

Protocolos de Comunicacion:

POP3

MARCO SCILIPOTI
mscilipoti@itba.edu.ar

LAUTARO HERNANDO
lhernando@itba.edu.ar

MARTIN IPPOLITO
mippolito@itba.edu.ar

23 de junio de 2023

ÍNDICE		VII Documento de diseño del monitor	
I. Introducción	1	I. RFC - Monitor POP3 Server . . .	6
II. Dificultades Encontradas y Decisiones de diseño	2	II. Resumen	6
I. Selector y stm	2	III. Estado de este Memo	7
II. Manejo de argumentos (args.c) .	2	IV. Descripción general del protocolo	7
III. Parser	2	V. Operación Básica	7
III. Diseño del proyecto - Arquitectura de la aplicacion	3	VI. Comandos	7
IV. Ejemplos de prueba	3	VI.1. Estado de AUTHORIZA- TION	7
I. Determinacion del buffer de salida	3	VI.2. Estado de TRANSACTION	8
I.1. Buffer de 2K	3	VII. Consideraciones de seguridad . .	10
I.2. Buffer de 4K	3	VIII. Consideraciones de implementa- ción	10
I.3. Buffer de 8K	4	IX. Consideraciones de extensibilidad	10
I.4. Buffer de 16K	4	VIII Posibles extensiones	11
I.5. Buffer de 32K	4	IX. Conclusiones	11
II. Testeo de concurrencia	4	X. Anexo A: Guía de instalación	11
II.1. Conclusiones	4	XI. Anexo B: Instrucciones para la configu- ración del servidor y cliente	11
III. Testeo de integridad	4	XII Anexo C: Ejemplos de monitoreo	12
III.1. Testeo de byte stuffing . .	4		
III.2. Testeo email 1GB	5		
IV. Testeo de concurrencia	5		
V. Testeo de concurrencia con lec- tura de email	5		
V. Limitaciones	6		
VI. Descripción aplicacion cliente	6		
I. Argumentos	6		

I. INTRODUCCIÓN

EN este informe se presentara una implementación de un servidor POP3 concurrente no bloqueante, asi como un protocolo de monitoreo planteado para el mismo, y un cliente que trabaja con este ultimo para poder monitorear el servidor. También se detallaran las diferentes decisiones de diseño asi como las dificultades encontradas, asi como los

diferentes tests a los cuales se sometio (y paso) el servidor.

II. DIFICULTADES ENCONTRADAS Y DECISIONES DE DISEÑO

I. Selector y stm

Dentro del trabajo tuvimos bastantes dificultades, gran parte de las mismas estan relacionadas tanto con la stm y como con el selector. En el selector tuvimos muchas dificultades al entender como debería ser la orquestación de escritura y lectura. Este mismo problema fue solucionado en clase, pero luego de entender dicha orquestación fue directa la implementación. Tambien dentro del selector tuvimos el problema de como liberar la memoria de cada fd, tanto en el momento de hacer un *quit* como en el momento en el que el usuario hago un *ctrl c*, este inconveniente nos tomo bastante tiempo ya que tomamos como prioridad no tener ningun memory leak y es por eso que luego de testear tanto con *-fsanitize=address* como con *valgrind* nos dio que no tenemos ningun memory leak.

En la state machine inicialmente el problema fue la transición entre los estados de la conexión. Luego de haberse podido transicionar rápidamente surgió la idea de poder agrupar el comportamiento de los comandos en sus respectivos archivos. De esta manera logramos encapsular la lógica específica de cada comando en su respectivo archivo. Inicialmente planteamos esto para la etapa de lectura a través el comando *read_commands* que interactuaba directamente con el parser y era el responsable de ejecutar el handler para el comando parseado. Luego de implementar la lógica de escritura, se considero oportuno hacer uso de la estructura existente de archivos para también hacer que cada comando administre la lógica de escritura en su respectivo archivo.

Por ultimo, la prueba de fuego de este diseño fue adaptarlo para el protocolo monitor. La verdad que hubo bastantes *bugs* al momento de repetir la estructura debido a la falta de tipos genéricos de C. Sin embargo se logro exitosamente implementar la misma estructura de

lectura/escritura de comandos para el monitor.

II. Manejo de argumentos (args.c)

Como parte del código otorgado por la cátedra, se utilizo *args.c*. Comprender el código fue un desafío leve, y termino siendo de mucha utilidad. Sin embargo, hubo que resolver un pequeño error, en el cual se asignaba para obtener el código del flag de argumento (*-h* código 'h' o *-help* código 0xD001) se almacenaba en una variable *char*, y en el caso del código 0xD001, se intentaba almacenar un numero de 2 bytes de longitud en un *char* (con tamaño de 1 byte).

III. Parser

El diseño del parser fue adaptar la librería *parser.c* proporcionada por la cátedra para parsear lo ingresado por el usuario. Debido a que el protocolo *pop3* tiene como máximo dos argumentos separados por un espacio por cada comando se diseño el parser tal que permita reconocer de manera genérica un comando y dos argumentos. Luego de leer el segundo argumento se setea un flag en la respuesta del *parser_feed*. Luego la función *read_commands* detecta este flag y llama a la función *process_commands* que compara el comando ingresado con el arreglo de comandos disponibles para ese estado. Si encuentra tal comando entonces ejecuta el handler correspondiente (a través de los punteros a función almacenados) con los respectivos argumentos y el estado de la conexión.

III. DISEÑO DEL PROYECTO - ARQUITECTURA DE LA APLICACION

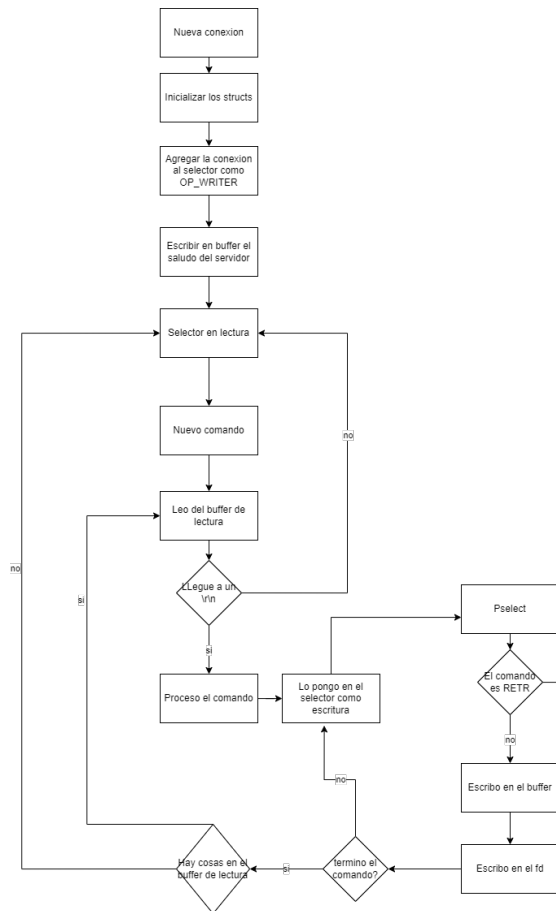


Figura 1: Diagrama de flujo de una nueva conexión

El diagrama demuestra el flujo desde que se establece una nueva conexión y como, mediante el selector, se va iterando a medida del estado de la conexión.

En el diagrama se simplifica la operación de RETR de un email del usuario. En este caso, se abre dicho archivo que contiene al correo y se lo añade al selector. La conexión en este momento pasa a estar en NOOP state.

IV. EJEMPLOS DE PRUEBA

i. Determinación del buffer de salida

Para la determinación del buffer de salida, se realizó un testeo de carga y según el tiempo que demoraba la respuesta del servidor se determinó el mejor tamaño de buffer. El testeo se basa en que un usuario tenga dentro de su casilla un email de 1GB, mediante el comando *curl* hacemos un *retr* de dicho email. Cuando el email se terminó de procesar se tomó el tiempo que nos arrojó el comando y lo comparamos con los diferentes con los tiempos de los diferentes tamaños de buffer. También fue analizado que porcentaje del CPU era utilizado por el servidor para asegurar que en todos los casos el mismo este utilizando el 100 %.

i.1. Buffer de 2K

Para el primer caso de testeo se probó con un buffer de 2k, debemos tener en cuenta que el servidor fue compilado mediante la optimización *-O3* y sin el flag *fsanitize* para poder obtener la mejor performance del mismo. Cabe aclarar que durante el desarrollo si se utilizó el flag *fsanitize* y también se testeo con *valgrind* para no tener ningún memory leak. Como podemos ver en la figura 2 el tiempo que le toma al servidor en procesar un email de 1GB con un buffer de escritura y de lectura de 2048Bytes, en este caso le toma un total de 13 segundos.

```

laucha@laucha:~$ curl pop3://laucha:laucha@localhost:8114/1 > output
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
   100    1319M   0 1319M    0     0   100M   0 --:--:--  0:00:13 --:--:-- 103M
  
```

Figura 2: Tiempo en procesar con buffer 2K

i.2. Buffer de 4K

Para el segundo testeo se decidió duplicar el buffer tanto de lectura como de escritura para poder compararlos con los otros casos. En este caso el tiempo que se presenta es de 11 segundos, siendo así 2 segundos más rápido

que con el buffer de 2K, analizandolo nos parecio una gran mejorio ya que mejoro casi un 15 % el rendimiento del mismo. Tambien fue testead el uso del CPU y en el momento en que se esta procesando el comando retr el uso es del 100 % pero cuando se termina de procesar, el mismo baja abruptamente. Esto nos demuestra de que cuando se termino de procesar el mail, se liberan todos los recursos y no se sigue procesando basura.

```
laucha@laucha:~$ curl pop3://laucha:laucha@localhost:8114/1 > output
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
   Dload  Upload   Total   Spent    Left     Speed
100 1319M    0 1319M    0     0   116M    0:00:11 0:00:11 0:00:00 115M
```

Figura 3: Tiempo en procesar con buffer 4K

i.3. Buffer de 8K

En el tercer testeo decidimos aumentar aun mas los buffers para ver como se comportaba el servidor a dicho cambio, nos dimos cuenta que el rendimiento del mismo seguia mejorando ya que se redujeron los tiempos en 1s que es el 10 % del rendimineto actual.

```
laucha@laucha:~$ curl pop3://laucha:laucha@localhost:8114/1 > output
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
   Dload  Upload   Total   Spent    Left     Speed
100 1319M    0 1319M    0     0   126M    0:00:10 0:00:10 0:00:00 127M
```

Figura 4: Tiempo en procesar con buffer 8K

i.4. Buffer de 16K

Se decidio duplicar el buffer a 16K en este caso resulto algo muy interesante que es que no aumento la perfomance del mismo si no que quedo en el mismo tiempo que con un buffer de 8K. Esto mismo no lleva a pensar que un buffer de mayor tamaño no quiere decir que el servidor va a tener una mayor performance.

```
laucha@laucha:~$ curl pop3://laucha:laucha@localhost:8114/1 > output
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
   Dload  Upload   Total   Spent    Left     Speed
100 1319M    0 1319M    0     0   126M    0:00:10 0:00:10 0:00:00 120M
```

Figura 5: Tiempo en procesar con buffer 16K

i.5. Buffer de 32K

Por ultimo se ha realizado el mismo test con un buffer de 32K. Como podemos ver en la

figura 6 el tiempo a sido el mismo que cuando corrimos el test con un buffer de 8K, esto mismo nos afianza mas la hipotesis que la mejora del rendimiento por parte del tamaño del buffer tiene un techo el cual lo alcanzamos.

```
laucha@laucha:~$ curl pop3://marco:marco@localhost:8114/1 > output
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
   Dload  Upload   Total   Spent    Left     Speed
100 1319M    0 1319M    0     0   126M    0:00:10 0:00:10 0:00:00 112M
```

Figura 6: Tiempo en procesar con buffer 32K

II. Testeo de concurrencia

ii.1. Conclusiones

Gracias a estos testeos pudimos concluir que la mejora del rendimiento del servidor no solo viene por parte del tamaño del buffer ya que, en este caso, a partir de un buffer con tamaño de 8K el rendimiento fue exactamente el mismo. Es por eso que llegamos a la conclusion de que el tamaño final del buffer es de 8K.

III. Testeo de integridad

Dentro del testeo de integridad se han tomado 2 casos, el primer caso se enfoca en el testeo del correcto funcionamiento del byte stuffing y en el segundo testeo es un testeo de integridad con un mail de gran tamaño.

iii.1. Testeo de byte stuffing

Para este testeo lo que se realizo fue crear un archivo el cual al pasar el mismo le aplicara byte stuffing. El mismo se a creado como se muestra en la figura 7, es por eso que el mismo servidor le aplicara el byte stuffing. Cuando utilizando el comando *diff* entre el email original y la respuesta del servidor, en este caso obtuvimos que hay una diferencia entre los contenidos de los mails. Dicha diferencia se encuentra al final de email y es que el output file tiene un */n* mas. Este caso se ha hablado en clase, y se llego a la conclusión que dicho */n* fue agregado por el comando curl. Es por eso que llegamos a la conclusion que el email no fue modificado por el servidor POP3.

Tambien se puede ver en la figura 9 que los archivos son identicos menos que el output tiene el `/n` anteriormente nombrado.

```
laucha@laucha:~/testprotos/marco/curl$ printf "\r\n holaaaa \r\n \r\n \r\n hola" > mail2
```

Figura 7: Creacion del mail

```
laucha@laucha:~/testprotos/marco/curl$ diff -u mail2 output
--- mail2      2023-06-19 12:37:12.729317079 -0300
+++ output     2023-06-19 12:38:39.114306054 -0300
@@ -1,4 +1,4 @@
 .
 holaaaa .
 ..
- hola
+ hola
 \ No newline at end of file
```

Figura 8: Diff entre la respuesta y el mail

```
laucha@laucha:~/testprotos/marco/curl$ cat output
.
holaaaa .
..
hola
laucha@laucha:~/testprotos/marco/curl$ cat mail2
.
holaaaa .
..
hola
```

Figura 9: Cat entre mail y output

iii.2. Testeo email 1GB

Para dicho testeo lo que se realizo fue crear un archivo de 1G y pasarlo a base64. Luego lo que se hizo fue hacer un `retr` de dicho email y hacer un `diff` entre ambos emails.

```
laucha@laucha:~$ diff -u output /home/laucha/testprotos/marco/curl/mail1
--- output      2023-06-18 20:06:26.636077160 -0300
+++ /home/laucha/testprotos/marco/curl/mail1 2023-06-18 19:54:28.392757021 -0300
@@ -17960911,4 +17960911,3 @@
HQUoVYfaHqJULiXUjUHI13uGwBaHXJmbvNoZTVgw+xA+mE1sn1hQuC12W1MN33tw+S+P0ma4H
HX3ASYM8SyhLQTEHZ2CnLAjnzA6nK47ohG2z6QeWTW88nu/kk/3MmAR30/FoPqDnpMbnjjQ6Bi
tvd2P8fZf778UNM28vXGkw==
```

Figura 10: Diff entre 2 emails 1GB

Como podemos ver en la figura 10 la diferencia entre ambos emails es únicamente en la ultima linea, como vimos en el caso anterior. Es por esto que podemos decir que no hay diferencia entre el email original y el que devolvió el servidor POP3.

iv. Testeo de concurrencia

Mediante un script de bash (el cual esta en el repositorio versionado dentro del directorio test), se crean 500 conexiones de 500 usuarios diferentes del servidor POP3. Lo que se busca es validar que las conexiones concurrentes sean manejadas correctamente. Adicionalmente, luego de la autenticacion del usuario, se realiza un LIST.

Con esto queda se valida el correcto manejo de concurrencias, de 500 usuarios. Recordando que el servidor tendra un manejo de hasta 1024 file descriptors, habria margen por el manejo de file descriptors por parte del servidor, pero al tener un maximo de 500 usuarios, no podria haber alguno mas conectado (ya que no se pueden tener dos conexiones con un mismo usuario).

Para corroborar que se han conectado los 500 usuarios, configuramos el logger para que muestre los mensajes [DEBUG]. Luego transferimos la salida estandar a un archivo y se cuentan la cantidad de lineas con mensaje '[DEBUG] yyyy-mm-dd hh:mm:ss ADD_USER' (donde el monitor añade a cada usuario) y '[DEBUG] yyyy-mm-dd hh:mm:ss Registered a new connection' donde se registra una nueva conexion.

Tambien podria realizarse un wordcount sobre dicho documento.

```
$ wc log_file
```

v. Testeo de concurrencia con lectura de email

Extendiendo el testeo anterior, se busca poner a prueba la lectura de un email para cada una de las 500 conexiones concurrentes.

Para esto se creo otro script de bash (tambien presente en el directorio test) donde se crean las carpetas necesarias para la lectura de los emails de cada usuario. Analogamente al test anterior, se realizan las 500 conexiones y por cada una, se realiza RETR 1. Todos los usuarios tienen un mismo de email de aproximadamente 1 MB.

Para corroborar, ademas de utilizar el logger como en el testeo anterior. Se debe corroborar

que la salida en el script del testeo se corresponda con el email presente en el directorio de cada usuario.

V. LIMITACIONES

En la realizacion del servidor POP3 se tuvieron que tomar ciertas medidas las cuales recaen en limitaciones que tiene dicho servidor. Una de ellas es que como maximo se podran tener 500 usuarios en simultaneo ya que se utiliza el *select()*.

Dentro de las limitaciones tambien tenemos con el largo del nombre de un directorio, el cual decidimos tomar el maximo que tiene linux que es de 4096. Es por esto que nombre de directorios con mas de 4096 no seran aceptados.

Otra limitante que tuvimos en la longitud de los nombres de usuario, pero esta misma no es tal limitante ya que dentro del RFC de POP3, nos indica que el maximo largo que puede tener un nombre de usuario es de 40 caracteres y ese mismo es nuestro tope.

Por ultimo, tenemos como limitante la cantidad de emails que puede tener un usuario en su carpeta curl. Durante el desarrollo se tomo la desicion de que la cantidad maxima de emails que pueda tener una persona sea de 500 emails, en el caso de que el usuario tenga mas de 500 emails el servidor POP3 va a tomar en su procesamiento unicamente los primeros 500 emails.

VI. DESCRIPCION APLICACION CLIENTE

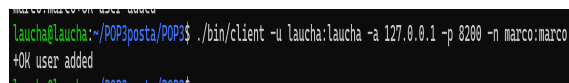
La aplicacion cliente fue creada para tener una comunicacion simple con el servidor monitor. Es por eso que las funcionalidades que tiene el mismo esta limitadas con lo que el protocolo monitor ofrece. Hay que tener en cuenta tambien que el protocolo realizado es un protocolo TCP por lo que se propuso generar una autentificacion *USER* y *PASS*.

I. Argumentos

La aplicacion monitor recibira por argumentos todo lo necesario para su ejecucion. Es por eso que tiene argumentos obligatorios como argumentos no obligatorios.

Dentro de los argumentos obligatorios estan *-p* el cual recibe el puerto en el que esta corriendo el servidor monitoreo, tambien se especifica mediante el argumento *-a* el address en donde esta corriendo el servidor monitor. Para la autentificacion se requiere utilizar el argumento *-u user:password*.

Dentro de los argumentos no obligatorios hay dos, el argumento *-n user:password* el cual agregara un nuevo usuario al servidor POP3 y el argumento *-m* el cual imprimira todas las metricas que tiene el servidor POP3. Por ultimo argumento no obligatorio es *-d directoryPath* el cual recibe un directorio el cual es el nuevo directorio principal del servidor POP3.

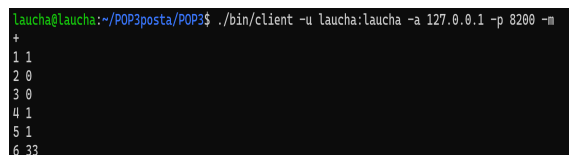


```

laucha@laucha:~/POP3posta/POP3$ ./bin/client -u laucha:laucha -a 127.0.0.1 -p 8200 -n marco:marco
+OK user added

```

Figura 11: Ejemplo aplicacion cliente usando *-n*



```

laucha@laucha:~/POP3posta/POP3$ ./bin/client -u laucha:laucha -a 127.0.0.1 -p 8200 -m
+
1 1
2 0
3 0
4 1
5 1
6 33

```

Figura 12: Ejemplo aplicacion cliente usando *-m*

VII. DOCUMENTO DE DISEÑO DEL MONITOR

Al modelar el protocolo que utilizara el monitor, se escribió un documento similar a un RFC

I. RFC - Monitor POP3 Server

II. Resumen

Este documento propone un protocolo para monitorear un servidor POP3. El protocolo define dos estados: AUTHORIZATION

y TRANSACTION. El estado de AUTHORIZATION tiene dos comandos, USERNAME y PASSWORD, mientras que el estado de TRANSACTION tiene tres comandos, METRICS, ADD_USER y EXIT.

III. Estado de este Memo

Este es un RFC informativo.

IV. Descripción general del protocolo

El protocolo para monitorear un servidor POP3 consta de dos estados: AUTHORIZATION y TRANSACTION. El estado inicial es AUTHORIZATION, y el cliente debe autenticarse enviando los comandos USERNAME y PASSWORD. Una vez autenticado, el cliente entra en el estado de TRANSACTION.

En el estado de TRANSACTION, el cliente puede enviar tres comandos: METRICS, ADD_USER y EXIT. El comando METRICS devolverá métricas sobre el uso del servidor. El comando ADD_USER agregará un nuevo usuario al servidor POP3. El comando EXIT finalizará la sesión.

V. Operación Básica

Inicialmente, el equipo servidor al iniciar un servicio POP3, habilitará conexiones en otro puerto TCP a definir por la implementación, para aquellos usuarios administradores que deseen utilizar el siguiente protocolo para administrar el servidor POP3.

De ahora en adelante, se denotará servidor al proceso de monitoreo del servidor POP3, evitándose confundir servidor con este último.

Cuando la conexión se ha establecido, el servidor PUEDE enviar un saludo. Luego quedará esperando al cliente y estos dos intercambiarán órdenes y respuestas (respectivamente) hasta el cierre o interrupción de la conexión.

Las órdenes en el protocolo monitor POP3 consisten en una serie de palabras claves sin diferenciar mayúsculas de minúsculas, posiblemente seguidas de uno o ningún argumento.

Todas las órdenes y sus argumentos están compuestos de caracteres ASCII y terminarán con un par CRLF (Carriage Return Line Feed). Cada argumento puede tener hasta 40 caracteres.

Las respuestas del monitor están formadas por un indicador de estado, posiblemente seguida de información adicional. Todas las respuestas están terminadas por un par CRLF. Los indicadores de estado son el positivo ("+") y el negativo ("-"). Los servidores PUEDEN enviar información complementaria a los indicadores de estado, dicha información no debe exceder los 20 caracteres.

Existen respuestas a ciertas órdenes que son multilínea. En esos casos, tras enviar la primera línea de la respuesta y un LF (Line Feed), irán más líneas terminadas por un LF. Cuando se han enviado todas las líneas de la respuesta, se envía una línea final que consiste en un par CRLF.

Un servidor DEBE responder a una orden no reconocida, no implementada o no válida sintácticamente, con un indicador de estado negativo. El servidor DEBE responder a una orden enviada en una fase incorrecta con un indicador de estado negativo.

Un servidor PUEDE tener un contador de inactividad para cerrar la conexión. Ese contador DEBE ser de al menos 10 minutos de duración. La recepción de cualquier orden durante el intervalo debería bastar para reiniciar el contador. Cuando el contador se agota, la sesión TCP deberá cerrarse.

VI. Comandos

vi.1. Estado de AUTHORIZATION

USERNAME El comando USUARIO se utiliza para identificar al usuario en el servidor. El cliente envía el nombre de usuario como argumento del comando.

C: USERNAME {usuario}

S: +

Argumentos: Una cadena que identifica un usuario monitor del servidor POP3. Este usuario no tiene relación con algún usuario POP3 del servidor.

Restricciones: Solo puede darse en la fase de AUTHORIZATION después de establecerse la conexión con el servidor monitor o tras unas órdenes USERNAME o PASSWORD fallidas.

Comentario: Para autenticar utilizando la combinación de órdenes USERNAME y PASSWORD, el cliente primero debe enviar la orden USERNAME con un indicador de estado positivo (+). Entonces el cliente puede enviar tanto la orden PASSWORD para completar la autenticación. Si el servidor POP3 responde con un indicador de estado negativo (-) a la orden USERNAME, entonces el cliente puede enviar una nueva orden de autenticación.

El servidor monitor debe devolver una respuesta positiva solo en caso de que exista el usuario monitor en el servidor.

Respuestas posibles: +, -
Ejemplos:

```
C: USERNAME admin
S: +
...
C: USERNAME notadmin
S: -
```

PASSWORD El comando PASSWORD se utiliza para autenticar al usuario en el servidor. El cliente envía la contraseña como argumento del comando.

```
C: PASSWORD {contraseña}
S: +
```

Argumentos: Una clave específica que da acceso a cierto usuario.

Restricciones: Solo puede darse en la fase AUTHORIZATION, después de una orden USERNAME con éxito.

Comentario: Cuando el cliente envía la orden PASSWORD, el servidor monitor utiliza el par de argumentos de las órdenes USER y PASS para determinar si al cliente debería proporcionársele el acceso al servidor monitor. El servidor monitor puede tratar los espacios en el argumento como parte de la contraseña en lugar de como separador de argumentos.

Respuestas posibles: +, -, -
Ejemplos:

```
C: USERNAME admin
S: +
C: PASSWORD secret
S: -
...
C: USERNAME admin
S: +
C: PASSWORD secret
S: +
```

vi.2. Estado de TRANSACTION

METRICS El comando METRICS se utiliza para recuperar métricas sobre el uso del servidor.

```
C: METRICS (n)
S: +
S: {métricas}
```

Argumentos: Puede recibir OPCIONALMENTE el número de la métrica que se quiere pedir. En el caso que no se ingrese el número de métrica se imprimirán todas.

Restricciones: Solo puede darse en la fase TRANSACTION.

Comentario: El servidor monitor enviará una respuesta multilínea positiva, incluyendo un listado de las métricas del servidor POP3 monitoreado.

Los valores de las métricas están codificados con un número entero positivo de la siguiente manera:

- 1: Cantidad de conexiones actuales
- 2: Cantidad de mensajes obtenidos
- 3: Cantidad de mensajes eliminados
- 4: Cantidad de conexiones concurrentes
- 5: Cantidad de conexiones históricas
- 6: Cantidad de bytes transferidos

Respuestas posibles:

- +
- 1 n
- 2 n
- 3 n
- 4 n
- 5 n
- 6 n

- ...
- +
- n
- ...
- -

Donde n representa el valor de una métrica en cada caso. El primer caso es una respuesta multilínea, donde se muestran todas las métricas. Y el segundo, una respuesta también multilínea, donde se muestra el valor para la métrica solicitada.

Ejemplo:

C: METRICS

S: +
S: 1 6
S: 2 120
S: 3 60
S: 4 2
S: 5 8
S: 6 54030
...
C: METRICS 2
S: 15

CHANGE_DIRECTORY El comando **CHANGE_DIRECTORY** modificara el repositorio donde el servidor POP3 buscara los correos de cada usuario.

C: CHANGE_DIRECTORY <path>
S: +

Argumentos: Path al directorio que contiene los correos de los usuarios

Restricciones: Solo se puede utilizar en el estado TRANSACTION

Comentario: El monitor retornara un mensaje con estado postivo en el caso de poder modificar dicho directorio en el servidor POP3. El servidor PUEDE retornar un estado de error si el servidor POP3 no pudo utilizar el path otorgado.

Respuestas posibles:

+
-

Ejemplos:

C: CHANGE_DIRECTORY /root/emailDirectory

S: +

CAPA El comando **CAPA** retorna una lista de las capacidades soportadas por el servidor monitor. Estará solo disponible en el estado TRANSACTION.

C: CAPA
S: +

El servidor PUEDE listar comandos adicionales en caso de que los haya implementado. Adicionalmente, puede listar información adicional sobre el servidor.

El servidor PUEDE retornar un mensaje con estado positivo sin ninguna información adicional, solamente en el caso en el que no desee listar información adicional sobre el servidor y no haya implementado ningún comando por fuera del protocolo.

Respuestas posibles: +

Ejemplos:

C: CAPA
S: +
...
C: CAPA
S: +
S: ADD_USER
S: LIST_USERS
S: ...

LIST_USERS El comando **LIST_USERS** se utiliza para mostrar un listado de todos los usuarios del servidor POP3.

C: LIST_USERS
S: +
S: <usuario1>
S: <usuario2>
...

Argumentos: Ninguno

Restricciones: Solo se puede utilizar en el estado TRANSACTION

Comentario: El monitor retornará un listado de los usuarios creados en el servidor POP3.

Respuestas posibles: +

Ejemplo:

C: LIST_USERS

S: +

S: <usuario1>

S: <usuario2>

S: <usuario3>

S: ...

ADD_USER El comando ADD_USER se utiliza para agregar un nuevo usuario al servidor POP3.

C: ADD_USER {usuario}:{contraseña}

S: +

Argumentos:

- *usuario*: El nombre del usuario que se desea agregar.
- *contraseña*: La contraseña del usuario.

Restricciones: Solo se puede utilizar en el estado TRANSACTION

Comentario: El servidor monitor agregará un nuevo usuario con el nombre de usuario y contraseña proporcionados al servidor POP3.

Respuestas posibles: +, -

Ejemplo:

C: ADD_USER newuser:password

S: +

...

C: ADD_USER existinguser:password

S: -

EXIT El comando EXIT se utiliza para finalizar la sesión con el servidor monitor.

C: EXIT

S: +

Argumentos: Ninguno

Restricciones: Solo se puede utilizar en el estado TRANSACTION

Comentario: Al recibir el comando EXIT, el servidor monitor finalizará la conexión.

Respuestas posibles: +

Ejemplo:

C: EXIT

S: +

VII. Consideraciones de seguridad

El protocolo monitor POP3 no establece mecanismos de seguridad adicionales a los ya presentes en el protocolo POP3. La autenticación del cliente al servidor monitor se realiza utilizando las órdenes USERNAME y PASSWORD, que deben estar protegidas adecuadamente para evitar el acceso no autorizado al servidor monitor.

Se recomienda implementar medidas de seguridad adicionales, como la encriptación de las comunicaciones, para garantizar la confidencialidad y la integridad de los datos transmitidos entre el cliente y el servidor monitor.

VIII. Consideraciones de implementación

El servidor monitor debe ser capaz de gestionar múltiples conexiones de clientes y debe implementar las funcionalidades definidas en el protocolo.

La comunicación entre el cliente y el servidor monitor se realiza a través de conexiones TCP. El servidor monitor debe estar configurado para escuchar en un puerto específico y gestionar las solicitudes de conexión entrantes de los clientes.

IX. Consideraciones de extensibilidad

El protocolo monitor POP3 es extensible y puede adaptarse para incluir nuevas funcionalidades y comandos en futuras versiones. Sobre todo en caso de implementar nuevas métricas.

Al agregar nuevos comandos o métricas, es importante seguir las convenciones establecidas en el protocolo, como el formato de los mensajes de comando y respuesta, para garantizar la interoperabilidad entre diferentes implementaciones.

Es recomendable documentar claramente cualquier extensión realizada al protocolo y proporcionar la información necesaria para que los clientes puedan utilizar las nuevas funcionalidades de manera adecuada.

VIII. POSIBLES EXTENSIONES

En términos de la implementación del servidor POP3 se podrían agregar mas comandos que actualmente no se soportan, como los mecanismos de spam para los mails.

Por otro lado, al protocolo de monitorización se le podría agregar la capacidad de realizar análisis en tiempo real de las métricas del servidor, tales como la identificación de patrones de uso o la detección de anomalías en el comportamiento del servidor. Posiblemente extendiendo también el cliente del monitor para poder graficar las métricas del servidor. Por ultimo se podrían ampliar los comandos que permitan la configuración remota del servidor, lo que proporcionaría a los administradores un mayor control sobre la operación del servidor POP3. Una de las principales posibles extensiones a futuro es la adición del comando `delete_user`, el cual pueda eliminar usuarios del servidor POP3. Análogamente para los usuarios del monitor, podría agregarse la posibilidad de crear y eliminar usuarios administradores, con acceso al monitor. Actualmente, los usuarios del monitor, se definen en ejecución.

IX. CONCLUSIONES

La implementación del protocolo POP3 y su servidor monitor ha permitido profundizar en el funcionamiento de los protocolos de internet y en el diseño e implementación de sistemas de servidor-cliente. A través de este proyecto, hemos podido aplicar y desarrollar habilidades en la programación concurrente, el uso de máquinas de estado y el manejo de sockets.

Además, la introducción del diseño de un protocolo de monitoreo (y su respectivo cliente) ha proporcionado un mecanismo para entender aun mejor las consideraciones al momento

de crear un protocolo y su respectivo RFC.

En resumen, este proyecto ha sido una valiosa oportunidad para aprender sobre el diseño e implementación de protocolos a través la implementación de sistemas de servidor-cliente y el diseño de nuestro propio protocolo.

X. ANEXO A: GUÍA DE INSTALACIÓN

- **Generación de binarios:** Situado en la raíz del repositorio, para compilar tanto el monitor como el cliente se ejecuta el comando `make all`. Asimismo, se pueden eliminar todos los binarios con el comando `make clean`. Cabe destacar que los binarios generados fueron testeados tanto en linux como en macos.
- **Compilación del cliente:** Para compilar el cliente, estando en la raíz del repositorio, se ejecuta el comando `make client`.
- **Compilación del servidor:** Para compilar el servidor, estando en la raíz del repositorio, se ejecuta el comando `make server`.

XI. ANEXO B: INSTRUCCIONES PARA LA CONFIGURACIÓN DEL SERVIDOR Y CLIENTE

- **Ejecución del cliente:** Para ejecutar el cliente se corre el comando `bin/client -a ipServer -p portServer -u user:pass`. Donde `-a ipServer` especifica la dirección IP del servidor que soporta el protocolo monitor, `-p portServer` especifica el puerto del servidor que soporta el protocolo monitor, y `-u user:pass` especifica el usuario administrador que ya existe dentro del servidor para poder ingresar como tal.
- **Ejecución del servidor:** Para ejecutar el servidor, se puede correr el comando `bin/server -u user:pass -u user2:pass2 -a admin:admin-pass -a admin2:admin-pass2 -d mails/`. Donde el `-u user/pass` define el usuario con nombre 'user' y contraseña 'pass', y `-a admin:admin-pass` define el usuario

administrador con nombre 'admin' y contraseña 'admin-pass'. Por último el argumento -d mails/ viene a ser el path donde se encuentran los mails que va a administrar el servidor.

```
laucha@laucha:~/POP3posta/POP3$ ./bin/server -u laucha:laucha -u marco:marco -a laucha:laucha -d /home/laucha/testprotos
[INFO] 2023-06-28 11:49:35 Starting the server
[INFO] 2023-06-28 11:49:35 Registered the pop3 server socket ipv4 to attend new connection
[INFO] 2023-06-28 11:49:35 Registered the pop3 server socket ipv6 to attend new connection
[INFO] 2023-06-28 11:49:35 Registered the monitor server socket to attend new connection
```

Figura 13: *Ejemplo de ejecucion del servidor*

XII. ANEXO C: EJEMPLOS DE MONITOREO

- **Agregar un nuevo usuario pop3 al servidor** se ejecutaría, bin/client -n new-user:new-user-pass -p portServer -u user:pass. donde -u user:pass se ingresan las credenciales del administrador del servidor
- **Ver las métricas del servidor** se ejecutaría bin/client -m -p portServer -a serverAddress -u user:pass, donde metric-id es el identificador de la métrica en en RFC del monitor y -u user:pass se ingresan las credenciales del administrador del servidor. Otro ejemplo, es con el argumento -n