Trabajo Práctico Especial

Protocolos de Comunicación

Grupo 6

1C - 2024

Tabla de Contenidos

1. Equipo

Nombre	Apellido	Legajo	E-mail
Sol	Rodriguez	63029	solrodriguez@itba.edu.ar
Maria Agustina	Sanguinetti	63115	msanguinetti@itba.edu.ar
Uriel Ángel	Arias	63504	uarias@itba.edu.ar

2. Introducción

Este informe detalla la implementación de un servidor para el protocolo SMTP (Simple Mail Transfer Protocol) desarrollado como parte del Trabajo Práctico Especial de la materia de Protocolos de Comunicación. Se proporciona una visión detallada de la estructura y los componentes de la aplicación desarrollada en C para implementar el servidor. La aplicación permite la transferencia de correos electrónicos dentro de una red local y proporciona funcionalidades básicas de gestión de correo, al igual que un sistema de monitoreo y métricas.

3. Descripción de los protocolos y aplicaciones desarrolladas

3.1. Protocolo SMTP[1]

Para la implementación del protocolo se inició abriendo un socket para conexiones TCP desde el archivo main e incluyendo el mismo en el selector. El selector permite gestionar múltiples conexiones simultáneamente de forma no bloqueante. Luego del selector, se hace uso de una máquina de estados para saber cual es la operación a realizar utilizando handlers y funciones definidas en el archivo de smtp.c.

Para la lectura se hace uso de parsers que se encargan de leer lo que se recibe por argumentos y validar si cumple con la especificación del protocolo sobre los comandos. Para esto se tiene un parser para el manejo de requests que se reciben hasta el verbo DATA y otro que se encarga de leer de aquí en adelante hasta encontrar el fin de la transacción. Nuevamente se aprovecha el uso de estados para poder asegurar el correcto orden en cada caso y se complementa con funciones que procesan el texto recibido para ir almacenando la información pertinente para la confección del mail. Se mantienen para detectar errores semánticos que deben ser acompañados de un mensaje de error con un código más específico.

Para guardar el contenido del mail que ingresa el usuario luego de enviar el comando de DATA, se crea un archivo en la carpeta temporal (tmp) del cliente en la dirección absoluta */var/Maildir/<nombre>/tmp/<time.random>*, donde *<nombre>* es el hostname del remitente y *<time.random>* es un string único generado, en el cual a medida que se van recibiendo líneas separadas con CRLF se va escribiendo en el mismo. Finalmente, una vez ingresado <CR><LF>.<CR><LF> se cierra este archivo y se lo mueve al directorio *new* del hostname. A su vez, se agrega una línea al archivo reports.txt, el cual contiene un registro de accesos con todos los mails enviados, incluyendo el/los remitentes, el/los destinatarios y el fecha de envío.[2] Cabe destacar que para llevar a cabo este movimiento se emplea la función *rename*(), la cual puede ser bloqueante por manipular archivos. No obstante, dado que según el RFC 5321, una vez que se envía el 250.0K, el mail necesariamente tuvo que haberse enviado, por lo que si se emplea algún

[1] Ver apéndice III: Diseño del servidor SMTP
[2] Ver apéndice I: Ejemplos de prueba

método que lo envíe asincrónicamente como para evitar el bloqueo, no se estaría cumpliendo el RFC.

3.2. Mecanismos para la recolección de métricas y configuración del sistema[3]

Para el monitoreo, la recolección de métricas y los cambios de configuración del servidor SMTP, se implementó un protocolo basado en UDP (User Datagram Protocol). El objetivo de este protocolo es permitir que un cliente solicite información específica al servidor, como métricas y estados de configuración, recibiendo una respuesta inmediata. El uso de UDP se seleccionó por su baja sobrecarga y su capacidad para manejar rápidamente los datos, al no requerir de una conexión, donde la pérdida ocasional de paquetes es aceptable.

Para ello se creó un nuevo servidor de métricas y una aplicación cliente que permite enviar datagramas al servidor de una forma que sea legible para los usuarios[4]. Para la creación del servidor de métricas se crea un nuevo socket en el main del proyecto que escucha en un puerto específico. Este socket se define como UDP mediante los flags SOCK_DGRAM y IPPROTO_UDP. Para manejar múltiples conexiones de manera eficiente se utiliza el mismo selector mencionado anteriormente para el servidor SMTP el cual permite manejar las solicitudes de forma no bloqueante. Luego, se creó el archivo *metrics_handler.c* donde se implementa la lógica para procesar los paquetes recibidos y generar respuestas basadas en las métricas y configuraciones solicitadas.

La aplicación cliente proporciona una interfaz de línea de comandos interactiva, para que los usuarios indiquen la solicitud que desean realizar al servidor de métricas. La aplicación cliente crea un socket UDP con el mismo puerto y envía datagramas con las solicitudes de los usuarios al servidor de métricas. Después de enviar una solicitud, la aplicación cliente espera y recibe la respuesta del servidor de métricas a través del socket UDP. Finalmente, la respuesta se interpreta y se muestra al usuario de manera que sea legible. Las opciones de solicitud disponibles en la aplicación son conexiones históricas y concurrentes, la cantidad de bytes transferidos, el estado del modo verboso

[3] Ver apéndice IV: Diseño de la aplicación de métricas y monitoreo
[4] Ver apéndice II: Ejemplos de configuración y monitoreo

(prendido o apagado) y la configuración del mismo (apagar y prender) en tiempos de ejecución, sin reiniciar el servidor.

Con el fin de cumplir con los estándares de un protocolo UDP, se crearon las siguientes estructuras de datos para la creación de los datagramas de solicitud y respuesta.

Datagrama de solicitud

```
struct metrics_request {
    uint16_t signature;        // Validación del protocolo
    uint8_t version;           // Versión del protocolo
    uint16_t identifier;       // Identificador de la solicitud
    uint8_t auth;              // Token de autenticación
    uint8_t command;           // Comando solicitado
};
```

Datagrama de respuesta

```
struct metrics_response {
    uint16_t signature;
    uint8_t version;
    uint16_t identifier;       // Identificador de la solicitud asociada
    uint8_t status;            // Estado de la respuesta
    uint8_t response;          // Respuesta específica
};
```

Los estados posibles que se pueden recibir en un datagrama de respuesta se definen de la siguiente manera:

```
#define STATUS_OK 0x00
#define STATUS_AUTH_FAILED 0x01
#define STATUS_INVALID_VERSION 0x02
#define STATUS_INVALID_COMMAND 0x03
#define STATUS_INVALID_REQUEST_LENGTH 0x04
#define STATUS_UNEXPECTED_ERROR 0x05
```

4. Problemas encontrados

En cuanto a los problemas enfrentados en el trabajo, uno de ellos fue el uso de *htons()* y *ntohs()*, los cuales convierten datos al orden de red (big-endian) y viceversa, asegurando así la correcta transferencia de datos desde la aplicación cliente al servidor de métricas y viceversa. No fue posible lograr una correcta transformación de los datos usando estas funciones, por lo que se implementó la transferencia sin las mismas. Sin embargo, a pesar de que se entiende que hacerlo de esta manera no asegura que los datos enviados y recibidos se interpreten correctamente sin importar la arquitectura de las máquinas involucradas, se testeó en múltiples computadoras y en ninguna de ellas se vió un resultado no deseado con los datos particulares que se transfieren en nuestro protocolo.

5. Limitaciones de la aplicación.

Entre las limitaciones del proyecto, se encuentra la falta de autenticación en el servidor SMTP. El servidor SMTP implementado no soporta autenticación de usuario y contraseña (AUTH PLAIN) como se especifica en RFC 4954. Esto implica que cualquier cliente puede enviar correos electrónicos a través del servidor sin necesidad de autenticarse, lo que puede llevar a problemas de seguridad y abuso del servidor para enviar correos no deseados (spam).

A su vez, no se implementaron mecanismos de cifrado como STARTTLS, TLS o SSL. La falta de cifrado hace que la transferencia de correos electrónicos y datos sensibles se realice en texto plano, lo que puede ser interceptado y leído por terceros malintencionados. El protocolo UDP implementado para la recolección de métricas tampoco incluye un mecanismo de seguridad para la transferencia de datos.

Otra limitación que podría considerarse es el tipo de almacenamiento para los registros. Aunque se implementó un sistema de registro de acceso para rastrear las acciones de los usuarios, este se limita al almacenamiento en un archivo de texto

simple. Esto puede ser ineficiente y dificultar la búsqueda y análisis de datos históricos a gran escala. Una solución más robusta podría incluir bases de datos.

6. Posibles extensiones.

Entre las posibles futuras extensiones se encuentra la persistencia de las métricas. Actualmente éstas se restablecen cuando se reinicia el servidor, por lo que sería de gran relevancia la implementación de un mecanismo de almacenamiento persistente para las mismas, garantizando la disponibilidad continua de datos históricos incluso después de reiniciar el servidor.

A su vez, se deberían realizar mejoras de seguridad, incorporando características adicionales como autenticación del usuario, cifrado de datos y seguridad en la conexión, para proteger aún más la privacidad y la integridad de los correos electrónicos transmitidos.

Por último, se podría continuar optimizando el rendimiento del servidor mediante técnicas avanzadas de gestión de recursos, permitiendo manejar cargas aún mayores con eficiencia, así como se podrían implementar mecanismos que permitan transformar los correos electrónicos utilizando aplicaciones externas.

7. Pasos de instalación y ejecución

7.1. Requisitos previos

Es necesario tener instaladas ciertas herramientas de desarrollo como un compilador de C, como puede serlo GCC, y herramientas básicas como make para construir el proyecto.

Esta guía asume que se está utilizando Linux o algún sistema Unix-like. De no contar con ello, se pueden considerar distintas opciones para la ejecución del servidor como la utilización de máquinas virtuales, contenedores o emuladores y subsistemas de Linux como WSL.

7.2. Pasos de instalación y configuración

1. Obtener el código fuente: descargar el proyecto que contiene el código fuente del servidor SMTP.

2. Abrir una terminal y navegar al directorio raíz del proyecto.
   ```
   $ cd /TP-PROTOS
   ```

3. Ejecutar el siguiente comando para compilar el servidor SMTP y el servidor de métricas:
   ```
   $ make all
   ```
   Esto generará dos ejecutables: smtpd para el servidor SMTP y metrics_client para el servidor de métricas.

4. Dado que el servidor SMTP crea nuevos directorios en la carpeta /var/Maildir, es necesario otorgar permisos a la carpeta var. Lo mismo se puede hacer mediante el siguiente comando:
   ```
   $ sudo chmod 777 /var
   ```

   Otra alternativa es siempre correr el servidor con sudo.

5. Ejecutar el servidor SMTP mediante el siguiente comando:
   ```
   $ ./smtpd
   ```

   Si desea seleccionar un puerto en particular para el servidor SMTP puede indicarlo de la siguiente manera:
   ```
   $ ./smtpd -p <número_de_puerto>
   ```
   El puerto default si no se ingresan argumentos es 2525.

   Si desea seleccionar un puerto en particular para el servidor de métricas puede indicarlo de la siguiente manera:
   ```
   $ ./smtpd -M <número_de_puerto>
   ```
   El puerto default si no se ingresan argumentos es 7030.

Asegurarse que los puertos seleccionados estén disponibles y no estén siendo utilizados por otro servicio en el sistema.

6. Ejecutar la aplicación cliente de métricas mediante el siguiente comando:
   ```
   $ ./metrics_client
   ```

   Si desea seleccionar un puerto en particular con el cual la aplicación se comunicará con el servidor de métricas puede utilizar el siguiente comando:
   ```
   $ ./metrics_client -M <número_de_puerto>
   ```

   Asegurarse que el puerto seleccionado coincida con el puerto en el que está escuchando el servidor de métricas, ya que de no coincidir no será posible solicitar métricas al servidor. El puerto default si no se ingresan argumentos es 7030.

7. Utilizar alguna herramienta de red, como *netcat* o *telnet* para establecer una conexión con el servidor SMTP.
   ```
   $ ncat -C localhost <número_de_puerto>
   ```

   Asegurarse que el puerto coincida con el puerto en el que está escuchando el servidor SMTP.

8. Pruebas de stress

Para determinar la capacidad del servidor SMTP en términos de la cantidad máxima de conexiones simultáneas que puede soportar. Para realizar la prueba de stress se implementó un script de Bash que intenta abrir 1000 conexiones simultáneas al servidor SMTP utilizando netcat. Cada conexión se establece y se mantiene abierta para simular una carga continua en el servidor. Los resultados de las pruebas cumplieron con los requisitos establecidos, pudiendo manejar un promedio de 700 conexiones simultáneas. En algunas ocasiones, el servidor logró manejar hasta 900 conexiones simultáneas antes de que se observara una degradación significativa del rendimiento o abortese.

9. Conclusiones

En la implementación del servidor SMTP y el servidor de métricas, se ha conseguido desarrollar una solución robusta para la transferencia y monitoreo de correos electrónicos. A lo largo del proyecto, se han aplicado diversas técnicas y protocolos para asegurar el correcto funcionamiento y la capacidad de manejo de múltiples conexiones de manera simultánea y no bloqueante. Sin lugar a dudas se ha logrado aplicar los temas vistos en clase y adquirir nuevos conocimientos al realizar una amplia investigación. Aunque se identifican áreas de mejora, la solución desarrollada ofrece una base sólida y funcional que puede ser expandida y mejorada en futuras iteraciones.

10. Apéndice

1. Ejemplos de prueba

Ejemplo 1:

Iniciar el servidor SMTP tal que escuche en el puerto 2020. El servidor de métricas escuchará en el puerto 3333.

```
solrodriguez@Sol-BOOKPro:/mnt/c/Users/solro/TP-PROTOS/TP-PROTOS$ sudo ./smtpd -p 2020 -M 3333
Listening on TCP port 2020
Listening on UDP port 3333
```

Iniciar una conexión al servidor SMTP en el puerto 2020 y enviar un mail.

```
solrodriguez@Sol-BOOKPro:/mnt/c/Users/solro$ ncat -C localhost 2020
220 proto.leak.com.ar SMTP TPE-Protos
helo example
250 proto.leak.com.ar at your service
MAIL FROM: <example@proto.leak.com.ar>
250 2.1.0 OK
RCPT TO: <example_rcpt@proto.leak.com.ar>
250 2.1.5 OK
DATA
354 Go ahead
This is an example mail
.
250 2.0.0 Ok: queued as 1719339813.1598690910
```

Revisar el envío del correo en el directorio correspondiente

```
solrodriguez@Sol-BOOKPro:/var/Maildir/example$ cd /var/Maildir/example/new
solrodriguez@Sol-BOOKPro:/var/Maildir/example/new$ ls
1719339813.1598690910
solrodriguez@Sol-BOOKPro:/var/Maildir/example/new$ vim 1719339813.1598690910
```

```
From: example@proto.leak.com.ar
Sender: example@proto.leak.com.ar
To: example_rcpt@proto.leak.com.ar
Date: Tue, 25 Jun 2024 15:24:24 -0300

This is an example mail
```

Ejemplo 2 (en modo verboso y usando un pipe):

Iniciar el servidor SMTP. Puertos 2525 para el servidor SMTP y 7030 para el de mérticas por defecto

```
solrodriguez@Sol-BOOKPro:/mnt/c/Users/solro/TP-PROTOS/TP-PROTOS$ sudo ./smtpd
Listening on TCP port 2525
Listening on UDP port 7030
```

Iniciar una conexión al servidor SMTP en el puerto 2020 y enviar un mail a través de un pipe.

```
solrodriguez@Sol-BOOKPro:/mnt/c/Users/solro$ cat mail.txt | ncat -C localhost 2525
220 proto.leak.com.ar SMTP TPE-Protos
250-proto.leak.com.ar at your service
250-sol
250-PIPELINING
250 SPACE 10240000
250 2.1.0 OK
250 2.1.5 OK
354 Go ahead
250 2.0.0 Ok: queued as 1719340275.1742338831
```

Ejemplo 3: Manejo de errores

```
solrodriguez@Sol-BOOKPro:/mnt/c/Users/solro$ ncat -C localhost 2525
220 proto.leak.com.ar SMTP TPE-Protos
hola example
502 5.5.1 Unrecognizable command
helo example
250 proto.leak.com.ar at your service
MAIL FROM: <example@gmail.com>
550 Requested action not taken: mailbox unavailable
MAIL FROM: <example@proto.leak.com.ar>>>
502 5.5.1 Unrecognizable command
MAIL FROM: <example@proto.leak.com.ar>
250 2.1.0 OK
MAIL TO <example@proto.leak.com.ar>
502 5.5.1 Unrecognizable command
RCPT TO: <example@proto.leak.com.ar>
250 2.1.5 OK
DATA
354 Go ahead
This is an email example
.
250 2.0.0 Ok: queued as 1719340540.1676204530
```

2. Ejemplos de configuración y monitoreo

Realizar consultas de conexiones históricas y concurrentes.

```
solrodriguez@Sol-BOOKPro:/mnt/c/Users/solro/TP-PROTOS/TP-PROTOS$ ./metrics_client
Select the metric to request:
0: Historical Connections
1: Concurrent Connections
2: Bytes Transferred
3: Verbose mode ON
4: Verbose mode OFF
5: Verbose mode status
Pealse, enter your choice: 0
Waiting for response...
Status: OK
Number of historical connections: 15

Do you want to make another request? (y/n): y
Select the metric to request:
0: Historical Connections
1: Concurrent Connections
2: Bytes Transferred
3: Verbose mode ON
4: Verbose mode OFF
5: Verbose mode status
Pealse, enter your choice: 1
Waiting for response...
Status: OK
Number of current connections:10

Do you want to make another request? (y/n):
```

Consultar y cambiar el modo verboso

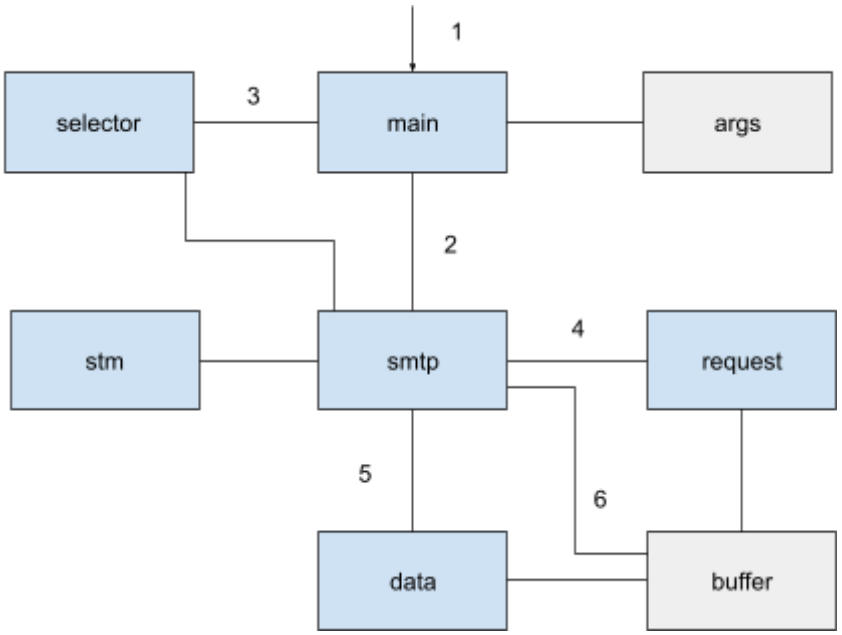
```
solrodriguez@Sol-BOOKPro:/mnt/c/Users/solro/TP-PROTOS/TP-PROTOS$ ./metrics_client
Select the metric to request:
0: Historical Connections
1: Concurrent Connections
2: Bytes Transferred
3: Verbose mode ON
4: Verbose mode OFF
5: Verbose mode status
Pealse, enter your choice: 5
Waiting for response...
Status: OK
Verbose mode is OFF

Do you want to make another request? (y/n): y
Select the metric to request:
0: Historical Connections
1: Concurrent Connections
2: Bytes Transferred
3: Verbose mode ON
4: Verbose mode OFF
5: Verbose mode status
Pealse, enter your choice: 3
Waiting for response...
Status: OK
Verbose mode is ON
```

```
Do you want to make another request? (y/n): y
Select the metric to request:
0: Historical Connections
1: Concurrent Connections
2: Bytes Transferred
3: Verbose mode ON
4: Verbose mode OFF
5: Verbose mode status
Pealse, enter your choice: 5
Waiting for response...
Status: OK
Verbose mode is ON
```

Realizar una consulta con un comando inválido.

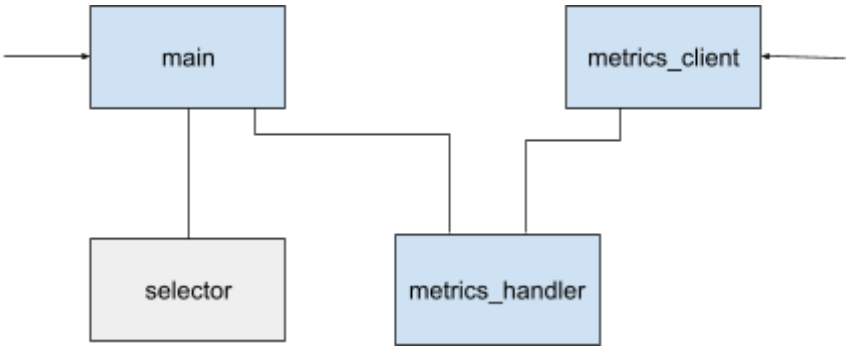
```
solrodriguez@Sol-BOOKPro:/mnt/c/Users/solro/TP-PROTOS/TP-PROTOS$ ./metrics_client
Select the metric to request:
0: Historical Connections
1: Concurrent Connections
2: Bytes Transferred
3: Verbose mode ON
4: Verbose mode OFF
5: Verbose mode status
Pealse, enter your choice: hola!
Invalid input. Please enter a number.
10
Waiting for response...
Status: Invalid command
```

3. Diseño del servidor SMTP



1. Se crea un socket TCP que escucha en el puerto especificado en args.
2. Un cliente se conecta al servidor SMTP y se acepta la conexión entrante.
3. *Se r*egistra el socket creado para la nueva conexión SMTP en el selector, el cual permite manejar múltiples conexiones.
4. Se parsean y procesan los comandos SMTP recibidos del cliente. Por ejemplo, comandos como MAIL FROM, RCPT TO, etc.
5. Al recibir el comando DATA, el servidor espera el contenido del mensaje de correo y la finalización con `<CR><LF>.<CR><LF>`.
6. Los datos del correo y las transacciones SMTP se almacenan en un buffer.

4. Diseño de la aplicación de métricas y monitoreo



1. *main* crea un socket UDP y lo enlaza a un puerto específico.
2. El socket se registra en el selector para manejar múltiples solicitudes de manera no bloqueante.
3. La aplicación cliente envía un paquete UDP al servidor de métricas, con la solicitud del usuario.
4. Cuando el servidor recibe un paquete, el *metrics_handler* procesa la solicitud.
5. Utilizando el socket UDP registrado, el servidor envía la respuesta al cliente que realizó la solicitud de métricas.

11. Bibliografía

Sockets UPD en C para Linux: https://old.chuidiang.org/clinux/sockets/udp/udp.php

How to implement UDP sockets in C:
https://www.educative.io/answers/how-to-implement-udp-sockets-in-c