72.07 Protocolos de Comunicación
Informe Trabajo Práctico Especial: “Servidor para el protocolo POP3”

Grupo 3

Integrantes:

- Marcos Gronda, 62067
- Máximo Gustavo Rojas Pelliccia, 62353
- Marcos Casiraghi, 62003

Tabla de Contenidos

1. Prefacio

Este informe presenta el desarrollo del trabajo práctico especial de la materia 72.07 - Protocolos de Comunicación del primer cuatrimestre del año 2023. El objetivo de este fue implementar un servidor para el protocolo POP3 y en simultáneo desarrollar un protocolo propio para el monitoreo y configuración de dicho servidor.

2. Protocolos

2.1 POP3 (RFC 1939, RFC 2449)

Teniendo que desarrollar un servidor que utilice el protocolo de POP3, es solo natural que como equipo tuvimos que afianzarnos con este protocolo e implementarlo. Se incorporaron las funcionalidades que tanto el RFC 1939, como el RFC 2449 describen como obligatorias, estas siendo: 'USER', 'PASS', 'CAPA', 'STAT', 'LIST [msg]', 'RETR <msg>', 'DELE <msg>', 'RSET', 'NOOP', 'QUIT' .

La implementación, permite a un cliente acceder y administrar sus correos electrónicos. Se debe destacar que la implementación de dicho servidor se hizo de tal forma que sea no bloqueante. En otras palabras, el servidor en ningún momento debería esperar para leer o escribir a un cliente o de un archivo. Esto lleva a una mejor utilización de los recursos de un servidor y evita que los clientes tengan que esperar de más. El servidor utiliza como protocolo de transporte a TCP que es confiable y orientado a conexión.

2.2 M^3

Para desarrollar el protocolo de monitoreo y configuración, se decidió optar por utilizar UDP como protocolo de transporte. Esto implicó que es más simple la lógica del protocolo dado que no es necesario establecer una conexión o manejar el caso en cual el cliente solo pudo mandar parte del comunicado. Creíamos también que sería enriquecedor probar usar UDP, dado que nuestra implementación del protocolo POP3 solo usa TCP.

Siguiendo con la directiva de desarrollar un protocolo simple y fácil de implementar, decidimos que este sea orientado a binario y no orientado a texto como es POP3. Esto eliminaría la necesidad de diseñar ambos parsers, uno para el servidor y otro para el cliente.

Finalmente, M^3 implementa un sistema de seguridad en base de tokens (números de 32 bits). Este permite a un potencial cliente autenticarse al hacer un pedido al servidor, por lo que, debe ser compartido de forma segura con cualquier cliente que lo necesite.

A diferencia de otros protocolos, M^3 distingue entre los requests y los response. Dicho esto, ambos cuentan con un header y pueden tener una longitud variable.

2.2.1 Estructura de un Request

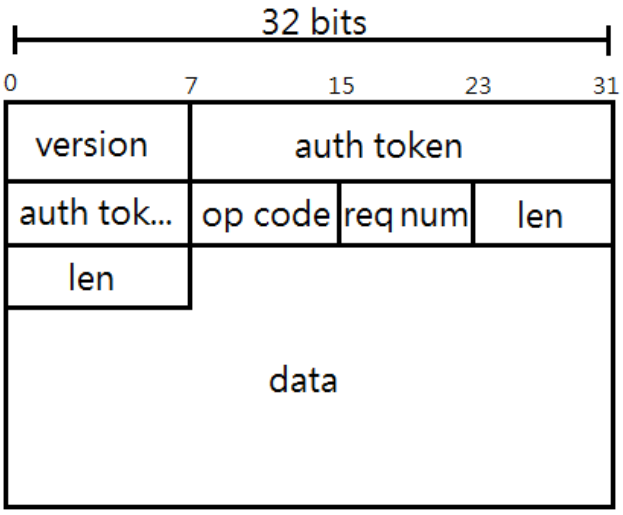


Figura 1: estructura de Request

Versión: similar a otros protocolos, el primer campo que se encuentra en el request es la versión del protocolo. Le permite al cliente indicar que versión quiere usar durante la transacción. Consiste de número de 1 byte, sin signo.

Nota: toda versión siguiente del protocolo usará este primer byte para indicar la versión.

Authentication Token: como se mencionó anteriormente, permite al usuario autenticarse. Consiste de un número de 4 bytes, sin signo.

Operation Code: el cliente indica qué operación quiere realizar. Consiste de un número de 1 byte, sin signo.

En la siguiente tabla se indican las operaciones válidas, con sus argumentos y valores de retorno:

Op Code	**Nombre**	**Operación a ejecutar**	**Valor de retorno (en data)**	**Argumentos**
0	GET_BYTES_SENT	Obtiene el total de los bytes enviados por el servidor POP3.	Número de 4 bytes	-
1	GET_BYTES_RECEIVED	Obtiene el total de los bytes recibidos por el servidor POP3.	Número de 4 bytes	-

2	GET_TOTAL_CONNECTIONS	Obtiene el total de las conexiones históricas al servidor POP3.	Número de 4 bytes	-
3	GET_CURR_CONNECTIONS	Obtiene el total de las conexiones concurrentes al servidor POP3.	Número de 4 bytes	-
4	ADD_USER	Agrega un nuevo usuario al servidor. POP3.	-	Nombre de usuario, seguido por ":", seguido por contraseña.
5	NOOP	Ninguna operación	-	-

(Ver apartado 10 para la lista de los posibles comandos).

Restricciones para el comando ADD_USER:

1) El nombre del usuario y la contraseña no pueden contener el valor ":" y debe tener como longitud mínima 1 byte y máxima 256 bytes.
2) El usuario y la contraseña deben estar separados por únicamente el carácter ":"
3) La combinación de usuario y contraseña debe ser null terminated.
4) El nombre de usuario debe ser único.

Request number: permite al usuario identificar el request con un número. El servidor mandará la respuesta con el mismo valor. Consiste de un número de 1 byte, sin signo.

Length: el usuario debe indicar la longitud de la sección de *data* en bytes. Consiste de un valor de 2 bytes, sin signo. Puede tomar valores entre 0 y 522.

Data: aquí se encuentra cualquier parámetro o argumento para la operación pedida. Tiene como longitud máxima 522 bytes.

2.2.2 Estructura de un Response

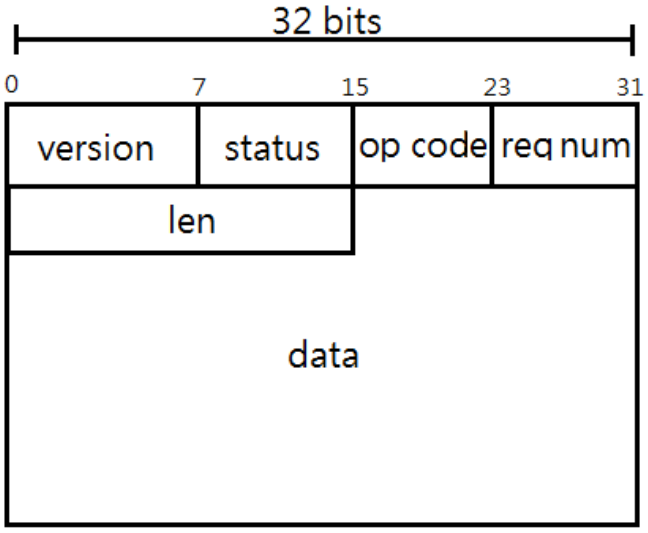


Figura 2: estructura de Response

Versión: indica la versión del protocolo que está usando el servidor. Consiste de un número de 1 byte, sin signo.

Nota: toda versión siguiente del protocolo usará este primer byte para indicar la versión. Si el cliente hace un pedido con una versión no soportada por el servidor, se usa este campo para indicar qué versión debería usar.

Status: indica el estado de la operación. Consiste de un número de 1 byte, sin signo.

Status	Nombre	Descripción
0	SUCCESS	La operación se ejecutó de forma exitosa.
1	INVALID_VERSION	La versión del request no es soportada.
2	INVALID_OPCODE	La operación del request no es válida.
3	INVALID_TOKEN	El número de autenticación (auth token) no es válido.
4	INVALID_ARGS	Los argumentos enviados para dicha operación son inválidos

Operation Code: el servidor envía el número de la operación que se ejecutó. Consiste de un número de 1 byte, sin signo.

Request number: el servidor reenvia el valor de *request number* del request. Consiste de un número de 1 byte, sin signo.

Length: el servidor debe indicar la longitud de la sección de *data* en bytes. Consiste de un valor de 2 bytes, sin signo. Puede tomar valores entre 0 y 522.

Data: aquí se encuentra la respuesta de la operación ejecutada. Tiene como longitud máxima 522 bytes.

3. Aplicaciones desarrolladas

3.1 Main

La aplicación main (archivo main.c) se encarga de configurar inicialmente el servidor y ejecutarlo. Es decir: cuando se inicia la aplicación, primero analiza los argumentos recibidos y configura el servidor con ellos.

A continuación, el main es el encargado de abrir los sockets pasivos para tanto el servidor POP3 como el servidor de monitoreo y configuración, haciéndolo en el puerto 44443 y 55552 por defecto respectivamente.

Finalmente, se inicializa el selector en el cual se suscriben el timeout y el handler para el servidor POP3 y el de monitoreo y configuración, así comenzando la ejecución de las rutinas de atención correspondientes.

3.2 Servidor POP3

3.2.1 Funcionamiento General

El servidor implementado, utiliza un selector de manera no bloqueante utilizando la syscall “pselect()” la cual le permite monitorear un máximo de 1024 file descriptors en simultáneo. Dependiendo del estado actual del cliente y el servidor durante la conexión, el selector permite suscribir los fd a distintas operaciones, lectura o escritura. Sobre esto, a cada fd se le asigna un handler propio para cada acción que realice.

La interacción entre el cliente y el servidor se lleva a cabo de acuerdo a la definición del RFC 1939. Primero, ante una conexión, el socket pasivo recibe la conexión y se genera un socket activo donde el cliente va a tener almacenada toda su información e interactuar con el servidor. Luego, el handler de ese socket activo se suscribe a escritura y se envía un saludo al cliente dándole la bienvenida así entrando en el estado de “AUTHENTICATION”. Es en este donde el usuario debe emitir las credenciales, previamente inicializadas por parámetro en el Main. El servidor siempre contesta a una acción de “USER” con un “+OK”, para no generar vulnerabilidades en cuanto a qué usuarios son correctos y cuáles no. Al momento de emitir “PASS”, si el usuario no existe, o la contraseña es incorrecta devuelve “-ERR”, solo así devolviendo “+OK” en el caso de que ambas credenciales sean válidas y mudando al cliente, ahora autenticado, al estado de “TRANSACTION”. Es en este estado que el cliente tiene acceso al manejo y lectura de la casilla de correos asociada al usuario (el usuario debe tener un subdirectorio con su nombre dentro del directorio general enviado como argumento al Main en formato de una ruta

absoluta). Finalmente, el tercer y último estado es el de “UPDATE” al que solo se puede llegar desde el estado de “TRANSACTION” al emitir el comando “QUIT”. En este estado el servidor procede a enviar un saludo de despedida al usuario e intentar eliminar los mails previamente marcados como eliminados en estado anterior. Si se logran eliminar, el servidor envía “+OK goodbye!” y en el caso contrario “-ERR some messages not deleted”. En ambos casos se procede a terminar la conexión con el cliente así cerrando el socket activo previamente generado.

3.2.2 Correos electrónicos

Es necesario indicar, al momento de iniciar el servidor, donde es que están almacenados las casillas de correos con el indicador de -f en los argumentos. Este directorio debe tener subdirectorios dentro, llamados con el nombre de usuario que luego se crearán con el argumento -u. Estos funcionarán como casillas independientes de cada usuario. (Ver 10 para un ejemplo del directorio raíz)

Una vez que un usuario termine el estado de “AUTHENTICATION”, el servidor irá a buscar dentro del directorio, el subdirectorio que tenga su nombre. Si existe uno, lo abrirá y leerá los nombres de los archivos, que funcionan como correo, y los almacenará en la memoria propia de la conexión con el cliente hasta llegar a 100. Una vez alcanzados, dejará de leer (ver puntos 3 y 4 de 5) y continuará con la ejecución normal. En el caso que no se de una ruta existente, o el usuario no tenga un subdirectorio a su nombre, los pasos anteriores no ocurrirán y el usuario no tendrá disponible ningún correo.

Decidimos que en el caso de que se elimine un correo desde fuera del servidor durante la ejecución de este, que el nombre del mensaje siga existiendo en memoria, por lo que puede seguir siendo referenciado. En el caso de intentar leerlo, como por ejemplo, con un ‘RETR’, este indicara que se accedió al mensaje pero luego dará error (ver 8 para ejemplo visual). En el caso de que se intente eliminar por el medio de ‘DELE’, este fallara al momento de hacer ‘QUIT’ en el estado de ‘UPDATE’ y avisara que no se pudo eliminar. En el caso de que se agreguen nuevos mensajes, estos no estarán disponibles durante la misma ejecución en la que se agregaron, teniendo que reiniciar el proceso de autenticación para hacerlo.

3.2.3 Estados de conexión

Por un lado, la conexión con el cliente se desarrolla en varios pasos, algunos mantienen los lineamientos del protocolo pop3 mientras que otros no. La primera diferencia es que el estado de “AUTHENTICATION” está dividido en 2, el primero siendo 'auth_ini' y el segundo 'auth_password'. En el primero, el usuario solo puede ejecutar los comandos de 'user', 'capa' y 'quit' y en el segundo 'pass', 'capa' y 'quit'. Mientras el usuario ingrese como argumento al comando 'user' uno invalido, se mantendrá en 'auth_ini' aunque se devuelva "+OK", solo cambiando su estado en uno correcto. Una vez en 'auth_password' tiene 2 opciones posibles, avanzar a 'transaction', que se mantiene fiel al rfc 1939 dando una contraseña correcta, o retroceder al primer pseudo estado de 'authorization' si la contraseña es incorrecta o se emite un comando erróneo.

El estado de 'transaction' como se dijo en el párrafo anterior, se mantiene fiel al rfc 1939, con la única diferencia siendo la transición al estado de 'update'. Decidimos que este no tenía sentido pues el usuario, autenticado o no, debe realizar los mismos pasos para cerrar la conexión que tiene abierta con el servidor a excepción de una, eliminar mensajes.

Es por eso que la acción de eliminar mensajes no tiene un estado propio, si no que al momento de cerrar la conexión, la acción se fija si el usuario viene de el estado de 'transaction' o no, procediendo a buscar y eliminar mensajes en el caso de que si.

Finalmente, todos los usuarios, sin importar su estado, al ejecutar quit transicionan al estado de 'client_finished' indicando que están listos para cerrar la conexión.

Por otro lado, el servidor tiene dos estados, uno de lectura y otro de lectura que van alternando. Explicar cómo y cuándo se suscriben

3.2.4 Parser

Analizando los casos de uso del socket activo, reconocimos que era necesario que el parser pueda mantener un estado con el fin de poder realizar una aplicación no bloqueante. Es posible que un comando no esté completo al momento de pasarlo al parser. Por estos motivos, la consumición del parser recibe una flag “finished” que indica si terminó de consumir un comando (si reconoce el \r\n al final).

La interpretación de comandos funciona de la siguiente manera:
El parser tiene internamente un arreglo con la información necesaria de todos los comandos disponibles. También, la estructura del parser tiene un arreglo booleano con una posición por cada comando; En cada posición de este arreglo mantiene un registro de si ese comando puede llegar a ser uno válido a partir de los caracteres que se pasaron hasta ese momento.

Cuando se inicializa el parser, se setea el estado en “COMMAND_RECOGNITION”, donde a medida que recibe caracteres descarta o no los comandos. Si no reconoce ningún comando (el arreglo mencionado está completo), termina indicando que es un comando invalido. En cambio, si todos los caracteres coincidieron para un comando, se marca el estado del parser como “WITH_ARGUMENTS_COMMAND” o “NO_ARGUMENTS_COMMAND” dependiendo del comando (si puede recibir argumentos o no)

Cuando el parser reconoce un comando que no debe recibir argumentos, se verifica que no haya ningún parámetro (ignorando espacios) y que termine correctamente, con ´\r\n´. En caso que no lo haga, se lo considera como comando invalido.

Cuando el parser reconoce un comando que puede recibir un argumento, continúa consumiendo caracteres para guardar el argumento. Si detecta que no se pasó ningún argumento cuando terminó el comando, se cambia el tipo de comando al de ningún argumento. Cuando consumió este tipo de comandos, se verifica que la longitud del argumento no exceda el máximo definido por el rfc.

Por último, cuando termina un comando, se prende el flag “finished” y se reinician las variables del parser para que esté listo para consumir más comandos.

3.2.5 Pipelining

Para manejar el pipelining y la ejecución de varios comandos sin tener que esperar a la respuesta del servidor, decidimos que el parser no debía encargarse de tener que interpretar varios comandos a la vez. Sino que consuma de a uno y que se encargue el socket activo de llamar reiteradas veces al parser si es que sigue habiendo comandos para interpretar en el buffer.

3.2.6 Logging

Durante la implementación del logging, se usó como principio guiante la frase mencionada durante una clase práctica: “menos es más”. Con eso en mente, decidimos usar estas reglas:

1) El nivel INFO, solo debería contener información en lenguaje no técnico, de tal forma que un usuario no experto podría entender que está ocurriendo. De la misma forma, solo se registran eventos importantes en la ejecución del servidor, como su correcta inicialización o cuando un usuario se autentica.
2) En el nivel DEBUG, se encuentra la mayoría de los eventos que se registran. Nuevamente se intenta sólo registrar los eventos importantes que ocurren pero, ahora sí se permite el uso de lenguaje técnico, por lo que, permite registrar más eventos.
3) En el nivel ERROR, se encuentran la mayoría de los errores que ocurren durante la ejecución del servidor, excluyendo aquellos que causen que el servidor tenga que frenar su ejecución. Estos, se encuentran en el nivel FATAL. No es ninguna coincidencia que la mayoría de los eventos en ERROR, ocurren al atender al cliente, como por ejemplo, no poder abrir un archivo de mail.
4) Como se mencionó previamente, en en nivel FATAL, solo se encuentran eventos de los que no se puede recuperar y por lo tanto, se debe cortar la ejecución.

3.4 Rendimiento del servidor

3.4.1 Tamaño de buffer

El servidor usa buffers a la hora de leer y escribir a un socket, y también leer un archivo de mail. Estos buffers tienen un tamaño fijo, predeterminado. Una pregunta que nos preguntamos al implementar el servidor fue: ¿Qué tamaño se le debería dar a dichos?

Para responder esta pregunta, analizamos el tiempo de transmisión de un mail con distintos tamaños de buffers. Para minimizar el impacto de los procesos como el establecimiento de la conexión, la autenticación, etc, se utilizó el comando curl y su funcionalidad de POP3. De la misma manera, se generó un archivo de 2.2 GB para ser transmitido.

A continuación se muestran los resultados:

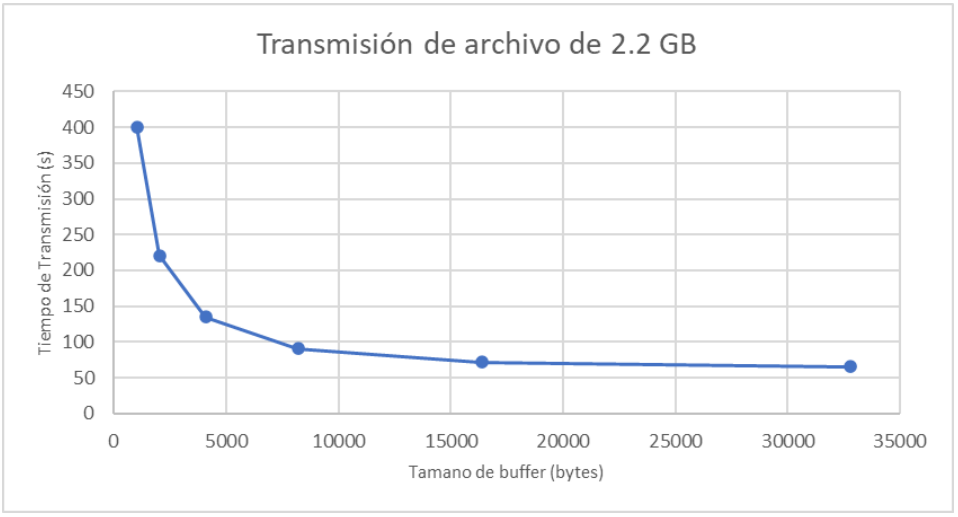


Figura 3: transmisión de archivo de 2.2GB

Como se puede observar, el tiempo de transmisión decrementa a medida de que se aumenta el tamaño de los buffers de forma exponencial. Se notó que entre 1024 y 2048 bytes, decrementa el tiempo de transmisión por un 45%, entre 2048 y 4096 bytes, un 38% y 4096 y 8192 bytes solo un 32%.

Con el fin de optimizar los tiempos de transmisión, pero al mismo tiempo disminuir la cantidad de recursos usados por el servidor, se eligió usar buffers de 8192 bytes. Se calculó que, si se tienen 500 clientes, cada uno con 3 buffers (de entrada, salida y lectura de archivo) de 8192 bytes, solo se usaría 11.7MB de memoria. Esta cifra es relativamente pequeña considerando que muchas de las computadoras hoy en día tienen 4, 8 o 16 GB de RAM.

3.4.2 Degradación de performance

Con el fin de ver cómo se degrada el rendimiento del servidor al aumentarse la cantidad de clientes, se diseñó un experimento en el cual se incrementaron la cantidad de usuarios haciendo *retr* de un mail grande. Se probaron con 1 a 5 clientes, que hacían pedidos de un mail de 2.2GB. Creemos que de esta forma, se pondría al límite al servidor y pondría en evidencia si hay una notable degradación en el rendimiento.

A continuación se muestran los resultados:

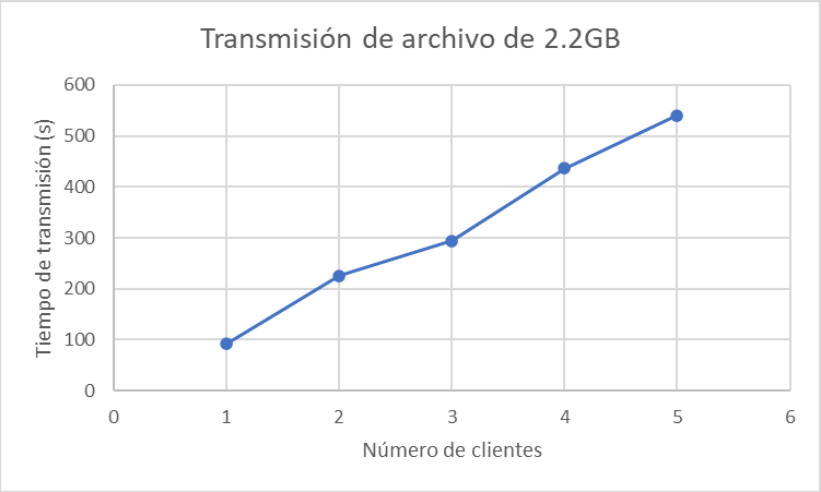


Figura 4: transmisión de archivo de 2.2GB para varios clientes

Como bien se puede observar, la degradación de la performance, ejemplificada en el aumento en la cantidad de tiempo que tarda las transmisiones, sigue una trayectoria lineal. Este resultado es el esperado ya que el servidor usa un único hilo de ejecución, por lo que, consideramos que el servidor tiene un uso eficaz de los recursos.

3.5 Monitoreo y configuración del servidor

Se creó una estructura llamada server_metrics, en la cual se almacenan los datos de cantidad de conexiones concurrentes, históricas, cantidad de bytes recibidos y enviados durante el periodo de vida del servidor. Cuando el socket pasivo del servidor pop3 recibe una nueva conexión, el main (ver 3.1) aumenta tanto la cantidad de conexiones históricas

como la concurrente, luego decrementando las concurrentes cuando se cierra la conexión con un cliente. Por otro lado, abordando la cantidad de bytes, se cuentan todos los bytes que recibió por parte de un cliente al momento de conectarse y los que manda el servidor, incluyendo los saludos. Es con estos datos que se puede contestar a los pedidos del cliente manager.

4. Problemas encontrados

4.1 Parseo

El primer problema que afrontamos fue diseñar un parser para POP3 que permita no solo el uso de *pipelining*, sino que también pueda manejar el caso en cual se recibe una transmisión del cliente incompleta. Por este motivo, quedó claro que era necesario mantener un estado del parser por cada conexión cliente. Para resolver los casos de una transmisión incompleta, el parser considera como incompleto al comando pasado hasta que lee un '\r\n'. El uso de un parámetro puntero "finished" indica si está listo para ser procesado el comando interpretado. Con respecto al pipelining, decidimos que el parser no debería ocuparse de interpretar varios comando a la vez, sino que consuma de a uno y que se encargue el handler del socket activo a llamarlo otra vez si es que sigue habiendo contenido para consumir en el buffer de lectura del cliente.

4.2 Máquina de estados

Otra dificultad que encontramos surgió al revisar el código que habíamos producido para la escritura y lectura de los sockets, lo cual nos hizo preguntarnos: ¿Realmente tiene sentido usar una máquina de estados? Nos dimos cuenta que, debido a que solo usábamos la máquina de estados para manejar el entrada y salida, lo cual llevaba a que esta tenga solo 2 estados posibles, *lectura* y *escritura*, más algunos estados auxiliares, no tenía sentido complejizar el diseño usandola.

Somos conscientes de que deben haber maneras más elegantes de manejar el estado del cliente pero creemos que pudimos encontrar un balance entre compresibilidad, simplicidad, y eficiencia.

4.3 Apertura de directorio con emails

Una incógnita que enfrentamos fue un error que ocurría cuando se intentaba abrir el directorio donde se encuentran los mails. El servidor, una vez que se registra exitosamente un usuario, abre el directorio pasado por parámetro (con el flag -f, ver 3.2.2) con el nombre del usuario anexado para conseguir la información necesaria de los mails. De todas formas, a un integrante del equipo, solamente le funcionaba correctamente la apertura del directorio si es que se hacía un printf de la ruta absoluta a abrir, mientras que para los demás no hacía falta. Después de pruebas en pampero para verificar la funcionalidad, este paso falló para todos. Con eso nos dimos cuenta que habían errores lógicos de la función que anexaba la ruta con el nombre del usuario y se pudo solucionar. Más allá de la solución, seguimos sin entender porque es que funcionaba localmente o porque el printf solucionaba el error para el integrante.

4.4 Sobre M^3

Este protocolo lo implementamos después de haber terminado el servidor pop3. Dado que el servidor requiere una conexión, fue necesario crear un socket pasivo para escuchar conexiones de TCP. Cuando quisimos desarrollar el M^3, consideramos hacer el protocolo orientado a conexión para reutilizar el comportamiento de la creación del socket. De todas formas, concluimos que es más apropiado hacer que el protocolo se maneje con UDP. Esto implicó tener que hacer otra configuración para el socket. A partir de esto, surgieron un par de problemas al configurar el socket que requirieron de un análisis extenso para poder resolverlos.

5. Limitaciones

- Una limitación en el parser de comandos del servidor pop3 es que este solo contempla comandos que únicamente pueden recibir a lo sumo un argumento. Esto fue para simplificar el código del parser dado que los comandos posibles cumplen esta condición. Si posteriormente se quiere agregar un comando como “TOP”, será necesario reescribir la estructura y lógica del parser para contemplar este caso.
- La implementación del servidor solo soporta hasta 20 usuarios, por lo que solo podrían abrirse hasta 20 casillas en simultáneo y solo 20 de todos los clientes conectados van a poder utilizar el máximo del servidor.
- La implementación del servidor puede acceder hasta 100 mails en el directorio de cada usuario. Se decidió esto dado a que, en la mayoría de los casos de uso, el usuario obtiene un mail y luego lo elimina, por lo que se consideró razonable no complejizar la aplicación y simplemente tener un máximo definido.
- Solo se leen, y por ende son accesibles, los primeros 100 correos electrónicos almacenados en el subdirectorio del usuario iniciado sesión. Se eligió hacerlo de tal manera, para simplificar el flujo de inicialización de un usuario, con la justificación de que el enfoque del trabajo práctico no es este.
- Actualmente, la cantidad de conexiones concurrentes está limitada por la cantidad de file descriptors que es capaz de monitorear la función pselect() utilizada por el Selector, ésta siendo 1024. Actualmente, tenemos: 1024- 2 (stderr, stdout) - 2 (socket de pop3 + M^3) = 1020 / 2 = 510 (en el peor caso posible) ya que un usuario dado tiene un fd para su socket activo y puede llegar a tener 1 más debido a la acción “RETR” para abrir y leer un correo.

6. Posibles extensiones

Una posible extensión al servidor de pop3 presentado podría ser completar las funcionalidades posibles del para el cliente. Esto involucra extender la lista de comandos posibles, incluyendo a APOP, TOP y UIDL, comandos que actualmente no están disponibles. Como se mencionó previamente en las limitaciones, realizar este cambio implicaría tener que cambiar la implementación del parser dado que este no puede interpretar más de un argumento por comando y el comando TOP, debe recibir dos argumentos.

Por otro lado, también se podría extender el protocolo de monitoreo. Actualmente, el administrador tiene una lista limitada de comandos que solamente permite informarle sobre cuantos conexiones hubieron, la cantidad de usuarios actualmente conectados y la cantidad de bytes transferidos. Se podría expandir esta lista para incluir comandos como saber la cantidad de mails que fueron leídos por los usuarios o la cantidad de mails que fueron borrados sin haber sido leídos. Estas son algunas de las estadísticas que podría tener disponible el administrador para saber en mayor profundidad como es el estado del servidor pop3.

Asimismo, también se podría agregar más funcionalidades que realicen cambios en tiempo de ejecución como: un comando para bloquear/remover a cierto usuario o para cambiar la contraseña de uno por si el cliente se la olvido y requiere otra.

7. Conclusiones

Durante el desarrollo de este proyecto creemos que pudimos poner a prueba los conocimientos adquiridos de la parte teórica y práctica de la materia. Al principio invertimos bastante tiempo en el diseño del servidor pop3 y más adelante al del protocolo de monitoreo. Pudimos implementar el protocolo de correo pop3 habiendo escuchado su historia en las clases.

Descubrimos que con solo saber la teoría y práctica no alcanza para plasmarlo en el trabajo práctico. Fue un trabajo largo de mucho aprendizaje e investigación donde pudimos comprender en mayor profundidad cómo es que funciona la tecnología informática que se sigue utilizando hoy en día.

Consideramos que pudimos organizarnos como equipo y llegamos a un resultado que cumple satisfactoriamente con el enunciado. Teniendo en cuenta también que el cierre del cuatrimestre tiene varias entregas y parciales, pudimos dividir los tiempos para completar este proyecto en tiempo y forma.

8. Ejemplos de prueba

-Flag de help cuando se inicializa el servidor pop3

```
 2023/Protocolos de Comunicacion/TP/pop-server/src$ ./server.out -h
INFO: main.c:96, Server initializing...
Usage: ./server.out [OPTION]...

   -h               Imprime la ayuda y termina.
   -p <POP3 port>   Puerto entrante conexiones POP3.
   -P <conf port>   Puerto entrante conexiones configuracion.
   -u <name>:<pass> Usuario y contraseña de usuario que puede usar el proxy. Hasta 10.
   -f <path>        Path absoluto de la carpeta donde se encuentran los mails.
   -v               Imprime información sobre la versión versión y termina.
```

-Inicialización del servidor con dos usuarios y una ruta a las casillas

```
machi@[redacted]:~/protos/src$ ./server.out -u machi:machi -u mario:luigi -f \\wsl.localhost\Ub
untu-20.04\home\machi\mails
INFO: main.c:96, Server initializing...
DEBUG: main.c:143, Pop3 passive socket created successfully with fd: 3
DEBUG: main.c:159, Management passive socket created successfully with fd: 4
INFO: main.c:162, Server now accepting connection requests
```

-Conexión cliente a servidor con puerto y dirección por defecto

```
machi@[redacted]:~$ netcat -C 127.0.0.1 44443
+OK pop3-server ready
```

-Cambio de puerto de conexiones pop3 (-p 12345)

```
icacion/TP/pop-server/src$ ./server.out -f /mnt/c/Users/Marcos\ Casiraghi/Do
cuments/ITBA/Primer\ Cuatrimestre\ 202\ Cuatrimestre\ 202
3/Protocolos\ de\ Comunicacion/TP/pop-server/mails -p 12345 -u marcos:123 -u
 santi:122
INFO: main.c:96, Server initializing...
DEBUG: main.c:143, Pop3 passive socket created successfully with fd: 3
DEBUG: main.c:159, Management passive socket created successfully with fd: 4
```

```
unicacion/TP/pop-server/src/client$ nc -C localhost 12345
+OK pop3-server ready
```

-Registro exitoso de cliente del servidor pop3

```
+OK pop3-server ready
USER marcos
+OK
PASS 123
+OK
```

-Fracaso de registro (contraseña incorrecta)

```
+OK pop3-server ready
user marcos
+OK
pass 12
-ERR
```

-Fracaso de registro (usuario no declarado en configuración de servidor)

```
+OK pop3-server ready
user santiago
+OK
pass 12345
-ERR
```

-Comando CAPA

```
CAPA
+OK
CAPA
PIPELINING
USER
.
```

-Comando LIST (con directorio válido y archivos para leer)

```
LIST
+OK 5 messages (73 octets)
1 15
2 9
3 5
4 40
5 4
.
```

-Comando LIST N° (con número de archivo válido)

```
LIST 3
+OK 3 5
```

-Comando LIST (sin archivos disponibles)

```
LIST
+OK 0 messages (0 octets)
```

-Comando LIST N° (con número de archivo inválido)

```
LIST 2
-ERR no such message
LIST -1
-ERR no such message
LIST 9999
-ERR no such message
```

-Comando STAT

```
STAT
+OK 5 73
```

-Comando NOOP

```
NOOP
+OK
```

-Comando RETR N°

```
RETR 1
+OK message follows
Este es un mail
.
RETR -1
-ERR no such message
RETR 999
-ERR no such message
```

-Comando DELE N°

```
DELE 1
+OK message deleted
DELE a
-ERR no such message
DELE 999
-ERR no such message
```

-Comando RSET

```
RSET
+OK maildrop has 5 messages (73 octets)
```

-Comando QUIT

```
QUIT
+OK goodbye!
```

-Pipelining

```
op-server/src$ printf 'user marcos\npass 123\nLIST\nRETR 1\n'|n
c -C localhost 44443
+OK pop3-server ready
+OK
+OK
+OK 5 messages (73 octets)
1 15
2 9
3 5
4 40
5 4
.
+OK message follows
Este es un mail
.
```

-Borrado de mensaje por fuera del servidor durante la ejecución

```
retr 30
+OK message follows

.
list 30
+OK 30 0
retr 30
+OK message follows
-ERR no such message
quit
+OK goodbye!
```

En este caso el mensaje 30 (que estaba vacío) fue eliminado durante la ejecución.

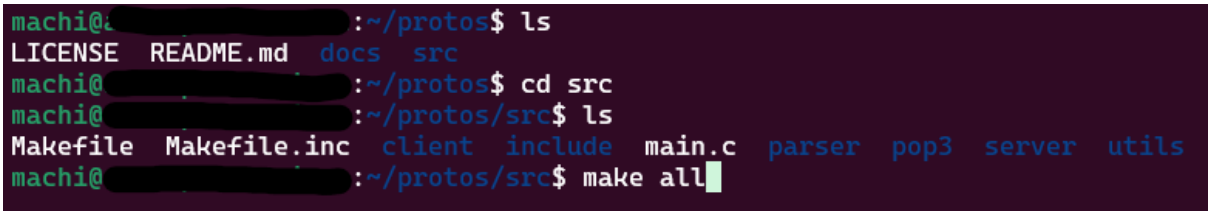
```
retr 30
+OK message follows
ejemplo borrado

.
dele 30
+OK message deleted
quit
-ERR some messages not deleted.
```

En este caso, se intentó eliminar un mensaje en el estado de ‘update’ que ya había sido eliminado por fuera durante la ejecución.

9. Guia de instalación

Para instalar el servidor, tanto de POP3 como de monitoreo y configuración, primero se debe clonar el repositorio de GitHub (https://github.com/MSGronda/pop-server). Segundo, dentro del directorio del servidor, se debe ir hacia el subdirectorio src/ y ejecutar el comando ´make all´. Al hacer esto, se crea un ejecutable llamado server.out que ejecuta la aplicación Main mencionada en 3.1. También se crea el ejecutable client.out dentro de la carpeta client del repositorio. Este ejecuta el cliente gerente mencionado en 3.4.



Debe también definir el token de autenticación usado por el protocolo de monitoreo y configuración, para hacer esto debe:

1) En el directorio en donde se encuentra el *server.out*, debe correr el comando:
   export M3_AUTH_TOKEN=<valor numérico>
2) De la misma manera, debe ejecutar el mismo comando en el directorio que se encuentra el *client.out*

Cabe aclarar que, existe un default, que se usará en el caso que no se haya generado esta variable de entorno, pero no se recomienda usarla.

10. Instrucciones para la configuración

Previo a ejecutar el programa server.out, se puede especificar por argumento varias opciones de configuración. Estos siendo:

-h : imprime la ayuda y finaliza la ejecución de la aplicación.

-v: imprime la información sobre la versión del servidor y finaliza la ejecución de la aplicación.

-p <puerto POP3> : indica el puerto donde se desea inicializar el socket pasivo para recibir conexiones (TCP). Este argumento es opcional, si no se especifica un puerto en particular, por defecto, la aplicación elige el <44443>.

-P <puerto M^3> : indica el puerto donde se desea inicializar el socket para contestar pedidos UDP. Este argumento es opcional, si no se especifica un puerto en particular, por defecto la aplicación elige el <55552>.

-u <usuario>:<contraseña> : indica un usuario con su contraseña que puede ser usado en el servidor para acceder a una casilla de correo electrónico. El servidor permite hasta 10 usuarios distintos en simultáneo. * Los usuarios son necesarios para poder iniciar sesión dentro del servidor y acceder a una casilla de correo particular.

-f <ruta> : ruta absoluta del directorio donde se encuentran los directorios de los usuarios con sus correos dentro. * La ruta es necesaria para poder acceder a los correos. Si una ruta no es dada, el servidor funcionará pero ningún usuario registrado tendrá mails para consumir. En caso de que la ruta contenga espacios, es necesario escapar estos para que se pueda acceder correctamente. * El directorio debe contener subdirectorios llamados por el nombre de usuario que va a poder acceder y dentro de cada subdirectorio, los correos de cada uno. Por ejemplo:

```
correos
├── Juan
│   ├── mail1.txt
│   ├── mail2.txt
│   ├── muy_importante.txt
│   ├── no_compartir.txt
│   └── mail5.txt
└── Esteban
    ├── Feliz_Cumpleaños.txt
    ├── mail2.txt
    └── mail.txt
└── Francisco
    ├── Empleo.txt
    ├── rechazado.txt
    └── rechazado2.txt
```

De esta manera, solo los usuarios que tengan el nombre de usuario Juan, Esteban y Francisco tendrán acceso a una casilla con correos dentro. Cualquier otro usuario no tendrá acceso a ningún correo.

Comandos posibles para el servidor de configuración

Comando	Función
help	Devuelve los comandos que se pueden ejecutar
noop	Devuelve un mensaje
btsent	Devuelve la cantidad de bytes enviados por el servidor
btrec	Devuelve la cantidad de bytes recibidos por el servidor
currusers	Devuelve la cantidad de usuarios conectados en el servidor
hisusers	Devuelve la cantidad de usuarios que se conectaron al servidor
adduser <usuario:contraseña>	Agrega un usuario al servidor
quit	cierra la conexión del usuario

11. Ejemplos de configuración y monitoreo

-Conexión a servidor de configuración

```
machi@MACHI:~/protos/src/client$ ./client.out 127.0.0.1 5
5552
+-----------------------------------------+
|Possible commands for client admin in m3   |
+-----------------------------------------+
|`help`: prints help table                |
|`btsent`: prints bytes sent              |
|`btrec`: prints bytes received           |
|`currusers`: prints amount ofcurrent users |
|`hisusers`: prints historic amount of users|
|`adduser <user>:<pass>`: adds a new user   |
|`noop`: noops                            |
|`quit`: quits session                    |
+-----------------------------------------+
```

dirección 127.0.0.1 y puerto 55552 por defecto (si se utiliza el parámetro de -P <puerto> al momento de iniciar el servidor, utilizar ese puerto).

-Conexión a servidor de configuración (mala dirección o puerto)

```
machi@MACHI:~/protos/src/client$ ./client.out 192.168.5.63 55552
+-----------------------------------------+
|Possible commands for client admin in m3   |
+-----------------------------------------+
|`help`: prints help table                |
|`btsent`: prints bytes sent              |
|`btrec`: prints bytes received           |
|`currusers`: prints amount ofcurrent users |
|`hisusers`: prints historic amount of users|
|`adduser <user>:<pass>`: adds a new user   |
|`noop`: noops                            |
|`quit`: quits session                    |
+-----------------------------------------+
> btsent
```

Al momento de correr un comando que no sea ‘help’, la línea de comandos quedará congelada sin volver a poder recibir comandos

-Ayuda en servidor de configuración

```
> help
+-----------------------------------------+
|Possible commands for client admin in m3   |
+-----------------------------------------+
|`help`: prints help table                |
|`btsent`: prints bytes sent              |
|`btrec`: prints bytes received           |
|`currusers`: prints amount ofcurrent users |
|`hisusers`: prints historic amount of users|
|`adduser <user>:<pass>`: adds a new user   |
|`noop`: noops                            |
|`quit`: quits session                    |
+-----------------------------------------+
```

-Comando ‘btsent’

```
> btsent
Bytes sent: 490
```

Indica cuantos bytes envío el servidor.

-Comando ‘btrec’

```
> btrec
Bytes recieved: 91
```

Indica cuantos bytes recibió el servidor por parte de los usuarios.

-Comando ‘currusers’

```
> currusers
Total current connections: 4
```

Indica la cantidad de usuarios concurrentes al momento de enviar el comando.

-Comando ‘hisusers’

```
> hisusers
Total historic connections: 15
```

Indica la cantidad de clientes que se conectaron al servidor desde que se inicializó.

-Comando ‘adduser’

```
> adduser p:p
User succesfully added
```

Permite agregar un usuario

-Comando ‘noop’

```
> noop
noop
```

Devuelve la palabra “noop”

-Comando ‘quit’

```
> quit
machi@MACHI:~/protos/src/client$
```

Cierra la conexión del cliente con el servidor

-Monitoreo en el servidor con servidor de configuración

```
DEBUG: server/management.c:139, New UDP packet (9 bytes) reci
eved by manager
DEBUG: server/management.c:154, UDP packet (6 bytes) sent by
manager
DEBUG: server/management.c:139, New UDP packet (9 bytes) reci
eved by manager
DEBUG: server/management.c:154, UDP packet (10 bytes) sent by
 manager
```

Indica a nivel ‘DEBUG’ cuando se reciben y se envían paquetes UDP

-Monitoreo de iniciación correcta de servidor

```
machi@MACHI:~/protos/src$ ./server.out -u m:m -f /home/machi/
protos/mail -p 55565 -P 25565
INFO: main.c:96, Server initializing...
DEBUG: main.c:143, Pop3 passive socket created successfully w
ith fd: 3
DEBUG: main.c:159, Management passive socket created successf
ully with fd: 4
INFO: main.c:162, Server now accepting connection requests
```

-Monitoreo de mala iniciación del servidor (falla de sockets)

```
machi@MACHI:~/protos/src$ ./server.out -u m:m -f /home/machi/
protos/mail -p 110 -P 25565
INFO: main.c:96, Server initializing...
FATAL: main.c:176, Error binding passive socket
```

En este caso el puerto 110 ya está siendo utilizado (es reservado) por lo que falla.

```
machi@MACHI:~/protos/src$ ./server.out -u m:m -f /home/machi/
protos/mail -p 55565 -P 100
INFO: main.c:96, Server initializing...
DEBUG: main.c:143, Pop3 passive socket created successfully w
ith fd: 3
FATAL: main.c:176, Error binding passive socket
```

En este caso el servidor de pop3 se inicializa correctamente pero el de configuración no ya que intenta abrir un socket pasivo en un puerto que está reservado.

-Monitoreo de recepción de nueva conexión

```
DEBUG: pop3/pop3.c:109, Initialized client data for client wi
th fd: 5
DEBUG: pop3/pop3.c:310, New connection recieved with fd: 5
```

Se indica el file descriptor asignado al cliente y que se recibió la conexión.

-Monitoreo de inicio de sesión

```
INFO: pop3/pop3_actions.c:217, User m logged in successfully
[2023-06-22 20:02:24]
```

Se indica qué usuario inició sesión y en qué fecha y a qué hora lo hizo.

-Monitoreo de interacción con cliente

```
DEBUG: pop3/socket_io_actions.c:36, Recieved 4 bytes from cli
ent with fd: 5
DEBUG: pop3/socket_io_actions.c:36, Recieved 2 bytes from cli
ent with fd: 5
```

Envío del comando list

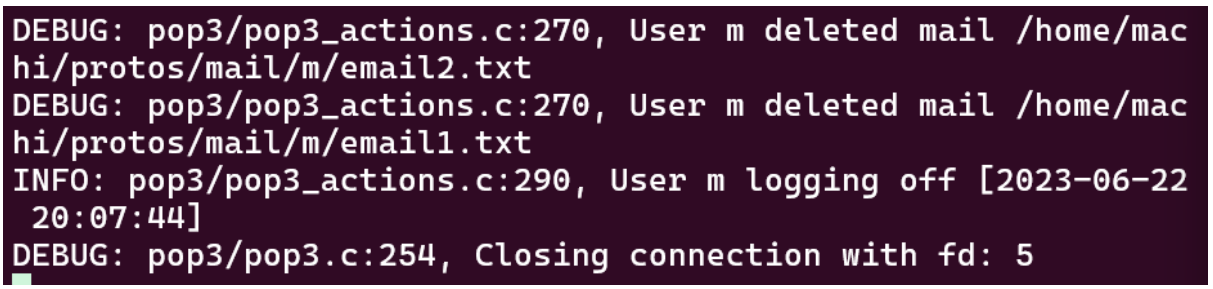
```
DEBUG: pop3/socket_io_actions.c:36, Recieved 6 bytes from cli
ent with fd: 5
DEBUG: pop3/socket_io_actions.c:36, Recieved 2 bytes from cli
ent with fd: 5
DEBUG: pop3/mail.c:508, User m retrieving mail email2.txt
```

ejecución de ‘retr 1’

-Monitoreo de cierre de sesión

```
INFO: pop3/pop3_actions.c:290, User m logging off [2023-06-22
 20:07:02]
DEBUG: pop3/pop3.c:254, Closing connection with fd: 5
```

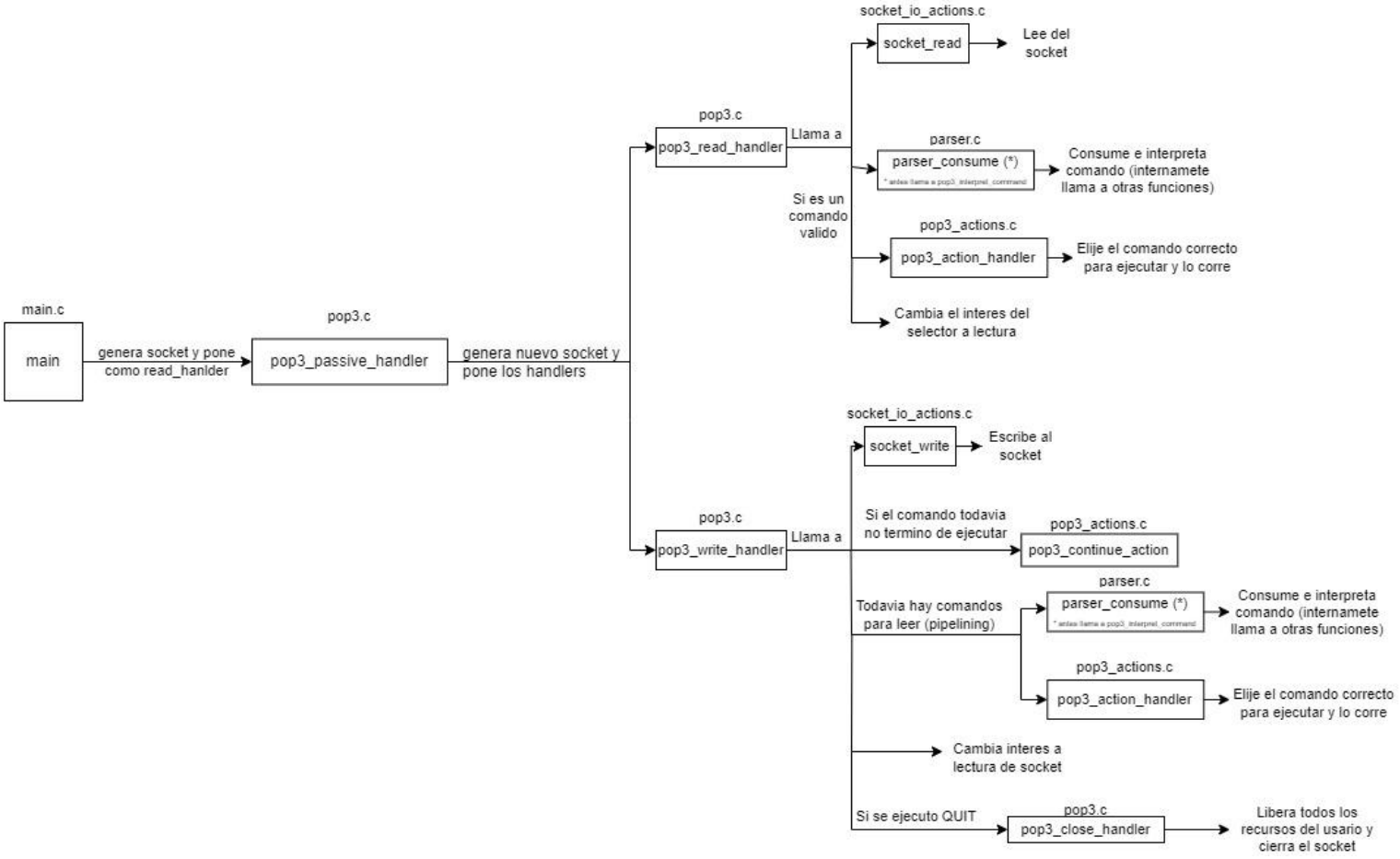
Cierre de sesión sin eliminar correos



Cierre de sesión eliminando correos.

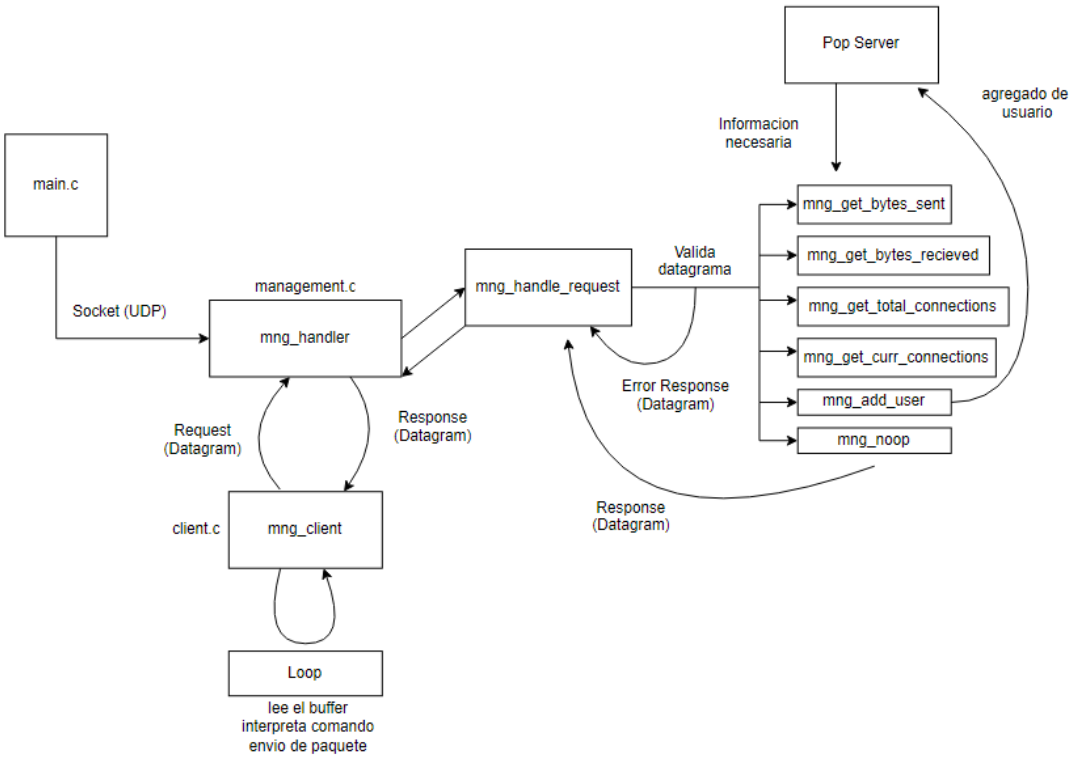
12. Diseño del proyecto

Abajo se encuentra el comportamiento general del servidor POP3 incluyendo las funciones que se ejecutan y a que archivo pertenecen. Esto no incluye el sistema de usuarios y la funcionalidad de RETR de un mail. El diagrama sigue el flujo de ejecución de un usuario.

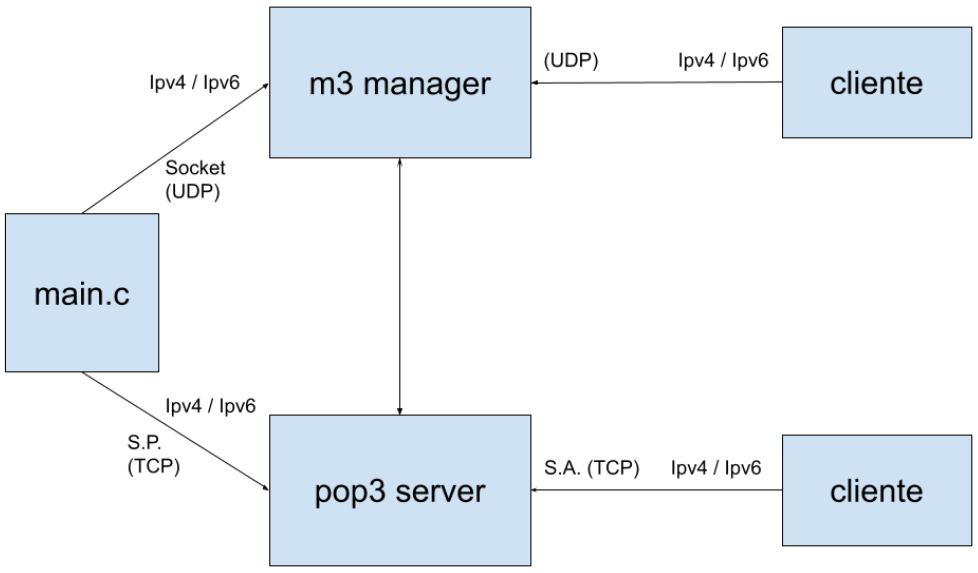


El diagrama completo se puede encontrar aquí.

Comportamiento general del protocolo de monitoreo y configuración



Interacción entre main, servidor pop3, servidor m^3 y clientes (de POP3] y M3):



S.P. - socket pasivo
S.A. - socket activo