Trabajo Práctico Especial 2024

Diseño e Implementación de un servidor SMTP

1. Equipo

Nombre	Apellido	Legajo	E-mail
Nicolas	Casella	62.311	**ncasella@itba.edu.ar**
Timoteo	Smart	62.844	**tsmart@itba.edu.ar**
Catalina	Müller	63.199	**camuller@itba.edu.ar**
Manuel	Quesada	63.580	**mquesada@itba.edu.ar**

2. Protocolos y aplicaciones desarrolladas

2.1 Servidor SMTP

En esta ocasión, se desarrolló la implementación de un servidor para Simple Mail Transfer Protocol (RFC 5321), con el objetivo de enviar correos electrónicos de forma rentable y eficiente.

El servidor va transicionando hacia diferentes estados a lo largo de las sesiones de conexión que inician los clientes, a través de comandos que son respondidos por el servidor. Empezando por el comando EHLO (o en su defecto HELO para mantener compatibilidad con clientes antiguos) para autenticarse ante el servidor con un nombre de usuario; MAIL FROM, que indica el remitente del correo electrónico; RCPT TO, que recibe por parámetro el receptor, siendo posible ejecutar este comando varias veces para indicar más de un receptor. Al transicionar al estado DATA, es cuando se introduce el contenido a enviar en el correo electrónico, según en el formato establecido en el RFC mencionado. Finalmente, QUIT termina la conexión con el servidor, abortando cualquier transacción en curso.

El protocolo ofrece los siguientes comandos:

- **EHLO**:

Comando que envía el usuario al servidor indicando su identidad, normalmente tras un mensaje de bienvenida de parte del servidor.

- **HELO**:

Mismas características que EHLO, mantenido por retrocompatibilidad con implementaciones anteriores de SMTP.

- **MAIL**:

Comando para especificar la dirección de correo del usuario que envía el mail. Este servidor soporta el envío de mensajes de direcciones del dominio *smtpd.com*.

- **RCPT**:

Comando para especificar la dirección de correo del destinatario. Pueden indicarse múltiples destinatarios, ejecutando este comando cuantas veces sea necesario, indicando un usuario por vez. Este servidor soporta el envío de mensajes a direcciones del dominio *smtpd.com*.

- **DATA**:

Indica el comienzo de la sección de data del mail. Aquí se incluye el contenido que irá en el mail. Se puede indicar el título del mismo utilizando el header “Subject:”, seguido por el título deseado en la misma línea. Para finalizar la sección de data, se debe escribir una línea con únicamente un punto (finalizar con <CR><LF>.<CR><LF>).

- **RSET:**

Devuelve el estado hasta antes de mandar el comando MAIL. Es decir, descarta cualquier información que se haya ingresado para los comandos MAIL, RCPT o DATA si aún no se envió el mail.

- **NOOP**

No realiza ninguna operación.

- **QUIT**

Termina la conexión con el servidor.

- **XSTAT**

Muestra en consola métricas sobre la operación del servidor. Recibe como parámetro una de las 4 palabras TOTAL, CURRENT, BYTES, o MAILS.

El servidor devuelve distintos códigos de error ante las siguientes situaciones:

- 451 (status_local_error_in_processing): ante un error por parte del servidor.
- 500 (status_syntax_error_no_command): cuando se intenta ejecutar un comando no reconocido que comienza por X.
- 501 (status_syntax_error_in_parameters): error en la cantidad o sintaxis de los parámetros de un comando. El servidor responde indicando cuál es la sintaxis adecuada para utilizar ese comando.
- 502 (status_cmd_not_implemented): cuando se intenta ejecutar un comando no reconocido.
- 503 (status_bad_seq_cmds): cuando se intenta ejecutar un comando que precisa que otros comandos se ejecuten anteriormente, sin haber ejecutado esos comandos previos.
- 550 (status_mailbox_not_found): dominio de mail del remitente inválido.

2.2 Aplicación cliente para monitoreo

El cliente de monitoreo consiste en una aplicación que envía *requests* al servidor SMTP con el fin de conseguir métricas. Para seguir el protocolo establecido por la aplicación, el servidor SMTP debe disponer de los siguientes comandos:

- **NOOP:**

  Utilizado por la aplicación para calcular el delay que tiene el servidor a la hora de responder *requests*. El valor puede estar representado por milisegundos.

- **XSTAT**

  Este comando de uso privado es utilizado por la aplicación para hacer los siguientes *requests*, en caso de que el servidor no disponga de XSTAT, el mismo no será compatible con el protocolo de monitoreo.:

  1. **XSTAT BYTES:**

Devuelve la cantidad de bytes que transfirió el servidor. La respuesta debe tener como *response* la forma: *total bytes of data transferred*: X, donde X representa un entero.

2. **XSTAT TOTAL:**
   Devuelve la cantidad de conexiones totales históricas del servidor, estas pueden ser volátiles, debe tener como *response* la forma: *total connections*: X, donde X representa un entero.
3. **XSTAT CURRENT:**
   Devuelve la cantidad de conexiones concurrentes del servidor, debe tener como *response* la forma: *current connections*: X, donde X representa un entero.
4. **XSTAT MAILS:**
   Devuelve la cantidad de mails enviados por el servidor, debe tener como *response* la forma: *total mails sent*: X, donde X representa un entero.

2.3 Aplicación cliente para configuración

El cliente de monitoreo consiste en una aplicación que envía mensajes al servidor a través de un puerto predefinido. Para que el servidor sea compatible con el cliente, debe seguir el siguiente protocolo:

- Debe disponer de un puerto que reciba comandos.
- Debe enviar dos estados como respuesta a cada mensaje entrante:
  1. **OK** Para representar una operación exitosa.
  2. **ERR** Para representar una operación no exitosa.
- Todos los mensajes deben tener la forma <*param*>=<*integer*>. En caso de que haya un valor no entero, debe ser rechazado con ERR.
- Debe disponer de un comando QUIT, el cual corta la conexión entre el cliente y el servidor. El servidor debe responder con OK.
- Debe ser *case-insensitive*.
- Los parámetros posibles pueden depender de la implementación.

En la implementación específica a este proyecto, solo se dispone del parámetro:

- **logger**

El cual activa/desactiva los *logs*, donde 0 los desactiva y cualquier otro valor los activa.

3. Problemas encontrados durante el diseño y la implementación

Un problema que surgió fue la concurrencia de múltiples clientes simultáneamente. Para implementarlo se hizo uso de threads, atendiendo a cada cliente en un thread. El problema que surgió de esto fue que las variables globales son compartidas entre threads, por lo que varios clientes las accedían y modificaban simultáneamente, y se pisaban los valores entre distintos threads.

Para solucionarlo, se creó una estructura *client_state*, la cual contiene varias de las variables necesarias para el funcionamiento del protocolo, y se agregó a la estructura *smtp*. Por cada conexión se crea una estructura smtp, por lo que las variables pasaron a ser únicas para cada thread.

Además, surgieron problemas a la hora de crear e implementar el cliente de configuración, principalmente en todo lo relacionado a la espera y recibimiento de mensajes. Se pudo solucionar al agregar un segundo listener y handler específicamente diseñado para este propósito: *config_handler.c*.

Por otro lado, la falta de autenticación de usuario limitó las posibilidades en lo que respecta al monitoreo y configuración dinámica del servidor. Una futura extensión del proyecto permitiría mejorar la seguridad del mismo. Aún así, el puerto de configuración es distinto del puerto de conexión, permitiendo la ofuscación del mismo mediante otras herramientas.

4. Limitaciones de la aplicación

- No permite autenticación de usuario.
- Tiene un límite de 800 usuarios concurrentes, cualquier valor mayor es inestable y no recomendado.

5. Posibles extensiones

- **Autenticación de usuario**: el servidor podría soportar autenticación usuario/contraseña (AUTH PLAIN) [RFC4954[1]].
- **Implementación de POP3**: se podría diseñar una implementación de POP3 compatible con el servidor SMTP.

[1] https://datatracker.ietf.org/doc/html/rfc4954

- **Servidor como relay**: el servidor podría aceptar correspondencia para usuarios externos, no solamente locales.

6. Conclusiones

A pesar de las dificultades encontradas a lo largo del desarrollo, el proyecto fue de gran utilidad para adentrarse y familiarizarse con el uso, creación e implementación de protocolos de red.

7. Ejemplos de prueba

Para todos los casos de prueba es necesario

- Seguir los pasos detallados en la guía de instalación

- Abrir una terminal y correr el comando:

```
nc -C localhost 2525
```

Enviar un correo genérico desde localhost

- Dentro de la terminal ejecutar los siguientes comandos:

```
EHLO <User>
MAIL FROM: <User>@smtpd.com
RCPT TO: <Recipient>
DATA
<Contenido del correo>

<CR><LF>.<CR><LF>
```

- El correo debería estar encolado y presente en la carpeta mail_dir/<User>

Obtener respuestas de error de parte del servidor

- Dentro de la terminal ejecutar los siguientes comandos:

```
RCPT TO: rcpt@smtpd.com
MAIL FROM: user@smtp.com
DATA
EHLO
MAIL FROM: user
```

```
RCPT TO:
INVALID COMMAND
XSTAT
```

Mandar dos correos diferentes concurrentemente desde 2 terminales

- Abrir otra terminal y correr en ambas el comando:

```
nc -C localhost 2525
```

- En la primera terminal ejecutar los siguientes comandos:

```
EHLO user1
MAIL FROM: user1@smtpd.com
RCPT TO: user2
DATA
Subject: para user2
Este correo está destinado al usuario 2.
<CR><LF>.<CR><LF>
```

- En la segunda terminal ejecutar los siguientes comandos:

```
EHLO user2
MAIL FROM: user2@smtpd.com
RCPT TO: user1
DATA
Subject: para user1
Este correo está destinado al usuario 1.
<CR><LF>.<CR><LF>
```

Mandar un correo a distintos remitentes

- Dentro de la terminal ejecutar los siguientes comandos:

```
EHLO <User>
MAIL FROM: <User>@smtpd.com
RCPT TO: <Recipient1>
RCPT TO: <Recipient2>
RCPT TO: <Recipient3>
RCPT TO: <Recipient4>
RCPT TO: <Recipient5>
DATA
<Contenido del correo>
```

```
<CR><LF>.<CR><LF>
```

Escribir un correo vacío

- Dentro de la terminal ejecutar los siguientes comandos:

```
EHLO <User>
MAIL FROM: <User>@smtpd.com
RCPT TO: <Recipient>
DATA

<CR><LF>.<CR><LF>
```

Mandar una secuencia repetida de comandos

- Dentro de la terminal ejecutar los siguientes comandos:

```
EHLO <User>
EHLO <User>
MAIL FROM: <User>@smtpd.com
MAIL FROM: <User>@smtpd.com
RCPT TO: <Recipient>
RCPT TO: <Recipient>
DATA
<Contenido del correo>

<CR><LF>.<CR><LF>
```

Visualizar datos sobre la operación del sistema

- Dentro de la terminal ejecutar los siguientes comandos:

```
XSTAT TOTAL
XSTAT CURRENT
XSTAT MAILS
XSTAT BYTES
EHLO <user>
MAIL FROM: <user>@smtpd.com
RCPT TO: <recipient>
XSTAT MAILS
DATA
Subject: test
data
.
```

```
XSTAT MAILS
```

- (*)Sin cerrar esta terminal, abrir una nueva terminal y correr el comando:

```
nc -C localhost 2525
```

- En la primera terminal correr el comando:

```
XSTAT TOTAL
XSTAT CURRENT
```

- Terminar la conexión de la segunda terminal con ctrl + c
- Volver a ejecutar los dos comandos:

```
XSTAT TOTAL
XSTAT CURRENT
```

- Volver a realizar los pasos desde (*), pero esta vez terminando la conexión de la segunda terminal ejecutando el comando QUIT en lugar de ctrl + c

8. Guía de instalación

Para poder compilar y correr el servidor es necesario descargar o clonar el repositorio de Github: TPE - PDC[2].

Para compilar el código, se debe correr el siguiente comando dentro de la carpeta */src*:

```
make clean all
```

Una vez compilado se creará el archivo smtpd dentro de la misma carpeta. Para ejecutarlo, correr:

```
./smtpd  -<flag> <param>
```

Para ver una lista de parámetros posibles, se puede usar el flag *-h*

Es necesario disponer de un entorno Linux para poder compilar y ejecutar el servidor.

9. Instrucciones para la configuración

Para poder configurar el servidor, primero situarse en el directorio /config y correr:

```
make clean all
```

Luego de la compilación, se creará el archivo client_config que se usará para configurar el servidor. Ejecutalor:

```
./client_config  -<flag> <param>
```

Marcando el -h como flag mostrará los argumentos posibles.

Una vez ejecutado el programa y habiendo hecho conexión con el servidor, se abrirá una línea de comandos en el que se podrán ejecutar los comandos que se encuentran disponibles ejecutando el comando HELP.

10. Ejemplos de configuración y monitoreo

Un registro de las acciones del servidor puede ser encontrado dentro del archivo *logs/log.txt*.

[2] https://github.com/catamuller/TPE-PDC

Se puede correr el comando `XSTAT <argumento>` con alguno de los siguientes 3 argumentos para obtener información sobre la operación del sistema:

- TOTAL: Muestra la cantidad de conexiones totales históricas al servidor desde la última vez que se inició (este valor se reinicia si se reinicia el servidor).
- CURRENT: Muestra la cantidad de conexiones concurrentes al momento de ejecutar el comando.
- BYTES: Muestra la cantidad total de bytes de data transferidos por mail a través del servidor (este valor se reinicia si se reinicia el servidor).
- MAILS: Muestra la cantidad total de mails enviados (este valor se reinicia si se reinicia el servidor).

Estos comandos son utilizados por el cliente de monitoreo para recolectar información del servidor. A modo de ejemplo se puede seguir los siguientes pasos:

1. Iniciar el servidor en el puerto default, si lo inicia en otro, hace falta incluirlo en el cliente de monitoreo.
2. Posicionarse dentro de la carpeta *monitoring*.
3. Correr el comando *make clean all*.
4. Correr el comando *./metrics -n -v -P <puerto>*.
5. Estas flags específicas permiten al usuario ver la versión del cliente y conexiones detectadas por el comando *netstat*, a modo de comparación. Una lista de flags puede verse con el flag -h.
6. A continuación se debería ver la interfaz del cliente de monitoreo.

Actualmente el cliente de configuración solo permite activar/desactivar los logs mencionados anteriormente. Para poder correrlo hace falta seguir los siguientes pasos:

1. Iniciar el servidor, por default el puerto de configuración es 2526 y la ip de configuración es la misma que la del servidor, en caso de querer modificarlo, incluir el flag -C *<ip>* -c *<puerto>*.
2. Posicionarse dentro de la carpeta *config*.
3. Correr el comando *make clean all*.
4. Correr el comando *./client_config -i <ip> -P <puerto>*
5. Correr el comando *help* dentro del cliente para ver una lista de comandos posibles.
6. Enviar *logging*=0.
7. Entrar al servidor como cliente, enviar un mail siguiendo las instrucciones en la sección *Ejemplos de prueba.*
8. Enviar *logging*=1 desde el cliente de configuración.
9. Enviar *quit*.

10. Apagar el servidor.
11. Si se revisan los logs debería figurar *Server Initialized y Server Shutting Down*, pero ningún registro del cliente.

11. Diseño del proyecto: arquitectura de la aplicación

La aplicación servidor de SMTP consiste en tres máquinas de estados principales: *client_state_machine*, *smtp_parser*, y *stmp_data_parser*.

La *client_state_machine* es una máquina de estados que se desplaza dependiendo en qué estado (de secuencia de comandos) se encuentra el cliente que lo usa. La misma consiste de la misma máquina de estados provista por la cátedra, con dos extensiones: la máquina puede volver hacia el estado anterior; y guarda la IP y puerto, o el *hostname* que ingresa el usuario al escribir el comando *EHLO* o *HELO*. La *state machine* tiene un total de 42 estados, donde los principales son se dividen en *requests* del cliente y *responses* del servidor. Los restantes son códigos de error por parte del servidor.

Los estados de la *client_state_machine* de las *requests* son los siguientes:

- **CLIENT_HELLO**: recibe la *request* del usuario para el comando *EHLO* o *HELO*.
  Si el usuario ya ingresó el comando, retorna un código de error de que ya realizó la acción.
  Si el *hostname* ingresado es inválido, retorna un código de error de que no puede ingresar un *hostname* vacío.
  En el caso de ser exitoso, la máquina de estados salta a *SERVER_HELLO* o *SERVER_EHLO*, dependiendo si ingresó *HELO* o *EHLO*, respectivamente.
- **CLIENT_RSET**: recibe la *request* del usuario para el comando *RSET*.
  Si el usuario ingresó el comando antes de realizar un *HELO* o *EHLO*, retorna un código de error de que debe hacerlo antes de correr el comando.
  En caso de ser exitoso, la máquina de estados salta a *SERVER_RSET*.
- **CLIENT_NOOP**: recibe la *request* del usuario para el comando *NOOP*.
  En todo caso, la máquina de estados salta a *SERVER_NOOP*.
- **CLIENT_HELP**: recibe la *request* del usuario para el comando *HELP*.
  En todo caso, la máquina de estados salta a *SERVER_HELP*.
- **CLIENT_VRFY**: recibe la *request* del usuario para el comando *VRFY*.
  Este comando no está implementado.
  En todo caso, la máquina de estados salta a *SERVER_VRFY*.
- **CLIENT_STAT_CURRENT_CONNECTIONS**: recibe la *request* del usuario para el comando *XSTAT CURRENT*.
  En todo caso, la máquina de estados salta a *SERVER_STAT_CURRENT_CONNECTIONS*.

- **CLIENT_STAT_TOTAL_CONNECTIONS**: recibe la *request* del usuario para el comando XSTAT TOTAL.
  En todo caso, la máquina de estados salta a SERVER_STAT_TOTAL_CONNECTIONS.
- **CLIENT_STAT_BYTES_TRANSFERED**: recibe la *request* del usuario para el comando XSTAT BYTES.
  En todo caso, la máquina de estados salta a SERVER_STAT_BYTES_TRANSFERED.
- **CLIENT_STAT_MAILS_SENT**: recibe la *request* del usuario para el comando XSTAT MAILS.
  En todo caso, la máquina de estados salta a SERVER_STAT_MAILS_SENT.
- **CLIENT_MAIL_FROM**: recibe la *request* del usuario para el comando MAIL.
  Si el usuario ingresó el comando antes de realizar un *HELO* o *EHLO*, retorna un código de error de que debe hacerlo antes de correr el comando.
  Si el usuario ya ingresó el comando anteriormente, retorna un código de error de que ya realizó la acción anteriormente.
  Si el usuario ingresó el comando incorrectamente, retorna un código de error con la sintaxis del mismo.
  En caso de ser exitoso, la máquina de estados salta a SERVER_MAIL_FROM.
- **CLIENT_RCPT_TO**: recibe la *request* del usuario para el comando RCPT.
  Si el usuario ingresó el comando antes de realizar un MAIL, retorna un código de error de que debe hacerlo antes de correr el comando.
  Si el usuario ingresó el comando incorrectamente, retorna un código de error con la sintaxis del mismo.
  En caso de ser exitoso, la máquina de estados salta a SERVER_RCPT_TO.
- **CLIENT_DATA**: recibe la *request* del usuario para el comando DATA.
  Si el usuario ingresó el comando antes de realizar un RCPT, retorna un código de error de que debe hacerlo antes de correr el comando.
  En caso de ser exitoso, la máquina de estados salta a SERVER_DATA.
- **CLIENT_MAIL_CONTENT**: recibe la request del comando para el contenido del MAIL a enviar.
  En todo caso, la máquina de estados salta a SERVER_MAIL_END.

Por otro lado, los estados de las *responses* son las siguientes:

- **SERVER_GREETING**: envía un mensaje de bienvenida al cliente al iniciar la conexión.
- **SERVER_HELLO**: realiza la *response* del CLIENT_HELLO.
  Ante un fallo, la máquina de estados vuelve a saltar a SERVER_HELLO.

En caso exitoso, el servidor retorna el código 250 de SMTP, y la máquina de estados salta a *CLIENT_MAIL_FROM*.

- **SERVER_EHLO**: realiza la *response* del CLIENT_HELLO.
  Ante un fallo, la máquina de estados vuelve a saltar a *SERVER_EHLO*.
  En caso exitoso, el servidor retorna el código 250 de SMTP, junto a todos los comandos disponibles, y la máquina de estado salta a *CLIENT_MAIL_FROM*.
- **SERVER_RSET**: realiza la *response* del *CLIENT_RSET*.
  Ante un fallo, la máquina de estados vuelve a saltar a *SERVER_RSET*.
  En caso exitoso, el servidor retorna el código 250 de SMTP, y la máquina de estados salta a *CLIENT_MAIL_FROM*.
- **SERVER_NOOP**: realiza la *response* del *CLIENT_NOOP*.
  Ante un fallo, la máquina de estados vuelve a saltar a *SERVER_NOOP*.
  En caso exitoso, el servidor retorna el código 250 de SMTP, y la máquina de estados salta al estado anterior.
- **SERVER_HELP**: realiza la *response* del *CLIENT_HELP*.
  Ante un fallo, la máquina de estados vuelve a saltar a *SERVER_HELP*.
  En caso exitoso, el servidor retorna el código 250 de SMTP, y la máquina de estados salta al estado anterior.
- **SERVER_VRFY**: realiza la *response* del *CLIENT_VRFY*.
  Ante un fallo, la máquina de estados vuelve a saltar a *SERVER_VRFY*.
  En caso exitoso, el servidor retorna el código 502 de SMTP, y la máquina de estados salta al estado anterior.
- **SERVER_STAT_CURRENT_CONNECTIONS**: realiza la *response* del *CLIENT_STAT_CURRENT_CONNECTIONS*.
  Ante un fallo, la máquina de estados vuelve a saltar a *SERVER_STAT_CURRENT_CONNECTIONS*.
  En caso exitoso, el servidor retorna la cantidad de conexiones en el momento que se solicitó, y la máquina de estados salta al estado anterior.
- **SERVER_STAT_TOTAL_CONNECTIONS**: realiza la *response* del *CLIENT_STAT_TOTAL_CONNECTIONS*.
  Ante un fallo, la máquina de estados vuelve a saltar a *SERVER_STAT_TOTAL_CONNECTIONS*.
  En caso exitoso, el servidor retorna la cantidad de conexiones totales en el momento que se solicitó, y la máquina de estados salta al estado anterior.
- **SERVER_STAT_BYTES_TRANSFERED**: realiza la *response* del *CLIENT_STAT_BYTES_TRANSFERED*.
  Ante un fallo, la máquina de estados vuelve a saltar a *SERVER_STAT_BYTES_TRANSFERED*.
  En caso exitoso, el servidor retorna la cantidad de bytes transferidos en el momento que se solicitó, y la máquina de estados salta al estado anterior.

- **SERVER_STAT_MAILS_SENT**: realiza la *response* del *CLIENT_STAT_MAILS_SENT*.
  Ante un fallo, la máquina de estados vuelve a saltar a *SERVER_STAT_MAILS_SENT*.
  En caso exitoso, el servidor retorna la cantidad de mails enviados en el momento que se solicitó, y la máquina de estados salta al estado anterior.
- **SERVER_MAIL_FROM**: realiza la *response* del *CLIENT_MAIL_FROM*.
  Ante un fallo, la máquina de estados vuelve a saltar a *SERVER_MAIL_FROM*.
  En caso exitoso, el servidor retorna el código 250 de SMTP, y la máquina de estados salta a *CLIENT_RCPT_TO*.
- **SERVER_RCPT_TO**: realiza la *response* del *CLIENT_RCPT_TO*.
  Ante un fallo, la máquina de estados vuelve a saltar a *SERVER_RCPT_TO*.
  En caso exitoso, el servidor retorna el código 250 de SMTP, y la máquina de estados salta a *CLIENT_DATA*.
- **SERVER_DATA**: realiza la *response* del *CLIENT_DATA*.
  Ante un fallo, la máquina de estados vuelve a saltar a *SERVER_DATA*.
  En caso exitoso, el servidor retorna el código 250 de SMTP, con la forma de finalizar la sección de *DATA*, y la máquina de estados salta a *CLIENT_MAIL_CONTENT*.
- **SERVER_MAIL_END**: realiza la *response* de *CLIENT_MAIL_CONTENT*.
  Ante un fallo, la máquina de estados vuelve a saltar a *SERVER_MAIL_END*.
  En caso exitoso, el servidor retorna el código 250 de SMTP, y la máquina de estados salta a *CLIENT_MAIL_FROM*.
- **QUIT**: realiza la *response* ante la *request* del comando *QUIT* por parte del usuario.
  En caso exitoso, el servidor retorna el código 221 de SMTP, y la máquina de estados salta a *CLOSE*, el cual finaliza la conexión con el usuario.

El *smtp_parser* es la segunda máquina de estados que se desplaza dependiendo de la entrada del usuario ante cualquier *request*, excepto luego de ingresar el comando *DATA*, luego de la secuencia para enviar un correo. Para eso se utiliza la otra máquina de estados, *smtp_data_parser*. La *smtp_parser* consta de un total de 117 estados, mientras que *smtp_data_parser* consta solamente de 6. Ambas, a su vez, se encargan de guardar los usuarios junto a su dominio ingresado en el comando *MAIL*, los destinatarios ingresados en el comando *RCPT*, y la información ingresada luego del comando *DATA*. La primera máquina de estados se encuentra en el archivo *smtp.c*, dentro del directorio *src*, mientras que las restantes se encuentran en el archivo *smtp_parsing.c*.

Por último, luego de recibir <*CR*><*LF*>.<*CR*><*LF*>, luego de recibir el comando *DATA*, se realiza un llamado a la función *sendMail*, el cual se encarga de: parsear el *Subject*, el cual será utilizado para el nombre del archivo; crear los directorios para los

valores ingresados luego del comando RCPT; y por último escribir el contenido sobre un archivo.

12. Código externo

Se utilizó código brindado por la cátedra, además de basar el proyecto en código existente también provisto por la cátedra.