



INSTITUTO TECNOLÓGICO DE BUENOS AIRES

Trabajo Práctico Especial - POP3 Server

72.07 Protocolos de comunicación

Integrantes:

Aramburu, Paz - 62556

Neimark, Luciano - 62086

Seggiaro, Luca - 62855

Tordomar, Nicolás - 62250

Fecha de entrega: 24 de noviembre de 2023

Índice

1. Introducción	2
2. Protocolos y aplicaciones desarrolladas.	2
2.1 Server POP3	2
2.2 Protocolo de administración	4
2.3 Aplicación cliente	5
3. Problemas encontrados durante el diseño y la implementación.	5
4. Limitaciones de la aplicación.	6
5. Posibles extensiones.	6
6. Conclusiones.	7
7. Ejemplos de prueba.	7
7.1 Comandos POP3	7
7.2 Pipelining	9
7.3 Conexiones IPv6 y IPv4	10
7.4 Enviar un mensaje a través de curl	11
7.5 Atender a múltiples clientes en forma concurrente y simultánea	13
8. Guía de instalación.	15
9. Instrucciones y ejemplos de configuración y monitoreo.	16
10. Documento de diseño del proyecto.	17
11. Referencias	18

1. Introducción

En el siguiente informe se detalla la implementación del trabajo práctico espacial de la materia Protocolos de Comunicación. En dicho trabajo se implementó un servidor POP3 (Post Office Protocol versión 3) siguiendo los RFCs 1939^[1] y 2449^[2].

A su vez, en el informe se detalla el protocolo de administración implementado por el grupo y su puesta en práctica. Además se encuentra explicado la aplicación cliente que hace uso de este protocolo.

2. Protocolos y aplicaciones desarrolladas.

2.1 Server POP3

El servidor POP3 es un protocolo estándar utilizado para la recepción de correos electrónicos. Se encarga de la descarga de mensajes de correo electrónico desde un servidor de correo a un cliente de correo electrónico (por ejemplo, Outlook, Thunderbird, etc.). La implementación realizada permite establecer una comunicación adecuada entre el servidor y los clientes que se conectan a él.

El servidor extrae los argumentos que se pasaron por línea de comando como parámetros al programa. En estos argumentos se encuentran, luego del -u, los usuarios y sus respectivas contraseñas así como, luego del -d, el directorio que contiene los distintos directorios de cada usuario. Es importante tener en cuenta que esta estructura que contiene los directorios de usuarios del servidor debe haber sido creada previamente y coincidir uno a uno con los usuarios y contraseñas proporcionados por los argumentos. Si no, esta etapa fallará y la ejecución se abortará.

Opcionalmente se pueden definir los puertos para los sockets, con -p para el socket TCP utilizado por el servidor POP3 y con -P para el socket UDP utilizado por el protocolo de gestión. Los puertos son números que identifican un extremo de conexión el servidor. Se permiten números del 1 al 65535 para definir estos puertos. Para conectarse, los clientes deben especificar el puerto luego de la ip con netcat. De no definirse ningún puerto en específico se utilizan por default el puerto 1110 para el POP3 y el 9090 para el manager.

Lo siguiente que hace el servidor es inicializar el selector, el cual monitorea el flujo de entrada y salida de cada cliente de manera eficiente. Este es quien permite que los sockets sean no bloqueantes y es quien ayuda a gestionar de manera eficiente múltiples conexiones. El servidor se registra en modo lectura en el selector y se agrega un handler que se encarga de aceptar de manera pasiva nuevos clientes. Si la inicialización del selector o suscribir al servidor en modo lectura falla, se aborta la ejecución. El servidor además crea un socket no pasivo para el protocolo UDP, pero eso se explica con más detalle en la sección [1.2 Protocolo de administración](#).

Si el servidor se ejecuta correctamente, al conectarse un cliente se llama al handler explicado en el párrafo anterior. El funcionamiento de este handler se basa en la creación de un socket que recibe conexiones entrantes para IPv6 y IPv4. Estos sockets abren conexiones en el puerto provisto por parámetro o en el puerto default si no se provee ninguno por parámetro. Si falla la creación del socket, una vez más se abortaría la ejecución.

Se crea la estructura del cliente y se le asigna un file descriptor, el cual es único para cada cliente conectado, una dirección de socket y un parser. Esta estructura posee dos buffers, uno del servidor y otro del cliente. El del servidor es el que contendrá las respuestas del servidor al cliente (las que se imprimen al cliente por salida estándar), y el del cliente es el que contiene lo que éste envía por entrada estándar, lo cual luego será analizado por el servidor. Se guarda en el buffer del servidor un saludo para avisarle al cliente que la conexión al servidor fue exitosa y que el protocolo está listo para recibir comandos. Luego se setea el estado de su máquina de estados en WRITE y se registra en el selector al cliente en modo escritura junto con los handlers para cada operación POP3 (write, read, close y block), los cuales tienen asignados una función para cada operación.

Como se registró el selector en modo escritura y el buffer del servidor contiene el saludo, se entra a la función *pop3WriteCommand* la cual se encarga de leer el buffer del servidor y enviarle con la función *send* el buffer leído al file descriptor del cliente. Esto genera que el mensaje aparezca en la salida estándar del cliente. Luego de que se lea todo el buffer, se registra el selector en modo lectura y se devuelve el estado de la máquina de estados READ para que el cliente pueda escribir los comandos.

Los comandos que soporta nuestra implementación de POP3 son aquellos que menciona el RFC 1939^[1] para que una aplicación servidor pueda ser considerada un servidor POP3. Estas son: QUIT, LIST, STAT, RETR, DELE, RSET, CAPA, NOOP y al menos un mecanismo de autenticación, para el cual usamos USER y PASS. Si no se autenticó un usuario los comandos LIST, STAT, RETR, DELE y RSET no pueden ser ejecutados.

Al enviar el cliente uno de estos comandos, se guarda en el buffer del cliente y como el selector se encontraba en modo de escritura y la máquina de estados en READ, se accede a la función *pop3ReadCommand* la cual para el buffer del cliente hasta que esté vacío. El parser compara la línea recibida del cliente con cada comando mencionado anteriormente y si coincide con alguno, configura el estado del parser en dicho comando. Si no, se configura en error. Luego, si se puede parsear el buffer correctamente, se llama a *executeCommand* quien recibe el parser y su objetivo es manejar lo que realiza dicho comando.

En líneas generales, el servidor devuelve un +OK, indicando que se ejecutó correctamente el comando, o un -ERR, indicando que hubo un error. Para los comandos USER, PASS, QUIT, STAT, DELE, RSET, CAPA y NOOP, el protocolo agrega un mensaje simple de +OK con información extra dependiendo del comando al buffer del servidor. Después de esto se registra el selector en modo escritura para poder devolver el mensaje del servidor y se devuelve el estado de la máquina de estados WRITE. El buffer del servidor luego se imprime en *pop3WriteCommand* al igual que se imprime el greeting.

Si se encuentra que el estado del cliente está en CLOSED, lo cual se asigna al ejecutar el comando QUIT, se llama a la función *closeConnection*, la cual se encarga de cerrar la conexión del cliente.

Para los comandos LIST y RETR la manera de abordar su funcionamiento es distinta. En el caso del LIST, este llama a *pop3ReadList*, función que se encarga de devolverle al cliente la lista de los correos que tiene en su directorio. Guarda esta lista con el formato “<id de correo> <tamaño>” en el server buffer y al terminar de guardar todos se registra el selector en modo escritura y se devuelve el estado WRITE_LIST. Este estado está asociado a la función de escritura *pop3WriteList*, la cual se encarga de enviarle al cliente, de forma parecida al *pop3WriteCommand*, el buffer del servidor en salida estándar.

En el caso de usar el comando LIST o RETR con el número de correo que se quiere ver, si efectivamente se encuentra dicho correo se llama a *pop3ReadFile*. Esta función se encarga de cargar en el server buffer el contenido del correo hasta llenar dicho buffer, agregando un `\r\n.\r\n` al final para indicar la finalización del correo. Si alguna línea arranca con `.`, se agrega otro `.` a ella de manera que queden `..` para no confundirse con el fin del correo (ver RFC 1939^[4], página 2). Al terminar de cargar el server buffer se registra el selector en modo escritura y se devuelve el estado WRITE_FILE, quien está asociado a la función *pop3WriteFile*. Esta se encarga de imprimirle al cliente el buffer. Si no se hubiese terminado de leer el correo se vuelve a llamar a *pop3ReadFile* para terminar, y así sucesivamente. Si se terminó de leer, se registra el selector en modo lectura y se devuelve el estado READ, el cual está asignado en la máquina de estados con la función *pop3ReadCommand* para que el cliente pueda seguir enviando comandos.

Es importante aclarar que ante alguna falla en el registro del select dentro de las funciones *pop3*, se registra a este mismo en el modo de no operación y se devuelve el estado de ERROR_STATE, donde se le avisa al cliente que hubo un error con “-ERR” y se continúan leyendo comandos. Lo mismo ocurre si no se puede parsear el comando correctamente.

2.2 Protocolo de administración

Con el objetivo de tener mayor control sobre el servidor, se realizó un protocolo de administración sobre este. A través de una conexión UDP, se pueden observar las métricas del servidor, y actualizar en tiempo real la lista de usuarios soportados.

Como se menciona en [RFC Manager Protocol - Grupo 05.txt](#), el protocolo busca mantener las convenciones estándar de POP3. La implementación de éste se puede observar más en detalle en el archivo.

Buscando mantener la consistencia, se creó un parser similar al del servidor para parsear las instrucciones, pero a diferencia de éste, las instrucciones llegan completas, ya que UDP envía un único paquete. Por ello, la conexión que se establece con el puerto UDP no

agrega fd's al selector, ya que cada vez que se le permite correr, puede completar su ciclo de funcionamiento.

Se puede utilizar el comando “nc” para realizar múltiples llamados consecutivos al protocolo. Ejemplo de uso: “nc -u localhost 9090”.

2.3 Aplicación cliente

Dado que no se conoce completamente el funcionamiento de herramientas externas como nc, se decidió generar una aplicación cliente, que permite comunicarse con el servidor a través de UDP, de la misma manera que hacía nc. Esto nos permitió abstraernos del uso de esta herramienta, ya que nosotros mismos generamos los sockets, y decidimos cómo va a esperar las instrucciones del usuario.

Esta aplicación se genera por separado, ejecutando “make mc”, generando el archivo ./manager_client, que recibe la dirección y el puerto como parámetro. A diferencia de nc, esta aplicación se puede finalizar con la interrupción de fin de línea Ctrl+D.

3. Problemas encontrados durante el diseño y la implementación.

Como se podrá observar en las siguientes secciones, un gran desafío del trabajo consistió en decidir qué funcionalidades implementar, y cuales dejar para más adelante. Como todo trabajo real, el tiempo es un factor determinante. Por ello, nos guiamos en base al valor que le proveen estas al usuario para decidir el orden de implementación. Obviamente este valor puede venir de muchas direcciones, sea la técnica, o sea la utilitaria.

Uno de los mayores problemas que nos encontramos al implementar el servidor fue la migración del servidor bloqueante a uno no bloqueante. Consideramos valioso que los usuarios no dependan del tiempo de respuesta por parte del servidor a otros usuarios, por lo que se utilizó el selector provisto por la cátedra para migrar el trabajo. Se tuvo que medir y testear el trabajo para identificar que todas las distintas funciones, previamente bloqueantes, cambiaran a no bloqueantes.

Un problema que llevo tiempo de resolver fue el manejo de los comandos multilinea LIST y RETR. Al tener que tomar en cuenta que podía no entrar la respuesta entera dentro de los buffers, se tuvo que implementar un mecanismo de lectura-escritura haciendo uso del selector. Los problemas se presentaban en varios casos bordes que eran difíciles de testear y también en poder terminar ese ciclo de lectura-escritura retornando al flujo normal de la aplicación.

4. Limitaciones de la aplicación.

La limitación principal del trabajo en relación a lo solicitado son las transformaciones de mails. Esta fue la única funcionalidad que no pudimos implementar debido al tiempo que nos tomó desarrollar las demás funcionalidades. Priorizamos el funcionamiento no bloqueante, el cliente y los casos de prueba para ver que esté funcionando de manera correcta.

Otra limitación de la implementación es el orden de los llamados a la hora de hacer una lectura con “RETR” o “LIST”. Si se ejecuta el servidor con strace, se podrá observar que el orden de llamados es: select-read-select-write. En base a las consultas y los conocimientos adquiridos a lo largo de la carrera, entendemos que ejecutar el read-write en el mismo ciclo resulta ser más eficiente, y mejora significativamente el rendimiento del servidor.

Se optó por mantener un único directorio para cada usuario, sin el manejo de carpetas usuario/cur, usuario/new, usuario/tmp. Esto se debe a que consideramos que no agrega valor para los usuarios, ya que en definitiva “LIST” y “RETR” deben mostrar ambas carpetas, y no hay distinción entre ellas. Thunderbird es compatible con la implementación.

Como se menciona en la guía RFC para la implementación del administrador, nuestro servidor trabaja con tamaños de buffer fijos. Si bien hemos seleccionado un valor que maximiza el throughput para nuestras pruebas (ver [7.5 Atender a múltiples clientes en forma concurrente y simultánea](#)), entendemos que una herramienta como POP3 puede ser utilizada con distintos objetivos, el hecho de adjuntar imágenes ya afecta por completo el cálculo de tamaño de buffer óptimo.

Una última limitación de la aplicación es que es single threading. Esto hace que todos los clientes corran en un mismo thread y es por eso que el servidor tiene una capacidad limitada para manejar múltiples solicitudes concurrentes de los clientes. El servidor se limita a escalar eficientemente, ya que no puede aprovechar completamente los recursos de hardware modernos, como múltiples núcleos de CPU, para procesar las solicitudes en paralelo. Una consecuencia de esto es que los clientes experimentan tiempos de espera más largos.

5. Posibles extensiones.

Dentro de las posibles extensiones del protocolo se encuentran:

Registro persistente de usuarios. El servidor actual no guarda de forma permanente las credenciales de los usuarios. Para mejorar esto, se sugiere implementar un método de almacenamiento persistente, como un archivo simple o una base de datos, para conservar y recuperar estas credenciales.

Aceptar una mayor cantidad de comandos POP3. Por el momento sólo aceptamos los comandos que permiten que el servidor sea un protocolo POP3 pero hay algunos comandos más que son opcionales en el RFC 1939^[1] que nos gustaría implementar a futuro.

Hacer un servidor multithreading. El servidor consta de un único thread. Para sacarle el máximo provecho al POP3 se podría implementar un servidor multithreading que maximice la eficiencia de éste usando todos los recursos del procesador de la computadora.

Soportar carpetas internas. En un futuro, nos gustaría aceptar las distintas carpetas internas de cada usuario. Esto permitiría que puedan distinguir entre correo nuevo y visto, que podría ser útil para otros contextos y aplicaciones.

Métricas no volátiles. Actualmente las métricas persisten a lo largo de la ejecución del servidor, pero si este colapsa se pierden. Creemos que almacenarlas en un archivo puede ser agregar valor una vez que se desee hostear el servidor.

6. Conclusiones.

Este trabajo práctico fue una buena forma de consolidar todo lo visto durante el cuatrimestre en la materia. Se logró profundizar en el protocolo POP3, comprendiendo a fondo la interpretación correcta de los RFCs. Este entendimiento fue esencial además para el desarrollo, implementación y documentación de nuestro propio protocolo. También se consiguió un aprendizaje muy grande sobre la programación con sockets bloqueantes y no bloqueantes, lo cual viene de la mano del uso del selector. A su vez, se usaron y reforzaron muchos conceptos de las materias Arquitectura de Computadoras y Sistemas Operativos.

A lo largo del desarrollo del trabajo surgieron muchas dificultades y dudas, pero todas ellas pudieron ser superadas. La mayor dificultad fue entender los conceptos nuevos y ponerlos en práctica al principio. Sin embargo, creemos que fue un trabajo sumamente enriquecedor y nos vamos con una gran cantidad de aprendizaje.

7. Ejemplos de prueba.

7.1 Comandos POP3

Manteniendo los estándares y convenciones solicitadas en Post Office Protocol - Versión 3 [RFC 1939^[1]] y [RFC 2449^[2]], se implementaron los siguientes comandos:

Primero nos conectamos al servidor


```

src git:(dev) x nc -c localhost 1110
+OK POP3 server ready
USER paz
+OK
PASS hola
-ERR [AUTH] Authentication failed.
USER paz
+OK
PASS paz
+OK Logged in.
LIST
+OK 2 messages
1 540
2 540
.
STAT
+OK 2 1080

```

Imagen 7.1.1: comandos USER, PASS, LIST y STAT

Siguiendo en la misma terminal con el usuario paz autenticado:

```

RETR 1
+OK 540 octets
MIME-Version: 1.0
Date: Thu, 16 Nov 2023 15:53:36 -0300
Message-ID: <CAAgBtGS3Bmf3AiUtX0MdZQ9Ltk=tB-uW+aEiEL12v2kcpX=c0g@mail.gmail.com>
Subject: Un asunto
From: LUCIANO NEIMARK <lneimark@itba.edu.ar>
To: LUCIANO NEIMARK <lneimark@itba.edu.ar>
Content-Type: multipart/alternative; boundary="00000000000008ac764060a49899b"

--00000000000008ac764060a49899b
Content-Type: text/plain; charset="UTF-8"
..

Prueba

--00000000000008ac764060a49899b
Content-Type: text/html; charset="UTF-8"

<div dir="ltr">Prueba</div>

--00000000000008ac764060a49899b--
.
DELE 1
+OK message 1 deleted
RSET
+OK maildrop has 2 messages (1080 octets)
DELE 3
-ERR no such message
RETR 5
-ERR no such message

```

Imagen 7.1.2: comandos RETR, DELE y RSET

Probamos los ultimos comandos

```
ERR no such message
CAPA
+OK Capability list follows
CAPA
QUIT
STAT
LIST
RETR
DELE
NOOP
RSET
.
NOOP
+OK
QUIT
+OK POP3 server signing off (2 messages left)
```

Imagen 7.1.3: comandos CAPA, NOOP y QUIT

Si nos conectamos pero no autentificamos el usuario e intentamos correr algunos de los comandos que son propios de un usuario

```
➔ src git:(dev) x nc -c localhost 1110
+OK POP3 server ready
LIST
-ERR Unknown command.
RETR 1
-ERR Unknown command.
STAT
-ERR Unknown command.
DELE 1
-ERR Unknown command.
RSET
-ERR Unknown command.
```

Imagen 7.1.4: comandos propios de un usuario sin estar autenticado

7.2 Pipelining

Presentamos un ejemplo de pipelining donde se autentica el usuario correctamente y luego se hace un RETR del primer correo

```

● → src git:(dev) x printf "USER paz\r\nPASS paz\r\nRETR 1\r\n" | nc -c localhost 1110
+OK POP3 server ready
+OK
+OK Logged in.
+OK 540 octets
MIME-Version: 1.0
Date: Thu, 16 Nov 2023 15:53:36 -0300
Message-ID: <CAAgBtGS3Bmf3AiUtX0MdZQ9LtK=tB-uW+aEiEL12v2kcpX=c0g@mail.gmail.com>
Subject: Un asunto
From: LUCIANO NEIMARK <lneimark@itba.edu.ar>
To: LUCIANO NEIMARK <lneimark@itba.edu.ar>
Content-Type: multipart/alternative; boundary="0000000000008ac764060a49899b"

--0000000000008ac764060a49899b
Content-Type: text/plain; charset="UTF-8"
..

Prueba

--0000000000008ac764060a49899b
Content-Type: text/html; charset="UTF-8"

<div dir="ltr">Prueba</div>

--0000000000008ac764060a49899b--
.
○ → src git:(dev) x █

```

Imagen 7.2.1: Ejemplo de pipelining

7.3 Conexiones IPv6 y IPv4

Si al correr el servidor preguntamos por las actividades de red esto es lo que nos devuelve

```

[Intordomar@pampero ~]$ netstat -nltp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)

```

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State	PID/Program name
tcp	0	0	0.0.0.0:1110	0.0.0.0:*	LISTEN	-
tcp	0	0	0.0.0.0:22	0.0.0.0:*	LISTEN	-
tcp6	0	0	:::3333	:::*	LISTEN	30152/./pop3d
tcp6	0	0	:::22	:::*	LISTEN	-
tcp6	0	0	:::9090	:::*	LISTEN	-

Imagen 7.3.1: netstat -nltp

Como se puede ver, parecería que el programa pop3d solo escucha conexiones en IPv6 porque el protocolo figura como tcp6. Es verdad que escucha conexiones en IPv6, pero al crear los sockets en el proyecto y definir su familia utilizamos AF_INET6 (ver imagen 7.3.2). Los sockets AF_INET6 se usan para IPv6, pero también aceptan conexiones IPv4. Las conexiones IPv4 se manejan en el mismo socket que IPv6: “IPv4 and IPv6 share the local port space. When you get an IPv4 connection or packet to an IPv6 socket, its source address will be mapped to v6” (ip6(7) — Linux manual page^[3]). Es por esto que con *netstat* aparece que solo hay una conexión abierta IPv6, lo cual es verdad pero esta conexión también acepta conexiones IPv4. Tomamos esta decisión porque creemos que hacer 2 sockets, uno para IPv4 y otro para IPv6, no es tan eficiente como solo tener uno para manejar ambas conexiones.

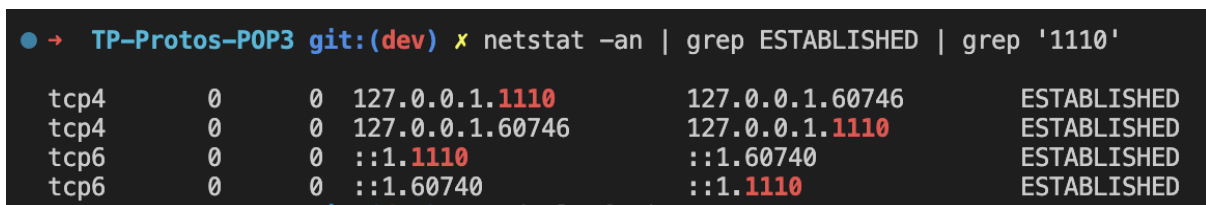
```
struct sockaddr_in6 serveradr;
const int server = socket(AF_INET6, SOCK_STREAM, IPPROTO_TCP);
```

Imagen 7.3.2: Creacion del socket con AF_INET6

Para conectarnos con IPv6 o IPv4 se lo indicamos a la terminal con un -6 o -4 respectivamente. Si lo probamos (Imagen 7.3.3) y luego vemos las conexiones actuales en el puerto 1110 veremos que aparecen ambas conexiones.



Imagen 7.3.3: Corriendo dos terminales con IPv4 y IPv6



Protocol	Local Address	Foreign Address	State
tcp4	127.0.0.1.1110	127.0.0.1.60746	ESTABLISHED
tcp4	127.0.0.1.60746	127.0.0.1.1110	ESTABLISHED
tcp6	::1.1110	::1.60740	ESTABLISHED
tcp6	::1.60740	::1.1110	ESTABLISHED

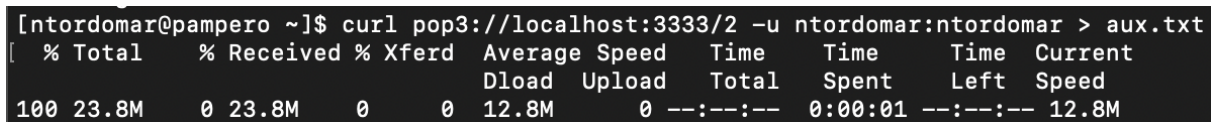
Imagen 7.3.4: Conexiones de la computadora en el puerto 1110

Podemos observar que aparecen ambas conexiones y cada una especifica su respectivo IP version. Por lo tanto, podemos afirmar que el servidor acepta tanto conexiones IPv4 como IPv6.

7.4 Enviar un mensaje a través de curl

Para verificar si efectivamente el servidor devuelve los mensajes de manera correcta, se decidió hacer una prueba con curl y diff.

En primer lugar, con el servidor corriendo en el puerto 3333, se hizo un curl del segundo correo de ntordomar y se guardó la salida en un archivo aux.txt.

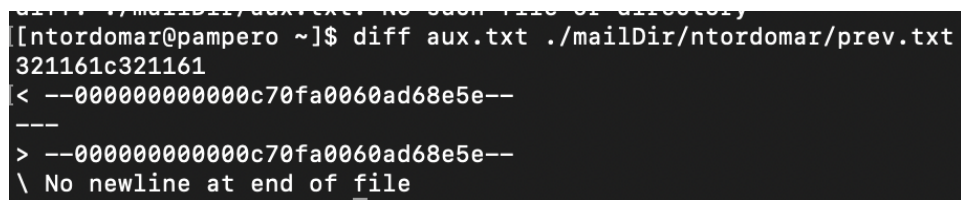


```
[ntordomar@pampero ~]$ curl pop3://localhost:3333/2 -u ntordomar:ntordomar > aux.txt
```

% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current
			Dload	Upload	Total	Spent	Left
100	23.8M	0 23.8M	0	0	12.8M	0 --:--:--	0:00:01 --:--:-- 12.8M

Imagen 7.4.1: Curl del correo 2 de ntordomar.

Este correo se corresponde con el archivo prev.txt que se encuentra en el directorio de ntordomar dentro del directorio mailDir (directorio que contiene las carpetas de los usuarios). Por lo tanto, se hizo un diff de aux.txt y prev.txt.

A terminal window with a dark background and light-colored text. The text shows a command prompt and the output of a 'diff' command. The output indicates a difference in the number of lines between two files, with a hexadecimal hash and a message about a missing newline at the end of the file.

```
diff: ./mailDir/aux.txt: No such file or directory
[ntordomar@pampero ~]$ diff aux.txt ./mailDir/ntordomar/prev.txt
321161c321161
< --000000000000c70fa0060ad68e5e--
---
> --000000000000c70fa0060ad68e5e--
\ No newline at end of file
```

Imagen 7.4.2: Diff de aux.txt y prev.txt

La única diferencia entre estos dos archivos parece ser la falta de un salto de línea al final de aux. Entonces al hacer curl se agrega un `\r\n` extra. Se atribuye entonces este comportamiento a dicho comando y se puede asegurar que los archivos son iguales.

7.5 Atender a múltiples clientes en forma concurrente y simultánea

El servidor implementado soporta conexiones de varios clientes a la vez y de forma eficiente ya que es no bloqueante.

Para poder testear cómo se comporta el servidor en situaciones de estrés y condiciones no favorables se hizo uso de la aplicación Apache JMeter como se propuso en las clases prácticas del TP. A continuación se detallan las pruebas hechas.

Test 1

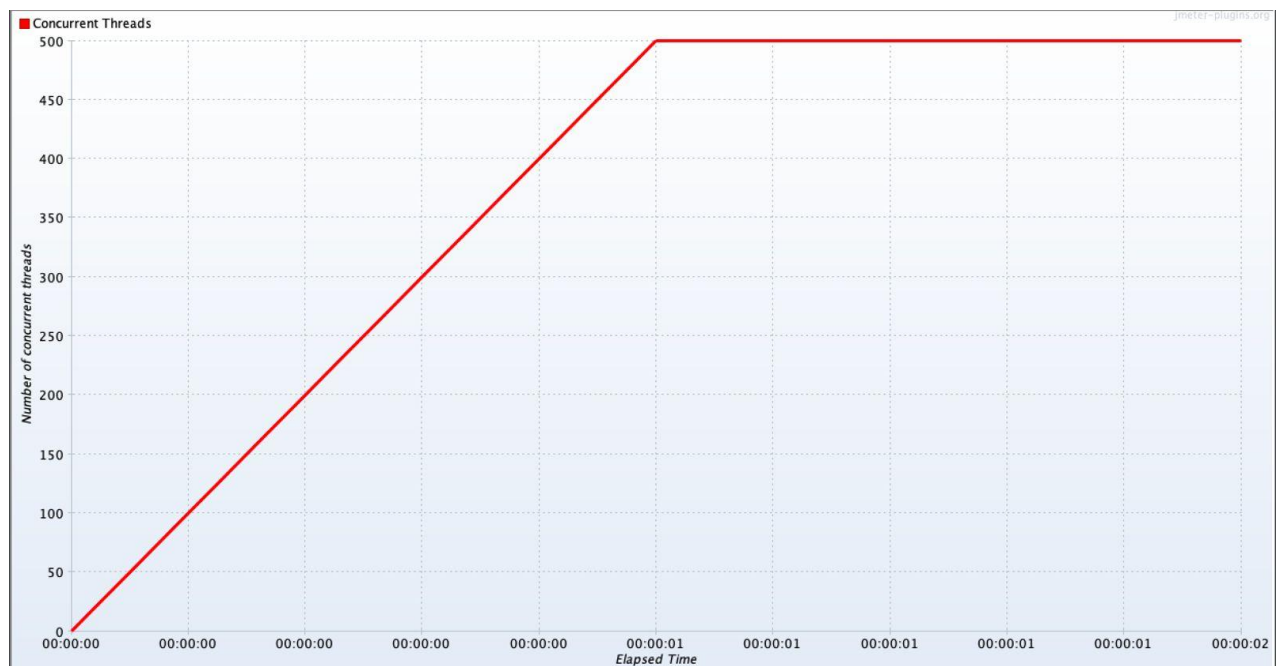


Imagen 7.5.1: Gráfico de cantidad de usuarios concurrentes en función del tiempo (Test 1)

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput
Mail Reader Sam...	519	16	0	45	9.80	0.00%	259.2/sec
TOTAL	519	16	0	45	9.80	0.00%	259.2/sec

Imagen 7.5.2: Resultados del Test 1

Como podemos observar, la cantidad de samples (clientes) que accedieron al servidor es de 519, con una tasa de error del 0%. Esta prueba fue realizada con el plugin Concurrency Thread Group (*Documentation*, n.d.) que te permite manejar hilos concurrentes que realizan operaciones de autenticación y lectura de mails en el servidor.

Test 2

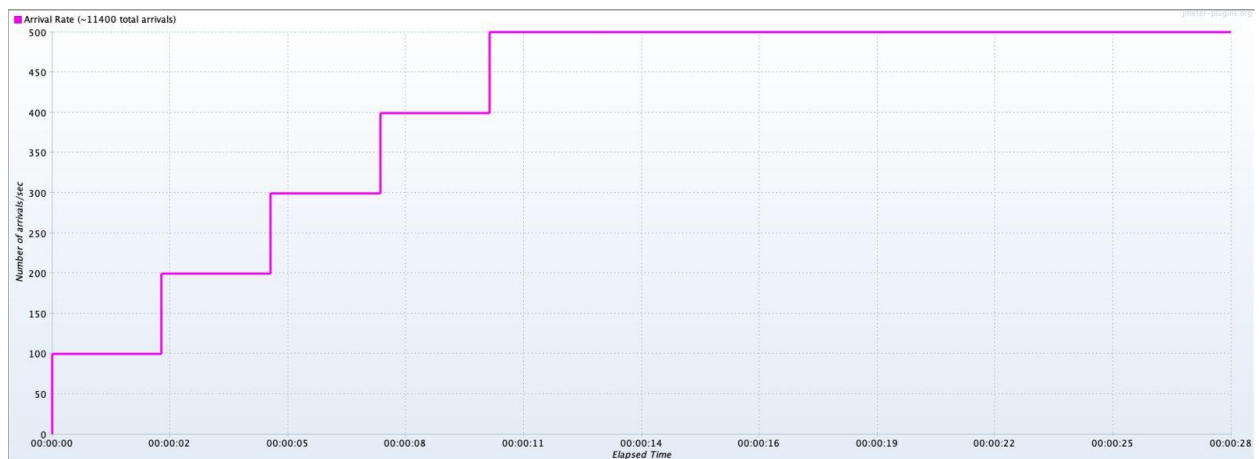
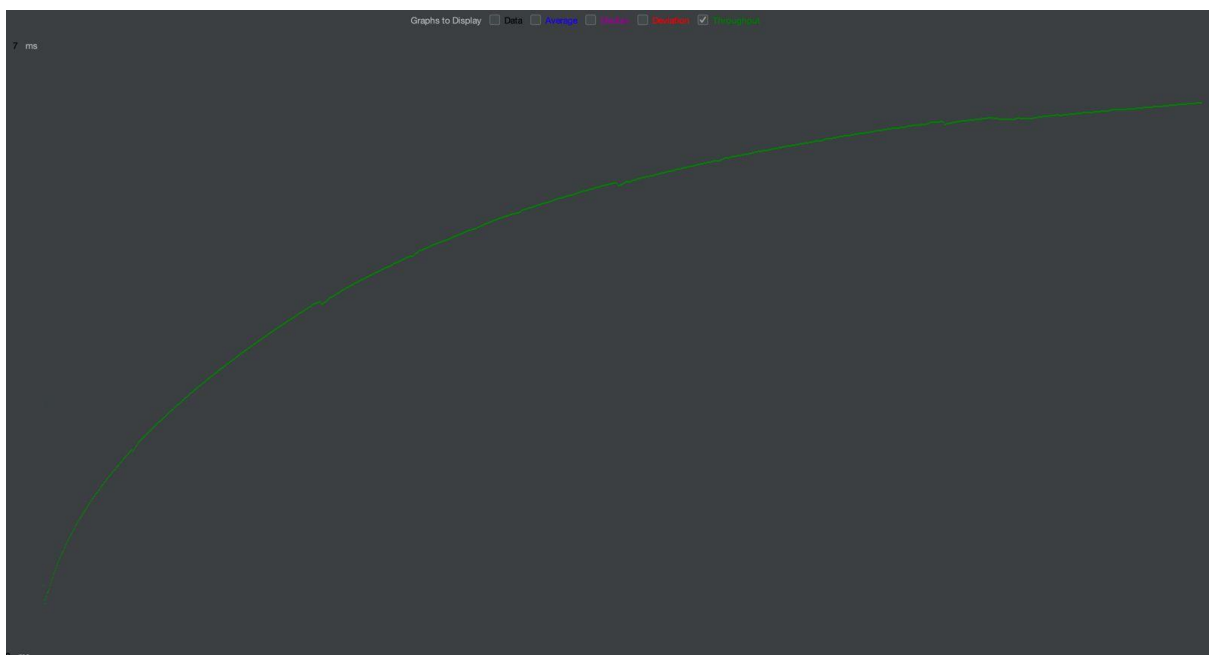


Gráfico de cantidad de llegadas concurrentes en función del tiempo. (Test 2)



Throughput del servidor en el test 2.

En este segundo test que se realizó, usamos la función de “concurrent arrivals” que trae el plugin. El test lo que logra medir y por eso nos pareció interesante para incluir en el informe, es una situación de estrés máxima donde cada segundo es mayor la cantidad de usuarios que ingresan al servidor. En la segunda imagen se puede ver cómo se va degradando el throughput.

8. Guía de instalación.

Una vez dentro de la carpeta del proyecto:

- I. Moverse a la carpeta src.
- II. Correr *make clean* y luego *make* y *make mc*.
- III. Para correr el servidor ejecutar el siguiente comando:

```
./pop3d -u <user>:<pass> -d <maildir> [-p <pop3_port>] [-P <manager_port>]
```

siendo lo que está entre corchetes opcional ([]) y donde

1. user es el usuario que se quiere autenticar
2. pass es la contraseña del usuario
3. maildir es el directorio de los mails del servidor
4. pop3_port es el puerto donde se conectan los clientes para usar el servidor POP3
5. manager_port es el puerto donde se conectan los clientes para usar el protocolo del manager

Para conectarse al servidor POP3 se debe ejecutar el siguiente comando

nc -C localhost <pop3_port> o *nc -C localhost 1110* si no hubiese un puerto POP3 definido.

- IV. En el caso de la aplicación de administración, ejecutar:

./manager_client localhost <manager_port> o *./manager_client localhost 9090* si no hubiese un puerto manager definido y donde manager_port tiene que coincidir con el puerto provisto al correr el servidor.

Alternativamente, se puede establecer la conexión con una herramienta externa, como nc:

nc 127.0.0.1 <manager_port> -u o *nc 127.0.0.1 9090 -u* si no hubiese un puerto manager definido.

9. Instrucciones y ejemplos de configuración y monitoreo.

Una vez compilado el servidor, su configuración inicial puede encontrarse ejecutando `./pop3d -h`. En líneas generales, se mantiene la siguiente estructura:

```
./pop3d -u <user>:<pass> -d <maildir> [-p <pop3_port>] [-P <manager_port>]
```

A modo de ejemplo:

```
./pop3d -u luciano:smtp -u paz:udp -d /tmp/maildir -p 1265 -P 9091
```

Establecerá `/tmp/maildir` como directorio raíz, establecerá el puerto 1265 para el servidor y 9091 para el manager.

Adicionalmente, se crearán los usuarios “luciano” y “paz”, y se espera que existan las correspondientes carpetas dentro de `/tmp/maildir`.

De manera similar, la aplicación de administración se puede ejecutar de la siguiente manera:

```
./manager_client localhost <manager_port>
```

A través de este, se podrían agregar usuarios con la instrucción:

```
C: ADD nicolás:arp
```

```
S: +OK
```

Creando el usuario “nicolás”, para poder ser utilizado por los clientes del servidor.

El manager también puede eliminar usuarios:

```
C: DEL luciano
```

```
S: +OK
```

Removiéndolo de la lista de posibles usuarios.

10. Documento de diseño del proyecto.

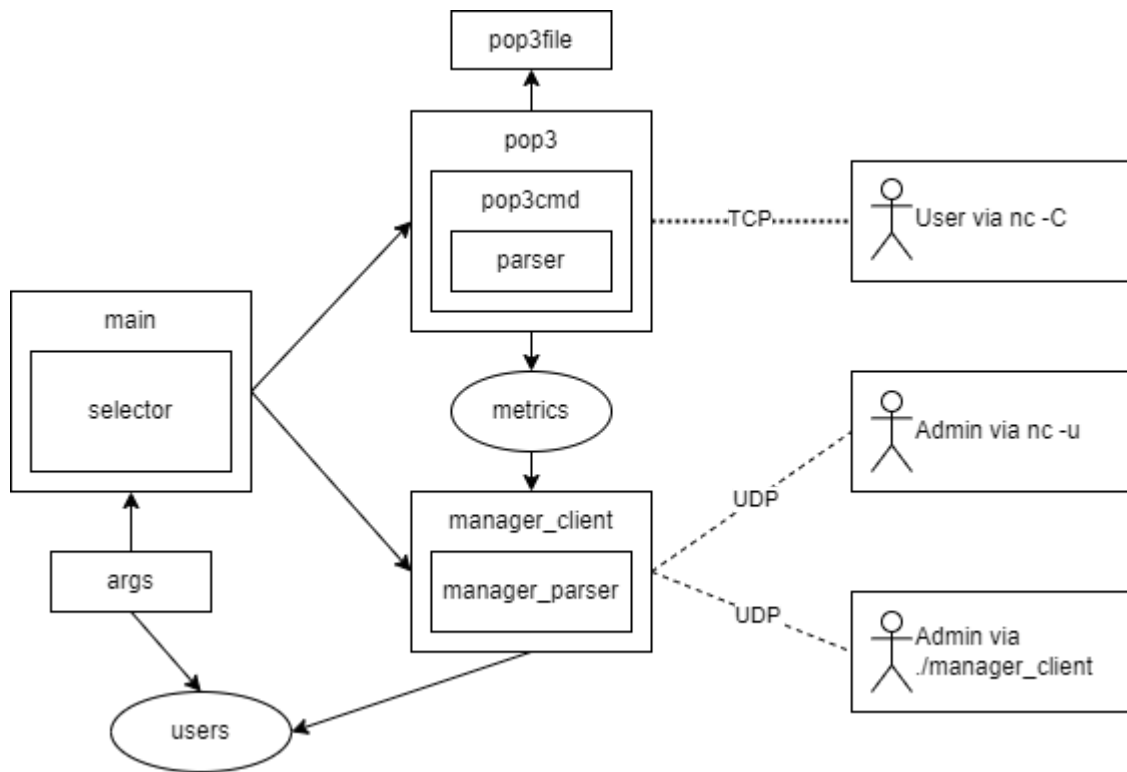


Imagen 10.1: Esquema de comunicación del proyecto

11. Referencias

^[1] *RFC 1939 POP3 Protocol*. (n.d.). IETF. Retrieved November 24, 2023, from

<https://www.ietf.org/rfc/rfc1939.txt>

^[2] *RFC 2449 POP3 Extension Mechanism*. (n.d.). IETF. Retrieved November 24, 2023, from

<https://www.ietf.org/rfc/rfc2449.txt>

^[3] Kerrisk, M. (n.d.). *ipv6(7) - Linux manual page*. man7.org. Retrieved November 24, 2023,

from <https://man7.org/linux/man-pages/man7/ipv6.7.html>

^[4] *Documentation*. (n.d.). Documentation :: JMeter-Plugins.org. Retrieved November 24,

2023, from <https://jmeter-plugins.org/wiki/ConcurrencyThreadGroup/>