectivamente lee, más que un problema, fue un desafío a resolver. Es una herramienta poderosa y consideramos que la podemos llegar a usar en el futuro.

Pop no es un protocolo complicado, lo que es más complicado es como los diferentes servidores pueden adoptarlo, ya que hay ciertas cosas no estandarizadas. Un ejemplo de esto fue que, en un momento, nos interesaba utilizar el dato de cuánto pesa un mail antes de recibirlo, o sea con Dovecot sería analizar la primera línea del *retr* que te dice cuanto pesa el mail - pero esto no es estándar! El rfc de Pop3 solo dicta que digas un “+Ok” y optativamente un mensaje, no implica que mensaje es ese. Efectivamente, encontramos un servidor que no devolvía el tamaño del mail después de un *retr*, entonces no podíamos aplicar funcionalidades a partir de esa línea ya que no todos los servidores lo implementan.

VI - Guía de Instalación

1. Asegurarse que el archivo “Configuration.xml” esté en la misma ruta que el jar con el tp.
2. Ejecutar el jar.

VII - Configuración

Dentro del proyecto hemos creado un archivo cuyo nombre es Configuration.xml. Este contiene información necesaria para iniciar el proxy y administrador. Si se desea, puede realizar cambios sobre el mismo. Modificaciones en tiempo de ejecución no se realizan sobre dicho archivo.

Para obtener los valores del XML se ha utilizado la librería JAXB (Java Architecture for XML Binding).

Para acceder al administrador se debe entrar al puerto 9999 de localhost y loggearse con el usuario y pass definidos en Configuration.xml (por defecto son “admin” y “1234” respectivamente).

La estructura del archivo es la siguiente:

- Configuration: Contiene toda la configuración.
- BufferSize: Indica el tamaño del buffer de lectura. El buffer de lectura es el único buffer configurable. Los buffers de escritura son de tamaño variable. El tamaño máximo que puede tener el mismo es 1000 bytes, ya que este es el tamaño máximo de línea soportado.
- ServerPort: Indica el puerto donde el proxy escuchará pedidos de conexión.
- pop3Port: Indica el puerto del servidor por defecto.
- Servername: Representa la dirección por la cual se accede al servidor.
- L33t y Rotation: Indican si están habilitadas las respectivas transformaciones. Pueden valer “true” o “false”. En el archivo de configuración por defecto las mismas se encuentran deshabilitadas.
- Admin: contiene el puerto desde el cual se accede a la consola de administración, y el usuario y pass con los que se debe conectar el admin.
- WelcomeMessage y GoodByeMessage: Representan los mensajes que imprime el administrador junto con el código “+OK” cuando se entra y cuando se sale de la consola, respectivamente.
- Multiplexing-config: Contiene un elemento “Multiplex” por cada usuario que desee acceder a un servidor distinto del default, el cual contiene el nombre de dicho usuario, y la dirección y puerto a los cuales se lo debe redirigir.

VIII - Ejemplos

A continuación se presenta un ejemplo de Configuration.xml

```
<Configuration>
   <BufferSize>1024</BufferSize>
   <ServerPort>7070</ServerPort>
   <pop3Port>110</pop3Port>
   <Servername>localhost</Servername>
   <L33t>false</L33t>
   <Rotation>true</Rotation>
   <Admin>
      <user>admin</user>
      <pass>1234</pass>
      <port>9999</port>
   </Admin>
   <WellcomeMessage>ProxyPOP3 ready.</WellcomeMessage>
   <GoodByeMessage>ProxyPOP3 says goodbye.</GoodByeMessage>
   <Multiplexing-config>
      <Multiplex>
         <User>user</User>
         <Host>host</Host>
         <Port>1111</Port>
      </Multiplex>
      <Multiplex>
         <User>user2</User>
         <Host>host</Host>
         <Port>2222</Port>
      </Multiplex>
   </Multiplexing-config>
</Configuration>
```

IX - Diseño del proyecto

El proyecto se ideó originalmente con una fuerte separación de lo que consiste la lógica de Pop3 y el Proxy. Esto se ve reflejado en el uso de interfaces y clases abstractas que permiten que el servidor pueda usar un handler sin saber si por detrás hay un handler Http, Pop, etc. Esta idea fue difícil de mantener al largo plazo, ya que al tener que incorporar ciertas funcionalidades (como por ejemplo, que el administrador pueda activar y desactivar las transformaciones de Pop), se requería más tiempo con el que contamos para hacer bien la separación. Por la dificultad que representaba, por la falta de tiempo, y porque no era requisito, al final la división no fue más respetada, y ahora todas las clases interactúan entre ellas sin filtro. Lo ideal hubiera sido mantener la división, y hacer un buen código que nos permita abstraernos del protocolo (se habló de esto en *Posibles Extensiones*).

El proyecto también se pensó para trabajar con múltiples threads, basándonos en el patrón *Reactor*. La idea es simple, tener un único thread que es el que tiene el selector, y él al recibir un set de keys, mandar cada key a un worker (thread) diferente, que proviene de un pool de threads. Se utilizan librerias que controlan recurrencia, y se toma en cuenta posibles condiciones de carrera que perjudican el funcionamiento del sistema. El sistema se termina entregando con un único thread (como discutido en la sección *Problemas*), pero toda la lógica para que funcione con varios threads se mantuvo.

X - Testing

Se han realizado testeos con la herramienta JMeter. Se corrieron 100 y 150 clientes. Los mails utilizados fueron 3 los cuales contenían:

- una imagen
- RFC de HTML
- RFC de POP3

El tamaño total de estos 3 mails fue de 2.7MB.

En la primer prueba, con 100 clientes, el throughput fue de 11.04MB/sec. Por su parte, con 150 clientes, fue de 9MB/sec. Al aumentar la cantidad de clientes, el throughput se ve afectado. Estas pruebas fueron sin las transformaciones aplicadas (leet y rotación). Además, al analizar el árbol de respuestas (*View Results Tree*), se ve que todos los mails se recibieron correctamente.

Luego, al intentar correr muchos más clientes, 500 por ejemplo, JMeter tira una excepción de OutOfMemory.