Protocolos de Comunicación

Trabajo Práctico Especial - Entrega final

Proxy POP3

Autores:

Alan Ezequiel Karpovsky - 49746

Gonzalo Virgilio Castiglione - 49138

José Santiago Galindo - 50211

Resumen

El objetivo del presente informe es documentar el las decisiones que se tomarán para la implementación del trabajo práctico especial. En las siguientes hojas se desarrollarán las decisiones sobre librerías y estrategias de programación necesarias para la implementación del proxy POP3 solicitado por la cátedra.

Martes 06 de Noviembre de 2012

Índice

1. Introducción

El objetivo de este informe es documentar las estrategias de implementación que se utilizarán a la hora de construír el servidor proxy POP3 solicitado por la cátedra. En la primeras secciones se informará sobre los RFCs a tener en cuenta, como así también las librerías necesarias para la implementación. Luego se explicará el funcionamiento general del servidor proxy junto a los servicios que éste provee seguido por las estrategias de configuración. Por último se comentarán algunas problemáticas detectadas durante el análisis de los RFC y los casos de prueba a realizarse.

2. RFCs relevantes para el desarrollo del trabajo practico

- **RFC 1939:** Post Office Protocol - Versión 3
- **RFC 2045, 2046, 2047:** Multipurpose Internet Mail Extensions
- **RFC 822:** Standard for the format ofarpa internet text messages
- **RFC 2119:** Key words for use in RFCs to Indicate Requirement Levels

3. Librerías externas a utilizar

- **Log4j:** Logueo de informacion.
- **JodaTime:** Manejo de fechas.
- **apache.common.net.util.SubnetUtils:** Manipulación de direcciones de red.
- **org.apache.commons.codec.binary:** Codificación y decodificación de Base64

4. Estructura del servidor proxy

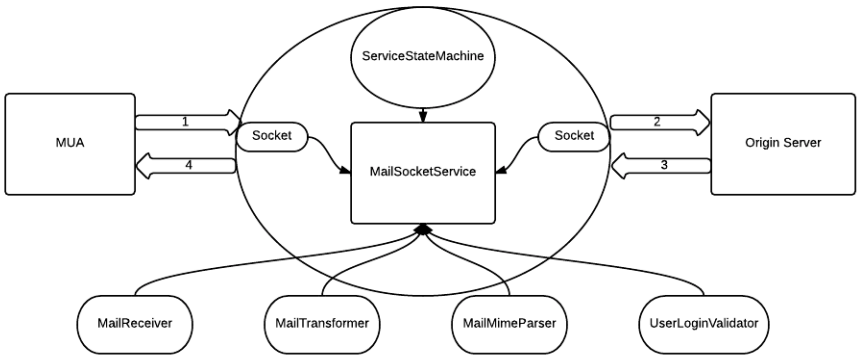
El servidor, en principio, proveerá 2 servicios:

Servicio de monitoreo (8081): El servicio de monitoreo sirve para poder visualizar, de manera remota, las estadísticas generales del servidor proxy. Basta con conectarse al puerto 8081 del servidor e ingresar la clave de administrador para la sección de monitoreo. Ejemplo de conexión al servicio de monitoreo:

```
nc localhost 8083
```

Servicio de configuración remota (8081): Por su parte, el servicio de configuración remota escuchará en el puerto 8082. En secciones posteriores se detalla el funcionamiento de este servicio.

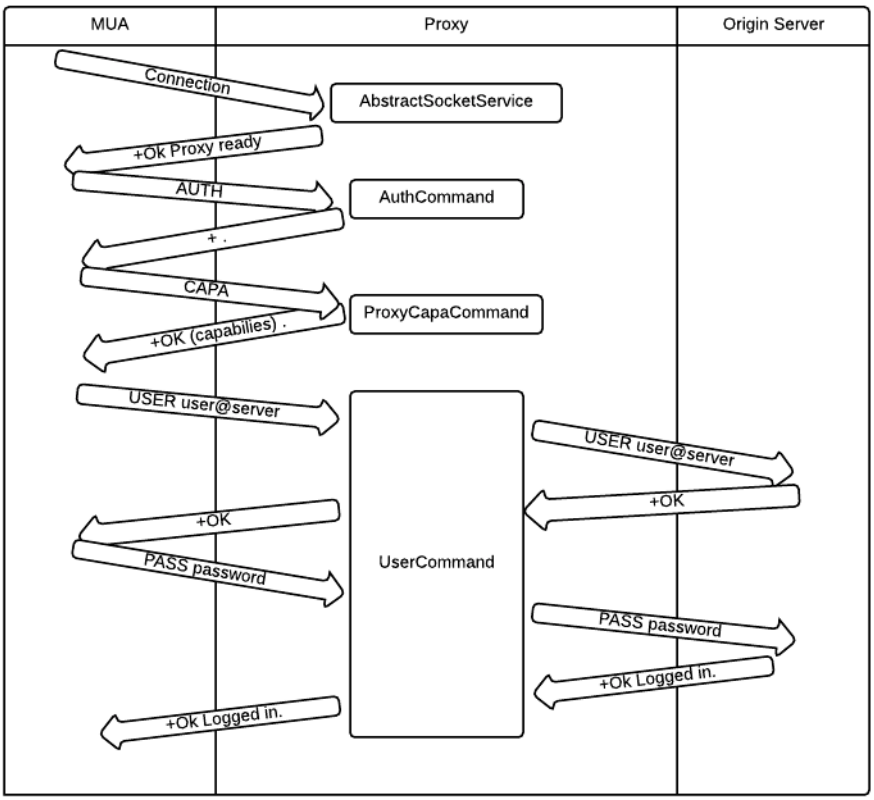
4.1. Arquitectura general



Código de AbstractSocketServer, método Run

```
public void run() {
	try {
		onConnectionEstabished();
		while (!endOfTransmission) {
			BufferedReader inFromClient = read();
			String clientSentence = inFromClient.readLine();
			if (clientSentence != null) {
				exec(clientSentence);
			} else {
				// Connection has been closed or pipe broken...
				endOfTransmission = true;
			}
		}
	} catch (Exception e) {
		logger.error("Exception on run(). Closing Connection.");
		e.printStackTrace();
	}
	try {
		onConnectionClosed();
	} catch (Exception e) {
		logger.error("Could not close connection " + e.getMessage());
	}
}
```

4.2. Ejemplo de Logueo Exitoso

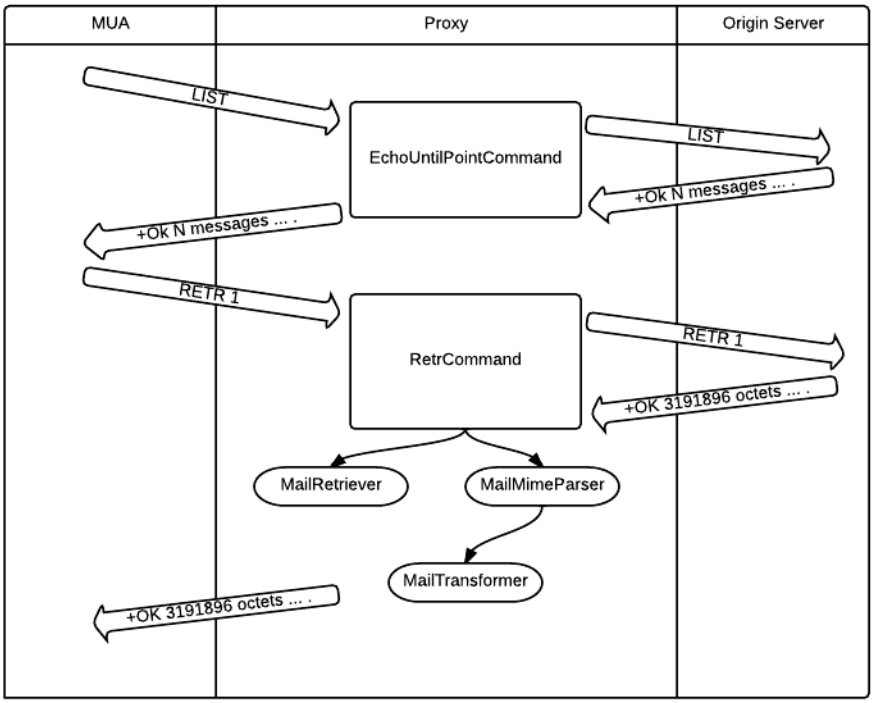


Multiplexación de cuentas

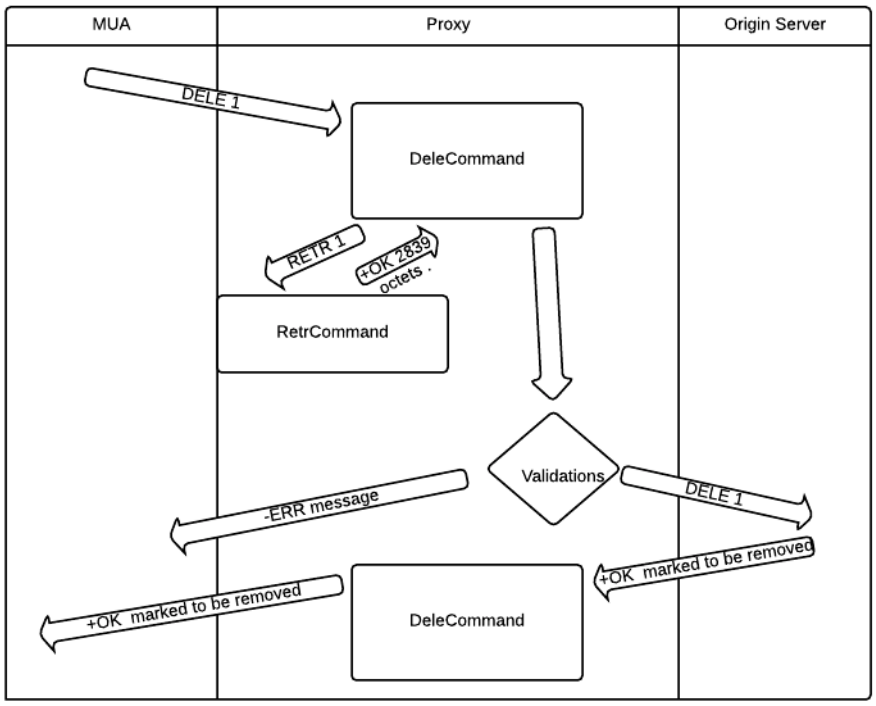
```
// UserCommand.java

private static final KeyValueConfiguration originServerConfig =
    Config.getInstance().getKeyValueConfig("origin_server");

public void exec(String[] params) {
    ...
    mailSocketService.setOriginServer(getMailServer(user));
    ...
}

private String getMailServer(User user) {
    String server = originServerConfig.get(user.getMail());
    return server == null ? originServerConfig.get("default") : server;
}

// MailSocketService.java

public String setOriginServer(String host) throws IOException {
    int port = Config.getInstance().getGeneralConfig().getInt("pop3_port");
```

```
    setOriginServerSocket(new Socket(host, port));
    String line = readFromOriginServer().readLine();
    logger.debug("Mail server new connection status: " + line);
    return line;
}
```

4.3. Ejemplo de Comando Retr Exitoso



4.4. Ejemplo de borrado

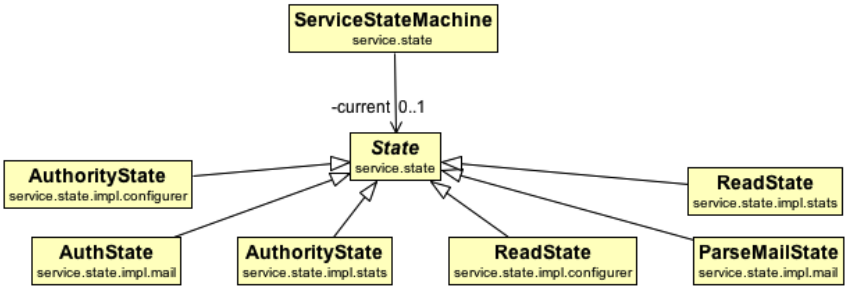


4.5. Máquina de estados

Se pensó la implementación como una máquina de estados donde, lógicamente, cada estado puede habilitar la ejecución de otros estados.

Esta estrategia fue útil al realizar la multiplexación de cuentas y a su vez nos permite tener un código simple, limpio y entendible a la hora de extender el desarrollo.

4.5.1. Arquitectura general: Máquina de estados

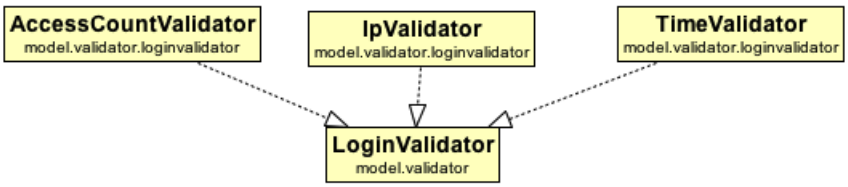


Ejemplo de clase estado

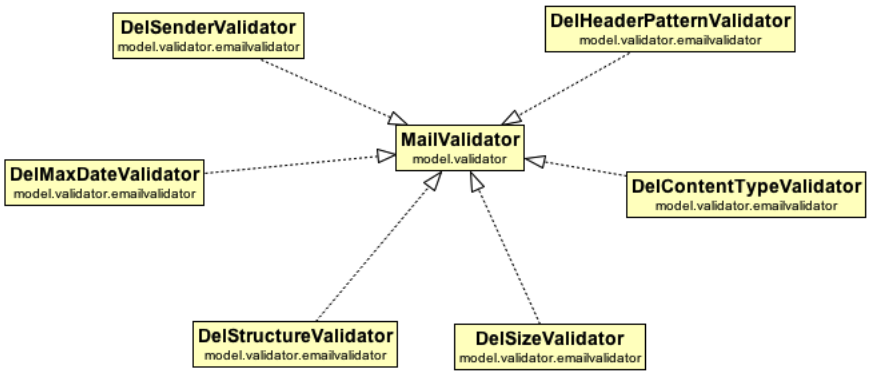
```
public class SomeState {

    ...
    // Each State registers all available commands
    commandRecognizer.register("PUT", PutCommand.class);
    commandRecognizer.register("DEL", DeleteLineCommand.class);
    commandRecognizer.register("GET", GetFileCommand.class);
    commandRecognizer.register("LIST", ListCommand.class);
    commandRecognizer.register("EXIT", LogOutCommand.class);
    ...

}
```

4.6. Control de accesos y validadores

4.7. Arquitectura general: Control de accesos



4.7.1. Arquitectura general: Validadores de borrado



4.7.2. Arquitectura general: Transformadores

Optimizaciones:

Se había optado por una estrategia multithread en la que se dedicaba un thread para parsear y convertir los datos del mail a tratar e ir escribiendolos a en un InputStream a medida que se encuentren listos, mientras que a la par, otro thread iba leyendo y enviando los datos al cliente.

Esta estrategia no pudo ser implementada debido a que las aplciaciones externas, reqiueren del mail completo para ejecutarse.

Código de la clase MailTransformer

```
public class MailTransformer {

    public void transformHeader(MimeHeader header) throws IOException {
        // Mime header transformation
    }

    public StringBuilder transformPart(Map<String, MimeHeader> partHeaders, StringBuilder
        part) throws IOException {
        // Mime part transformation
    }

    public void transformComplete(Mail mail) {
        // Full mail transformation (external apps)
    }
}
```

5. Configuración de la aplicación

La configuración del proxy se hará por medio de archivos que se escanearán permanentemente para detectar cambios en ellos. Por ejemplo, si que quieren agregar los filtros de ip, bastará con crear un archivo *bannedips.conf*, y copiarlo al directorio */config/filters/bannedips.conf*.

5.1. Archivos de configuración

A continuación se desarrollarán cada una de las distintas configuraciones soportadas por el servidor proxy como así también el formato de éstos.

5.1.1. origin_server.conf

Sirve para configurar un origin server distinto del default: se requiere la dirección del origin server deseado y el puerto. A través de este archivo se podría hacer el encadenamiento de proxys.

```
dest = mail.itba.edu.ar
port = 110
```

5.1.2. monitor.conf

Define todas las opciones de configuración referentes al servicio de estadísticas y monitoreo remoto que corre en el puerto 8081.

```
password = secreta
refresh_rate_ms = 2000
```

5.1.3. transformations.conf

Contendrá la lista de todas las modificaciones que se le deban aplicar a los mensajes de correo electrónico. Las transformaciones funcionan de manera global (no por usuario). Es decir, si se activa *l33t*, todos los mensajes que el servidor proxy forwardee a los MUA vendrán modificados con la estrategia *l33t*

```
l33t
rotateimages
hidesender
```

5.1.4. access_time.conf

Aquí se define el control de accesos por rangos horarios para cada uno de los usuarios

```
juan@zauberlabs.com = 9:00-11:00
juan@monits.com = 19:00-22:00
```

5.1.5. access_ip.conf

Aquí se define el control de accesos por IP

```
200.232.11.89
74.232.111.0/24
```

5.1.6. access_successfull_login.conf

Aquí se define el control de accesos por cantidad de logins exitosos. Los pares son del tipo [usuario, cantidad de logins admitidos por día]

```
juan@hotmail.com = 9
pepe@gmail.com = 2
jorge@gmail.com = 4
```

5.1.7. dest_servers.conf

Este archivo permite lograr que para un determinado usuario no se utilice el mail server default.

```
juan@gmail.com = pop3.alu.itba.edu.ar
```

5.1.8. notdelete_maxage.conf

Aquí se pondrán reglas para la eliminación condicional por antiguedad. Sólo se permitirá borrar un mail si la fecha es mayor a la fecha delcarada en este archivo.

```
juan@gmail.com = 15/02/2005
```

5.1.9. notdelete_sender.conf

Los usuarios que estén presentes en este archivo tendrán completamente prohibido el borrado de mensajes.

```
juan@gmail.com = foo@bar.com, foo2@bar.com
```

5.1.10. notdelete_header_pattern.conf

Sirve para definir borrados condicionales en base a headers que contengan los mensajes.

```
juan@gmail.com = List-Id == <foo.example.org>
```

5.1.11. notdelete_contenttype.conf

Sirve para definir borrados condicionales en base al tipo de archivos adjuntos.

```
juan@gmail.com = doc, avi
juan@gmail.com = pdf
```

5.1.12. notdelete_size.conf

Sirve para definir borrados condicionales en base al tamaño del mensaje.

```
juan@gmail.com = 10000
alan@gmail.com = 1000
```

5.1.13. notdelete_structure.conf

```
hasAttachments
```

6. Protocolos a desarrollarse: Configuración remota

Para configurar el servidor de forma remota lo que se debe hacer es conectarse a un servicio que éste provee en el puerto **8082** y hablar el **Protocolo de Configuración Remota** definido en la siguiente sección.

6.1. Protocolo de Configuración Remota

El protocolo de configuración remota es del tipo *request - response*, cada vez que el cliente ejecuta un comando, el servidor le responderá con **+OK** o **-ERR** en caso de una ejecución satisfactoria o una ejecución con errores respectivamente. Asimismo es un protocolo orientado a texto, más puntualmente orientado a linea.

6.1.1. Ejecución satisfactoria

En el caso de una ejecución satisfactoria, el servidor enviará el mensaje **+OK** y el *status code* 0. Se reservan los códigos de estado desde el 0 al 99 para respuestas relacionadas con ejecuciones correctas en futuras extensiones del protocolo.

```
+OK 0
```

6.1.2. Ejecución con errores

Los *status code* del 100 en adelante están intencionados para casos en los que el servidor deba comunicar cualquier tipo de error. El servidor puede incluír una entidad conteniendo una explicación de la situación de error e informando si ésta es una situación permanente o temporaria.

```
-ERR ERROR_CODE [HUMAN READABLE ERROR MESSAGE]
```

Los programas externos que implementen este protocolo podrían mostrarle al usuario cualquier entidad incluída junto a los códigos de error con el fin de ser más amigables para con el usuario.

Error codes	
100	Unrecognized command
101	Invalid parameters: missing arguments
102	Invalid parameters: file does not exists
103	Invalid parameters: unrecognized user
104	Invalid parameters: not a number
111	Invalid password

6.2. Comandos soportados

El servicio de configuración soporta los comandos: **AUTH**, **LIST**, **GET**, **PUT**, **DEL**, **EXIT**. A continuación se explicará en detalle el funcionamiento de cada uno de ellos.

6.2.1. AUTH password

El método **AUTH** sirve para autenticarse contra el servidor de configuración. Recibe como único parámetro la contraseña con la cual el cliente que se quiere autenticar.

Posibles códigos de estado relacionados con el método **AUTH**: 0, 200

6.2.2. LIST

El método **LIST** imprime una lista de todos los comandos soportados por el servidor. No recibe parámetros. Es importante destacar que se impone como fin del mensaje una línea conteniendo únicamente el caracter punto (ASCII period).

Posibles códigos de estado relacionados con el método **LIST**: 0, 100 Requiere estar autenticado para poder utilizarlo.

6.2.3. GET filename.conf

El método **GET** recibe como único parámetro un string que contiene el nombre de un archivo de configuración e imprime el contenido de éste. Al igual que el método *LIST*, utiliza el caracter punto para avisar el fin de su ejecución.

Posibles códigos de estado relacionados con el método **GET**: 0, 100, 101, 102 Requiere estar autenticado para poder utilizarlo.

6.2.4. PUT filename.conf string

El método **PUT** recibe dos parámetros: el nombre de un archivo de configuración y un *string* en este órden. Lo que hace es agregar el *string* recibido como parámetro en una nueva línea al final del archivo de configuración.

Posibles códigos de estado relacionados con el método **PUT**: 0, 100, 101, 102 Requiere estar autenticado para poder utilizarlo.

6.2.5. DEL N filename.conf

El método **DEL** rebice un número entero N y el nombre de un arcvhivo de configuración. Su función es eliminar la línea N del archivo en cuestión.

Posibles códigos de estado relacionados con el método **DEL**: 0, 100, 101, 102 Requiere estar autenticado para poder utilizarlo.

6.2.6. EXIT

El método **EXIT** varía su semántica según el usuario esté o no esté autenticado en el servidor de configuración: En caso de estar autenticado, el comando **EXIT** cierra la sesión; caso contrario, cierra la conexión con el servidor.

Posibles códigos de estado relacionados con el método **EXIT**: 0

6.3. Ejemplo de configuración remota

A continuación se presenta un ejemplo de configuración remota del servidor en el que el cliente es *kickeado* por ingresar su contraseña de manera errónea 3 veces seguidas:

```
S: +OK 0 [Server ready]
C: AUTH myUltraSecretPass
S: -ERR 200 [Invalid password]
C: AUTH anothedPassword
S: -ERR 200 [Invalid password]
C: AUTH forgottenPassword
S: -ERR 103 [Too many login attempts, try later.]
```

En el siguiente ejemplo el administrador del servidor consulta los comandos disponibles, imprime el archivo de control de accesos y luego bannea a un nuevo usuario por IP:

```
S: +OK 0 [Server ready]
C: AUTH myPassword
S: +OK 0 [Password accepted]
C: LIST
S: ADD
S: DEL
S: AUTH
S: LIST
S: PRT
S: EXIT
S: .  // Period marks end of transmission
S: +OK 0 [Command listed]
C: PRT access_ip.conf
S: 167.12.3.6
S: 162.11.23.5
S: .
S: +OK 0 [File access_ip.conf printed]
C: ADD access_ip 200.232.1.45
S: -ERR 102 [Configuration file not found]
C: ADD access_ip.conf 200.232.1.45
S: +OK 0 [File access_ip.conf updated]
S: 167.12.3.6
S: 162.11.23.5
S: 200.232.1.45
S: .
S: +OK 0 [File access_ip.conf printed]
C: EXIT
S: +OK 0 [Good bye]
C: EXIT
S: +OK 0 [Good bye] // Connection clossed
```

7. Protocolos a desarrollarse: Monitoreo y estadísticas

Para monitorear el servidor de forma remota lo que se debe hacer es conectarse a un servicio que éste provee en el puerto **8081** y hablar el **Protocolo de Monitoreo y Estadísticas**definido en la siguiente sección.

7.1. Protocolo de Monitoreo y Estadísticas

El Protocolo de Monitoreo y Estadísticas es del tipo *request - response*, cada vez que el cliente ejecuta un comando, el servidor le responderá con **+OK** o **-ERR** en caso de una ejecución satisfactoria o una ejecución con errores respectivamente. Asimismo es un protocolo orientado a texto, más puntualmente orientado a linea.

7.2. Comandos soportados

El servicio de configuración soporta los comandos: **AUTH**, **LIST**, **GET**, **PUT**, **DEL**, **EXIT**. A continuación se explicará en detalle el funcionamiento de cada uno de ellos.

7.2.1. AUTH password

El método **AUTH** sirve para autenticarse contra el servidor de configuración. Recibe como único parámetro la contraseña con la cual el cliente que se quiere autenticar.

Posibles códigos de estado relacionados con el método **AUTH**: 0, 200

7.2.2. LIST

El método **LIST** imprime una lista de todos los comandos soportados por el servidor. No recibe parámetros. Es importante destacar que se impone como fin del mensaje una línea conteniendo únicamente el caracter punto (ASCII period).

Posibles códigos de estado relacionados con el método **LIST**: 0, 100 Requiere estar autenticado para poder utilizarlo.

7.2.3. AUTO [start—stop]

El método **AUTO** recibe como único parámetro un string el cual podrá ser *start* o *stop*. Al ejecutar **AUTO start**, el servidor de estadísticas comenzará a imprimir en pantalla las estadísticas generales, refrescándolas cada N milisegundos; estando N definido previamente en el archivo de configuración *monitor.conf*. Por su parte, al ejecutar **AUTO stop**, se le indicará al servidor que se desea detener el *auto-refresh* de las estadísticas. Posibles códigos de estado relacionados con el método **GET**: 0, 100, 101 Requiere estar autenticado para poder utilizarlo.

7.2.4. HIST username

El método **HIST** recibe como único parámetro un string que contiene el nombre de un usuario y lo que hace es imprimir por salida estándar las estadísticas para el usuario en cuestión.

Posibles códigos de estado relacionados con el método **PUT**: 0, 100, 101, 103 Requiere estar autenticado para poder utilizarlo.

7.2.5. SHOW

El método **SHOW** no recibe parámetros e imprime en pantalla las estadísticas generales del servidor.

Posibles códigos de estado relacionados con el método **DEL**: 0, 100, 101, 102 Requiere estar autenticado para poder utilizarlo.

7.2.6. EXIT

El método **EXIT** varía su semántica según el usuario esté o no esté autenticado en el servidor de configuración: En caso de estar autenticado, el comando **EXIT** cierra la sesión; caso contrario, cierra la conexión con el servidor.

Posibles códigos de estado relacionados con el método **EXIT**: 0

7.3. Ejemplo de configuración remota

A continuación se presenta un ejemplo de configuración remota del servidor en el que el cliente es *kickeado* por ingresar su contraseña de manera errónea 3 veces seguidas:

```
S: +OK 0 [Server ready]
C: AUTH myUltraSecretPass
S: -ERR 200 [Invalid password]
C: AUTH anothedPassword
S: -ERR 200 [Invalid password]
C: AUTH forgottenPassword
S: -ERR 103 [Too many login attempts, try later.]
```

8. Problemas y dificultades detectados

8.1. Servicio y protocolo de configuración

Para implentar el servicio y el protocolo de configuración, en un primer momento se pensó es ir por una configuración orientada a archivos. Esta solución, cuya ventaja era era su rápida implementación y simplicidad, trajo consigo un conjunto de problemas:

- Imposiblidad de generar un protocolo coherente que funcionara con usuarios no humanos
- Imposibilidad para imprimir la configuración actual del proxy (ya que pasandole el archivo se pisaba los archivos de configuración anteriores sin siquiera poder consultar su previo valor)

8.2. Multiplexación de cuentas

El problema que surgió en este punto estuvo relacionado con la forma en que los MUAs se autentican son el servidor.

El nombre de usuario era requerido para saber con que servidor hablar, pero este se obtenia, a veces, luego de recibir pedidos de ejecucion de otros comandos.

Frente a esto se optó por emular un servidor hasta el momento de recibir el comando de login con el nombre de usuario.

8.3. Parser de mails MIME

El problema fue plantear una implementación del parser sin tener en cuenta que las partes de MIME pueden estar anidadas. Al descubrir esto se tuvo que hacer una remplementación del algoritmo de parseo, que se volvió considerablemente más complejo.

9. Limitaciones de la Aplicación

- Uso de Threads bloqueantes
- No utilización de SSL
- Baja escalibilidad

10. Posibles Extensiones

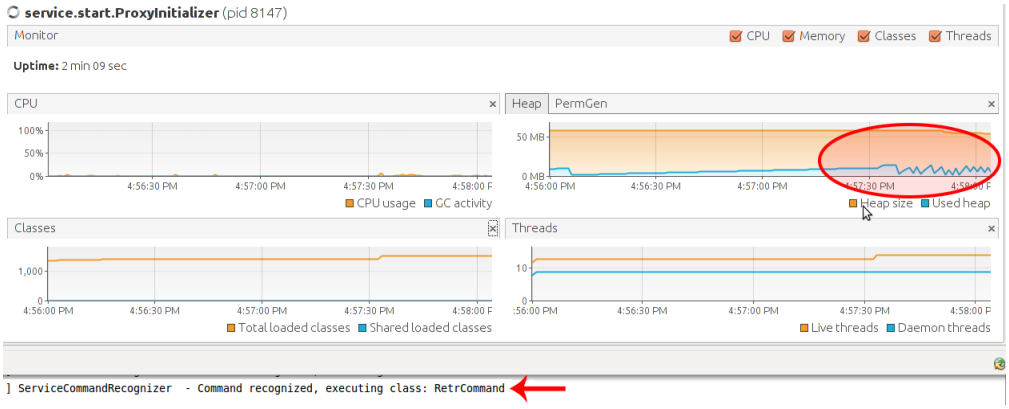
- Caches para mails para el caso de delete
- Uso de memoria exclusivo para mails pequeños
- Utilización de SSL

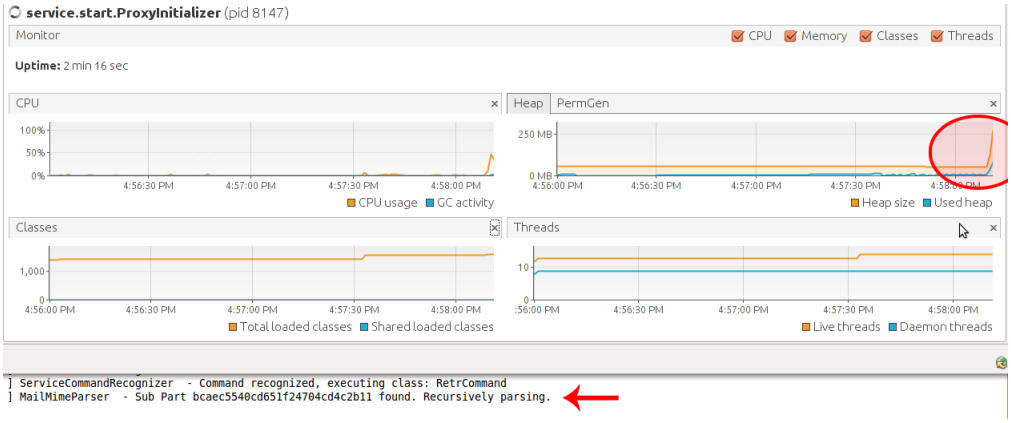
11. Casos de prueba

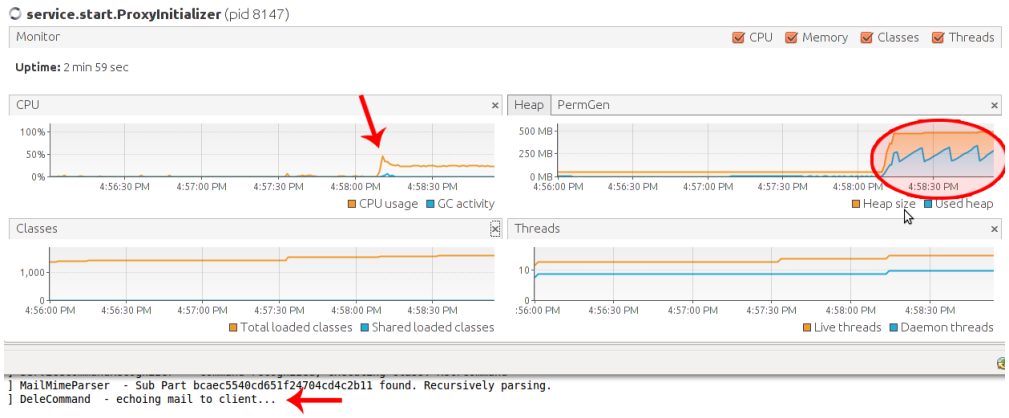
11.0.1. Tests

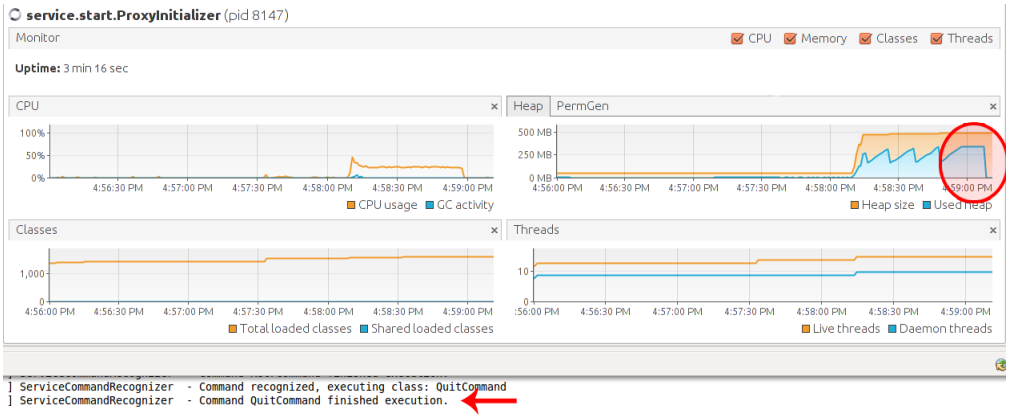
A continuación se muestran gráficos de consumo de CPU y memoria para el caso de un mail de 20 MB. La flecha roja indica el estado en el que se encontraba la aplicación a la hora de tomar la captura de pantalla. El proceso que se retrata contempla los siguientes estados:

1. Retriving mail: Consumo de memoria bajo
2. Recursive parsing: Consumo de memoria bajo/moderado
3. Echoing to client: Consumo de memoria alto
4. Finishing execution: Consumo de memoria bajo










```
] ServiceCommandRecognizer  - Command recognized, executing class: QuitCommand
] ServiceCommandRecognizer  - Command QuitCommand finished execution.
```

12. Conclusiones

En el transcurso de este trabajo se aprendío mucho, desde el funcionamiento de protocolos ampliamente usados como son POP3 hasta la implementación de un protocolo propio.
Queremos destacar especialmente la novedad de tener que toparse con las problemáticas relacionadas con la utilización de sockets en la realización de un programa de capa de aplicación, tanto al entender los protocolos implementados, como al momento de crear el protocolo de configuración.

En conclusión, fue un trabajo que demandó mucho esfuerzo y una programación muy cuidadosa, pero sentimos que nos aportó mucho al momento de entender los protocolos que gobiernan internet.

Una crítica, podría ser que tal vez, determinados requerimientos, como la rotación de imágenes, no nos sumaron mucho al entendimiento de la materia, agregando mucho tiempo de desarrollo, que si bien no fue desaprovechado podría haber sido usado en otro requerimiento más enriquecedor.

Temáticas destacadas que hemos comprendido luego de la realización del presente trabajo práctico:

- Protocolo POP3 extendido
- MIME
- Programación avanzada con threads y sockets
- Funcionamiento de TCP
- Probemáticas de la creación y definición de un protocolo

13. Guía de Instalación

Dependencias:
- Para la compilación del proyecto es necesario que el sistema cuente con Maven2. El cual, en el caso de Ubuntu, puede instalarse con el siguiente comando:

```
> sudo apt-get install maven2
```

- Una vez que el sistema cuenta con Maven, se procede a generar el ejecutable con el siguiente comando (desde la carpeta raíz del proyecto):

```
> mvn clean package
```