



TPE: Servidor POP3 Protocolos de Comunicación - 72.07

Comisión S

IÑAKI BENGOLEA (63515),.....IBENGOLEA@ITBA.EDU.AR
FELIX LOPEZ MENARDI (62707),.....FLOPEZMENARDI@ITBA.EDU.AR
JOAQUÍN EDUARDO GIROD (63512),.....JGIROD@ITBA.EDU.AR
CHRISTIAN TOMÁS IJJAS (63555),CIJJAS@ITBA.EDU.AR

24 de Noviembre de 2023

Contents

1	Introducción	2
2	Aplicaciones y Protocolos Desarrollados	2
2.1	Protocolo CONF1	2
2.1.1	Escalabilidad	2
2.1.2	Operación Básica	3
2.1.3	Client Side	3
2.1.4	Server Side	4
2.1.5	Comando GET INF	4
2.1.6	Ejemplo de Uso Servidor Config	5
2.2	Servidor CONF1, UDP	5
2.3	Servidor POP3, TCP	6
3	Problemas Encontrados	7
4	Limitaciones de la aplicación	8
5	Posibles Extensiones	9
6	Conclusión	10
7	Ejemplo de Uso POP3	10
8	Guía de Instalación	12
9	Ejemplo de Uso Servidor Config	12
10	Diagrama de Diseño	13
11	Anexo	14

1 Introducción

En el desarrollo de este trabajo, se exploraron y diseñaron tres componentes fundamentales: el protocolo CONF1, la implementación de dicho protocolo y un servidor POP3. El protocolo CONF1 surge como respuesta a la necesidad de facilitar la comunicación con servidores activos, permitiendo realizar cambios relevantes o acceder a información básica de manera eficiente y 'on demand'. Este protocolo se enfoca en proporcionar al administrador la capacidad de realizar ajustes dinámicos sin interrupciones innecesarias para los usuarios. Por otro lado en cuanto al desarrollo del servidor POP3 se utilizaron los correspondientes RFC y estándares de la industria para proporcionar una experiencia lo más cercana posible a la esperada por parte de un usuario común. En las siguientes secciones se procede a describir el correcto funcionamiento y las técnicas utilizadas para llevar a cabo el trabajo general.

2 Aplicaciones y Protocolos Desarrollados

2.1 Protocolo CONF1

Los servidores tienden a acumular información y transitar muchas etapas desde que son levantados, también suelen tener horas de baja demanda y alta demanda, al igual que problemas con los usuarios que ameritan pequeños cambios en él. Para resolver estos inconvenientes se desarrolló CONF1 que busca permitir una comunicación con un servidor activo y que a partir de esta comunicación se efectúen cambios relevantes dentro del servidor o simplemente se acceda a información básica de forma rápida. Por ende se evitarían reinicios innecesarios con downtime para los usuarios, el análisis extenso de logs para obtener información básica y se habilitaría la posibilidad de alterar el servidor dinámicamente esencialmente aumentando las capacidades iniciales del servidor. La intención de este protocolo es que sea utilizado por el administrador del mismo ya que permite acceso a tanto edición como obtención de información sensible.

2.1.1 Escalabilidad

La escalabilidad del protocolo es un aspecto importante del protocolo, buscamos permitir la mayor flexibilidad dentro de lo razonable a la hora de exponer servicios para que sean configurados, esto se denotará más adelante con la lista de Códigos de Objeto. Queda en plena libertad de quien este implementando este protocolo que tan permisivo o restrictivo quiere ser con los recursos puestos a disposición. A pesar de esto la sección de respuesta del servidor no permite mucha flexibilidad más allá del contenido en la respuesta. Esto es intencional y se debe a que el propósito de este protocolo es acceder a recursos básicos y obtener información rápida. De ser requerido cambios profundos dentro de la aplicación esperamos que el desarrollador se incline por otros protocolos, por lo menos uno que trabaje sobre TCP para garantizarse la efectividad de los

cambios con mayor seguridad. A una conclusión similar se llega con el aspecto de obtención de información, esperamos que un análisis extensivo de los datos sea hecho por otros medios, potencialmente analizando los logs y no en base a las respuestas de este protocolo.

Se opta por priorizar la simplicidad tanto en las peticiones del cliente como en las respuestas del servidor, incluso omitiendo un parámetro de versión ya que este aumentaría el tamaño de las peticiones y no se espera que vaya a ser trascendente en el futuro.

2.1.2 Operación Básica

Un servidor de configuración UDP es levantado en el puerto 9090, para hacer uso del servicio de configuración el cliente **debe** enviar datagramas UDP a dicho puerto. Los comandos se dividen en dos grupos: los de listado y los de edición. Ambos requieren autenticación, un código de objeto referente a lo que se esta solicitando o alterando y en caso de alteración se agrega un argumento. Ante la recepción de uno de estos datagramas el servidor responde con un código de éxito o fracaso y la información solicitada o pertinente a la acción.

2.1.3 Client Side

El cliente **debe** conformar comandos de 3 o 4 miembros en ASCII imprimibles separados por un único espacio y finalizados por '\r\n', de un máximo de 128 octetos teniendo en cuenta el CRLF. Además **deben** acatar con la siguiente estructura:

$$CR = <ATHTK> <VERB> <OC> <ARG> \quad (1)$$

Donde,

- **CR** es Client Request (Petición del Cliente)
- **ATHTK**, Authorization Token (Token de Autorización) **debe** ser una combinación de caracteres ASCII alfabéticos de un tamaño fijo de 10 caracteres. Este token de verificación es acordado entre el servidor a ser configurado y el cliente administrador de dicho servidor. Queda a disposición de la implementación si hacer la decidid si la comparación es case sensitive o case insensitive, aunque se recomienda case insensitive como el resto del protocolo.
- **VERB** (verbo) indica la operación a ejecutarse, tanto GET como SET **deben** estar acompañados de un Código de Objeto pero GET no **debe** tener el cuarto miembro opcional del esqueleto, el argumento. El caso del SET, destinado a señalar la operación de edición sobre las variables **debe** tener el argumento señalado en el cuarto miembro del comando, este refiere al nuevo valor tras la concreción de la operación.

- **OC**, Object Code (Código de Objeto) **deben** ser 3 caracteres que refieren al objeto sobre el cual se esta realizando la acción destacada en el verbo. Los objetos sobre los cuales se pueden realizar acciones son definidos por el servidor y **deben** ser notificadas de ser pedido. Dentro de los códigos de objetos siempre **debe** estar presente INF, que al hacerse un GET **debe** retornar la lista con los Códigos de Objetos del servidor encuestado.
- **ARG** es argument (argumento) es un parámetro **OPCIONAL** y **debe** ser una cadena de ASCII imprimibles alfanumérica, dependiendo de la acción y el objeto sobre el cual se este realizando, el tipo de dato al cual hace referencia esta cadena puede variar y queda a disposición del servidor esta relación entre el par (**VERB**, **OC**) y el tipo de dato requerido.

2.1.4 Server Side

El servidor **debe** decidir sobre los objetos que va a exponer para que sean configurados, al igual que las validaciones en los casos en que se reciban argumentos, el protocolo solo establece que los argumentos sean en ASCII imprimibles y no excedan el tamaño máximo. Entonces el servidor **debe** establecer una lista con la codificación de 3 caracteres para cada uno de estos objetos al igual que la validación acorde al par (**VERB**, **OC**), ejemplos de validaciones serán expuestos mas adelante. Esta lista **debe** ser accesible por medio de una petición GET INF, esta petición sera analizada con mas detalle más adelante.

Todas las respuestas **deben** estar conformadas por dos miembros separados por un único espacio escrito en ASCII imprimibles finalizados con '\r\n' y teniendo un máximo de 255 octetos teniendo en cuenta el CRLF. Además **deben** acatar a la siguiente estructura:

$$SR = < SC > < RESP > \quad (2)$$

Donde,

- **SR** es Server Response (Respuesta del Servidor)
- **SC**, Status Code (Código de Estado) **debe** preceder a todos los comandos, con una estructura de 4 caracteres indicativos del estado de la operación realizada, **debe** estar dentro de estos indicativos de estado '+SUC' y '-ERR'. El servidor **puede** incluir mas estados para las respuestas, pero estos deben ser especificados en la respuesta de GET INF.
- **RESP** Response (Respuesta) admite flexibilidad en cuanto a su contenido, siempre y cuando sean caracteres ASCII imprimibles, no se exceda el máximo de 255 caracteres y se finalice con CRLF. Se espera que para las respuestas de error se ofrezca información pertinente.

2.1.5 Comando GET INF

Una petición GET INF tiene la siguiente forma:

$$CR = < ATHTK > GET INF \quad (3)$$

Donde se mantienen las referencias de **CR** y **ATHTK** mencionadas previamente. Ante una petición con token valido de GET INF, el servidor **debe**, en caso de éxito, retornar en el segundo miembro **RESP** (Ver ecuación (2)) se **debe** incluir la lista de Códigos de Objetos que son accedidos únicamente por GET separados por una coma y un espacio luego un pipe una combinación de ' — ' (SPACE, PIPE, SPACE) y la lista de Códigos de Objetos que pueden ser accedidos tanto por GET como por SET, de nuevo separando cada uno por una coma y un espacio. El primero de los comandos en la lista de accesibles por medio de GET **debe** ser INF.

A continuación un ejemplo de un retorno de GET INF, donde el servidor expone como objetos accesibles por medio de GET, los buffers (BUF) y el recuento de usuario totales (TTU), mientras que accesible tanto por medio de GET como SET expone la locación de la carpeta de Logs (LGF) y el Authorization Token (ATT) utilizado para verificarse como administrador (ATT). Supongamos Authorization Token = 'adminadmin'.

```
C : adminadmin GET INF
S : +SUC Valid Object Codes : INF, BUF, TTU $|$
    LGF, TTU
```

2.1.6 Ejemplo de Uso Servidor Config

A continuación un ejemplo de uso del protocolo donde se definió la siguiente tabla de Object Codes y validaciones:

Cód. Objeto	Objeto	Acciones	Validación
INF	Information	GET	-
TTU	Total Users	GET	-
BUF	Buffers Size	GET/SET	Entero entre 8 y 65536

2.2 Servidor CONF1, UDP

Se implemento un thread para el servidor de configuración que atiende las consultas de UDP provenientes de IPv4 y las lee, parsea y contesta inmediatamente, ya que no se mantiene una sesión como es el caso de POP3/TCP, no se requiere hacer un thread por cliente. Se implementó el protocolo detallado previamente, la lista de Códigos de Objetos y sus validaciones correspondientes es la siguiente.

Cód. Objeto	Objeto	Acciones	Validación
INF	Information	GET	-
HTU	Historic Users	GET	-
CCU	Concurrent Users	GET	-
BTF	Bytes Transferred	GET	-
BUF	Buffer Sizes	GET	-
LGF	Log Folder	GET/SET	String no nulo de tamaño menor a 128
MDF	Maildir Folder	GET/SET	String no nulo de tamaño menor a 128
ATT	Authorization Token	GET/SET	String de tamaño 10
TRF	Transformation Active	GET/SET	String con valor 'true' o 'false'
TFN	Transformation Function	GET/SET	String no nulo de tamaño menor a 128

De esta forma se aprovecha el poder del protocolo que permite el monitoreo y configuración de ciertas variables del servidor.

2.3 Servidor POP3, TCP

Se hizo una implementación de servidor basado en el protocolo POP3 (RFC 1939 y RFC 2449) con threads bloqueantes que acepta conexiones tanto IPv4 como IPv6, se ofrece logging, un servicio de transformación de mensajes configurable por medio del servidor de configuración y los argumentos iniciales al levantarse el servidor, al igual que acceso restringido a los recursos de cada usuario por medio de semáforos y .

Se realizaron tres tests sobre la concurrencia de la aplicación. En el primer testeo (Anexo Figura 3), no se notó degradación significativa (Anexo Figura 4), mientras que en el segundo testeo (Anexo Figura 5), sí se pudo notar una degradación significativa de performance (Anexo Figuras (a)-(f)). El tercer test consistió en crear 500 conexiones en simultaneo. Resultó ser un testeo exitoso dado que el sistema mantuvo comportamiento normal a pesar de la gran cantidad de conexiones.

Los resultados que muestran degradación son lógicos y previsibles ya que el servidor esta siendo bombardeado con múltiples requests, la solución a esto seria escalar el poder computacional del servidor, una implementación no bloqueante también podría llegar a tener efectos positivos, aunque intuimos que tendría un problema similar.

La presencia de 500 usuarios conectados no afecta la experiencia del próximo usuario, lo cual es lógico ya que lo que se pierde seria memoria reservada y no

mucho poder computacional, es scheduler no le daría tiempo de procesador a los threads bloqueados. El verdadero problema provendría de 500 usuarios concurrentes interactuando activamente con el servidor y no simplemente conectados, en este caso sí se espera que el próximo usuario experimente una calidad de servicio decremada ya que el tiempo del procesador debe ser repartido entre mas threads.

Donde,

- **CR** es Client Request (Petición del Cliente)
- **CMD**, es el comando que **debe** ser una combinación de caracteres ASCII alfabéticos de un tamaño fijo de 4 caracteres. Los comandos válidos que se aceptarán son los siguientes: CAPA, USER, PASS, STAT, LIST, RETR, DELE, NOOP, RSET y QUIT.
- **ARG** es argument (argumento), un parámetro **OPCIONAL** y **debe** ser una cadena de ASCII imprimibles alfanumérica, cuyo valor depende del comando que se quiera ejecutar.

3 Problemas Encontrados

- El primer problema encontrado provino lógicamente de la implementación rigurosa del RFC, si bien los textos de RFC son concisos y detallados, hay secciones que se abren a la decisión del desarrollador o no se elabora con tanta rigurosidad. En estas disyuntivas siempre se opto por imitar el funcionamiento de Dovecot, por verificar la conducta que tomaba Dovecot a la situación en cuestión e imitarla lo mas posible.
- Otro de los problemas que preveíamos desde que vimos los contenidos teóricos, es el de aprovechar el funcionamiento que te garantiza el protocolo de transporte que va por debajo de la aplicación además de elegir el correcto para el protocolo a desarrollar. TCP que es utilizado por POP3 envía información de a streams, que si bien esta garantizada de llegar en orden, no esta garantizado que llegue la totalidad del mensaje en un stream, sin importar el tamaño del mensaje. Esta limitación del protocolo de transporte se combinaba con el pipelining para hacer de la tokenización del mensaje completo un desafío, esto se termino resolviendo llevando este problema al thread handler y haciendo que el parser siempre reciba comandos "limpios".
- Otro desafío fue el de lograr Logs coherentes y útiles para el administrador del servidor, al principio pensamos en una implementación simple donde todos los Logs fueran al mismo archivo, pero esto los hacia totalmente inusables a gran escala. Terminamos con una implementación donde cada thread notifica su accionar en un archivo aparte, esto se combina con los access logs para en caso de ser necesario verificar la transacción que llevo a cabo un usuario en particular, esto es particularmente útil en caso de que

se notifiquen bugs porque solo seria cuestión de buscar la sesión utilizada. Al descentralizar los logs, también pudimos mejorar la performance, si 20 threads quisiesen loggear al mismo archivo nos estaríamos llevando la peor parte del readers-writers en cuanto a semáforos y bloqueos, por ende también nos llevamos un saldo positivo en performance con esta implementación. Obviamente es necesario, como con cualquier logger, que sean regularmente eliminados para no generar una acumulación de archivos basura.

- Otro obstáculo que tiende a ser recurrente en cualquier proyecto es el de utilizar código de terceros, si bien en este caso fue el provisto por la cátedra, siempre existe ese periodo de adaptación al uso de una implementación externa.
- La escritura adecuada del Protocolo nuestro también presento dificultad, lograr ser consciente e intencional con cada aspecto que se deja a juicio de la implementación y cada aspecto que es obligatorio, es un trabajo mental exhaustivo donde la simpleza y el sentido común triunfan. Por ejemplo al principio habíamos obviado totalmente el hecho de que el administrador tiene que de alguna manera averiguar los objetos expuestos por el servidor, una ceguera que proviene de estar haciendo una implementación de dicho protocolo en paralelo.

4 Limitaciones de la aplicación

- La principal limitación de la aplicación es el hecho de ser no bloqueante, esto produce una degradación de la performance con el aumento de los usuarios, si bien con una cantidad baja es posible argumentar que hay un mejor uso de los cores de un procesador a una cantidad alta es indudable que una implementación no bloqueante es mejor.
- La descarga del mails de gran tamaño es lenta en comparación con Dovecot, esto se puede deber a los tamaños de buffers utilizados no siendo los mas optimos, al igual que la estrategia utilizada para lidiar con el byte-stuffing (Anexo Figura 8).
- Si bien se soporta pipelining, en el caso en que el string pipelineado exceda el tamaño máximo permitido por comando (255 bytes), aunque no lo exceda cada comando en particular, la aplicación lo toma como un comando demasiado grande.
- Si bien se soportan transformaciones el byte stuffing este se hace previo a la transformación y no posterior, suponemos que cualquier sistema de filtrado de mails esta al tanto de la problemática del '\r\n.\r\n'.
- El servidor de configuración CONF1/UDP no acepta conexiones IPv6.

- Access Log si sufre el bloqueo de readers-writers ya que muchos threads potencialmente escriben en este archivo, aunque la magnitud del bloqueo es despreciable ya que cada thread haría una sola escritura en este archivo como máximo.
- Dentro de las limitaciones evidentes de la aplicación están aquellos valores que fueron elegidos de forma arbitraria, como seria el numero de usuarios concurrentes que esta puesto en 500 o el numero de conexiones en espera de cada socket que esta puesto en 20.
- El mensaje que se recupera con la función RETR tiene un '\n' demás al final, hubo arduos intentos de arreglar este pequeño error y se llevo a una versión que lo removía pero de una forma muy forzada que no funcionaba bien con archivos grandes por ejemplo.

5 Posibles Extensiones

Cabe destacar que si nos atenemos a las palabras del RFC 2449, diseñado para posibilitar las extensiones a POP3, podemos ver notar una cierta resiliencia a extenderlo. En el mismo se encuentra lo siguiente : "This extension to the POP3 protocol is to be used by a server to express policy descisions taken by the server administrator. It is not an endorsement of implementations of further POP3 extensions generally. It is the general view that the POP3 protocol should stay simple, and for the simple purpose of downloading email from a mail server. If more complicated operations are needed, the IMAP protocol [RFC 2060] should be used." A pesar de esto las mejoras que proponemos son sobre las funcionalidades actuales.

- No bloqueante es el paso evidente que mejoraría drásticamente la implementación como fue mencionado en incisos previos.
- Mejorar autenticación del admin, aunque esto implicaría cambiar el protocolo a uno que ofrezca mas seguridad.
- Añadir Timeouts de conexiones, actualmente no existe un timeout.
- Añadir una capa de seguridad y autenticación al POP3, esta capa si es recomendada por el RFC 2449.
- Con el objetivo de simplificar el código y evitar errores, se utiliza strlen() en algunas funciones cuando se podría usar la versión con tamaño fijo de dicha función, esto tiene un costo en performance.
- Se podría hacer un implementación no volátil de las estadísticas del servidor.
- El parseo podría hacerse con una complejidad menor aunque mejoras en esta área no tiene efectos grandes en la experiencia de uso de la aplicación ya que el parseo siempre es de comandos pequeños.

6 Conclusión

La ejecución del proyecto posibilitó que el conjunto completo obtuviera una comprensión profunda sobre la implementación de un protocolo personalizado, así como también adquirir conocimientos sobre el funcionamiento de un protocolo POP3. El proyecto demandó un esfuerzo considerable por parte de todos, dado su alcance extenso, que implicó investigación en todos los aspectos vinculados al mismo.

Identificamos los siguientes temas como aquellos en los que experimentamos un mayor crecimiento en conocimientos: Protocolo POP3, Sockets, funcionamiento TCP y UDP, IPv4 e IPv6.

Finalmente, también ganamos experiencia al trabajar con herramientas de depuración como valgrind, address sanitizer y diversas flags que se pueden emplear en la compilación de un proyecto.

7 Ejemplo de Uso POP3

Primero se debe ejecutar el archivo llamado 'populator' quien cargará correos electrónicos en un usuario ejemplo:

```
$ ./populator.sh
```

Próximamente se debe ejecutar el ejecutable llamado pop3d para levantar el servidor POP3:

```
$ ./pop3d -u testuser:testuser -u testuser2:
testuser2 -u testuser3:testuser3 -t 'sed_s/[
Aa]/4/g|sed_s/[Ee]/3/g|sed_s/[iI]/1/g|sed_s/[
0o]/0/g'
```

En este ejemplo creamos tres usuarios, el primero de ellos con nombre de usuario 'testuser' y contraseña 'testuser'. Se puede agregar más de uno repitiendo el flag '-u' y las credenciales de dichos usuarios en el formato exhibido. También se especificó una transformación por si se quisiera utilizarla, que inter-cambia las vocales de un correo por su contraparte numérica similar.

Luego, desde otra terminal, nos conectaremos al servidor en el puerto 1110 utilizando netcat:

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

A continuación se exhiben posibles comandos y sus respectivas respuestas del servidor, donde C representa al cliente y S al servidor:

```
→ PDC-POP3D git:(master) ✕ nc -C localhost 1110
+OK POP3 server ready (^~^)\n
capa\n
+OK\n
CAPA\n
USER\n
PIPELINING\n
.\n
user testuser\n
+OK\n
pass wrongpassword\n
-ERR [AUTH] Authentication failed.\n
user testuser\n
+OK\n
pass testuser\n
+OK Logged in.\n
list\n
+OK 6 messages:\n
1 753\n
2 442\n
3 1008\n
4 52428800\n
5 753\n
6 1976\n
.\n
retr 2\n
+OK 442 octets\n
Return-Path: <joaquin@foo.pdc>\n
X-Original-To: joaquin@foo.pdc\n
Delivered-To: joaquin@foo.pdc\n
Received: from joaquin (localhost [127.0.0.1])\n
        by joaquin.foo.pdc (Postfix) with ESMTF id B6F7217E87A\n
        for <joaquin@foo.pdc>; Thu, 23 Nov 2023 00:05:12 -0300 (-03)\n
Message-Id: 950124.162336@example.com\n
Subject: Carpe Diem\n
from: joaquingirod@gmail.com\n
to: jgirod@itba.edu.ar\n
Date: Thu, 23 Nov 2023 00:05:12 -0300 (-03)\n\n
Quam minimum credula postero\n
.\n
[ ]
```

Figure 1: Pedidos y respuestas de servidor POP3

8 Guía de Instalación

Los archivos se pueden compilar ejecutando el siguiente comando en el directorio raíz del proyecto:

```
$ make clean; make all
```

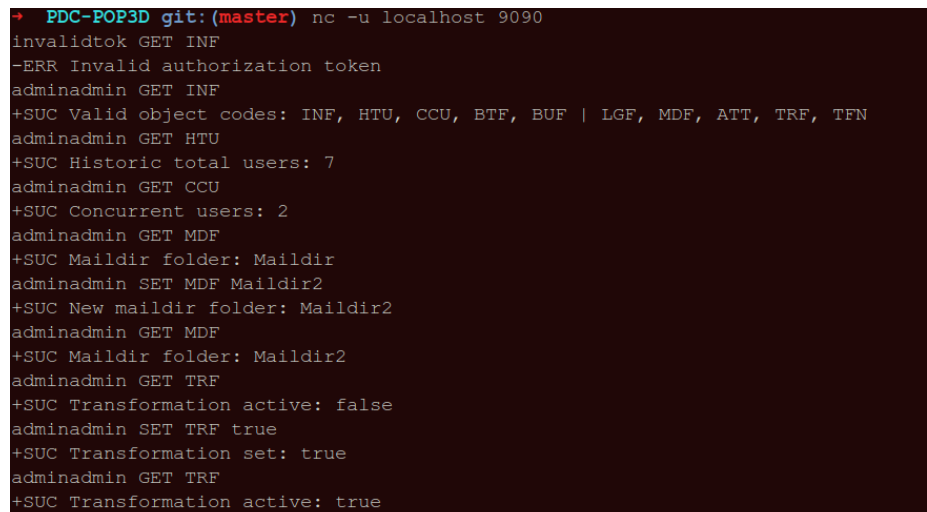
Luego se crearán ejecutables que se ejecutan como se exhibió en las secciones de ejemplos de uso.

9 Ejemplo de Uso Servidor Config

A continuación un ejemplo de uso del servidor de configuración. Se accedió utilizando netcat en el puerto 9090 y especificando el flag de UDP:

```
$ nc -u localhost 9090
```

Luego se utilizaron los códigos objeto y acciones definidos en la sección 2.2 y reconociendo como válido al authorization token 'adminadmin':



```
+ PDC-POP3D git:(master) nc -u localhost 9090
invalidtok GET INF
-ERR Invalid authorization token
adminadmin GET INF
+SUC Valid object codes: INF, HTU, CCU, BTF, BUF | LGF, MDF, ATT, TRF, TFN
adminadmin GET HTU
+SUC Historic total users: 7
adminadmin GET CCU
+SUC Concurrent users: 2
adminadmin GET MDF
+SUC Maildir folder: Maildir
adminadmin SET MDF Maildir2
+SUC New maildir folder: Maildir2
adminadmin GET MDF
+SUC Maildir folder: Maildir2
adminadmin GET TRF
+SUC Transformation active: false
adminadmin SET TRF true
+SUC Transformation set: true
adminadmin GET TRF
+SUC Transformation active: true
```

Figure 2: Ejemplo de uso del servidor de configuración

10 Diagrama de Diseño

A continuación se exhibe un diagrama que refleja del flujo del proyecto, especificando los puertos y protocolos utilizados.

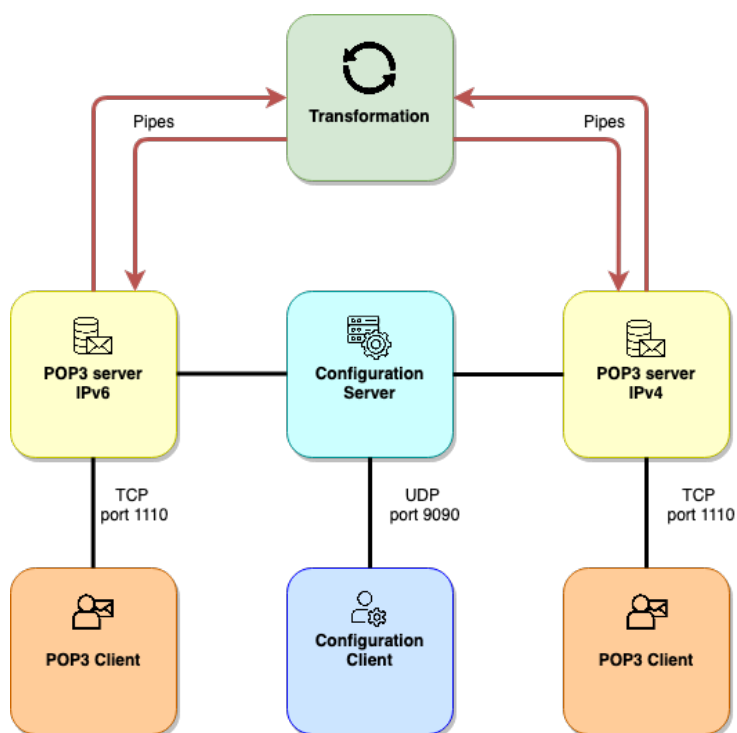


Figure 3: Diagrama del diseño de flujo del proyecto

11 Anexo

```
1  #!/bin/bash
2
3  # Number of users to connect
4  num_users=200
5
6  # Server details
7  server_host="localhost"
8  server_port=1110
9
10 # Function to connect a user in a separate thread
11 connect_user() {
12 |   nc -C $server_host $server_port
13 | }
14
15 # Record the start time
16 start_time=$(date +%s.%N)
17
18 # Loop to connect users
19 for ((i = 1; i <= num_users; i++)); do
20 |   # Call the connect_user function in the background
21 |   connect_user &
22 | done
23
24 # Wait for all background processes to finish
25 wait
26
27 # Record the end time
28 end_time=$(date +%s.%N)
29
30 # Calculate the duration
31 duration=$(echo "$end_time - $start_time" | bc)
32
33 # Display the throughput
34 echo "Throughput: $num_users connections in $duration seconds."
```

Figure 4: Test de Concurrency #1 (throughput.sh)

```

→ src git:(d67f081) x
./throughput.sh
Simulating load with concurrency: 20
Curl request 1 took 52 milliseconds
Curl request 2 took 50 milliseconds
Curl request 3 took 50 milliseconds
Curl request 4 took 50 milliseconds
Curl request 5 took 50 milliseconds
Curl request 6 took 50 milliseconds
Curl request 7 took 49 milliseconds
Curl request 8 took 50 milliseconds
Curl request 9 took 50 milliseconds
Curl request 10 took 51 milliseconds
Average curl response time: 50 milliseconds
Simulating load with concurrency: 50
Curl request 1 took 49 milliseconds
Curl request 2 took 51 milliseconds
Curl request 3 took 49 milliseconds
Curl request 4 took 50 milliseconds
Curl request 5 took 55 milliseconds
Curl request 6 took 49 milliseconds
Curl request 7 took 50 milliseconds
Curl request 8 took 50 milliseconds
Curl request 9 took 51 milliseconds
Curl request 10 took 54 milliseconds
Average curl response time: 50 milliseconds
Simulating load with concurrency: 100
Curl request 1 took 51 milliseconds
Curl request 2 took 50 milliseconds
Curl request 3 took 49 milliseconds
Curl request 4 took 50 milliseconds
Curl request 5 took 50 milliseconds
Curl request 6 took 54 milliseconds
Curl request 7 took 50 milliseconds
Curl request 8 took 50 milliseconds
Curl request 9 took 50 milliseconds
Curl request 10 took 50 milliseconds
Average curl response time: 50 milliseconds
Simulating load with concurrency: 200
Curl request 1 took 49 milliseconds
Curl request 2 took 50 milliseconds
Curl request 3 took 51 milliseconds
Curl request 4 took 53 milliseconds
Curl request 5 took 54 milliseconds
Curl request 6 took 50 milliseconds
Curl request 7 took 49 milliseconds
Curl request 8 took 54 milliseconds
Curl request 9 took 50 milliseconds
Curl request 10 took 54 milliseconds
Average curl response time: 51 milliseconds
Simulating load with concurrency: 400
Curl request 1 took 51 milliseconds
Curl request 2 took 50 milliseconds
Curl request 3 took 50 milliseconds
Curl request 4 took 50 milliseconds
Curl request 5 took 50 milliseconds
Curl request 6 took 50 milliseconds
Curl request 7 took 53 milliseconds
Curl request 8 took 50 milliseconds
Curl request 9 took 50 milliseconds
Curl request 10 took 51 milliseconds
Average curl response time: 50 milliseconds

```

Figure 5: Resultados Test de Concurrencia #1


```

1  #!/bin/bash
2
3  stress_test() {
4      local SERVER="localhost"
5      local PORT="1110"
6      local USERNAME="testuser"
7      local PASSWORD="testuserpassword"
8      local TOTAL_REQUESTS=$1
9
10     echo "Performing stress test with $TOTAL_REQUESTS curls"
11
12     total_time=0
13     for ((i = 1; i <= TOTAL_REQUESTS; i++)); do
14         (
15             start_time=$(date +%s%N)
16             curl -u "$USERNAME:$PASSWORD" "pop3://$SERVER:$PORT/1" >/dev/null 2>&1
17             end_time=$(date +%s%N)
18             elapsed_time=$((end_time - start_time) / 1000000) # Convert nanoseconds to milliseconds
19             echo "Curl request $i took $elapsed_time milliseconds"
20         ) &
21     done
22
23     # Wait for all background processes to finish
24     wait
25
26 }
27
28 # Call the function with increasing numbers of curls
29 stress_test 10
30 sleep 1
31 stress_test 50
32 sleep 1
33 stress_test 100
34 sleep 1
35 stress_test 200

```

Figure 6: Test de Concurrency #2 (stress.sh)

```

→ src git:(d67f081) x
./stress.sh
Performing stress test with 10 curls
Curl request 4 took 9 milliseconds
Curl request 8 took 9 milliseconds
Curl request 5 took 9 milliseconds
Curl request 9 took 9 milliseconds
Curl request 6 took 9 milliseconds
Curl request 7 took 9 milliseconds
Curl request 3 took 10 milliseconds
Curl request 1 took 11 milliseconds
Curl request 2 took 11 milliseconds
Curl request 10 took 10 milliseconds
Performing stress test with 50 curls
Curl request 5 took 14 milliseconds
Curl request 7 took 28 milliseconds
Curl request 4 took 29 milliseconds
Curl request 12 took 28 milliseconds
Curl request 6 took 29 milliseconds
Curl request 8 took 30 milliseconds
Curl request 10 took 30 milliseconds
Curl request 17 took 30 milliseconds
Curl request 1 took 32 milliseconds
Curl request 15 took 31 milliseconds
Curl request 2 took 33 milliseconds
Curl request 21 took 30 milliseconds
Curl request 3 took 33 milliseconds
Curl request 14 took 31 milliseconds
Curl request 23 took 27 milliseconds
Curl request 38 took 24 milliseconds
Curl request 30 took 24 milliseconds
Curl request 36 took 23 milliseconds
Curl request 13 took 33 milliseconds
Curl request 28 took 22 milliseconds
Curl request 35 took 23 milliseconds
Curl request 19 took 23 milliseconds
Curl request 34 took 25 milliseconds
Curl request 49 took 22 milliseconds
Curl request 16 took 33 milliseconds
Curl request 43 took 26 milliseconds
Curl request 37 took 24 milliseconds
Curl request 45 took 13 milliseconds
Curl request 44 took 13 milliseconds
Curl request 29 took 28 milliseconds
Curl request 47 took 21 milliseconds
Curl request 32 took 30 milliseconds
Curl request 39 took 21 milliseconds
Curl request 22 took 29 milliseconds
Curl request 31 took 210 milliseconds
Curl request 27 took 211 milliseconds
Curl request 18 took 220 milliseconds
Curl request 40 took 216 milliseconds
Curl request 11 took 223 milliseconds
Curl request 46 took 216 milliseconds
Curl request 50 took 214 milliseconds
Curl request 9 took 226 milliseconds
Curl request 42 took 217 milliseconds
Curl request 33 took 216 milliseconds
Curl request 24 took 217 milliseconds
Curl request 41 took 216 milliseconds
Curl request 25 took 218 milliseconds
Curl request 26 took 214 milliseconds

```

((a)) Resultados Test de Concurrency
#2

```

Curl request 20 took 216 milliseconds
Curl request 48 took 219 milliseconds
Performing stress test with 100 curls
Curl request 5 took 14 milliseconds
Curl request 3 took 17 milliseconds
Curl request 1 took 34 milliseconds
Curl request 7 took 35 milliseconds
Curl request 9 took 38 milliseconds
Curl request 10 took 53 milliseconds
Curl request 4 took 55 milliseconds
Curl request 6 took 56 milliseconds
Curl request 11 took 58 milliseconds
Curl request 12 took 59 milliseconds
Curl request 8 took 61 milliseconds
Curl request 14 took 58 milliseconds
Curl request 23 took 57 milliseconds
Curl request 15 took 59 milliseconds
Curl request 22 took 53 milliseconds
Curl request 20 took 55 milliseconds
Curl request 27 took 54 milliseconds
Curl request 24 took 54 milliseconds
Curl request 2 took 64 milliseconds
Curl request 13 took 61 milliseconds
Curl request 34 took 52 milliseconds
Curl request 30 took 54 milliseconds
Curl request 25 took 60 milliseconds
Curl request 49 took 52 milliseconds
Curl request 16 took 57 milliseconds
Curl request 69 took 47 milliseconds
Curl request 17 took 54 milliseconds
Curl request 41 took 52 milliseconds
Curl request 58 took 41 milliseconds
Curl request 39 took 23 milliseconds
Curl request 84 took 30 milliseconds
Curl request 99 took 17 milliseconds
Curl request 86 took 19 milliseconds
Curl request 87 took 17 milliseconds
Curl request 93 took 13 milliseconds
Curl request 94 took 22 milliseconds
Curl request 92 took 18 milliseconds
Curl request 85 took 16 milliseconds
Curl request 80 took 36 milliseconds
Curl request 97 took 17 milliseconds
Curl request 43 took 212 milliseconds
Curl request 18 took 210 milliseconds
Curl request 19 took 216 milliseconds
Curl request 42 took 212 milliseconds
Curl request 53 took 219 milliseconds
Curl request 62 took 211 milliseconds
Curl request 47 took 216 milliseconds
Curl request 33 took 216 milliseconds
Curl request 44 took 216 milliseconds
Curl request 37 took 219 milliseconds
Curl request 52 took 214 milliseconds
Curl request 73 took 214 milliseconds
Curl request 55 took 216 milliseconds
Curl request 36 took 223 milliseconds
Curl request 61 took 221 milliseconds
Curl request 46 took 216 milliseconds
Curl request 40 took 214 milliseconds
Curl request 38 took 224 milliseconds
Curl request 71 took 216 milliseconds

```

((b)) Resultados Test de Concurrency
#2

```

Curl request 21 took 228 milliseconds
Curl request 29 took 229 milliseconds
Curl request 50 took 224 milliseconds
Curl request 59 took 218 milliseconds
Curl request 91 took 211 milliseconds
Curl request 74 took 215 milliseconds
Curl request 31 took 228 milliseconds
Curl request 72 took 223 milliseconds
Curl request 88 took 211 milliseconds
Curl request 64 took 209 milliseconds
Curl request 83 took 221 milliseconds
Curl request 75 took 218 milliseconds
Curl request 81 took 213 milliseconds
Curl request 51 took 235 milliseconds
Curl request 68 took 227 milliseconds
Curl request 76 took 216 milliseconds
Curl request 63 took 228 milliseconds
Curl request 45 took 227 milliseconds
Curl request 32 took 231 milliseconds
Curl request 56 took 226 milliseconds
Curl request 65 took 234 milliseconds
Curl request 66 took 216 milliseconds
Curl request 57 took 214 milliseconds
Curl request 100 took 216 milliseconds
Curl request 54 took 239 milliseconds
Curl request 67 took 221 milliseconds
Curl request 77 took 222 milliseconds
Curl request 90 took 211 milliseconds
Curl request 26 took 244 milliseconds
Curl request 48 took 231 milliseconds
Curl request 78 took 216 milliseconds
Curl request 35 took 214 milliseconds
Curl request 79 took 230 milliseconds
Curl request 28 took 220 milliseconds
Curl request 96 took 216 milliseconds
Curl request 95 took 209 milliseconds
Curl request 82 took 208 milliseconds
Curl request 60 took 241 milliseconds
Curl request 70 took 233 milliseconds
Curl request 89 took 223 milliseconds
Curl request 98 took 221 milliseconds
Performing stress test with 200 curls
Curl request 1 took 18 milliseconds
Curl request 7 took 18 milliseconds
Curl request 2 took 26 milliseconds
Curl request 3 took 33 milliseconds
Curl request 4 took 33 milliseconds
Curl request 11 took 30 milliseconds
Curl request 10 took 27 milliseconds
Curl request 8 took 47 milliseconds
Curl request 5 took 50 milliseconds
Curl request 41 took 51 milliseconds
Curl request 13 took 58 milliseconds
Curl request 12 took 64 milliseconds
Curl request 24 took 59 milliseconds
Curl request 6 took 70 milliseconds
Curl request 15 took 71 milliseconds
Curl request 9 took 72 milliseconds
Curl request 16 took 72 milliseconds
Curl request 14 took 87 milliseconds
Curl request 42 took 87 milliseconds
Curl request 52 took 92 milliseconds

```

((c)) Resultados Test de Concurrencia
#2

```

Curl request 34 took 93 milliseconds
Curl request 27 took 92 milliseconds
Curl request 43 took 94 milliseconds
Curl request 40 took 96 milliseconds
Curl request 26 took 90 milliseconds
Curl request 49 took 98 milliseconds
Curl request 23 took 101 milliseconds
Curl request 47 took 99 milliseconds
Curl request 28 took 97 milliseconds
Curl request 57 took 96 milliseconds
Curl request 56 took 93 milliseconds
Curl request 31 took 95 milliseconds
Curl request 79 took 83 milliseconds
Curl request 83 took 84 milliseconds
Curl request 73 took 88 milliseconds
Curl request 93 took 72 milliseconds
Curl request 48 took 88 milliseconds
Curl request 37 took 96 milliseconds
Curl request 77 took 91 milliseconds
Curl request 74 took 93 milliseconds
Curl request 179 took 53 milliseconds
Curl request 91 took 80 milliseconds
Curl request 95 took 59 milliseconds
Curl request 19 took 65 milliseconds
Curl request 38 took 51 milliseconds
Curl request 161 took 52 milliseconds
Curl request 147 took 59 milliseconds
Curl request 182 took 45 milliseconds
Curl request 164 took 35 milliseconds
Curl request 187 took 35 milliseconds
Curl request 143 took 25 milliseconds
Curl request 160 took 37 milliseconds
Curl request 180 took 50 milliseconds
Curl request 101 took 36 milliseconds
Curl request 97 took 14 milliseconds
Curl request 102 took 17 milliseconds
Curl request 173 took 20 milliseconds
Curl request 89 took 14 milliseconds
Curl request 131 took 66 milliseconds
Curl request 133 took 14 milliseconds
Curl request 118 took 45 milliseconds
Curl request 36 took 112 milliseconds
Curl request 140 took 14 milliseconds
Curl request 112 took 45 milliseconds
Curl request 87 took 88 milliseconds
Curl request 115 took 14 milliseconds
Curl request 175 took 35 milliseconds
Curl request 46 took 216 milliseconds
Curl request 18 took 212 milliseconds
Curl request 51 took 220 milliseconds
Curl request 30 took 220 milliseconds
Curl request 35 took 225 milliseconds
Curl request 61 took 223 milliseconds
Curl request 39 took 226 milliseconds
Curl request 32 took 227 milliseconds
Curl request 45 took 229 milliseconds
Curl request 62 took 219 milliseconds
Curl request 44 took 223 milliseconds
Curl request 81 took 216 milliseconds
Curl request 71 took 215 milliseconds
Curl request 154 took 208 milliseconds
Curl request 76 took 222 milliseconds

```

((d)) Resultados Test de Concurrencia
#2

```

Curl request 25 took 238 milliseconds
Curl request 88 took 227 milliseconds
Curl request 72 took 224 milliseconds
Curl request 90 took 222 milliseconds
Curl request 84 took 221 milliseconds
Curl request 59 took 224 milliseconds
Curl request 58 took 239 milliseconds
Curl request 63 took 229 milliseconds
Curl request 53 took 227 milliseconds
Curl request 105 took 229 milliseconds
Curl request 125 took 219 milliseconds
Curl request 181 took 209 milliseconds
Curl request 78 took 228 milliseconds
Curl request 50 took 249 milliseconds
Curl request 21 took 233 milliseconds
Curl request 135 took 217 milliseconds
Curl request 94 took 235 milliseconds
Curl request 55 took 242 milliseconds
Curl request 82 took 240 milliseconds
Curl request 65 took 242 milliseconds
Curl request 168 took 213 milliseconds
Curl request 134 took 222 milliseconds
Curl request 75 took 229 milliseconds
Curl request 68 took 238 milliseconds
Curl request 150 took 227 milliseconds
Curl request 85 took 238 milliseconds
Curl request 113 took 214 milliseconds
Curl request 92 took 237 milliseconds
Curl request 196 took 213 milliseconds
Curl request 66 took 242 milliseconds
Curl request 142 took 217 milliseconds
Curl request 80 took 239 milliseconds
Curl request 122 took 227 milliseconds
Curl request 128 took 225 milliseconds
Curl request 153 took 227 milliseconds
Curl request 165 took 226 milliseconds
Curl request 64 took 240 milliseconds
Curl request 139 took 224 milliseconds
Curl request 86 took 228 milliseconds
Curl request 200 took 214 milliseconds
Curl request 146 took 228 milliseconds
Curl request 149 took 227 milliseconds
Curl request 124 took 231 milliseconds
Curl request 110 took 221 milliseconds
Curl request 70 took 246 milliseconds
Curl request 166 took 223 milliseconds
Curl request 109 took 228 milliseconds
Curl request 138 took 238 milliseconds
Curl request 100 took 231 milliseconds
Curl request 137 took 217 milliseconds
Curl request 104 took 236 milliseconds
Curl request 119 took 234 milliseconds
Curl request 99 took 219 milliseconds
Curl request 22 took 247 milliseconds
Curl request 156 took 220 milliseconds
Curl request 96 took 227 milliseconds
Curl request 157 took 218 milliseconds
Curl request 148 took 223 milliseconds
Curl request 120 took 234 milliseconds
Curl request 98 took 223 milliseconds
Curl request 126 took 228 milliseconds
Curl request 114 took 237 milliseconds

```

((e)) Resultados Test de Concurrency
#2

```

Curl request 130 took 230 milliseconds
Curl request 123 took 217 milliseconds
Curl request 144 took 237 milliseconds
Curl request 67 took 263 milliseconds
Curl request 33 took 280 milliseconds
Curl request 152 took 231 milliseconds
Curl request 116 took 223 milliseconds
Curl request 162 took 238 milliseconds
Curl request 167 took 224 milliseconds
Curl request 159 took 228 milliseconds
Curl request 127 took 222 milliseconds
Curl request 20 took 249 milliseconds
Curl request 117 took 235 milliseconds
Curl request 145 took 241 milliseconds
Curl request 176 took 228 milliseconds
Curl request 103 took 218 milliseconds
Curl request 170 took 227 milliseconds
Curl request 185 took 218 milliseconds
Curl request 60 took 283 milliseconds
Curl request 192 took 239 milliseconds
Curl request 197 took 224 milliseconds
Curl request 189 took 219 milliseconds
Curl request 132 took 231 milliseconds
Curl request 184 took 228 milliseconds
Curl request 136 took 240 milliseconds
Curl request 172 took 225 milliseconds
Curl request 111 took 225 milliseconds
Curl request 186 took 209 milliseconds
Curl request 183 took 229 milliseconds
Curl request 177 took 238 milliseconds
Curl request 190 took 217 milliseconds
Curl request 163 took 223 milliseconds
Curl request 178 took 216 milliseconds
Curl request 169 took 210 milliseconds
Curl request 198 took 229 milliseconds
Curl request 141 took 235 milliseconds
Curl request 129 took 238 milliseconds
Curl request 171 took 218 milliseconds
Curl request 174 took 230 milliseconds
Curl request 106 took 235 milliseconds
Curl request 151 took 227 milliseconds
Curl request 191 took 222 milliseconds
Curl request 155 took 218 milliseconds
Curl request 121 took 276 milliseconds
Curl request 108 took 232 milliseconds
Curl request 158 took 266 milliseconds
Curl request 107 took 228 milliseconds
Curl request 199 took 212 milliseconds
Curl request 17 took 305 milliseconds
Curl request 69 took 239 milliseconds
Curl request 195 took 228 milliseconds
Curl request 194 took 219 milliseconds
Curl request 188 took 211 milliseconds
Curl request 54 took 243 milliseconds
Curl request 193 took 224 milliseconds
Curl request 29 took 281 milliseconds

```

((f)) Resultados Test de Concurrency
#2

```

1  #!/bin/bash
2
3  # Number of users to connect
4  num_users=500
5
6  # Server details
7  server_host="localhost"
8  server_port=1110
9
10 # Loop to connect users
11 for ((i = 1; i <= num_users; i++)); do
12     # Attempt connection using netcat
13     nc -C $server_host $server_port &
14     echo "User $i connected."
15     # Sleep for a short duration to avoid overwhelming the system
16     sleep 0.1
17 done
18
19 # Wait for all background processes to finish
20 wait
21
22 echo "All users connected."

```

Figure 7: Test de 500 conexiones (interconnection_script.sh)

```

● cijjas@LAPTOP-RUP13L5R:~/tprotos/PDC-POP3D$ time curl pop3://testuser:testuser@localhost:1234/4 > /dev/null
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %         %         Dload  Upload   Total   Spent    Left   Speed
100 50.6M    0 50.6M    0     0  16.5M      0  --:--:--  0:00:03 --:--:-- 16.5M

real    0m3.076s
user    0m0.252s
sys     0m0.030s
○ cijjas@LAPTOP-RUP13L5R:~/tprotos/PDC-POP3D$ █

```

Figure 8: Test de velocidad