Trabajo Práctico Especial

Grupo 4

72.07 - Protocolos de Comunicación

Primer cuatrimestre 2024

Integrantes:

- Candisano, Gonzalo. (Legajo: 62616)
- Gonzalez Rouco, Lucas. (Legajo: 63366)
- Neme, Emilio Pablo. (Legajo: 62601)
- Shlamovitz, Theo. (Legajo: 62087)

Profesores:

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

Índice

1. Descripción detallada de los protocolos y aplicaciones desarrolladas

Protocolo SMTP

Funcionamiento

El protocolo SMTP (Simple Mail Transfer Protocol) es el estándar para el envío de correos electrónicos a través de redes IP. Fue definido inicialmente en el RFC 821 y su versión más actual se encuentra en el RFC 5321. A continuación, se describe detalladamente el funcionamiento de un servidor SMTP, teniendo en cuenta los requisitos dados por la cátedra.

Establecimiento de la conexión

Cuando un cliente (por ejemplo, Mail User Agents como Mozilla Thunderbird, Microsoft Outlook y Evolution) quiere enviar un correo, establece una conexión TCP con el servidor SMTP en el puerto 2525. El servidor debe estar preparado para aceptar conexiones entrantes y manejar múltiples clientes de forma concurrente.

Comandos soportados.

Una vez establecida la conexión, el servidor SMTP y el cliente intercambian una serie de comandos y respuestas para transferir el mensaje del correo deseado a enviar. Los comandos y respuestas aceptados son los siguientes:

- **HELO**: El cliente se presenta al servidor, indicando el nombre del host desde el cual se está conectando.
    - Respuesta del servidor: **250 localhost**
- **EHLO**: Mismo comando que el anterior, solo cambia la respuesta
    - **250-localhost**
    - **250-PIPELINING**
    - **250 SIZE 10240000**
- **MAIL FROM**: El cliente especifica la dirección de correo del remitente.
    - Respuesta del servidor: **250 2.1.0 Ok.**
- **RCPT TO**: El cliente especifica la dirección de correo del destinatario. Este comando puede repetirse para múltiples destinatarios.
    - Respuesta del servidor: **250 2.1.0 Ok**.
- **DATA**: El cliente indica que va a comenzar a enviar el cuerpo del mensaje de correo. El servidor responde con un código que indica que está listo para recibir los datos.
    - Respuesta del servidor: **354 End data with <CR><LF>.<CR><LF>.**
    - El cliente envía el cuerpo del mensaje, terminando con una línea que contiene solo un punto (.).

    - Respuesta del servidor: **250 OK: queued as <message-id>**.
- **QUIT**: El cliente termina la sesión SMTP.
    - Respuesta del servidor: **221 2.0.0 Bye.**

Manejo de errores

El servidor SMTP debe proporcionar respuestas según el comando recibido así permitiendo al cliente tomar las acciones adecuadas en caso de error. Los comandos de respuesta de errores implementados son los siguientes:

- **500 5.5.2 Error: bad syntax:** Indica que hay un error en la sintaxis
- **501 5.1.3 Bad recipient address syntax:** Indica que el mail del destinatario no es correcto
- **501 5.1.7 Bad recipient address syntax:** Indica que el mail del remitente no es correcto
- **502 5.5.2 Error: command not recognized:** Indica que el comando proporcionado no es aceptado.
- **503 5.5.1 Error: need RCPT command:** Indica que debe ejecutarse el comando “RCPT TO” antes del “DATA”
- **503 5.5.1 Error: need MAIL command:** Indica que debe ejecutarse el comando “MAIL FROM” antes del “RCPT TO”
- **503 5.5.1 Error: nested MAIL command:** Indica que el comando “MAIL FROM” ya ha sido ejecutado

Mensaje a enviar

Cuando el servidor recibe un comando DATA, comienza a recibir el cuerpo del mensaje de correo (puede incluir texto, imágenes, archivos adjuntos, etc). El servidor debe almacenar este mensaje de manera eficiente. Si el servidor implementa la transformación de mensajes, puede pasar el mensaje a través de aplicaciones externas para por ejemplo combatir el spam y el malware.

Cierre de conexión

Una vez que el cliente ha enviado todos los comandos necesarios y el mensaje de correo ha sido transferido correctamente, el cliente envía el comando QUIT. El servidor responde con el comando con código 221 (para indicar que la conexión se va a cerrar) y cierra la conexión TCP de manera limpia.

Métricas

El servidor SMTP debe implementar mecanismos para recolectar y reportar métricas operacionales. Dichas métricas sirven para registrar detalladamente la actividad del servidor y así ver tanto el desempeño como posibles problemas. Estas métricas son las siguientes:

- Conexiones históricas
- Conexiones concurrentes
- Bytes transferidos
- Mails enviados
- Estados de las transformaciones

Implementación

Para la implementación en C, específicamente con la variante C11 (ISO/IEC 9899:2011), se debe utilizar sockets para gestionar conexiones de red tanto IPv4 como IPv6, soportar simultaneidad mediante el uso de select, como también se podrían usar threads y/o procesos fork. También se debe tener una gestión eficientemente de los recursos de memoria, asegurar (en nuestro caso) que las operaciones sean no bloqueantes y, poder recolectar métricas para gestionar las configuraciones en tiempo de ejecución.

Conclusión

Implementar un servidor SMTP (sin autenticación de usuarios) implica cumplir con una serie de requerimientos funcionales y no funcionales. Un entendimiento detallado del funcionamiento del protocolo SMTP es esencial para asegurar que el servidor pueda interactuar correctamente con los clientes y manejar los correos electrónicos de manera efectiva.

Protocolo de Monitoreo

Introducción

Este documento especifica el Protocolo de Monitoreo, diseñado para consultar estadísticas y configuraciones a través de UDP, como también para modificar configuraciones. Opera de manera sin estado, con cada solicitud y respuesta encapsulada en un único datagrama UDP.

Descripción General del Protocolo

- Protocolo de Transporte: **UDP**
- Longitud del Mensaje: Fija en 14 bytes
- Gestión de Sesiones: Sin estado (no hay concepto de sesión)

Formatos de Mensaje

Mensaje de Solicitud

Un mensaje de solicitud tiene exactamente 14 bytes de longitud, consistiendo en los siguientes campos:

- Firma del Protocolo: 2 bytes, valor fijo 0xFF 0xFE
- Versión del Protocolo: 1 byte, valor fijo 0x00
- Identificador de Solicitud: 2 bytes, usado para correlacionar solicitudes y respuestas
- Autenticación: 8 bytes, campo de ancho fijo para contraseña
- Comando: 1 byte, especifica la operación a realizar

Valores de Comando:

Valor	Descripción
0x00	Cantidad de conexiones históricas
0x01	Cantidad de conexiones concurrentes
0x02	Cantidad de bytes transferidos
0x03	Cantidad de correos enviados
0x04	Ver estado de transformaciones
0x05	Activar transformaciones
0x06	Desactivar transformaciones

Estructura del Mensaje de Solicitud:

Campo	Longitud (Bytes)	Valor/Descripción
Firma del protocolo	2	0xFF 0XFE
Versión del protocolo	1	0x00
Identificador de solicitud	2	Valor único arbitrario
Autenticación	8	Usuario/Contraseña o token
Comando	1	Comando especificado

Mensaje de Respuesta

Un mensaje de respuesta tiene exactamente 14 bytes de longitud, consistiendo en los siguientes campos:

- Firma del Protocolo: 2 bytes, valor fijo 0xFF 0xFE
- Versión del Protocolo: 1 byte, valor fijo 0x00
- Identificador de Solicitud: 2 bytes, copiado de la solicitud
- Estado: 1 byte, indica éxito o tipo de error
- Datos de Respuesta: 8 bytes, contiene el resultado del comando

Valores de Estado:

Valor	Descripción
0x00	Éxito
0x01	Error: Autenticación fallida
0x02	Error: Versión inválida
0x03	Error: Comando inválido
0x04	Error: Solicitud inválida
0x05	Error: Programa transf. no definido
0x06	Error: Error inesperado
0x07-0xFF	Reservado para futuros usos

Estructura de Datos de Respuesta:

Campo	Longitud (Bytes)	Valor/Descripción
Firma del protocolo	2	0xFF 0xFE
Versión del protocolo	1	0x00
Identificador de solicitud	2	Copiado de la solicitud
Estado	1	Estado especificado
Cantidad	4	**Cantidad resultante (big-endian)**
Reservado	4	Para futuros usos

Campo	Longitud (Bytes)	Valor/Descripción
Firma del protocolo	2	0xFF 0xFE
Versión del protocolo	1	0x00
Identificador de solicitud	2	Copiado de la solicitud
Estado	1	Estado especificado
Booleano	1	**Valor booleano resultante**
Reservado	7	Para futuros usos

Ejemplos

Ejemplo de Solicitud

Un cliente envía una solicitud para obtener la cantidad de conexiones históricas:

FF FE 00 12 34 56 78 9A BC DE F0 12 34 00 00

- Firma del Protocolo: 0xFF 0xFE
- Versión del Protocolo: 0x00
- Identificador de Solicitud: 0x1234
- Autenticación: 0x56789ABCDEF01234 (token de ejemplo)
- Comando: 0x00 (cantidad de conexiones históricas)

Ejemplo de Respuesta

El servidor responde con la cantidad de conexiones históricas:

FF FE 00 12 34 00 00 00 00 02 00 00 00 00

- Firma del Protocolo: 0xFF 0xFE
- Versión del Protocolo: 0x00
- Identificador de Solicitud: 0x1234
- Estado: 0x00 (éxito)
- Cantidad: 0x00000002 (2 conexiones)
- Reservado: 0x00000000

Consideraciones de Seguridad

- Autenticación: Las solicitudes incluyen un token de autenticación de 8 bytes para verificar la identidad del cliente.

- Manejo de Errores: Los códigos de estado detallados ayudan a diagnosticar problemas, como fallos de autenticación y solicitudes inválidas.

Aplicación cliente

El cliente utiliza UDP. Este, está diseñado para enviar un mensaje a un servidor y recibir una respuesta. A continuación, se describe el funcionamiento del código en detalle:

Primero, se incluyen las cabeceras necesarias para manejar sockets, errores, direcciones de red y otras utilidades estándar de C. La función "*sockAddrsEqual*" compara dos direcciones de socket ("*sockaddr"*) para determinar si son iguales. Esta función es crucial para asegurar que las respuestas recibidas provengan del servidor correcto.

La función "*udpClientSocket*" configura y crea un socket UDP cliente. Esta función utiliza "*getaddrinfo*" para obtener una lista de direcciones que coinciden con el host y servicio especificados. El criterio de búsqueda ("*addrCriteria*") se configura para permitir tanto IPv4 como IPv6 y especificar que solo se necesitan sockets de datagramas (UDP). La primera dirección de la lista se utiliza para crear el socket cliente con la llamada a socket.

En la función principal (*main*), se validan los argumentos de entrada asegurando que se proporcionan los siguientes parámetros: una dirección del servidor, una contraseña de exactamente 8 caracteres, un puerto de servicio y un comando. Si los argumentos no son válidos, el programa muestra un mensaje de uso y termina.

Una vez validados los argumentos, se obtiene la dirección del servidor llamando a udpClientSocket. Se inicializa el generador produciendo valores aleatorios en cada ejecución. Luego, se prepara el datagrama a enviar. El datagrama tiene una longitud fija de 14 bytes y se estructura de la siguiente manera:

- Los dos primeros bytes (0xFF y 0xFE) son una firma de protocolo.
- El tercer byte (0x00) es la versión del protocolo.
- Los bytes cuarto y quinto son valores aleatorios generados con rand().
- Los siguientes ocho bytes contienen la contraseña proporcionada.
- El último byte contiene el comando convertido a su valor numérico restando '0'.

El datagrama se envía al servidor utilizando la función "*sendto*". Si el envío falla, se muestra un mensaje de error y el programa termina. Si el número de bytes enviados no coincide con el tamaño del datagrama, también se muestra un error.

Se establece un tiempo de espera de 5 segundos para la respuesta del servidor utilizando setsockopt con la opción *SO_RCVTIMEO*. Esto asegura que el programa no se bloquee indefinidamente esperando una respuesta.

La respuesta del servidor se recibe con recvfrom, almacenando la dirección y puerto de origen en "*fromAddr*". Se verifica que la respuesta provenga del servidor esperado utilizando "*sockAddrsEqual"*. Si la dirección no coincide, se muestra un mensaje de error.

El datagrama de respuesta se valida comprobando su firma de protocolo y versión. Si la firma no es 0xFF 0xFE o la versión no es 0x00, se muestra un mensaje de error y el programa termina. Si el estado en la respuesta es 0x00, se extraen y muestran el identificador y un contador de 64 bits. El contador se convierte de orden de bytes de red a orden de bytes de host con ntohl.

Si hay un error en la respuesta, se muestra un mensaje adecuado basado en el código de estado recibido:

- 0x01: Autenticación fallida.
- 0x02: Versión inválida.
- 0x03: Comando inválido.
- 0x04: Solicitud inválida (longitud incorrecta).
- 0x05: No hay definido un programa para las transformaciones.
- 0x06: Error inesperado.

Finalmente, el programa libera los recursos de la dirección del servidor con "*freeaddrinfo*" y cierra el socket antes de finalizar. Esto asegura que todos los recursos utilizados se liberen adecuadamente.

2. Problemas encontrados durante el diseño y la implementación

Debuguear Errores

Descripción del Problema: El proceso de debugging fue uno de los desafíos más significativos durante el desarrollo del trabajo. Los errores fueron surgiendo durante todas las etapas de desarrollo del proyecto.

Tipos de errores y herramientas de debugging utilizadas:

- Errores de Ejecución: Identificar errores en tiempo de ejecución que pueden hacer que el programa se detenga inesperadamente.
- Errores Lógicos: Detectar errores de lógica donde el código no produce los resultados esperados, aunque se ejecute sin fallos.
- Herramientas de Debugging: Uso de herramientas como gdb (GNU Debugger) y el debugger de CLion para inspeccionar el estado del programa en tiempo de ejecución,

establecer puntos de interrupción, y analizar el stack trace para entender mejor el flujo del programa y los valores de las variables.

Solución: La implementación de técnicas de debugging como la revisión minuciosa del código, permitió identificar y corregir errores.

Entender el Funcionamiento de Todo el Código

Descripción del Problema: Comprender el funcionamiento completo del código es muy importante para poder realizar su mantenimiento y extensión de forma correcta y exitosa.

Detalle:

- Código Complejo: La presencia de funciones complejas que realizan múltiples tareas dificulta la comprensión del flujo general del programa.

Solución: Dedicar tiempo a leer y analizar el código existente, agregando comentarios donde sea necesario.

Leer del Buffer y Escribir al Archivo Correctamente

Descripción del Problema: La utilización correcta de buffers y archivos es de suma importancia para la integridad de los datos y el correcto funcionamiento del programa.

Detalles a tener en cuenta:

- Sincronización: Asegurar que los datos se leen del buffer y se escriben en el archivo en el orden y formato correctos.
- Tamaño del Buffer: Determinar y manejar el tamaño adecuado del buffer para evitar desbordamientos o pérdidas de datos.
- Manejo de EOF (End of File): Detectar y manejar correctamente las condiciones de fin de archivo al leer o escribir datos.
- Manejo de Estados: Manejar correctamente los estados de nuestra máquina de estados y nuestras intenciones en los distintos file descriptors.

Solución: Implementar mecanismos para la lectura y escritura de datos, como el uso de funciones de manejo de archivos. Asegurarse de que el manejo de buffers se realiza de manera eficiente y segura, y utilizar técnicas como el control de flujo para evitar problemas de sincronización. Documentar bien el funcionamiento de cada uno de nuestros estados.

Manejar Correctamente los Errores

Descripción del Problema: El manejo adecuado de errores es de suma importancia para la estabilidad y fiabilidad del proyecto. Los errores pueden ocurrir en cualquier momento y deben ser manejados de manera que no comprometan la integridad del sistema.

Detalles:

- Detección de Errores: Identificar correctamente cuándo y dónde ocurren los errores.
- Manejo de Excepciones: Implementar estrategias para manejar excepciones y errores de manera que el sistema pueda recuperarse o finalizar de forma segura.
- Registro de Errores: Mantener un registro detallado de los errores para facilitar su análisis y corrección posterior.

Solución: Desarrollar un marco de manejo de errores consistente que incluya la verificación de códigos de retorno de funciones y el uso de estructuras de control de errores. Además, proporcionar mensajes de error claros y significativos que faciliten la identificación y resolución de problemas.

3. Limitaciones de la aplicación

Dentro de las limitaciones de la aplicación se encuentran las siguientes:

- Las comunicaciones entre los clientes y el servidor no estarán cifradas (ya que la implementación del servidor SMTP no soporta STARTTLS, TLS, ni SSL), dejando los datos vulnerables como por ejemplo a ataques de tipo man-in-the-middle.
- Puede llegar a resultar insuficiente la capacidad de atender múltiples clientes en forma simultánea (al menos 500) en un caso real de alto tráfico
- Las métricas se pierden en el caso de reinicio del servidor. Esto, en un caso de uso real, podría llegar a dificultar el monitoreo de los mismos a largo plazo.
- Limitación de rendimiento del sistema en el caso de usar aplicaciones externas como Spamassassin o RenAttach ya que no depende del sistema en sí si estas aplicaciones tienen fallos o si no están correctamente optimizadas.
- Aunque se permite la modificación de la configuración del servidor en tiempo de ejecución, la capacidad de realizar estos cambios puede estar limitada a ciertos parámetros específicos.

4. Posibles extensiones

Algunas posibles extensiones que podrían llegarse a implementar en un futuro para mejorar la eficiencia, seguridad y funcionalidad sobre el servidor SMTP ya implementado son las siguientes:

- Implementar métodos de autenticación seguros como por ejemplo OAuth2 o mismo la utilización de certificados digitales con el objetivo de mejorar la seguridad.
- Incorporar el uso de SSL, TLS y STARTTLS para la encriptación de la comunicación entre el cliente y el servidor con el objetivo de mejorar la seguridad.

- Implementar una base de datos para el manejo de métricas así almacenando los datos de manera persistente facilitando el monitoreo de los mismos a largo plazo ya que de esta forma no se borraran cada vez que se reinicia el servidor.
- Implementación de un dashboard (mediante una interfaz web) para facilitar la visualización de las métricas en tiempo real y poder recibir alertas sobre eventos críticos de una manera más amigable.
- Implementar filtros avanzados para el análisis y detección de spam y/o malware (además de los mencionados anteriormente que son programas externos como Spamassassin y RenAttach).
- "Mejorar" la transformación de mensajes con el objetivo de poder incluir características como la firma y encriptarlos mediante el uso de por ejemplo S/MIME.
- Implementar técnicas de balanceo de carga con el objetivo de expandir la cantidad de instancias simultáneas en el servidor.
- Ampliar el servidor para que pueda soportar correos salientes (Relay) así permitiendo el reenvío de correos a servidores externos.
- Implementar soporte para otras extensiones del protocolo SMTP definidos en diversos RFCs.

5. Conclusiones

El trabajo nos obligó a investigar y profundizar nuestros conocimientos. Fue un trabajo con muchos obstáculos, pero trabajando en equipo y, con ayuda de la cátedra, consideramos que pudimos superarlos. Se pudo trabajar en profundidad con el protocolo SMTP, aprendiendo en el camino interpretar RFCs de manera adecuada.

Por último, consideramos que como grupo trabajamos bien entre los 4, ya que por no decir todo, la gran mayoría del trabajo, lo realizamos mediante la plataforma "Discord" en llamada, así pudiendo discutir lo que se iba realizando el trabajo los 4 juntos mediante la utilización de la extensión "Live share", lo que permite escribir código de a más de a 1 persona a la vez en el mismo file.

6. Ejemplos de prueba

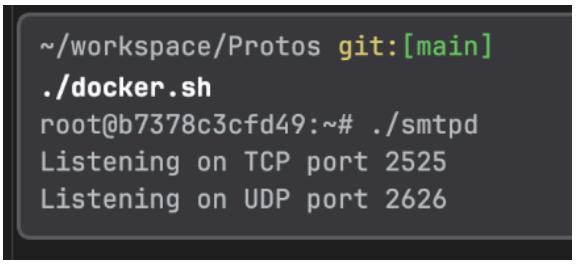
Conexión con IPV4 y helo:

```
gcandisano@DESKTOP-6EC9180:/mnt/c/Users/candi/OneDrive/Documentos/ITBA/TERCERO/2do/PROTOS/Protos$ nc 127.0.0.1 -C 2525
220 localhost SMTP
helo protos
250 localhost
```

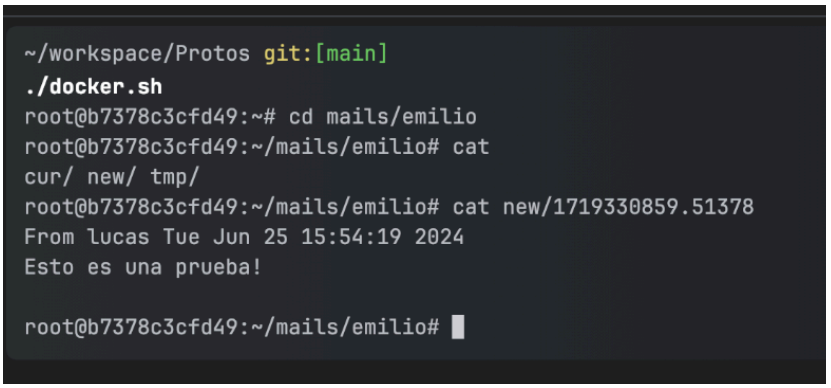
Conexión con IPV6 y ehlo:

```
gcandisano@DESKTOP-6EC9180:/mnt/c/Users/candi/OneDrive/Documentos/ITBA/TERCERO/2do/PROTOS/Protos$ nc ::1 -C 2525
220 localhost SMTP
ehlo protos
250-localhost
250-PIPELINING
250 SIZE 10240000
```

En las siguientes imágenes se puede observar como levantamos el servidor smtp por TCP en el puerto 2525 y por UDP 2626



Directorio de emails:



Transformaciones:
Podemos observar que con un servidor cuyo programa para las transformaciones es "*tac*", el resultado del mail es el inverso

```
root@b7378c3cfd49:~# nc -C localhost 2525
220 localhost SMTP
EHLO lucas
250-localhost
250-PIPELINING
250 SIZE 10240000
MAIL FROM: <lucas@pdc.com>
250 2.1.0 Ok
RCPT TO: <catedra@pdc.com>
250 2.1.0 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
5
4
3
2
1
.
250 2.0.0 Ok: queued as B9OAQ6ah2
```

```
root@b7378c3cfd49:~# ./smtpd -T tac
Listening on TCP port 2525
Listening on UDP port 2626
```

```
root@b7378c3cfd49:~# cat mails/catedra/new/1719336450.30516
From lucas Tue Jun 25 17:27:30 2024
1
2
3
4
5
```

Parámetros:

Podemos observar los distintos parámetros con los que el servidor cuenta

```
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 !3 ❱ ./smtpd -u mipass23 -P 8080 -p 2020 -T tac
Listening on TCP port 2020
Listening on UDP port 8080
Handling client ::1:40572, received 14 bytes
Handling client ::1:60853, received 14 bytes
^Csignal 2, cleaning up and exiting
closing: Interrupted system call
```

```
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 !3 ❱ ./smtpd -h
Usage: ./smtpd [OPTION]...

   -h               Imprime la ayuda y termina.
   -p <SMTP port>   Puerto entrante conexiones SMTP.
   -P <conf port>   Puerto entrante conexiones configuracion
   -u <pass>             Contraseña de admin de 8 caracteres.
   -T <program>     Prende las transformaciones.
   -v               Imprime información sobre la versión versión y termina.
```

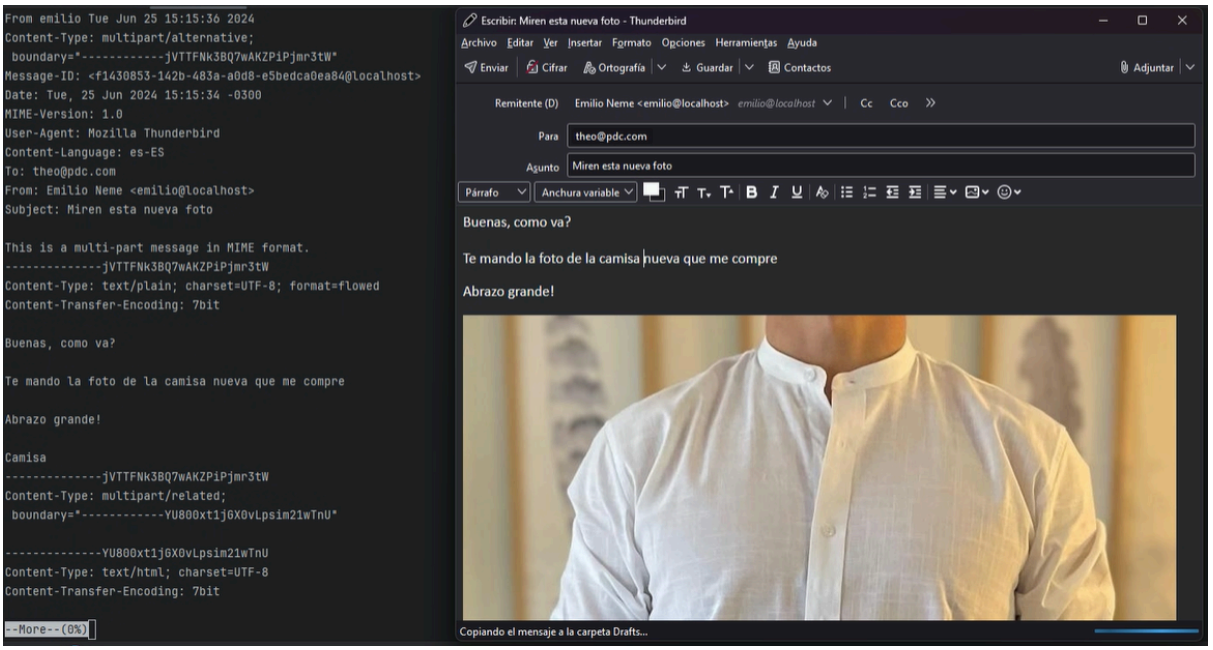
Por ejemplo, en la siguiente imagen vemos que si la contraseña no coincide, no permite el acceso.

```
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 !2 ❱ ./mng_client ::1 secretpa 8080 3
Error: Auth failed
```

A su vez, también podemos realizar un envío de correo electrónico mediante el uso de “curl”

```
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 !3 ) curl smtp://localhost:2020 --mail-from test@mail.com --mail-rcpt server@pdc.com --upload-file test.txt -v
*   Trying 127.0.0.1:2020...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0* Connected to localhost (127.0.0.1) port 2020 (#0)
< 220 localhost SMTP
> EHLO test.txt
< 250-localhost
< 250-PIPELINING
< 250 SIZE 10240000
> MAIL FROM:<test@mail.com> SIZE=58
< 250 2.1.0 Ok
> RCPT TO:<server@pdc.com>
< 250 2.1.0 Ok
> DATA
< 354 End data with <CR><LF>.<CR><LF>
} [58 bytes data]
* We are completely uploaded and fine
< 250 2.0.0 Ok: queued as sbutsAxb9
100    58    0     0  100    58      0   2669 --:--:-- --:--:-- --:--:--  2761
* Connection #0 to host localhost left intact
```

Y también mediante el uso de Mail User Agents, cómo Thunderbird.



También podemos apagar y prender las transformaciones de manera dinámica. En el siguiente ejemplo mostramos cómo se realiza, siendo el programa de la transformación ‘Tac’

```
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 ❱ cat test.txt | nc -C localhost 2525
220 localhost SMTP
250 2.1.0 Ok
250 2.1.0 Ok
354 End data with <CR><LF>.<CR><LF>
250 2.0.0 Ok: queued as GsBA0W4BV
221 2.0.0 Bye
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 ❱ cat mails/lucas/new/1719340060.25703
From emi Tue Jun 25 15:27:40 2024
2.0
Chauu
Holaa
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 ❱ ./mng_client ::1 mipass23 2626 6
Identifier: 31857
Transformations disabled
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 ❱ cat test.txt | nc -C localhost 2525
220 localhost SMTP
250 2.1.0 Ok
250 2.1.0 Ok
354 End data with <CR><LF>.<CR><LF>
250 2.0.0 Ok: queued as 6ZmfPby51
221 2.0.0 Bye
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 ❱ cat mails/lucas/new/1719340098.57778
From emi Tue Jun 25 15:28:18 2024
Holaa
Chauu
2.0
```

Además, contamos con la posibilidad de enviar un mismo mensaje a varias personas, mediante el uso de varios <RCPT TO>

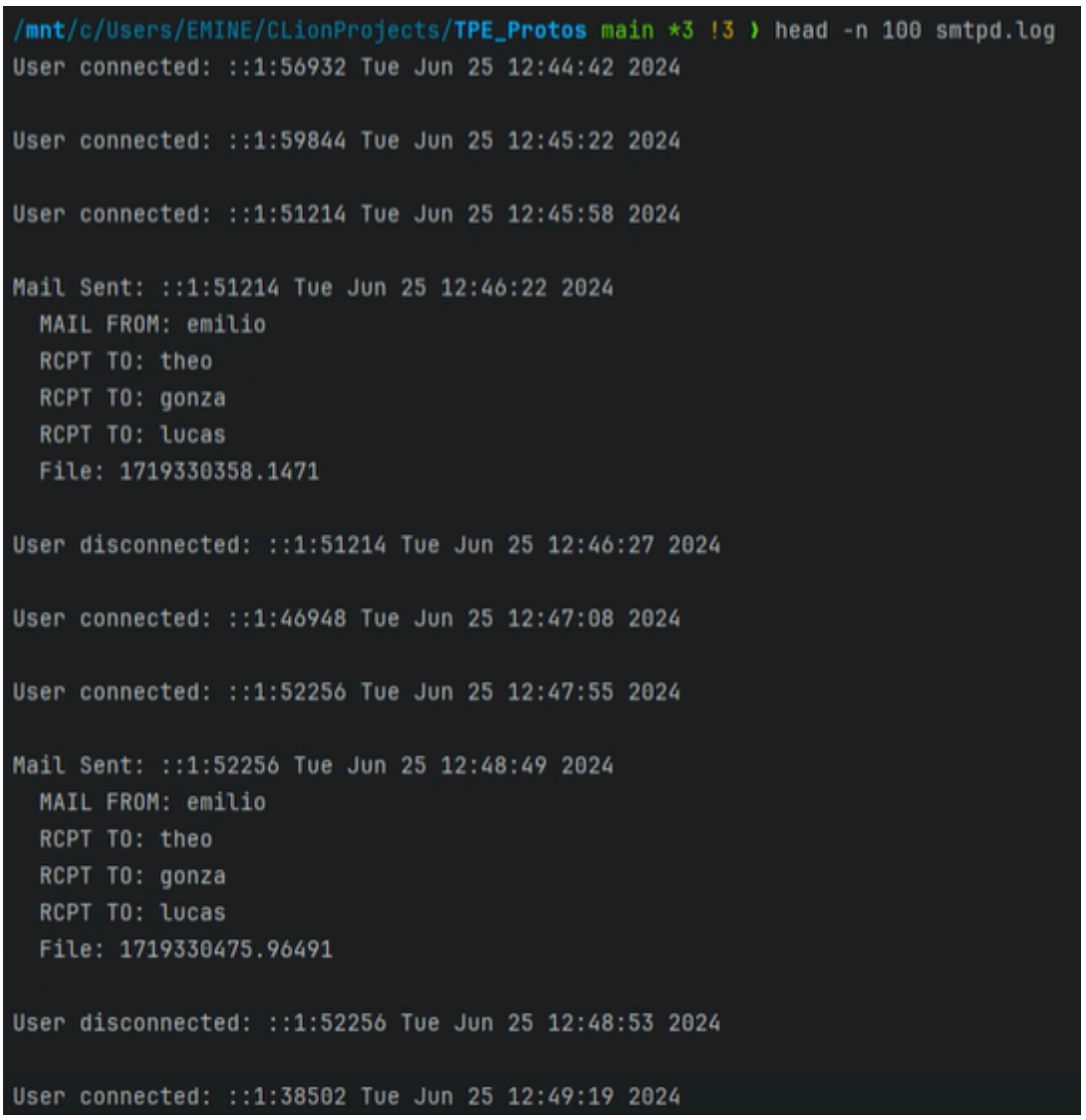
```
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 ❱ cat test2.txt | nc -C localhost 2525
220 localhost SMTP
250 localhost
250 2.1.0 Ok
250 2.1.0 Ok
250 2.1.0 Ok
354 End data with <CR><LF>.<CR><LF>
250 2.0.0 Ok: queued as ezhqKtceJ
221 2.0.0 Bye
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 ❱ cat test2.txt
HELO example.com
MAIL FROM:<sender@example.com>
RCPT TO:<theo@pdc.com>
RCPT TO:<lucas@pdc.com>
DATA
From: sender@example.com
To: recipient@example.com
Subject: Test Email

This is a test email.
.
QUIT
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 ❱ cat mails/theo/new/1719339983.61142
From sender Tue Jun 25 15:26:23 2024
From: sender@example.com
To: recipient@example.com
Subject: Test Email

This is a test email.
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 ❱ cat mails/lucas/new/1719339983.61142
From sender Tue Jun 25 15:26:23 2024
From: sender@example.com
To: recipient@example.com
Subject: Test Email

This is a test email.
```

Además se implementó un registro de acceso que permite a un administrador entender los accesos de cada uno de los usuarios.



7. Guía de instalación y Instrucciones para la configuración

Debido a que algunos de los participantes del grupo lo necesitaron, se escribieron dos archivos, “docker.sh” y “Dockerfile” para crear una imagen y un contenedor, permitiendo así la ejecución del mismo ahí dentro.
Sin embargo, utilizando o no un contenedor, la instalación es la misma.

- Correr make all en la carpeta raíz del proyecto.
    - Esto habrá generado los archivos “smtpd”, “mng_client”
- Para correr el servidor se debe correr:
    - ./smtpd [-p <smtp_port>] [-P <mng_port>] [-h] [-v] [-u <pass>] [-T <program>] donde:

        - -p es el puerto donde se correrá el servidor SMTP. Si no se especifica, su valor por defecto es 2525
        - -P es el puerto donde se correrá el servidor de métricas. Si no se especifica, su valor por defecto es 2626
        - -h imprima la ayuda y finaliza
        - -v imprime la versión y finaliza
        - -u es la contraseña que se tendrá que usar para acceder al servidor de métricas. Si no se especifica, su valor por defecto es “secretpa”
        - -T indica si las transformaciones tienen que estar o no apagadas. Si no lo están, el programa utilizado será pasado por este parámetro.

    - ./mng_client <server_address/name> <auth> <server_port> <command>
        - <server_address/name> es la dirección o nombre del servidor donde está corriendo el servidor de métricas
        - <auth> es la contraseña para acceder al servidor
        - <server_port> es el puerto del servidor al cual queremos acceder
        - <command> es el dato que queremos pedirle al servidor. Las opciones para esta campo son:
            - 0: conexiones históricas
            - 1: conexiones concurrentes
            - 2: bytes transferidos
            - 3: mails enviados
            - 4: estados de las transformaciones
            - 5: habilitar las transformaciones
            - 6: deshabilitar las transformaciones
- Correr make test en la carpeta raíz del proyecto
    - Esto correra todos los test realizados

8. Ejemplos de configuración y monitoreo

A continuación se mostrará, utilizando la aplicación cliente, algunos ejemplos de configuración y monitoreo del servidor a través del protocolo desarrollado.

Un usuario nueva intenta usar la aplicación cliente pero como no sabe qué comandos y opciones existen, se ejecuta:

```
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 !3 ❱ ./mng_client -h
Usage: ./mng_client <Server Address/Name> <Auth> <Server Port/Service> <Command>
Commands:
        0: Get historic connections
        1: Get concurrent connections
        2: Get bytes transfered
        3: Get mails sent
        4: Get transformations status
        5: Enable transformations
        6: Disable transformations
```

Servidor cliente:

```
~/workspace/Protos git:[main]
./docker.sh
root@b7378c3cfd49:~# ./mng_client localhost secretpa 2626 1
Identifier: 65428
Concurrent connections: 1
root@b7378c3cfd49:~#
```

En la siguiente imagen se muestra (de la parte de monitoreo) un ejemplo de los bytes transferidos, las conexiones históricas y las comunicaciones actuales:

```
root@b7378c3cfd49:~# ./mng_client localhost secretpa 2626 2
Identifier: 7713
Bytes transfered: 170
root@b7378c3cfd49:~# ./mng_client localhost secretpa 2626 0
Identifier: 5899
Historic connections: 1
root@b7378c3cfd49:~# ./mng_client localhost secretpa 2626 1
Identifier: 40640
Concurrent connections: 1
```

En la siguiente imagen se puede observar como se prenden y se apagan las transformaciones

```
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 !3 ) ./mng_client ::1 mipass23 8080 4
Identifier: 11544
Transformations: enabled
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 !3 ) ./mng_client ::1 mipass23 8080 6
Identifier: 36156
Transformations disabled
```

En la siguiente imagen se puede observar cómo se consultan distintas variables.

```
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 !3 ) ./mng_client ::1 mipass23 2626 1
Identifier: 17923
Concurrent connections: 8
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 !3 ) ./mng_client ::1 mipass23 2626 0
Identifier: 16963
Historic connections: 1094
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 !3 ) ./mng_client ::1 mipass23 2626 2
Identifier: 51645
Bytes transfered: 189705
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 !3 ) ./mng_client ::1 mipass23 2626 3
Identifier: 572
Mails sent: 1299
/mnt/c/Users/EMINE/CLionProjects/TPE_Protos main *3 !3 ) ./mng_client ::1 mipass23 2626 4
Identifier: 36987
Transformations: disabled
```

9. Prueba de estrés

"*stress_test.c*" implementa un test de estrés para un servidor SMTP utilizando threads. El programa crea 500 hilos, cada uno de los cuales establece una conexión TCP con un servidor SMTP local (127.0.0.1) en el puerto 2525. Cada hilo envía un mensaje de correo electrónico de prueba compuesto de varias etapas del protocolo SMTP (HELO, MAIL FROM, RCPT TO, DATA, y QUIT). Después de cada etapa, el hilo recibe y muestra la respuesta del servidor. Si una conexión o una operación de envío fallan, el hilo imprime un mensaje de error y termina.

¿Cuál es la máxima cantidad de conexiones simultáneas que soporta?

Mediante el uso del test mencionado se puede observar que soporta al menos 500 conexiones simultáneas

10. Documento de diseño del proyecto

El proyecto consta de dos componentes principales: un servidor SMTP y un cliente de monitoreo. A continuación se describe el funcionamiento general del proyecto:

Inicio del Servidor SMTP

- Configuración y Arranque: El servidor SMTP se inicia y se configura según los parámetros especificados (puertos, contraseña de métricas, programa de transformación).
- Preparación del Entorno: El servidor crea sockets para aceptar conexiones en los puertos especificados (SMTP y métricas).

Proceso de Envío de Correo (SMTP)

- Establecimiento de Conexión:
  - Un cliente (como Thunderbird, Outlook, etc.) establece una conexión TCP con el servidor SMTP en el puerto 2525.
- Intercambio de Comandos:
  - El cliente envía comandos como HELO/EHLO, MAIL FROM, RCPT TO, DATA.
  - El servidor responde con códigos adecuados (250, 354, etc.).
- Transferencia de Datos:
  - El cliente envía el cuerpo del mensaje.
  - El servidor almacena el mensaje y envía una confirmación (250 OK).
- Cierre de Conexión:
  - El cliente envía el comando QUIT.

    - El servidor responde con 221 Bye y cierra la conexión.

Manejo de Errores

- Validación de Comandos: El servidor valida los comandos y proporciona respuestas de error específicas (500, 501, 502, 503).
- Transformaciones de Mensajes: Si están habilitadas, los mensajes pasan por programas externos para la transformación (antivirus, anti-spam, transformación).

Recolección de Métricas

- Registro de Actividades: El servidor recolecta métricas como cantidad de conexiones, bytes transferidos, mensajes procesados.
- Persistencia de Métricas: Las métricas se registran en memoria y se pierden en caso de reinicio del servidor.

Protocolo de Monitoreo

- Recepción de Solicitudes UDP:
    - El cliente de monitoreo envía una solicitud UDP al servidor de métricas.
- Procesamiento de Solicitudes:
    - El servidor verifica la autenticación, valida la solicitud y ejecuta el comando (cantidad de conexiones, bytes transferidos, etc.).
- Envío de Respuesta:
    - El servidor responde con un mensaje UDP que incluye el resultado de la solicitud (éxito o error).

Interacción del Cliente de Monitoreo

- Preparación de la Solicitud: El cliente prepara un datagrama UDP con los campos necesarios (firma, versión, ID de solicitud, autenticación, comando).
- Envío de la Solicitud: El cliente envía el datagrama al servidor de métricas.
- Recepción de la Respuesta: El cliente recibe y valida la respuesta, mostrando el resultado o error al usuario.

Conclusión

- Manejo de Recursos: Ambos, el servidor SMTP y el cliente de monitoreo, liberan recursos y cierran conexiones adecuadamente al finalizar sus operaciones.