Protocolos de Comunicacion:

POP3

Marco Scilipoti mscilipoti@itba.edu.ar
Lautaro Hernando lhernando@itba.edu.ar
Martin Ippolito mippolito@itba.edu.ar

20 de junio de 2023

Índice

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. Por otro lado se detallaran las diferentes decisiones de diseño asi como las dificultadas encontradas.

II. Descripcion aplicacion cliente

III. Diseño y Dificultades Encontradas

I. Select y stm

Gran part

i.1. Manejo de argumentos (args.c)

Como parte del código otorgado por la cátedra, se utilizo args.c. Comprender el código fue un desafió 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 codigo 0xD001, se intentaba almacenar un numero de 2 bytes de longitud en un char (con tamaño de 1 byte).

IV. Ejemplos de prueba

I. Determinacion del buffer de salida

Para la determinacion del buffer de salida, se realizo un testeo de carga y segun el tiempo que demoraba la respuesta del servidor se determino 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 termino de procesar se tomo el tiempo que nos arrojo el comando y lo comparamos con los diferentes con los tiempos de los diferentes tamaños de buffer. Tambien 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 probo con un buffer de 2k, debemos tener en cuenta que el servidor fue compilado mediante la optimizacion *-O3* y sin el flag *fsanitize* para poder obtener la mejor performance del mismo. Cabe aclarar que durante el desarrollo si se utilizo el flag *fsanitize* y tambien se testeo con valgrind para no tener ningun memory leak.
Como podemos ver en la figura 1 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
                                 Dload  Upload   Total   Spent    Left  Speed
100 1319M    0 1319M    0     0   100M      0 --:--:--  0:00:13 --:--:--  103M
```

Figura 1: *Tiempo en procesar con buffer 2K*

i.2. Buffer de 4K

Para el segundo testeo se decidio 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 asi 2 segundos mas rapido que con el buffer de 2K, analizandolo nos parecio una gran mejorio ya que mejoro casi un 15 % el rendimiento del mismo. Tambien fue testeado 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 --:--:--  0:00:11 --:--:--  115M
```

Figura 2: *Tiempo en procesar con buffer 4K*

i.3. Buffer de 8K

En el tercer testeo dicidimos 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 --:--:--  0:00:10 --:--:--  127M
```

Figura 3: *Tiempo en procesar con buffer 8K*

i.4. Buffer de 16K

Se dicidio 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 perfomance.

```
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   120M      0 --:--:--  0:00:10 --:--:--  120M
```

Figura 4: *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 5 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 rendimineot por parte del tamaño del buffer tiene un techo el cual lo alcanzamos.

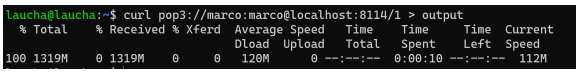

Figura 5: *Tiempo en procesar con buffer 32K*

i.6. Conclusiones

Gracias a estos testeos pudimos conlcuir 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 conlcusion de que el tamaño final del buffer es de 8K.

II. 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.

ii.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 6, 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 encutra 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 8 que los archivos son identicos menos que el output tiene el /*n* anteriormente nombrado.

Figura 6: *Creacion del mail*

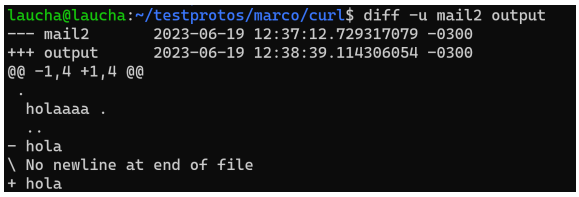

Figura 7: *Diff entre la respuesta y el mail*

Figura 8: *Cat entre mail y output*

ii.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.

Figura 9: *Diff entre 2 emails 1GB*

Como podemos ver en la figura 9 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.

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 dicidimos 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. Posibles extensiones

VII. Conclusiones

VIII. 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`.
- **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`.

IX. 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.

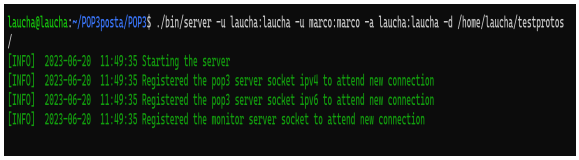

Figura 10: *Ejemplo de ejecucion del servidor*

X. 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 metric-id -p`

`portServer -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 Si no se incluye el `metric-id` entonces se imprimen todas las métricas (identificadas por su `metric-id`).

XI. Documento de diseño del monitor

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