Informe Trabajo Práctico Especial

72.07 - Protocolos de Comunicación

Primer cuatrimestre 2024

Grupo 5

Integrantes:

- Bendayan, Alberto (Legajo: 62786)
- Boullosa Gutierrez, Juan Cruz (Legajo: 63414)
- Quian Blanco, Francisco (Legajo: 63006)
- Stanfield, Theo (Legajo: 63403)

Profesores:

- Codagnone, Juan Francisco
- Garberoglio, Marcelo Fabio
- Kulesz, Sebastian

Índice

Protocolos y aplicaciones desarrolladas

Protocolo SMTP

Implementación

El protocolo SMTP (Simple Mail Transfer Protocol) es el estándar para la transmisión de correos electrónicos a través de redes IP. Este informe describe la implementación de un demonio SMTP que puede ser utilizado por Mail User Agents (MUAs) como Mozilla Thunderbird, Microsoft Outlook y Evolution para la redacción y envío de correos electrónicos. La conexión al servidor SMTP puede realizarse mediante herramientas como netcat o mediante aplicaciones cliente de correo electrónico.

Comandos

Luego de establecer la conexión con el servidor por TCP, el cliente tendrá el acceso al servidor y dispondrá de los siguientes comandos. Esto ocurre siempre y cuando el servidor le acepta la conexión al cliente.

- EHLO: utilizado por el cliente para identificarse al servidor y solicitar capacidades extendidas del servidor. Debe ser seguido por un nombre de dominio del cliente. El servidor responde con una lista de extensiones SMTP que admite.
    - Sintaxis: ‘EHLO domain’
    - Respuesta:
        - 250-localhost
        - 250-PIPELINING
        - 250 SIZE 10240000
- HELO: utilizado por el cliente para identificarse al servidor, similar a EHLO, pero sin solicitar capacidades extendidas.
    - Sintaxis: ‘HELO domain’
    - Respuesta: 250 localhost
- MAIL FROM: utilizado para iniciar una transacción de correo electrónico y especificar la dirección de correo del remitente. Debe ser seguido por la dirección de correo del remitente entre ‘<’ y ‘>’.
    - Sintaxis: ‘MAIL FROM:<sender@smtpd.com>’
    - Respuesta: 250 Mail from received - sender@smtpd.com
- RCPT TO: utilizado para especificar la dirección de correo de un destinatario de la transacción de correo actual. Debe ser seguido por la dirección de correo del destinatario entre ‘<’ y ‘>’. Puede haber múltiples comandos RCPT TO para un solo mensaje si hay varios destinatarios.
    - Sintaxis: ‘RCPT TO: <destinatario @smtpd.com>’
    - Respuesta: 250 Rcpt to received - destinatario @smtpd.com

- DATA: utilizado para iniciar la transferencia del contenido del correo electrónico, incluyendo el encabezado y el cuerpo del mensaje.
    - Sintaxis: ‘DATA’
    - Respuesta: 354 End data with <CR><LF>.<CR><LF>

  El cliente debe enviar el contenido del correo y finalizarlo con una línea que contenga solo un punto (.). Luego de finalizar el contenido del correo, el servidor responde con:
    - Respuesta: 250 Ok: queued
- QUIT: utilizado para terminar la sesión SMTP. El servidor responde con:
    - Sintaxis: ‘QUIT’
    - Respuesta: 221 Bye

Mails

Al enviar correos, nuestro servidor SMTP los guarda en la carpeta ‘mails’. Se crea una carpeta individual para cada destinatario, dentro de la cual se almacenan los correos correspondientes.

Para la generación del contenido del correo, además del texto tomado desde DATA, añadimos un encabezado al inicio el cual incluye el remitente@smtpd.com y la fecha y la hora actuales.

Para el nombre del archivo donde se guarda cada correo, utilizamos un UUID (Identificador Único Universal), garantizando así nombres de archivo únicos y evitando colisiones.

Límite de conecciones

El servidor SMTP implementado tiene un límite de 500 clientes, si se intentan conectar más clientes el único comando que podrán ejecutar es el comando QUIT para que se les corte la conexión. La respuesta que da el servidor frente a este tipo de acciones es:

- 554 failed connection to localhost SMTP - Use QUIT to close

Reporte de fallos

El servidor SMTP implementado proporciona reportes de fallos a los clientes según el error que ocurrió al detectar el comando del cliente. A continuación se detallan los posibles errores y sus respectivos mensajes:

- 500 Syntax error. Expected: *expected syntax*
    - Este error se genera cuando el servidor no reconoce el comando recibido debido a un error de sintaxis.
    - Ejemplo: ‘500 Syntax error. Expected: HELO domain or EHLO domain’
- 550 Invalid domain. The domain specified does not exist
    - Este error se produce cuando el dominio especificado en la dirección de correo del remitente o del destinatario no es ‘***@smtpd.com***’.
- 554 failed connection to localhost SMTP - Use QUIT to close

    - Como bien explicamos anteriormente, este error indica que hubo un problema al intentar conectar con el servidor SMTP, más precisamente, ya no se admiten más conexiones.
- 503 Bad sequence of commands: *Valid Sequence of commands*
    - Este error ocurre cuando los comandos no se envían en el orden correcto
    - Ejemplo: '503 Bad sequence of commands. MAIL FROM command must precede RCPT TO command'

Protocolo de Supervisión

El diseño del protocolo de supervisión fue el resultado de un arduo debate y colaboración entre todos los miembros del equipo. Considerando la extensión del proyecto y las restricciones de tiempo, decidimos optar por una solución más pragmática. Tras evaluar varias alternativas, optamos por implementar un protocolo UDP basado en texto. Esta elección se fundamentó en su simplicidad y eficiencia para nuestro caso de uso, permitiéndonos enfocarnos en otras áreas críticas del proyecto sin comprometer la funcionalidad básica.

UDP fue elegido debido a su bajo overhead y su simplicidad en comparación con TCP, lo cual es beneficioso para aplicaciones donde la rapidez es crucial. Implementamos este protocolo de tal manera que soporta conexiones tanto en IPv4 como en IPv6, asegurando una mayor flexibilidad y compatibilidad con diferentes redes y dispositivos.

A este protocolo se puede acceder mediante el puerto 6969 de nuestra aplicación a través del comando netcat

nc -u 127.0.0.1 6969

Una vez dentro, nos va a solicitar un inicio de sesión para el cual utilizamos

Usuario: "user"

Password: "user"

Una vez finalizado este inicio de sesión accedemos a los comandos que nos ofrece este servidor, siendo estos:

- *actual:* devuelve la cantidad de usuarios conectados al servidor en el instante en el que se ejecuta el comando.
- *bytes:* devuelve la cantidad de bytes transferidos hasta el momento en el que se ejecuta el comando.
- *historico:* devuelve la cantidad de usuarios que en algún momento estuvieron conectados hasta el momento en el que se ejecuta el comando.
- *mail:* devuelve la cantidad de mails enviados hasta ese momento. Este comando nos pareció útil para tener una estadística más de cómo se comporta nuestro servidor.

- *transon*: activa las transformaciones en el servidor SMTP.
- *transoff*: Desactiva las transformaciones en el servidor SMTP.
- *cant*: Muestra la cantidad máxima de usuarios que se pueden conectar al servidor.
- *help:* este comando nos devuelve todas las opciones de comandos que tenemos.

Problemas encontrados durante el diseño y la implementación

A lo largo del proyecto, hemos enfrentado una serie de desafíos que, gracias a la colaboración y el trabajo en equipo, hemos logrado superar de manera efectiva.

El primer desafío importante fue comprender en profundidad el código proporcionado por la cátedra. Este código constituía la base de nuestro trabajo y era esencial dominarlo completamente para avanzar en el desarrollo del proyecto. Para ello, dedicamos un tiempo considerable a sesiones de análisis y discusión en equipo. Durante estas sesiones, examinamos detalladamente cada parte del código, identificando sus funcionalidades y cómo podríamos integrarlas en nuestro proyecto. Este esfuerzo conjunto no solo nos permitió adquirir un entendimiento sólido del código, sino que también estableció una base de conocimiento compartido entre todos los miembros del equipo, lo cual resultó ser muy fructífero para el desarrollo exitoso y eficiente del trabajo práctico.

Avanzando en el mismo, nos hemos encontrado con errores que, a primera vista, parecían ser menores, pero que en realidad demandaron una gran cantidad de pruebas y un exhaustivo proceso de depuración para poder localizarlos y corregirlos. Estos problemas aparentemente insignificantes resultaron ser más complejos de lo anticipado, lo que nos obligó a emplear diversas técnicas de debugging y a realizar múltiples iteraciones de pruebas. Este proceso detallado y minucioso fue fundamental para garantizar la estabilidad y funcionalidad del código. La perseverancia y la atención al detalle de nuestro equipo fueron cruciales para superar estos obstáculos y avanzar de manera efectiva en el desarrollo del proyecto.

Otro problema importante fue el tema del parser. Se nos fue muy complicado realizar un parser que funcione adecuadamente. Lo que nos sucedía era que, por alguna razón, el servidor se quedaba esperando algo en un ‘select’ y no podía proceder con la ejecución del programa. Otro de los problemas surgidos con el parser era que los errores se imprimían dos o más veces. Todos los problemas que tuvimos con el parser fueron resueltos tras extenso análisis del código y mucho uso del ‘gdb’. Probablemente esta fue la tarea más tediosa y complicada de todo el trabajo práctico, pues sin este funcionamiento no podíamos probar las otras características del trabajo práctico.

Siguiendo con los problemas encontrados, uno surgió a la hora de crear la posibilidad para mandar el mail a distintos destinatarios. Esto surgió crear una lista de destinatarios y luego modificar todas las acciones, que anteriormente eran individuales, para que las realice

a toda la lista de destinatarios, tales como crear los directorios apropiados, la creación del archivo de mail y aplicar las transformaciones.

Limitaciones de la aplicación.

Una de las limitaciones que tenemos es que al usar UDP como protocolo de transporte, no podemos darle un mensaje de "bienvenida" en el momento que un nuevo usuario se conecta. Es por eso que el usuario que se conecta tiene que mandar un mensaje cualquiera con el fin de establecer conexión. Una vez establecida la conexión, el servidor le pide al cliente que inicie sesión. Finalmente se termina obteniendo un funcionamiento óptimo.

Uno de los desafíos que podríamos enfrentar a medida que nuestro sistema evoluciona es la restricción en la cantidad de usuarios que pueden estar conectados simultáneamente. Actualmente, el sistema está configurado para soportar un máximo de 500 usuarios concurrentes. Esta limitación, aunque suficiente para nuestras necesidades presentes, podría devenir en un obstáculo significativo en el futuro.

Posibles extensiones.

En aras de mejorar la funcionalidad y la seguridad de nuestro servidor, proponemos la implementación de las siguientes extensiones:

1. **Sistema de autenticación segura**: La implementación de un sistema de autenticación segura es crucial para garantizar un mayor nivel de seguridad en nuestro servidor. Este sistema debe ser capaz de verificar la identidad de los usuarios junto al uso de protocolos seguros como OAuth2. La autenticación segura protegerá los datos sensibles contra accesos no autorizados.
2. **Gestión de Datos Persistentes**: Para lograr un manejo efectivo de la información y asegurar la continuidad de las estadísticas y datos críticos tras cada reinicio del servidor, es indispensable integrar una base de datos. Esta base de datos permitirá almacenar de manera persistente todos los datos relevantes. La persistencia de datos sería algo esencial para mantener la integridad y la consistencia de la información, facilitando la recuperación y el análisis histórico de datos.
3. **Servidor de Relay (Relay Server)**: Implementar la capacidad del servidor para actuar como un relay server, es decir, como un intermediario que permite enviar correos electrónicos a otros servidores, puede expandir significativamente la funcionalidad del sistema. Esta capacidad es fundamental para aplicaciones que requieran comunicación continua y fiable mediante correo electrónico. A través de la configuración adecuada de un relay server, es posible asegurar la entrega eficiente y segura de mensajes, mejorando la interoperabilidad con otros sistemas y servicios. Este servicio debe estar configurado para manejar grandes volúmenes de correos y garantizar la entrega en tiempo real, manteniendo políticas estrictas contra el spam y asegurando la protección contra amenazas externas.

En resumen, la incorporación de estas extensiones no sólo elevará el nivel de seguridad y funcionalidad de nuestro servidor, sino que también contribuirá a mejorar la eficiencia operativa. La autenticación segura protegerá contra accesos no autorizados, la gestión de datos persistentes garantizará la continuidad y la integridad de la información, y la capacidad de actuar como un relay server facilitará la comunicación eficiente con otros servidores. Cada una de estas mejoras representa un paso significativo hacia la creación de un entorno de servidor más robusto y fiable.

Conclusiones.

El proyecto nos llevó a investigar y expandir nuestros conocimientos de manera considerable. A lo largo del proceso, enfrentamos numerosos desafíos, pero gracias al trabajo en equipo y a las clases especiales brindadas de la cátedra, creemos que logramos superarlos eficazmente. Durante este trabajo, tuvimos la oportunidad de profundizar en el protocolo SMTP, lo que nos permitió aprender a interpretar adecuadamente los RFCs, un aspecto fundamental para comprender y aplicar correctamente las especificaciones técnicas.

Finalmente, queremos destacar que como grupo de trabajo, compuesto por cuatro integrantes, logramos una colaboración eficaz. Aunque no todo el trabajo, sí la mayor parte de él, fue realizado mediante reuniones en la plataforma “Discord”. Esta herramienta nos permitió discutir en tiempo real las diferentes etapas del proyecto. Además, utilizamos la extensión “Live Share” de Visual Studio Code, la cual facilitó enormemente la colaboración simultánea, permitiéndonos escribir y editar código en conjunto, en el mismo archivo, de manera eficiente y sincronizada.

Esta metodología de trabajo no solo mejoró nuestra productividad, sino que también fortaleció nuestra capacidad para resolver problemas y tomar decisiones en equipo. Cada miembro del grupo aportó sus habilidades y conocimientos, lo que resultó en una experiencia de aprendizaje enriquecedora y un producto final de alta calidad. En resumen, la combinación de esfuerzo colaborativo, apoyo académico y herramientas tecnológicas avanzadas nos permitió superar los obstáculos y alcanzar los objetivos del proyecto con éxito.

Ejemplos de prueba.

Para esta sección del informe decidimos dividirlo en dos partes dado que son dos protocolos distintos y el buen funcionamiento de uno no depende del otro a pesar de sus interconexiones.

Protocolo SMTP

El servidor SMTP espera conexiones en el puerto 1209. Nos conectamos con el comando:

nc -C localhost 1209

Luego, va a estar esperando el saludo con el comando EHLO o HELO seguido de un dominio cualquiera. Como se observa en la imagen a continuación, si no se incluye dominio muestra un error y vuelve a esperar el comando.

```
220 localhost SMTP
EHLO
500 Syntax error. Expected: HELO domain or EHLO domain
EHLO smtpd.com
250-localhost
250-PIPELINING
250 SIZE 10240000
```

Luego se espera que el usuario ingrese al emisor del email con el comando:

mail from: <nombre@smtpd.com>

Solamente acepta dominios smtpd.com. En caso de ingresar un dominio invalido lo indica con un mensaje de error y vuelve a esperar el comando mail from.

```
mail from: <user@otherdomain.com>
550 Invalid domain. The domain specified does not exist
mail from: <user@smtpd.com>
250 Mail from received - user@smtpd.com
```

A continuación, se espera que el cliente ingrese a los receptores utilizando el comando:

rcpt to: <nombre@smtpd.com>

El comportamiento es análogo al del comando mail from.

```
rcpt to: <rector@smtpd.com>
250 Rcpt to received - rector@smtpd.com
rcpt to: <profesor@smtpd.com>
250 Rcpt to received - profesor@smtpd.com
```

Tras finalizar de indicar los receptores, se espera el comando DATA. Donde luego de ingresarlo, el usuario enviará el contenido del mail y finalizará el cuerpo ingresando ".\r\n"

```
data
354 End data with <CR><LF>.<CR><LF>
Buenas tardes,
Espero que se encuentren bien
.
250 Ok: queued
```

Por último, está el comando quit que se utiliza para cerrar la conexión.

```
mail from:<aa@smtpd.com>
250 Mail from received - aa@smtpd.com
quit
221 Bye
```

Protocolo de Supervisión

Este protocolo va a estar esperando conexiones en el puerto 6969. Mediante el comando netcat, podemos conectarnos al mismo con un comando como este:

nc -u 127.0.0.1 6969

Luego, la aplicación va a esperar una entrada cualquiera sea para poder establecer la conexión:

```
Ingrese usuario: user
Ingrese contraseña: user
Acceso concedido. Puede escribir los comandos.
HELP
 - Ingrese 'historico' para obtener el historico de usuarios conectados
 - Ingrese 'actual' para obtener los usuarios conectados ahora
 - Ingrese 'mail' para obtener la cantidad de mails enviados
 - Ingrese 'bytes' para obtener la cantidad de bytes transferidos
 - Ingrese 'status' para ver el estado de las transformaciones
 - Ingrese 'transon' para activar las transformaciones
 - Ingrese 'transoff' para desactivar las transformaciones
 - Ingrese 'cant' para obtener la maxima cantidad de usuarios
```

Una vez iniciada la conexión e iniciada la sesión, se muestra el comando help para que el usuario pueda saber qué es lo que debe hacer.

```
actual
Cantidad actual 1

historico
Cantidad historica 4
```

Siendo este un ejemplo de la cantidad de usuarios conectados en ese momento y luego la cantidad de usuarios históricos. Los comandos mail y bytes tienen un funcionamiento análogo.

Guía de instalación e instrucciones para la configuración

Para el correcto uso de este proyecto, es necesario tener instalado uuid-dev. El cual se puede instalar mediante el comando:

sudo apt install uuid-dev

Luego, para compilar el proyecto, se utiliza el comando:

CC=gcc make clean all

Por último se levanta el servidor smtp corriendo el comando:

./build/smtpd

Ejemplos de configuración y monitoreo.

Dentro de la configuración tenemos diversas opciones para configurar nuestro servidor.

Activación y desactivación de transformaciones

Para contar con las transformaciones contamos con tres comandos:

status

Nos devuelve si las transformaciones están activadas o desactivadas y luego tenemos dos comandos más:

transon

Que enciende las transformaciones y el siguiente comando que las desactiva:

transoff

```
transon
Transormaciones activadas

transoff
Transormaciones desactivadas
```

Cantidad de bytes transferidos

Para la configuración de la los bytes transferidos tenemos el comando:

bytes

Este comando devuelve la total de bytes que fueron transferidos desde la inicialización del servidor.

```
bytes
Bytes transferidos 0

cant
Cantidad maxima de usuarios 500
```

Cantidad de email enviados

Una estadística que nos parecio util para observar el buen funcionamiento del proyecto es:

mail

Este comando devuelve el total de mail enviados hasta el momento en el que se ejecuta este comando.

```
mail
Mails enviados 0
```