Instituto Tecnológico de Buenos Aires

# Trabajo Práctico Especial

*Protocolos de Comunicación*

*Segundo cuatrimestre 2024*


**Legajos:**

Agustin Alonso: 63316

Valentina Marti Reta: 63225

Tomas Becerra: 63732

# Índice

# 1. Introducción

El objetivo del presente trabajo práctico es desarrollar un servidor que soporte el protocolo POP3 (Post Office Protocol versión 3), teniendo en cuenta los principios presentes en el RFC [1939].

A su vez, se implementó un protocolo de administración cliente-servidor, a partir de las métricas definidas por el grupo.

A continuación se detalla cómo se llevó a cabo la implementación de dichos protocolos, junto con el servidor llamado POPCORN.

# 2. Descripción de los protocolos y aplicaciones desarrolladas

## 2.1. Servidor POP3

Un servidor POP3 es un tipo de servidor de correo electrónico que permite a los clientes recuperar mensajes de correo electrónico almacenados en un servidor remoto. Está diseñado para descargar correos desde el servidor al cliente (como Outlook, Thunderbird, entre otros).

Para la implementación de dicho servidor y siguiendo las instrucciones del enunciado, se optó por la opción de manejar una serie de argumentos para la ejecución del programa:

- -d Argumento seguido del directorio donde residen las direcciones de mail. Default: Maildir
- -t Argumento seguido de la transformación que se quiere aplicar. Default: cat
- -p Argumento seguido por el puerto para el protocolo POP3. Default: 8085
- -P Argumento seguido por el puerto entrante del servicio de management. Default: 8086
- -l Argumento seguido de la dirección donde servirá el servicio de POP3. Default: 0.0.0.0
- -L Argumento seguido de la dirección donde servirá el servicio de management. Default: 127.0.0.1
- -u Argumento seguido del usuario y contraseña (<name>:<pass>) de usuario que puede usar el servidor. Hasta 10.

Es importante que se el directorio donde residen las direcciones de mail que se pasa como parámetro exista y que dicha estructura contenga directorios que coincidan uno a uno con los usuarios proporcionados por los argumentos.

El servidor inicia configurando el selector, una herramienta clave para gestionar de forma eficiente el flujo de datos de entrada y salida de cada cliente. Este permite que los sockets sean no bloqueantes y que se puedan manejar múltiples conexiones en simultáneo. Durante la inicialización, el servidor registra dicho selector para operaciones de escritura y configura un handler que acepta conexiones de clientes de manera pasiva. Si ocurre un error al inicializar el selector o registrar el servidor, el proceso aborta. Adicionalmente se crea un socket no pasivo para el protocolo UDP, cuya implementación corresponde al Protocolo de administración, explicado en la sección 2.2. Cuando un

cliente intenta conectarse, el handler gestiona la solicitud. Este componente utiliza un socket que admite conexiones entrantes tanto para IPv6 como para IPv4, asignando el puerto indicado como parámetro, sino el default. Si se produce algún error al crear el socket, el servidor aborta.

Luego, para cada cliente conectado se define la estructura *pop3_session_data*, que almacena toda la información necesaria para manejar una sesión de cliente POP3. Dentro de esta se incluye información sobre el cliente, como la dirección de la red y el nombre del usuario. También, se define una máquina de estados que permite gestionar el flujo de las operaciones del cliente y se incluyen dos buffers, uno de lectura y otro de escritura, que se utilizan para almacenar los datos entrantes y salientes. Estos son luego encapsulados en las estructuras *buff_read* y *buff_write* para facilitar su manejo. Además, en la estructura se tiene el parser de comandos POP3, que se encarga de interpretar los comandos enviados por el cliente.

También, para indicar el estado de operación se utiliza el campo OK, que se setea en true or false y se utiliza a la hora de analizar los comandos, y la variable *next_state*, que define el siguiente estado de la máquina de estados. Además, se incluye un manejador de mail (*m_manager*), que facilita la administración de los mensajes durante la sesión.

En cuanto al flujo de trabajo, cuando se recibe una conexión de un cliente, se inicializa el buffer del servidor con un mensaje de bienvenida. Esto se guarda en el buffer_write, el cual luego se enviará al cliente a través del file descriptor. Posteriormente, el servidor pasa a escuchar en modo de lectura, esperando los comandos del cliente. Cuando el cliente envía un comando, este se guarda en el *buffer_read*, y el parser lo interpreta para determinar si corresponde a un comando válido. Si el comando es válido, se ejecuta y el servidor responde con un mensaje adecuado, que se guarda nuevamente en el buffer_write y se envía al cliente. Este proceso permite que el servidor gestiona las conexiones y operaciones de manera eficiente y dinámica.

Esta implementación de POP3 soporta los comandos especificados en el RFC 1939: ***QUIT***, ***LIST***, ***STAT***, ***RETR***, ***DELE***, ***NOOP*** y ***RSET***, además, de los comandos de autenticación ***USER*** y ***PASS***. Hasta que el cliente no se autentique correctamente, ningún comando que no sea ***QUIT***, ***USER*** o ***PASS*** están habilitados. Cuando un cliente envía un comando, este es almacenado en el buffer correspondiente y luego transformado en *verb* (la primera parte del comando) y *arg1* (la segunda parte del comando) por el *pop3_parser*. Luego, se procesa el comando en la función *process_command*. Si reconoce el *verb* y el *arg1* es el correcto para dicho comando, se llama a la función correspondiente que contesta con una respuesta estándar +**OK** (seguido de la información del comando) si el mismo se ejecuta correctamente o con **-ERR** (especificando el error con un mensaje) en caso de error. Luego de realizar esto, el selector se registra en modo escritura para que el cliente reciba la respuesta.

Luego de ejecutar el comando ***QUIT***, se cambia el estado de cliente a **UPDATE**. Al arribar al estado, se ejecuta la función *checkout*, la cual se encarga de liberar la estructura manejada de mails y de poner en el buffer un mensaje de despedida. Luego, vamos a la función *goodby_message*, la cual cambia el estado a **CLOSE_CONNECTION** y se imprime el mensaje. Luego se da de baja el selector, se cierra el file descriptor y se elimina la estructura de la sesión.

## 2.2. Protocolo de administración

El protocolo de administración es no orientado a conexión, utilizando el protocolo de transporte UDP. En primer lugar, se debe proveer un **token** como forma de autorización de quien quiere acceder a la configuración del servidor. Esta clave consiste en 8 caracteres arbitrarios establecidos en el servidor. A su vez, el protocolo soporta diversos comandos a ser ejecutados en forma de configuración. En cuanto a las métricas que se pueden consultar, podemos obtener información de las **conexiones** del servidor, las **actuales**, **históricas** y el **récord** de conexiones concurrentes atendidas. En adicción, es posible consultar la cantidad totales de bytes que se transfirieron por el servidor, especificando por los salientes, entrantes y totales. Finalmente, se puede alterar la configuración de las máximas conexiones que soportara el servidor, entre un número de 1 y 500 conexiones.

Implementar el protocolo de administración de manera no orientada a conexión nos provee de ciertos beneficios así como impedimentos. En cuanto a performance, es beneficioso esta implementación debido a que el servidor no requiere de la creación de un socket activo para atender cada llamado. De esta manera, se reduce la cantidad de conexiones concurrentes en el servidor en todo momento. Si hablamos de los impedimentos, al no ser orientado a conexión, cada configuración debe ser ejecutada de manera singular, así como siempre especificar un token, el cual es único.

## 2.3. Aplicación cliente

La aplicación cliente recibe los distintos argumentos por entrada estándar, y se encarga de procesar los argumentos para armar el datagrama que requiere el protocolo. En caso de éxito, imprime en salida estándar la información de la petición solicitada, así como en caso de falla informa porque se dio la misma. A su vez, se puede especificar con argumentos en que puerto y dirección se quiere hacer la solicitud.

# 3. Problemas encontrados durante el diseño y la implementación

Durante el transcurso de este proyecto nos encontramos con algunas dificultades. El primero de ellos, fue migrar nuestra implementación del *echo server* de un modelo bloqueante a uno no bloqueante, utilizando el selector proporcionado por la cátedra. Este cambio fue necesario para mejorar el desempeño y evitar que el servidor se viese afectado por solicitudes concurrentes. Esta migración implicó ajustar múltiples funciones para garantizar que operaran correctamente; nuestro principal problema fue la falta de cambiar el interés del selector al finalizar con ciertas acciones.

Otro problema importante al cual nos enfrentamos, fue la necesidad de desarrollar un parser propio. Aunque la cátedra ofreció uno rediseñado, encontramos dificultades al integrarlo con nuestras necesidades específicas. El resultado de esto fue desarrollar un parser propio que nos ayude a nuestro caso específico.

Luego, la implementación del comando *RETR* también nos presentó complicaciones. Para evitar que el servidor se bloqueará al realizar la transformación del contenido del archivo, fue necesario *forkear* procesos. Esto permitió que las operaciones relacionadas con el comando se ejecutarán de manera independiente, pero también introdujo complejidades adicionales en la gestión de procesos.

Por último, añadir la funcionalidad de modificar parámetros del servidor durante la ejecución resultó ser complicado debido a que fue lo último que se realizó del protocolo de administración. Integrar esta nueva característica sin alterar la estructura existente fue un desafío que requirió de una cuidadosa adaptación del protocolo.

Un problema adicional que enfrentamos fue con el soporte de *pipelining*. El sistema no funcionaba correctamente, ya que solo aceptaba un único comando antes de que la terminal se "muriera" y dejara de responder. Este inconveniente nos dificultó la implementación adecuada de múltiples comandos en una misma conexión iniciada por *pipelining*.

# 4. Limitaciones de la aplicación

Una de las principales limitaciones de la aplicación, está relacionada con la falta de soporte para ciertos comandos del protocolo POP3. Si bien hemos implementado los comandos básicos como QUIT, LIST, RETR, entre otros, no pudimos integrar todos los comandos definidos en el RFC 1939. Esto fue producto de las restricciones de tiempo durante el desarrollo, lo que obligó a priorizar las funcionalidades obligatorias y asegurar que el servidor fuese funcional y estable.

Otra limitación relevante está relacionada con la gestión de logs. Actualmente, los registros de las conexiones al servidor se escriben en un archivo log al cual cualquier usuario con acceso al servidor puede consultar si se conecta a la máquina. Esto podría representar un riesgo en la seguridad y privacidad del cliente.

En cuanto al protocolo de monitoreo, la principal limitación es que al no ser orientado a conexión, por cada pedido que se desea realizar, ya sea por una métrica o una configuración, se tiene que especificar el token de autorización por cada petición que se desea realizar.

Además, una limitación importante es que el *pipelining* funciona de manera intermitente, solo procesando correctamente una única línea de comandos a la vez. Cuando se intenta enviar múltiples comandos en una sola conexión, el servidor no responde de manera adecuada, lo que impide el uso eficiente de este mecanismo.

# 5. Posibles extensiones

Una extensión importante sería agregar soporte para más comando POP3, como lo son TOP y UIDL, y otros mecanismos de autenticación, como APOP, lo que mejoraría la seguridad y la compatibilidad con una mayor cantidad de clientes de correo. Otro aspecto a mejorar sería el sistema de logs. Actualmente, los logs se almacenan en un archivo accesible por todos los usuarios conectados, lo que puede representar un riesgo para la seguridad y privacidad. Sería una buena extensión implementar un sistema de acceso controlado a los logs, además de agregar funcionalidades para filtrar y analizar los registros de forma más eficiente.

Otra mejora adicional sería la implementación de métricas no volátiles. Actualmente, las métricas sólo persisten durante la ejecución del servidor, pero se pierden si el servidor colapsa. Para evitar esto, sería una gran extensión almacenar estas métricas en un archivo o base de datos, lo que permitiría su conservación y análisis posterior.

Por otro lado, el registro persistente de usuarios también es una gran extensión. El servidor actual no conserva permanentemente las credenciales de los usuarios, lo que puede ser un inconveniente en entornos más exigentes. Implementar un método de almacenamiento persistente, como un archivo o base de datos, para almacenar las credenciales de los usuarios garantiza que los datos estén disponibles incluso después de reinicios del servidor, mejorando la experiencia del usuario y la administración del servidor.

Por último, otra extensión relevante sería la implementación de una mayor cantidad de configuraciones dinámicas para el servidor.

Además, una extensión relevante sería la mejora del soporte para el *pipelining*, que actualmente solo funciona correctamente cuando se envía un único comando. Resolver esta limitación permitiría que el servidor procese múltiples comandos de manera más eficiente en una sola conexión, mejorando el rendimiento general y la experiencia de usuario.

# 6. Conclusiones

Este proyecto resultó ser una buena oportunidad para afianzar los conocimientos adquiridos de la materia durante el cuatrimestre. La realización del mismo nos permitió adentrarnos en el protocolo POP3 y comprender mejor la correcta interpretación de los RFCs, lo cual fue clave para desarrollar, implementar y documentar nuestro propio protocolo. También, aprendimos más acerca de la programación con sockets, tanto bloqueantes como no bloqueantes.

A través de este proyecto, también pudimos aplicar varios conceptos de materias relacionadas, como Arquitectura de Computadoras y Sistemas Operativos, que fueron clave para entender el funcionamiento interno del servidor. La integración de estos conocimientos resultó ser un desafío,

pero también una gran oportunidad de aprendizaje práctico que nos permitió ver cómo se interconectan diversas áreas de la informática en proyectos reales.

La mayor dificultad que enfrentamos fue comprender y aplicar correctamente los nuevos conceptos al principio, especialmente aquellos relacionados con la implementación de protocolos y la programación con sockets. Sin embargo, con el tiempo, superamos esas dificultades y conseguimos avances significativos. Consideramos que este proyecto ha sido una experiencia altamente enriquecedora, que no solo nos permitió mejorar nuestras habilidades técnicas, sino también desarrollar un enfoque más sólido y práctico en el campo de las redes y la programación.

# 7. Ejemplo de prueba

Todos estos comandos son conectando al servidor ejecutado con las siguientes configuraciones:

*./serverx -l 127.0.0.1 -p 8085 -u vale:root -u vale2:root2 -t cat -d Maildir*

Donde vale y vale2 corresponden a directorios válidos dentro de la estructura Maildir.

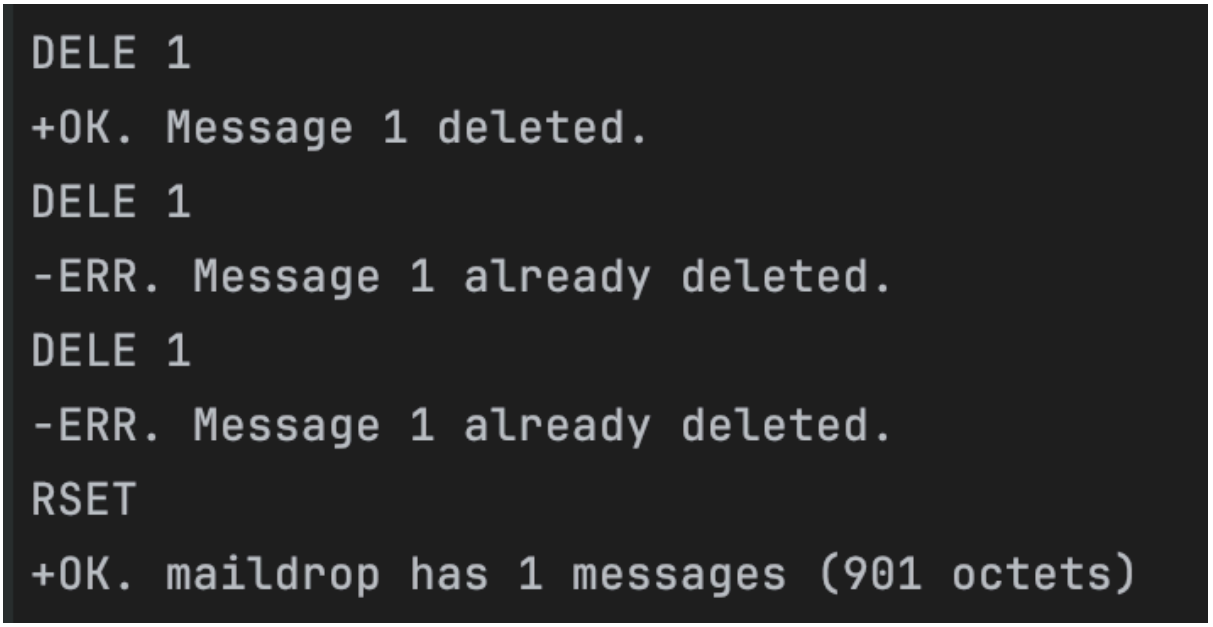
Primero nos conectamos al servidor, listamos los mails del usuario y traemos el número 1:

```
valentinamartireta@MacBook-Pro-de-Valentina-2 TPE-PROTOS % nc localhost 8085
+OK Welcome to POPCORN
USER vale
+OK.
PASS root
+OK.
LIST
+OK. 2 1043
1 901
2 142
.
LIST 1
+OK. 1 901
RETR 1
+OK. 901 octets.
Subject: Prueba
From: <vmartireta@itba.edu.ar>
Message-ID:<CAECUWH41tDfiS24s5PW2UWmATgnjrsQ4z=TYoWot_SYC+ccFgw@mail.gmail.com>

¿Qué es Lorem Ipsum?
Lorem Ipsum es simplemente el texto de relleno de las imprentas y archivos de texto.
Lorem Ipsum ha sido el texto de relleno estándar de las industrias desde el año 1500,
cuando un impresor (N. del T. persona que se dedica a la imprenta) desconocido usó una
galería de textos y los mezcló de tal manera que logró hacer un libro de textos especimen.
No sólo sobrevivió 500 años, sino que tambien ingresó como texto de relleno en documentos electrónicos,
quedando esencialmente igual al original. Fue popularizado en los 60s con la creación de las hojas
"Letraset", las cuales contenian pasajes de Lorem Ipsum, y más recientemente con software de autoedición,
como por ejemplo Aldus PageMaker, el cual incluye versiones de Lorem Ipsum.
.
```

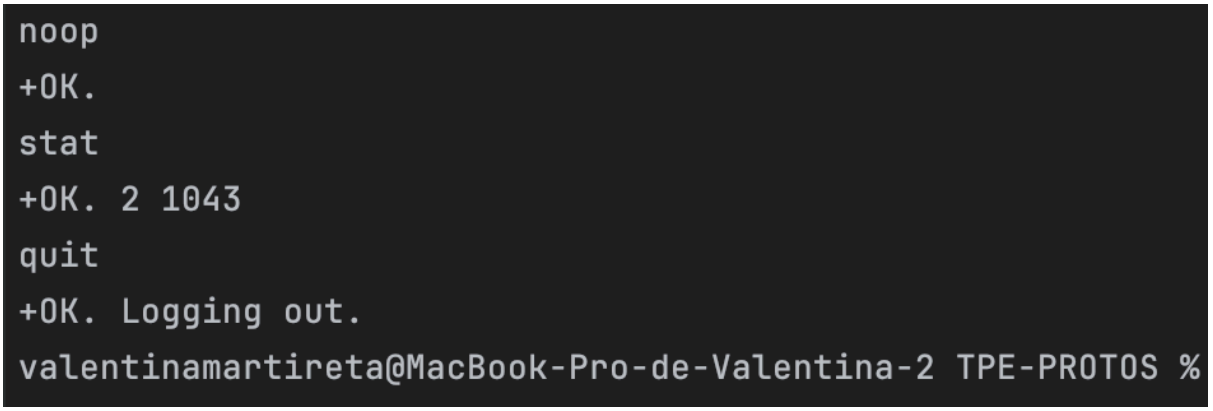
*Comandos USER, PASS, LIST, RETR*

Luego,  siguiendo en la misma terminal eliminamos un mensaje, intentamos eliminarlo otra vez y luego reseteamos las eliminaciones:




*Comandos DELE y  RSET*

Para finalizar en la misma sesión probamos los comando *NOOP* y  *STAT* y luego nos desconectamos (lo hacemos todo en lowercase para mostrar que no es case sensitive):




*Comandos STAT, NOOP, QUIT*

Luego vemos que cuales son los comandos permitidos para los usuarios que no están autenticados:

```
valentinamartireta@MacBook-Pro-de-Valentina-2 TPE-PROTOS % nc localhost 8085
+OK Welcome to POPCORN
LIST
-ERR. First enter USER.
pass
-ERR. First enter USER.
PASS
-ERR. First enter USER.
RETR 1
-ERR. First enter USER.
QUIT
+OK. Logging out.
```

*Comandos de un usuario no autenticado*

Si bien tuvimos problemas con Pipelining, como bien se mencionó en dificultades encontradas, este funciona  a medias. Al pasar el comando por Pipelining este se ejecuta, pero luego la terminal queda trabada y deja de funcionar. No pudimos resolver este problema.

```
valentinamartireta@MacBook-Pro-de-Valentina-2 TPE-PROTOS % printf "USER vale\nPASS root\nRETR 1\n" | nc localhost 8085
+OK Welcome to POPCORN
+OK.
+OK.
+OK. 901 octets.
Subject: Prueba
From: <vmartireta@itba.edu.ar>
Message-ID:<CAECUWH41tDfiS24s5PW2UWmATgnjrsQ4z=TYoWot_SYC+ccFgw@mail.gmail.com>

¿Qué es Lorem Ipsum?
Lorem Ipsum es simplemente el texto de relleno de las imprentas y archivos de texto.
Lorem Ipsum ha sido el texto de relleno estándar de las industrias desde el año 1500,
cuando un impresor (N. del T. persona que se dedica a la imprenta) desconocido usó una
galería de textos y los mezcló de tal manera que logró hacer un libro de textos especimen.
No sólo sobrevivió 500 años, sino que tambien ingresó como texto de relleno en documentos electrónicos,
quedando esencialmente igual al original. Fue popularizado en los 60s con la creación de las hojas
"Letraset", las cuales contenian pasajes de Lorem Ipsum, y más recientemente con software de autoedición,
como por ejemplo Aldus PageMaker, el cual incluye versiones de Lorem Ipsum.
.
```

*Ejemplo de Pipelining*

# 8. Guia de instalación detallada y precisa

Para la correcta instalación del proyecto, nos movemos a la carpeta principal. Una vez parados en la carpeta principal,  corremos el siguiente comando:

```
make clean all
```

Esto genera tanto el ejecutable del server (serverx) como del monitor (mng).

Luego, para correr el servidor ejecutamos el siguiente comando:

```
./serverx
[-l <dirección servicio POP3>]
[-p <puerto POP3>] -u <name>:<pass>
[-u <name>:<pass>] [-t <transformación>]
-d <maildir>
```

Donde lo que está entre corchetes es opcional y donde:

**name** = usuario a autenticar

**pass** = contraseña del usuario

**maildir** = directorio de los mails del servidor

Para conectarse al servidor POP3 se debe ejecutar el siguiente comando:

```
nc localhost <Puerto POP3>
o nc localhost 8085 (si no hubiese puerto POP3)
```

# 9. Instrucciones para la configuración

Para conectarse a la aplicación de administración

```
./mng
[-L <manager address>]
[-P <manager port>]
[-t <token>]
[-c <comando>]
[-n <valor>]
```

El argumento -c puede tomar los distintos valores en base a qué comando se desea ejecutar.

- Conexiones históricas - *chistory*
- Conexiones actuales - ccurrent
- Récord de conexiones - *crecord*
- Bytes transferidos - *bytess*
- Bytes recibidos - *bytesr*
- Bytes totales - *bytest*
- Máxima cantidad de conexiones - cmax

# 10. Elementos de configuración y monitoreo

En cuanto al monitoreo del servidor, contamos con métricas que consultan principalmente las conexiones del mismo y la cantidad de datos que se transfirieron.

- Conexiones históricas
  Cantidad de conexiones totales que atendió el servidor, incluye las ya finalizadas.
- Conexiones actuales
  Cantidad de conexiones que el servidor se encuentra atendiendo.
- Récord de conexiones
  Cantidad récord de conexiones que el servidor supo manejar.
- Bytes transferidos
  Cantidad total de bytes que transfirió el servidor.
- Bytes recibidos
  Cantidad total de bytes que recibió el servidor.
- Bytes totales
  Cantidad total de bytes que circularon de entrada o salida del servidor.

Finalmente, es posible configurar la cantidad máxima de conexiones que puede atender el servidor. Esta configuración es útil ya que limitar la máxima cantidad de conexiones afecta directamente la performance de respuesta a cada una de las peticiones.

# 11. Documento de diseño del proyecto