



# Trabajo Práctico

## Proxy POP3

Protocolos de Comunicación

Segundo Cuatrimestre 2018

Grupo 3

Integrantes:

Della Sala Rocío - 56507

Giorgi, María Florencia - 56239

Rodríguez, Ariel Andrés - 56030

Santoflaminio, Alejandro - 57042

# 1. Índice

Protocolo.....	2
Aplicaciones desarrolladas.....	6
Problemas encontrados.....	6
Diseño.....	7
Implementación.....	8
Limitaciones de la aplicación.....	10
Posibles extensiones.....	11
Conclusiones.....	12
Ejemplos de prueba.....	12
Guía de instalación.....	16
Instrucciones para la configuración.....	16
Ejemplos de configuración y monitoreo.....	16
Documento de diseño del proyecto.....	17

## 2. Descripción

### 2.1 Protocolo

Definimos el protocolo de administración entre el proxy y el cliente de configuración a partir de sus características más generales, para luego definir las funcionalidades y su estructura.

#### **Características**

En primer lugar, establecimos que queríamos implementar un protocolo orientado a caracteres, pero como la conexión entre el administrador y proxy es SCTP, nos dimos cuenta que era conveniente aprovechar el uso de datagramas para enviar las tareas a ejecutar. Finalmente, optamos por un *protocolo orientado a bit*. Este tipo de protocolo nos permitirá hacer un parseo más sencillo de los comandos en el proxy.

En segundo lugar, definimos el mecanismo de comunicación entre ambos siendo este *request/response*. De este modo se definen dos tipos de mensajes:

- Los tipo request que envían tareas a realizar como por ejemplo setear el programa externo o pedir información de una métrica.
- Los tipo response que envían la respuesta de ejecutar la tarea que el request específico. Dependen del tipo de request que se envió.

Por último, el protocolo es orientado a conexión debido a que mantenemos una sesión del cliente. Esto se realizó mediante la autenticación del mismo. Esto es similar a POP3 ya que se tendrán dos estados: uno de autenticación y otro de transacción.

#### **Funcionalidades (comandos)**

Se pensaron los comandos que el usuario pueda ejecutar según los requerimientos pero también en base a otras cuestiones como la seguridad del proxy, su futuro testeo, su independencia con programas externos entre otras cuestiones. Cada uno de los comandos tiene una posible respuesta, que incluye un estado y un mensaje de respuesta si es necesario.

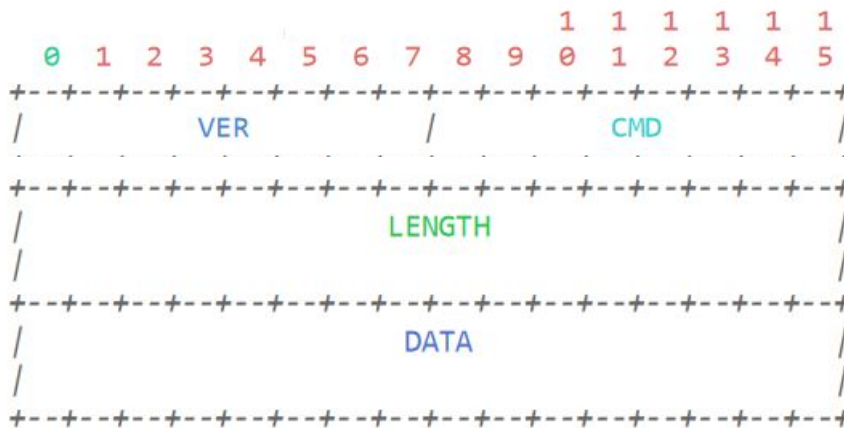
- Autorización
  - AUTH pass
  - Argumentos: Se envía como argumento la contraseña.

- Necesitamos autenticación, debido a que es conectado a sesión. Esto nos permitirá evitar controles como por ejemplo que el administrador nunca intentara cambiar el programa transformador por un comando bash que pueda ser peligroso para el funcionamiento del proxy (rm).
- Devuelve +OK en caso satisfactorio o -ERR si la contraseña especificada es incorrecta.
- Setear el programa externo de transformación
  - SET TRANSFORMATION PROGRAM transformationprogram
  - Argumentos: Se envía como argumento el programa transformador.
  - Útil para que la implementación del proxy no quede restringida al correcto funcionamiento del programa de transformación. Además nos permite probar con otros programas además del mimeFilter para detectar si los errores (si los hay) provienen del funcionamiento del proxy o del programa externo.
- Obtener el programa externo de transformación
  - GET TRANSFORMATION PROGRAM
  - Argumentos: No tiene argumentos.
- Obtener una métrica
  - GET METRIC metric
  - Argumentos: Se envía como argumento una constante que define la métrica a utilizar. Tenemos 5 métricas en total:
    - CONEXIONES CONCURRENTES
    - MÁXIMAS CONEXIONES CONCURRENTES
    - ACCESOS HISTÓRICOS
    - BYTES TRANSFERIDOS
    - ADMINISTRADORES CONECTADOS ACTUALMENTE
  - Para el caso de los bytes transferidos estos representan tanto los comandos enviados por el cliente (que se envían al origin server) más los mails enviados al programa transformador si esta prendido.
  - Devuelve +OK en caso satisfactorio o -ERR si la métrica especificada no existe
- Apagar/Prender la transformación
  - SWITCH TRANSFORMATION PROGRAM
  - Argumentos: No tiene argumentos
  - Prende y apaga el programa transformador, de esta forma podemos pasar los mails devueltos por el servidor origen por un programa externo a voluntad nuestra.

- Obtener los mimes prohibidos
  - GET MIME
  - Argumentos: No tiene argumentos.
  - Comando para obtener los mimes que están siendo censurados por el programa transformador.
- Agregar un mime a la lista de mimes prohibidos
  - FORBID mime
  - Argumentos: Se envía como argumento el mime a agregar de la lista de mimes censurados. No se puede enviar una lista de mimes.
  - Comando para agregar un mime a la lista de los mimes que estan siendo censurados por el programa transformador. Permite prohibir todo un tipo, indicando "\*" como subtipo. Por ejemplo si hacemos FORBID image/\* prohibiremos todos los tipos image.
- Quitar un mime de la lista de mimes prohibidos
  - ALLOW mime
  - Argumentos: Se envía como argumento el mime a sacar de la lista de mimes censurados. No se puede enviar una lista de mimes.
  - Comando para remover un mime de la lista de los mimes que estan siendo censurados por el programa transformador.
  - Aclaración: Si teníamos censurado todo un tipo, no podemos luego permitir un subtipo de ese tipo. Es decir, si teníamos censurado image/\*, no podemos hacer ALLOW image/png, debemos hacer primero ALLOW image/\* y luego prohibir los subtipos deseados.
- Cerrar la sesión
  - QUIT
  - Argumentos: No tiene argumentos.

### Estructura del mensaje

A partir de las funcionalidades que deseabamos implementar, definimos la estructura del request

Request

VER: Un byte que detalla la versión del protocolo. Se agrega para poder pensar el protocolo escalable a otras versiones y mantener retrocompatibilidad con clientes con versiones más viejas.

CMD: Un byte que detalla el comando a ejecutar. Estos estarán definidos como constantes donde:

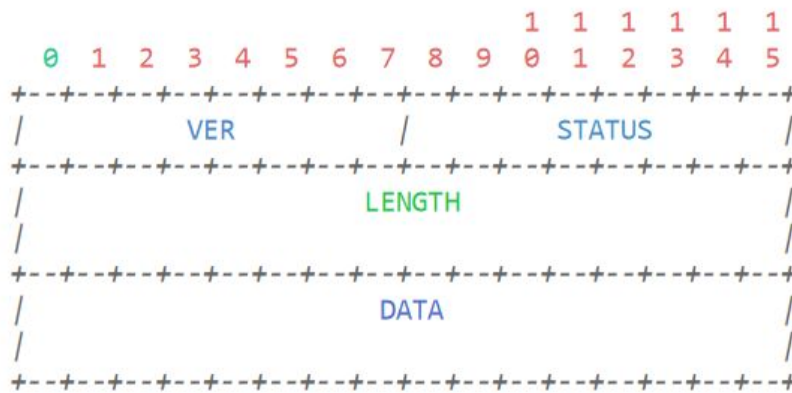
- 1 Autenticarse
- 2 Setear el programa externo de transformación
- 3 Obtener el programa externo de transformación
- 4 Apagar/Prender la transformación
- 5 Obtener una métrica
- 6 Obtener los mimes prohibidos
- 7 Agregar un mime a la lista de mimes prohibidos
- 8 Quitar un mime de la lista de mimes prohibidos
- 9 Cerrar la sesión

LENGTH: 4 bytes que indican (de existir un mensaje) el tamaño del mismo.

DATA: Mensaje de tamaño variable indicado parámetros de tipo texto.

Response

Para la respuesta algunos campos se mantienen igual que la estructura del request.



STATUS: 1 byte para especificar el estado de la respuesta que puede ser +OK indicando que la tarea se realizó satisfactoriamente o -ERR para indicar un mensaje de error

## 2.2 Aplicaciones desarrolladas

En el proyecto encontramos tres aplicaciones desarrolladas: el servidor proxy para el protocolo POP3, un cliente para configurar el servidor proxy y un programa cuya tarea es filtrar media types de un mail.

### 2.2.1 Pipelining

Soportamos pipelining tanto para los pedidos del cliente como para los pedidos del servidor. Se encolan los pedidos del cliente en una estructura `msg_queue` para poder procesarlos al llegar las respuestas del servidor.

Si no soporta pipelining, los comandos se desencolan de a uno y luego se envían. Si se soporta se desencolan varias request y se envían todas juntas. Cuando se recibe una respuesta se sabe el orden en el que llegan los comandos y a partir de este orden se procesa.

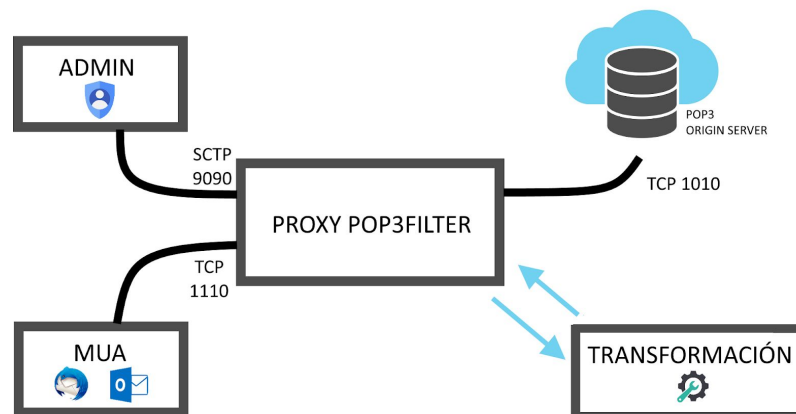
## 3. Problemas encontrados

Se presentaron varias dificultades durante la realización del trabajo:

- Muchos de los problemas encontrados se relacionan a que el buffer utilizado en las funciones de la máquina de estados principal es acotado a un tamaño de 30, por ende no se podía leer completamente una request (o un saludo) y/o enviar una respuesta de forma completa en un único send.
- Nos pasó que **al ejecutar pipelining había casos específicos en los cuales en una computadora funcionaba diferente que en otra**, entonces cada vez que se realizaba un cambio todos los miembros del equipo tenían que chequear si funcionaba.

- Falta de conocimiento en el uso de selectores a gran escala.
- Falta de conocimiento sobre los distintos tipos de formato que pueden presentarse en un mail.
- Problemas para filtrar un mail anidado dentro de otro. Finalmente se resolvió mediante el manejo de stack de boundaries.
- Un error donde los archivos de los casos de prueba estaban corrompidos e impedían el correcto funcionamiento del TP.

### 3.1 Diseño



El diseño del proxy se basó en los archivos provistos por la cátedra como `stm.c` y `selector.c`. El primero maneja la máquina de estados para los clientes que se conecten y el segundo los intereses de los file descriptors que se conectan al proxy y el flujo a través de la máquina de estados. El archivo que maneja el flujo principal de la aplicación y une la máquina de estado con el selector es `pop3nio.c`.

La estructura más importante del trabajo es `pop3` ubicada en `pop3nio.h` que contiene información de una conexión, el file descriptor del cliente conectado, el file descriptor del servidor origen al cual nos estamos conectando, información de la sesión, los buffers que fueron definidos con un límite de 45 para ahorrar espacio, los file descriptor de los pipes de la transformación externa, la request actual del cliente y la response del servidor, entre otras estructuras.

Una aclaración importante respecto al parser del response es que **es muy estricto en cuanto a las respuestas del estado**.

Claramente si corremos con Dovecot como servidor origen no hay limitación y el trabajo funciona bien. Pero si nos hacemos pasar por el servidor origen haciendo netcat en una terminal, el parser detecta un error ante una respuesta que no



**comience con +OK o -ERR** (salvo en el saludo principal) o si por ejemplo ingresamos user sin ningún nombre de usuario. En estos casos no saltan excepciones pero sí cerramos la conexión con el usuario y le dejamos un mensaje *“-ERR Response error. Disconnecting ...”*.

Hicimos esto porque creíamos que el usuario no debería recibir nada inválido. De todas formas es una pequeña aproximación a un parser mucho mas sofisticado de respuestas POP3.

## 3.2 Implementación

### 3.2.1 POP3 FILTER

Una vez parseado el input, lo primero que se realizó fue establecer dos socket pasivos para escuchar conexiones entrantes. El primero es el socket para la conexión con el administrador, el mismo escucha en el puerto 9090 a través del protocolo SCTP y el otro es para las conexiones con los clientes, escuchando en el puerto 1110 a través del protocolo TCP.

Una vez configurados ambos sockets se los registra en el selector junto con los handlers para manejar las conexiones entrantes, escrituras y lecturas.

Para los clientes se definió una máquina de estados con dos estados terminales *DONE* para el caso satisfactorio y *ERROR* para el caso donde se produzca un error en la ejecución.

Cuando llega una conexión lo primero que se realiza es aceptarla y crear un cliente nuevo. Una vez hecho esto se procede al primer estado llamado *ORIGIN SERVER RESOLUTION* el cual realiza una resolución DNS para conectarse al origin server a través de un thread para que la misma no sea bloqueante.

Luego se pasa al siguiente estado *CONNECTING TO OS* en el cual nos conectamos al servidor.

Seteamos los intereses del file descriptor del cliente para leer y esperamos el mensaje de bienvenida del servidor en el siguiente estado *WELCOME READ* y escribimos el resultado en el file descriptor del cliente en el estado *WELCOME WRITE*.

Una vez conectados se pasa al estado *CAPA* se envía el comando *CAPA* para saber si el servidor soporta pipelining. Si en la respuesta del servidor se encuentra

esta característica entonces sabemos que podemos enviarle más de un mensaje a la vez.

Volvemos a setear los intereses del file descriptor del cliente para leer y esperamos que mande la request. Pasamos al estado `REQUEST` en el cual leemos las request de los clientes y se la parseamos. Si se produjo un error (por ejemplo el comando supera los cuatro caracteres) no se envía nada al servidor, sino que se le notifica al cliente del error cometido y se espera nuevamente que el mismo ingrese un request nuevo.

Si no se produjo un error primero se transforma el request en un buffer y se lo envía al servidor origen. Luego pasamos al estado `RESPONSE` en el cual parseamos la respuesta del servidor. Si el comando es un `RETR` o `TOP` (es decir devuelve un mail) entonces chequeamos si la transformación externa está activada, y si es así, le pasamos el mail a través de un pipe en el estado `EXTERNAL TRANSFORMATION`. Si no, se la envía directamente al cliente.

Para los administradores lo que se hace es leer el pedido del cliente, deserealizarlo en la estructura *request* utilizada, ejecutar alguna función en el caso que sea necesario como por ejemplo agregar una media type para filtrar, serealizarlo como *response* y devolver la respuesta.

### 3.2.2 ADMINISTRADOR

Lo que se realiza una vez conectados con el proxy es un read del input del usuario, se parsea lo que se ingresa de tal forma que esto no lo tenga que procesar el proxy y asegurarse de que lo que se esté mandando sea un request válido. Se lo serealiza y luego se lo envía. Una vez enviado, se bloquea esperando una respuesta para dicho comando. Al llegar, se deserealiza la respuesta y se la imprime en stdout.

### 3.2.3 STRIPMIME

Para la realización de este programa se hizo uso de los parches provistos por la cátedra en campus y se trabajó a partir de ellos.

Requiere de las variables de entorno `FILTER_MEDIAS` y `FILTER_MESSAGE`. De esta forma, en el programa se realiza un listado donde se almacenan los tipos con sus respectivos subtipos en forma de nodos a partir de la información recibida en `FILTER_MEDIAS`.

Se hace uso de una estructura para mantener el estado durante el parseo del mail, la cual contiene diferentes parsers para interpretar las diferentes partes de un correo.

Además de parsers, en esta estructura también almacenamos un stack de delimitadores correspondientes a los valores que tenemos como boundaries en un correo. Esto permite el filtrado de contenido en mensajes anidados.

Si no recibe ningún mensaje a reemplazar en la variable de entorno correspondiente, se reemplazará el contenido a filtrar con el texto "Parte reemplazada."

Por otro lado si no se recibe nada en FILTER\_MEDIAS el programa retornará sin transformar el texto, por lo tanto es importante saber que si se setea "./stripmime" como programa de transformación, es necesario tener al menos un media type censurado. Si no es así luego el cliente no podrá ver la transformación de su mensaje, ya que stripmime habrá retornado antes.

## 4. Limitaciones de la aplicación

Existen varias limitaciones en el proyecto. Una limitación a agregar fue que quedaron varios leaks de memoria al usar valgrind que por falta de tiempo no llegamos a quitar. De todos modos se utilizó esta herramienta a lo largo del desarrollo del proyecto y tanto el ejecutable del admin como el de stripmime son aquellos que menos errores tienen.

Dentro del pop3filter:

- No se loguea a través de un thread, por ende la operación puede llegar a bloquearse.
- El socket pasivo TCP escucha direcciones IPv6. No pudimos probar el caso 200 de prueba.
- Buffers acotados. Por ejemplo no se pueden ingresar parametros de mas de 40 caracteres.
- La aplicación no funciona y se cuelga si nos hacemos pasar por el origin server, pedimos un mail y estan las transformaciones activadas. Desconocemos la causa y a falta de tiempo no pudimos resolverlo.
- **Cuando elegimos ingresar un programa transformador al iniciar pop3filter** (es decir, haciendo -t ./example), **para luego filtrar mediante tal programa es necesario switchear la transformación a ON desde el admin.** No es tanto una limitación sino más una cuestión de diseño.

Dentro del administrador:

- Su comunicación con el proxy es bloqueante, es decir que al solicitar un comando se queda esperando la respuesta del mismo y no puede enviar otro.

Dentro de stripmime tenemos:

- La aplicación no funciona bien si se censura alguno de los tipos multipart.

- Al ejecutar el comando diff -u, se muestra como si hubiera diferencia entre el mensaje pasado con stripmime incluso si este no reemplazó ninguna parte. Haciendo diff -u y utilizando cat -t detectamos que al final de cada línea en el archivo original está el carácter “^M”, que se corresponde con el *carriage return*. No logramos detectar la causa, debido a falta de tiempo, sin embargo el contenido de los mails filtrados es el correcto y visualizando el resultado con un editor de archivos mbox el contenido es visualizado de manera correcta. Si usamos diff -b, se obtiene el resultado esperado.

## 5. Posibles extensiones

### 5.1 Buffers

La aplicación en general hace uso de buffers acotados, que aunque se contempla los casos para que no se genere Segmentation Fault, esto limita la implementación. Se podría definir el buffer con un tamaño de solo byte y enviar cada bytes apenas se reciben.

### 5.2 Autenticación del administrador

Se podría mejorar autenticación del administrador de modo tal que no se envíen en texto plano ya que con Wireshark puede ser interceptada fácilmente.

### 5.3 Administrador bloqueante

Se podría mejorar la comunicación del administrador utilizando un selector. De esta forma las lecturas y escrituras dejarían de ser bloqueantes y pasarían a implementarse a través de un mecanismo de suscripción.

### 5.4 Parser Response

Debido a que se parsea la response, en el caso que nos hagamos pasar por el origin server el parser no contempla muchísimos casos en los cuales el “servidor” intente romper la aplicación.

### 5.5 Censurar multipart

También se podría alterar el funcionamiento para que censure correctamente los tipos multipart de un mail, reemplazando todo el contenido correspondiente.

## 6. Conclusiones

La realización del trabajo permitió a todo el grupo obtener un aprendizaje profundo acerca de la implementación de un protocolo propio, así como también entender el funcionamiento de un protocolo POP3.

El trabajo requirió de mucho esfuerzo de todos, ya que era extenso y requirió de investigación en todos los ámbitos relacionados al proyecto.

También es importante destacar que se aplicaron conocimientos de la materia Sistemas Operativos.

Consideramos que los siguientes temas fueron en los que más aprendizaje desarrollamos:

- ❑ Protocolo POP3
- ❑ Media types
- ❑ Sockets
- ❑ Funcionamiento de SCTP y TCP

Por último, también adquirimos experiencia trabajando con herramientas de debuggeo como valgrind, address sanitizer y diferentes flags que se pueden utilizar para la compilación de un proyecto.

## 7. Ejemplos de prueba

Los siguientes comandos son ejemplos para ejecutar desde el directorio root del proyecto y entender el funcionamiento de los ejecutables.

Para ejecutar pop3filter podríamos hacerlo de la siguiente manera:

```
$ ./pop3filter localhost
```

Es decir tenemos que especificar la dirección del servidor origen como parámetro.

Luego para ejecutar el administrador desde otra terminal podríamos hacer:

```
$ ./pop3ctl 9090 127.0.0.1
```

Es decir tenemos que especificar el puerto y la dirección del servidor como parámetros.

De esta forma podremos correr los comandos de administrador mencionados anteriormente. Es necesario recordar que para loguearse como administrador hay que utilizar la contraseña *"protosgrupo3"*.

Luego, desde una tercera terminal, podríamos hacer:

```
$ nc -C localhost 1110
```

En este caso si tenemos, por ejemplo, Dovecot inicializado, podremos utilizar los comandos correspondientes y traer un mail que tengamos en nuestra carpeta correspondiente. Si no se modificó el programa de transformación, entonces debería traer el contenido del mensaje sin modificaciones ya que al iniciar la aplicación el programa se encuentra apagado y seteado como cat. A continuación se presentan algunos ejemplos.

Ejemplo 1: En la siguiente imagen podemos ver como utilizar el proxy como clientes si tenemos Dovecot corriendo en nuestras PCs. A la izquierda podemos ver los logs y a la derecha la vista del cliente.

The image shows a terminal window with two panes. The left pane displays Dovecot logs for a client connection from ::ffff:127.0.0.1:51826. The logs show the client sending 'user rocio', 'pass Rd12345', and 'list' commands, and the server responding with '+OK' and message counts. The client then sends 'quit' and disconnects. The right pane shows a netcat listener on port 1110 receiving the same commands and responses from the client.

```

rocio@rocio-Lenovo-G50-30 ~/Escritorio/SEGUNDO CUATRIMESTRE 2018/Protocolos de Comu
No errors found on input
Listening on TCP port 1110
Listening on SCTP port 9090
Waiting for connections ...
[2018-11-06T 14:32:51Z]: CLIENT connection success - ::ffff:127.0.0.1:51826
[2018-11-06T 14:32:51Z]: ORIGIN SERVER RESOLUTION success.
[2018-11-06T 14:32:51Z]: ORIGIN SERVER CONNECTION success.
[2018-11-06T 14:32:54Z]: REQUEST send - cmd user - args rocio
[2018-11-06T 14:32:54Z]: RESPONSE received - cmd: user - status:+OK
[2018-11-06T 14:32:57Z]: REQUEST send - cmd pass - args Rd12345
[2018-11-06T 14:32:57Z]: RESPONSE received - cmd: pass - status:+OK
[2018-11-06T 14:33:01Z]: REQUEST send - cmd list - args (null)
[2018-11-06T 14:33:01Z]: RESPONSE received - cmd: list - status:+OK
[2018-11-06T 14:33:05Z]: REQUEST send - cmd quit - args (null)
[2018-11-06T 14:33:06Z]: RESPONSE received - cmd: quit - status:+OK
[2018-11-06T 14:33:06Z]: CLIENT disconnected - ::ffff:127.0.0.1:51826

rocio@rocio-Lenovo-G50-30 ~ $ nc localhost 1110
+OK Dovecot (Ubuntu) ready.
user rocio
+OK
pass Rd12345
+OK Logged in.
list
+OK 13 messages:
1 2494
2 402
3 402
4 346
5 343
6 343
7 339
8 337
9 366
10 336
11 537
12 496
13 4300
.
quit
+OK Logging out.
rocio@rocio-Lenovo-G50-30 ~ $

```

**Figura 1**

Ejemplo 2: En la siguiente imagen podemos ver como utilizar el proxy con pipelining de comandos.

```

rocio@rocio-Lenovo-G50-30 ~/Escritorio/SEGUNDO CUATRIMESTRE 2018/Protocolos de Comu
nicación/Proxy POP3/IP-PROTOS $ ./pop3filter -p 5670 localhost
Option argument for '-p' is 5670
No errors found on input
Listening on TCP port 5670
Listening on SCTP port 9090
Waiting for connections ...
[2018-11-06T 14:35:12Z]: CLIENT connection success - ::ffff:127.0.0.1:58241
[2018-11-06T 14:35:12Z]: ORIGIN SERVER RESOLUTION success.
[2018-11-06T 14:35:12Z]: ORIGIN SERVER CONNECTION success.
[2018-11-06T 14:35:13Z]: REQUEST send - cmd user - args rocio
[2018-11-06T 14:35:13Z]: REQUEST send - cmd pass - args Rd12345
[2018-11-06T 14:35:13Z]: REQUEST send - cmd retr - args 5
[2018-11-06T 14:35:13Z]: REQUEST send - cmd quit - args (null)
[2018-11-06T 14:35:13Z]: RESPONSE received - cmd: user - status:+OK
[2018-11-06T 14:35:13Z]: RESPONSE received - cmd: pass - status:+OK
[2018-11-06T 14:35:13Z]: RESPONSE received - cmd: retr - status:+OK
[2018-11-06T 14:35:13Z]: RESPONSE received - cmd: quit - status:+OK
[2018-11-06T 14:35:13Z]: CLIENT disconnected - ::ffff:127.0.0.1:58241

rocio@rocio-Lenovo-G50-30 ~ $ printf 'USER rocio\nPASS Rd12345\nRETR 5\nQUIT\n'
c -C foo 5670
+OK Dovecot (Ubuntu) ready.
+OK
+OK Logged in.
+OK 343 octets
Date: Tue, 30 Oct 2018 02:22:13 -0300
From: Rocio <rocio@rociodellasala56507>
To: rociomail@rociodellasala56507
Subject: protos
Message-ID: <20181030052213.GA17906@rociodellasala56507>
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Disposition: inline
User-Agent: Mutt/1.5.21 (2010-09-15)
Lines: 1

holaaaaa!!
.
+OK Logging out.
rocio@rocio-Lenovo-G50-30 ~ $

```

Figura 2

Ejemplo 3: En la siguiente imagen podemos ver como utilizar el proxy haciéndonos pasar por el origin server. Esto es invisible para el usuario quien salvo por el mensaje de saludo no sabrá que no está corriendo Dovecot.

```

rocio@rocio-Lenovo-G50-30 ~/Escritorio/SEGUNDO CUATRIMESTRE 2018/Protocolos de Comu
nicación/Proxy POP3/IP-PROTOS $ ./pop3filter -P 9898 localhost
Option argument for '-P' is 9898
No errors found on input
Listening on TCP port 1110
Listening on SCTP port 9090
Waiting for connections ...
[2018-11-06T 14:38:59Z]: CLIENT connection success - ::ffff:127.0.0.1:51842
[2018-11-06T 14:38:59Z]: ORIGIN SERVER RESOLUTION success.
[2018-11-06T 14:38:59Z]: ORIGIN SERVER CONNECTION success.
[2018-11-06T 14:39:24Z]: REQUEST send - cmd user - args rocio
[2018-11-06T 14:39:29Z]: RESPONSE received - cmd: user - status:+OK
[2018-11-06T 14:39:34Z]: REQUEST send - cmd pass - args rocio
[2018-11-06T 14:39:51Z]: RESPONSE received - cmd: pass - status:+OK
[2018-11-06T 14:39:55Z]: REQUEST send - cmd list - args (null)
[2018-11-06T 14:40:03Z]: RESPONSE received - cmd: list - status:+OK
[2018-11-06T 14:40:04Z]: REQUEST send - cmd quit - args (null)
[2018-11-06T 14:40:11Z]: RESPONSE received - cmd: quit - status:+OK
[2018-11-06T 14:40:11Z]: CLIENT disconnected - ::ffff:127.0.0.1:51842

rocio@rocio-Lenovo-G50-30 ~ $ nc localhost 1110
Bienvenido a POP3 !!
USER rocio
+OK Hola RO!
PASS rocio
+OK Autenticada
list
+OK La lista de mensajes es:
1
2
3
.
quit
+OK
rocio@rocio-Lenovo-G50-30 ~ $

```

Figura 3

También funcionan otros ejemplos en los cuales tenemos prendido el programa transformador y podemos ver el mail solicitado con los comandos RETR o TOP modificado o por ejemplo enviamos pipelining con espacios detras de los comandos y estos no se ven afectados.

### Stripmime (standalone):

Podríamos probar también, ejecutar stripmime de forma independiente.

El siguiente comando filtrará los tipos image del archivo ii\_images.mbox y la salida se podrá ver en el archivo out generado en el directorio root.

```
$ FILTER_MEDIAS=image/* ./stripmime <./test_cases/ii_images.mbox>
out
```

Con respecto a stripmime corriendo el archivo grande de prueba provisto por la cátedra (i\_big.mbox) pudimos obtener los siguientes tiempos para los siguientes comandos:

- 1) Primero filtrando un media type no presente en el archivo:

```
export FILTER_MEDIAS=a/b
time cat i_big.mbox | ./stripmime | pv > out
real    1m33.663s
user    1m28.835s
sys     0m4.509s
```

- 2) Luego filtrando un media type presente en el archivo, cuyo contenido era extenso.

```
export FILTER_MEDIAS=application/x-cd-image
time cat i_big.mbox | ./stripmime | pv > out
real    1m21.334s
user    1m21.096s
sys     0m1.604s
```

Podemos compararlo con el tiempo que se obtiene al realizar cat con el mismo archivo:

```
time cat ../test_cases/i_big.mbox > out
real    0m9.727s
user    0m0.000s
sys     0m0.587s
```



## 8. Guía de instalación

Al igual que como se indica en el archivo `Readme.md`, se pueden compilar todos los archivos realizando desde el directorio principal del proyecto:

```
$ cmake .  
$ make all
```

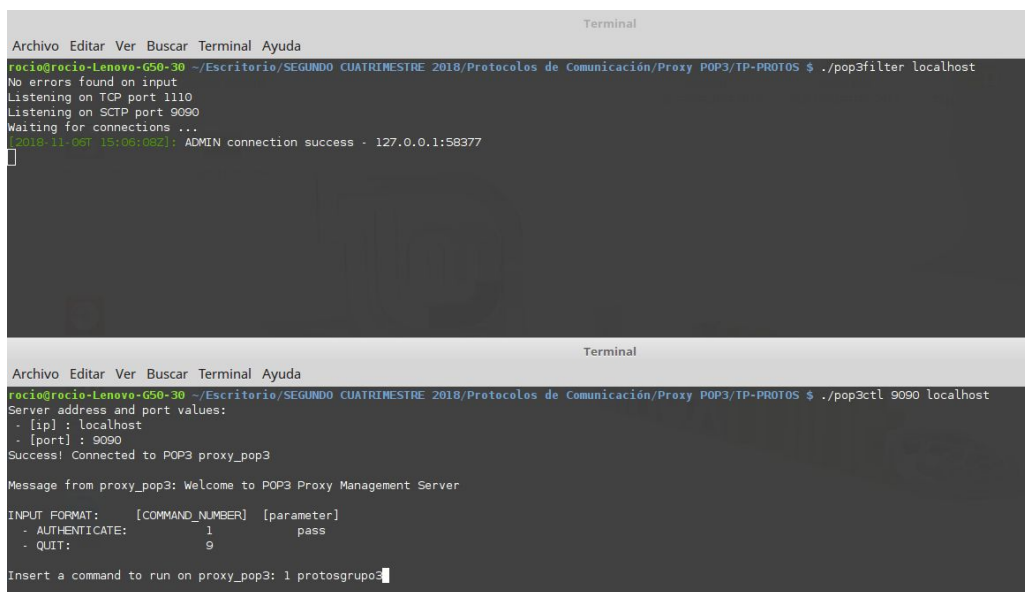
Esto generará tres ejecutables en dicho directorio. Bastará ejecutarlos como se indica en la sección *7 Ejemplos de Prueba*.

## 9. Instrucciones de configuración

Con respecto a la configuración que podemos realizar al loguearnos como admin ejecutando `pop3ctl`, se pueden ejecutar los comandos ya mencionados.

### 9.1 Ejemplos de configuración y monitoreo

En las siguientes imágenes podemos ver un ejemplo de como correr el administrador , el menú que se despliega al ingresar y al estar autenticado y enviar diferentes comandos.



```
Terminal  
Archivo Editar Ver Buscar Terminal Ayuda  
rocio@rocio-Lenovo-G50-30 ~/Escritorio/SEGUNDO CUATRIMESTRE 2018/Protocolos de Comunicación/Proxy POP3/TP-PROTOS $ ./pop3filter localhost  
No errors found on input  
Listening on TCP port 1110  
Listening on SCTP port 9090  
Waiting for connections ...  
[2018-11-06T 15:08:08Z]: ADMIN connection success - 127.0.0.1:58377  
[  
]  
  
Terminal  
Archivo Editar Ver Buscar Terminal Ayuda  
rocio@rocio-Lenovo-G50-30 ~/Escritorio/SEGUNDO CUATRIMESTRE 2018/Protocolos de Comunicación/Proxy POP3/TP-PROTOS $ ./pop3ctl 9090 localhost  
Server address and port values:  
- [ip] : localhost  
- [port] : 9090  
Success! Connected to POP3 proxy_pop3  
  
Message from proxy_pop3: Welcome to POP3 Proxy Management Server  
  
INPUT FORMAT: [COMMAND_NUMBER] [parameter]  
- AUTHENTICATE: 1 pass  
- QUIT: 9  
  
Insert a command to run on proxy_pop3: 1 protogsupos
```

**Figura 4**

```

Insert a command to run on proxy_pop3: 1 protosgrupo3
Answer from proxy_pop3: +OK
Now you are authenticated!

INPUT FORMAT:
[COMMAND_NUMBER]      [parameter]
- SET TRANSFORMATION PROGRAM: 2      transformationprogram
- GET TRANSFORMATION PROGRAM: 3
- SWITCH TRANSFORMATION PROGRAM: 4
- GET METRIC: 5      metric
                        where metric is:
                        CONCURRENT CONNECTIONS 0
                        MAX CONCURRENT CONNECTIONS 1
                        HISTORICAL ACCESSES 2
                        TRANSFERED BYTES 3
                        CURRENT ADMINS CONNECTED 4
                        MAX ADMINS CONNECTED 5

- GET MIME: 6
- ALLOW MIME: 7      mime
- FORBID MIME: 8      mime
- QUIT: 9

Insert a command to run on proxy_pop3: 2 sed s/o/0/g
Answer from proxy_pop3: +OK

Insert a command to run on proxy_pop3: 3
Answer from proxy_pop3: +OK: sed s/o/0/g

Insert a command to run on proxy_pop3: 8 text/image
Answer from proxy_pop3: +OK

Insert a command to run on proxy_pop3: 6
Answer from proxy_pop3: +OK: text/image

Insert a command to run on proxy_pop3: 5 4
Answer from proxy_pop3: +OK: current admins connected: 1

Insert a command to run on proxy_pop3: 9
Answer from proxy_pop3: +OK
See you later!

```

**Figura 5**

## 10. Documento de diseño del proyecto

A continuación se presenta el mismo gráfico mostrado anteriormente respecto al diseño del proyecto:

