Instituto Tecnológico de Buenos Aires

TRABAJO PRÁCTICO

ENTREGA TPE PROTOCOLOS DE COMUNICACIÓN (72.03)

Grupo 3

Integrantes:

Mauro Joaquín Vella - Legajo N° 62134 (mvella@itba.edu.ar)

Bruno Enzo Baumgart - Legajo N° 62161 (bbaumgart@itba.edu.ar)

Santiago José Hirsch - Legajo N° 62169 (shirsch@itba.edu.ar)

Docentes:

Juan Francisco Codagnone

Marcelo Fabio Garberoglio

Hugo Javier Stupenengo Faus

Pedro Martín Valerio Ramos

Sebastian Kulesz

Robert Oscar Axt

Thomas Mizrahi

1. Introducción

El servidor POP3 inicia su implementación extrayendo los parámetros proporcionados al programa. Estos parámetros definen el directorio raíz, opcionalmente el puerto en donde se levanta el servidor y usuarios con sus respectivas contraseñas. Es crucial destacar que esta estructura de directorios debe estar creada con anterioridad y, los sub-directorios deben coincidir exactamente con los usuarios proporcionados como argumentos; de lo contrario, esta fase fracasará, y la ejecución se detendrá.

Tras completar la fase de inicialización, el servidor se suscribe para la lectura en el selector y añade un controlador (*handler*) encargado de aceptar de forma pasiva a los nuevos clientes.

Cuando llegan nuevos clientes, se crea una nueva sesión para cada uno de ellos. Cada sesión representa una conexión con el cliente y se libera una vez que el cliente finaliza su conexión con el servidor.

Al iniciar una nueva sesión, se establece un estado base de lectura (*READ*), que permanece constante, junto con un estado de proceso (*PROCESS*). Además, la sesión se suscribe para operaciones de escritura y se configura el controlador correspondiente. Posteriormente, cuando el servidor intenta escribir al cliente, detecta que el estado 'PROCESS' se encuentra en la cima de la pila de estados. En consecuencia, ejecuta la función '*continue_session'*. Durante esta función, se procesa el saludo inicial, y una vez completado, se coloca en el búfer de escritura, se elimina el estado de la cima de la pila (es decir, el estado PROCESS), y se agrega el estado de escritura (WRITE). De este modo, cuando sea su turno de escritura según el selector, podrá realizar esta operación.

Cuando llega el momento de escribir, se presentan dos escenarios. Si todo puede ser enviado correctamente al cliente, se elimina el estado de escritura (WRITE) y se vuelve a suscribir a lectura, ajustando el controlador correspondiente. Sin embargo, si no se logra enviar todo al cliente, se añade un estado adicional de escritura (WRITING), indicando que existen elementos en el búfer de escritura que deben completarse. Por consiguiente, en su próximo turno, la sesión retomará desde donde se quedó. Una vez que se haya enviado todo, se quitan tanto el estado WRITE cómo WRITING, y se vuelve a suscribir la escritura, ajustando su controlador.

Una vez finalizado el saludo inicial (greeting), cuando el cliente envíe un comando, se activará la sesión. Se leerá y almacenará lo enviado por el cliente en un búfer, el cual se pasará al analizador sintáctico (parser). El parser no consumirá todo el contenido, sino que se detendrá únicamente al encontrar un comando. Si no logra encontrar una línea completa (finalizada por \r\n), se guardará hasta el punto donde haya leído y se detendrá temporalmente. Este proceso se repetirá hasta que logre identificar el final de la línea. Es importante destacar que en esta etapa, el estado READ seguirá siendo el dominante en la pila de estados.

Cuando el parser identifica un comando completo y el búfer de lectura queda vacío, este comando se almacenará en la sesión del cliente, y se añadirá el estado de proceso (PROCESS) a la pila de estados. Posteriormente, se invocará este estado, el cual ejecutará la función "state_machine_run". Esta función verifica el estado actual del cliente en la máquina de estados (los estados del cliente están descritos en el RFC 1939 [RFC1939]) y comprueba

si el comando enviado corresponde a dicho estado. En caso de ser un comando incorrecto, se enviará un mensaje de error. Sin embargo, si el comando es correcto, se procederá con su procesamiento. Al finalizar este procesamiento, se añadirá el estado de escritura (WRITE) a la pila de estados, se volverá a suscribir para escritura en el selector, y se finalizará este proceso.

Puede ocurrir que el parser procese una línea y una porción adicional de información. En este caso, para evitar que el sistema intente leer de inmediato del cliente después de procesar el comando, se introduce en la pila un estado adicional de lectura (READING). Este estado indica que aún hay datos pendientes en el búfer que deben completarse antes de poder retomar la lectura desde el cliente. El flujo de acción sigue un patrón similar al mencionado previamente, con la pila de estados conformada por READ, READING y PROCESS. Una vez que se completa el proceso de procesamiento, el estado PROCESS se elimina de la pila y se introduce WRITE para responder al cliente. Si la respuesta no puede enviarse completamente de una vez, se agrega el estado WRITING a la pila.

La complicación surge al sacar los estados WRITE/WRITING y si la lectura del búfer aún no ha finalizado (es decir, si quedan por procesar más comandos o líneas). En esta situación, no es viable suscribirse de nuevo a READ debido al riesgo de bloquear el proceso. Aquí es donde el estado READING se vuelve crucial. En la siguiente iteración, manteniendo al cliente suscrito para escritura en el selector, se invoca su controlador. Dado que el estado READING está en la cima de la pila, se efectúa una operación de lectura. En otras palabras, se accede al controlador de READ sin tener una suscripción activa para lectura en el selector. Esto conlleva a dos posibles escenarios: si al leer nuevamente del búfer de lectura no se completa otro comando, se elimina el estado READING y se mantiene READ en la cima de la pila. En este caso, se espera a que el usuario termine de enviar la información restante para completar el comando. Sin embargo, si se logra completar el comando, el flujo continúa su curso normal.

Por último, existe un último caso que difiere de lo anteriormente explicado. Todo lo que se ha mencionado hasta ahora se aplica a la mayoría de los comandos, pero ese no es el caso de los comandos como RETR y LIST, que son comandos de múltiples líneas. En estas instancias, si la información no se procesa por completo en una iteración, se agrega el estado PROCESSING a la pila, seguido por WRITE. Se activa el controlador de escritura y, al completarse la escritura, se elimina el estado WRITE o se agrega WRITING si la escritura no ha concluido. Es relevante subrayar que el cliente permanece suscrito para escritura en el selector. Así, cuando el controlador de escritura se reactiva, y detecta el estado PROCESSING en la cima de la pila, en lugar de proceder con la escritura, se enfoca en procesar la información y llama a continue_session. Este proceso de procesamiento y envío al cliente se repite hasta que no quede más información pendiente.

Para una comprensión más detallada de este proceso, se encuentra un diagrama específico en la sección 8 que puede ser de gran utilidad.

2. Implementación del protocolo pop3

Para este trabajo se ha desarrollado una implementación de servidor no bloqueante sin un mecanismo de transformación de mensajes. En esta sección, nos enfocaremos en presentar diversos escenarios de uso de este servidor.

Dentro del contexto de la materia, partimos del entendimiento de que los usuarios con los que nos conectamos al ejecutar el servidor ya existen previamente, y que el servidor dispone de correos electrónicos para estos usuarios. Por consiguiente, la expectativa de uso implica inicializar el servidor, establecer conexión con un usuario y poder aprovechar todas las funcionalidades ofrecidas por el servidor.

Tener en cuenta que a lo largo de este informe, donde se utilice la notación <palabra> indica que "palabra" debe estar presente y la notación <?palabra?> indica que "palabra" puede o no estar presente.

Terminal 1:

```
./pop3d -d <directorio de mail> <? -p <PORT> ?> <? -a <ADMIN>:<ADMINPASSWORD> ?> -u <USER>:<PASSWORD>
```

Terminal 2:

```
nc -c localhost <PORT>
```

Comandos disponibles en la implementación

A continuación se listan todos los comandos implementados en este trabajo.

```
USER <username>
PASS <password>
QUIT
CAPA
LIST <?mail number?>
DELE <mail number>
RETR <mail number>
NOOP
RSET
```

Detalles y ejemplos de utilización de los comandos en la implementación

En esta sección se detallara en específico la utilización de cada comando provisto. Cuando se habla de etapa de autorización, nos referimos a la etapa que finaliza luego de utilizar el comando PASS <password> y recibir una respuesta de inicio de sesión por parte del servidor. También es importante aclarar que todos los comandos son case insensitive.

USER <username>

- Argumentos:

  Nombre de usuario (requerido), el cual debe existir con anterioridad para poder ser autorizado.

- Restricciones:

  Puede ser utilizada únicamente en la etapa de autorización, una vez autorizado el usuario no puede volver a iniciar su sesión a menos que la finalice.

- Comentarios:

  Este comando se utiliza acompañado del comando PASS <password> los cuales en conjunto se utilizan para autorizar a un usuario para luego poder utilizar los demás comandos.
  Al no insertarse un usuario válido este comando no se encarga de dar una respuesta de error, pues elegimos preservar la privacidad de los usuarios y que el error se muestre a la hora de ingresar la contraseña.

- Respuestas posibles:
  +OK

  Ejemplos:

  U: USER user
  S: +OK
  —----------
  U: USER notuser
  S: +OK

PASS <password>

- Argumentos:

  Contraseña del usuario (requerido), el usuario debe existir con anterioridad para poder ser autorizado.

- Restricciones:

  Puede ser utilizada únicamente en la etapa de autorización, una vez autorizado el usuario no puede volver a iniciar su sesión a menos que la finalice.

- Comentarios:

  Este comando se utiliza acompañado del comando USER <username> los cuales en conjunto se utilizan para autorizar a un usuario para luego poder utilizar los demás comandos.
  Chequea que la contraseña del usuario elegido sea la correspondiente y se encarga de autorizar al usuario.

- Respuestas posibles:
  +OK Logged in.
  -ERR [AUTH] Authentication failed

  Ejemplos:

  U: USER user
  S: +OK
  U: PASS pass
  S: +OK Logged in.
  —----------
  U: USER user
  S: +OK
  U: PASS notpass
  S: -ERR [AUTH] Authentication failed

QUIT

- Argumentos:

  No recibe argumentos.

- Restricciones:

  Sin restricciones.

- Comentarios:

Se utiliza para finalizar la sesión.

- Respuestas posibles:
  +OK POP3 server signing off

Ejemplos:

```
U: USER user
S: +OK
U: PASS pass
S: +OK Logged in.
U: QUIT
S:+OK POP3 server signing off
—----------
U: USER notuser
S: +OK
U: QUIT
S:+OK POP3 server signing off
```

CAPA

- Argumentos:

  No recibe argumentos.

- Restricciones:

  Sin restricciones.

- Comentarios:

  Este comando te devuelve las cosas que puede hacer el servidor

- Respuestas posibles:
  OK

Ejemplos:

```
U: USER user
S: +OK
U: PASS pass
```

S: +OK Logged in.
U: CAPA
S: +OK
CAPA
PIPELINING
.

LIST <?mail number?>

- Argumentos:

  Recibe el número de mail (opcional).

- Restricciones:

  Puede ser utilizada únicamente en la etapa posterior a la autorización. Un usuario sin autorización no puede utilizar el comando LIST.

- Comentarios:

  Este comando se utiliza para listar los mails del usuario, en caso de recibir como argumento el número de mail, otorga únicamente la información del mismo. En caso contrario, muestra una lista de todos los mails mostrando
  el número de mail y el espacio que ocupa.

- Respuestas posibles:
  +OK <mails count> messages (<octets> octets)
  +OK <mail number> <tamaño de archivo>
  -ERR There's no message <mail number>.

  Ejemplos:

U: USER bruno
S: +OK
U: PASS pass
S: +OK Logged in.
U: LIST
S: +OK 3 messages (119 octets)
S: 1 13
S: 2 71
S: 3 35
—----------
U: USER user
S: +OK
U: PASS pass
S: +OK Logged in.
U: LIST 0
S: -ERR There's no message 0.
—----------
U: USER user
S: +OK
U: PASS pass
S: +OK Logged in.
U: LIST 1
S: +OK 1 71

DELE <mail number>

- Argumentos:

  Recibe el número de mail (requerido).

- Restricciones:

  Puede ser utilizada únicamente en la etapa posterior a la autorización. Un usuario sin autorización no puede utilizar el comando DELE.

- Comentarios:

  Este comando se utiliza para marcar como eliminado cierto mail, no se elimina directamente, debido a que se le da la posibilidad al usuario de utilizar el comando RSET para recuperar la información.
  El mail elegido se elimina al finalizar la sesión del usuario.

- Respuestas posibles:
  +OK Marked to be deleted.
  -ERR There's no message <mail number>
  Ejemplos:

```
U: USER bruno
S: +OK
U: PASS pass
S: +OK Logged in.
U: LIST
S: +OK 3 messages (119 octets)
S: 1 13
S: 2 71
S: 3 35
S: .
U: DELE 1
S: +OK Marked to be deleted.
—----------
U: USER bruno
S: +OK
U: PASS pass
S: +OK Logged in.
U: LIST
S: +OK 3 messages (119 octets)
S: 1 13
S: 2 71
S: 3 35
S: .
U: DELE 4
S: -ERR There's no message 4
```

RETR <mail number>

- Argumentos:

  Recibe el número de mail (requerido).

- Restricciones:

  Puede ser utilizada únicamente en la etapa posterior a la autorización. Un usuario sin autorización no puede utilizar el comando RETR.

- Comentarios:

  Este comando es utilizado para devolver el contenido de un mail en específico.

- Respuestas posibles:
  +OK
  -ERR There's no message <mail number>.

  Ejemplos:

```
U: USER bruno
S: +OK
U: PASS pass
S: +OK Logged in.
U: LIST
S: +OK 2 messages (106 octets)
S: 1 71
S: 2 35
U: RETR 2
S: +OK
S: Este es el contenido de un mail
S: .
—----------
U: USER bruno
S: +OK
U: PASS pass
S: +OK Logged in.
U: LIST
S: +OK 2 messages (106 octets)
S: 1 71
S: 2 35
U: RETR 3
S: -ERR There's no message 3
```

NOOP

- Argumentos:

  No recibe argumentos

- Restricciones:

  Puede ser utilizada únicamente en la etapa posterior a la autorización. Un usuario sin autorización no puede utilizar el comando NOOP.

- Comentarios:

  La operación cumple la función de no realizar ninguna operación.

- Respuestas posibles:
  +OK

  Ejemplo:

```
U: USER user
S: +OK
U: PASS pass
S: +OK Logged in.
U: NOOP
S: +OK
```

RSET

- Argumentos:

  No recibe argumentos.

- Restricciones:

  Puede ser utilizada únicamente en la etapa posterior a la autorización. Un usuario sin autorización no puede utilizar el comando RSET.

- Comentarios:

  Realiza un reseteo de los cambios realizados por el usuario en la sesión activa, volviendo a los datos originales al comienzo de la sesión.

- Respuestas posibles:
  +OK

Ejemplos:

```
U: USER user
S: +OK
U: PASS pass
S:+OK Logged in.
U: LIST
S: +OK 3 messages (119 octets)
S: 1 13
S: 2 71
S: 3 35
S: .
U: DELE 1
S: +OK Marked to be deleted.
U: RSET
S: +OK
```

3. Implementación de protocolo de usuario

Para poder utilizar el protocolo de usuario debe primero correrse el servidor con las especificaciones necesarias, y luego correr el comando deseado con las credenciales del administrador.

Terminal 1:

./pop3d -d <directorio de mail> <? -a <ADMIN>:<ADMINPASSWORD> ?> -u <USER>:<PASSWORD> <? -p <PORT> ?>

Terminal 2:

./pop3d-user -a <ADMIN>:<ADMINPASSWORD> <COMMAND> <? ARG1 <? ARG2?> ?>

A continuación se listaran los posibles comandos a utilizar en este protocolo:

```
current
history
bytes
password <user> <new password>
delete <user>
```

```
concurrent <max concurrent users>
```

current

- Argumentos:

  No recibe argumentos.

- Restricciones:

  No tiene restricciones.

- Comentarios:

  El comando current devuelve la cantidad de usuarios que están usando el servidor en ese momento.

- Respuestas posibles:

```
OK
Current users: <userCount> (cantidad de usuarios activos)
```

Ejemplos:

```
$ ./pop3d-user -a admin:pass current
Request:
user protocol
username: admin
password: pass
id: 1
command: current
.

OK
Current users: 0
```

history

- Argumentos:

  No recibe argumentos.

- Restricciones:

  No tiene restricciones.

- Comentarios:

  El comando history devuelve la cantidad histórica de usuarios.

- Respuestas posibles:

```
OK
Historic user count: <userCount> (cantidad histórica de usuarios)
```

Ejemplos:

```
$ ./pop3d-user -a admin:pass history
Request:
user protocol
username: admin
password: pass
id: 1
command: history
.

OK
Historic user count: 0
```

bytes

- Argumentos:

  No recibe argumentos.

- Restricciones:

  No tiene restricciones.

- Comentarios:

  El comando bytes devuelve la cantidad de bytes transferida entre quien usa el servidor y el servidor.

- Respuestas posibles:

```
OK
Transferred bytes: <bytes> (bytes transferidos)
```

Ejemplos:

```
$ ./pop3d-user -a admin:pass bytes
Request:
user protocol
username: admin
password: pass
id: 1
command: bytes
.

OK
Transferred bytes: 0
```

password

password <user> <new password>

- Argumentos:

Recibe como argumentos <user> (nombre de usuario) y <new password> (nueva contraseña)

- Restricciones:

  No tiene restricciones.

- Comentarios:

  El comando password se utiliza para cambiar la contraseña del usuario especificado.

- Respuestas posibles:

```
OK
Changed password for user <user> (Nombre de usuario) to
<new_password> (Nueva contraseña)
```

Ejemplos:

```
$ ./pop3d-user -a admin:pass password not_admin new_password
Request:
user protocol
username: admin
password: pass
id: 1
command: password
arg1: not_admin
arg2: new_password
.

OK
Changed password for user not_admin to new_password
```

delete <user>

- Argumentos:

  Recibe como argumento <user> (nombre de usuario).

- Restricciones:

  No tiene restricciones.

- Comentarios:

  El comando delete se utiliza para eliminar al usuario especificado.

- Respuestas posibles:

```
OK
Deleted user: <user> (nombre de usuario)
```

Ejemplos:

```
$ ./pop3d-user -a admin:pass delete not_admin
Request:
user protocol
username: admin
password: pass
id: 1
command: delete
arg1: not_admin
.

OK
Deleted user not_admin
```

concurrent <max concurrent users>

- Argumentos:

  Recibe como argumento <max concurrent user> (número máximo de usuarios que pueden utilizar el servidor a la vez).

- Restricciones:

  No tiene restricciones.

- Comentarios:

  El comando concurrent

- Respuestas posibles:

```
OK
Set max concurrent users to: <max concurrent users> (Máxima cantidad de usuarios)
```

Ejemplos:

```
$ ./pop3d-user -a admin:pass concurrent 123
Request:
user protocol
username: admin
password: pass
id: 1
command: password
arg1: 123
.

OK
Set max concurrent user to: 123
```

Estructura de los datagramas

Datagrama de cliente

```
user protocol
username: <username>
password: <password>
id: <id>
command: <command>
<?
arg1: <arg1>
?>
<?
arg2: <arg2>
?>
.
```

- La primera línea debe estar conformada por la frase “user protocol”, exactamente de ese modo.
- En la segunda línea debe decir “username: ” que debe estar seguido de <username> (nombre del usuario).
- En la tercera línea debe decir “password: ” que debe estar seguido de <password> (contraseña del usuario).
- En la cuarta línea debe decir “id: “ que debe estar seguido de <id> (id de la request)
- La quinta línea debe decir “command: “ que debe estar seguido de <command> (comando).
- La sexta y séptima líneas son opcionales en caso de recibir uno o dos argumentos.
- En caso de haber un argumento la sexta línea debe decir “arg1: “ seguido de <arg1> (primer argumento).
- Y en caso de haber 2 argumentos la sexta línea será igual al caso anterior, seguido de una séptima línea que debe decir “arg2: “ seguido de <arg2> (segundo argumento).
- Notar que el datagrama debe finalizar con “.\r\n”

Datagrama de respuesta del Servidor

```
user protocol
request_id: <request id>
status_code: <status code>
value: <value>
```

- La primera línea está conformada por la frase “user protocol”.
- La segunda línea dira “request_id: ” <request id> (id de la request).
- La tercera línea dirá “status_code: ” <status code> (código de status).

- Y por último la cuarta línea el value que dirá “value :” seguido de <value> (valor). Notar también que se finaliza con un “\n”.

Códigos de *status*

Código	Significado	Descripción	Mensaje
20	Éxito	La operación fue exitosa	OK
40	Error de usuario	Error por parte del usuario, podría significar el envío de argumento invalido.	User error
41	Error de autorizacion del usuario	No tiene permisos para ejecutar los comandos.	User error: unauthorized
42	Error usuario inexistente	No existe el usuario.	User error: user does not exist
50	Error del servidor	Ocurrió un error inesperado por parte del servidor	Server error

4. Guia de instalacion

Requerimientos previos:

- GCC
- Make

Para compilar el proyecto se debe correr el comando “make” o “make all” en el directorio del proyecto (TPE-PROTOS)

Una vez compilado genera dos ejecutables “./pop3d” y "./pop3d-user” que corresponden al servidor y a la aplicación del usuario respectivamente.

En caso de querer más detalles sobre cómo utilizar los ejecutables leer el README.md en la raíz del proyecto.

5. Problemáticas encontradas durante el desarrollo.

Para comenzar terminar el programa con una señal de resulta en un memory leak que no logramos manejar. Este es el único error que se pudo encontrar analizando el trabajo

Por otra parte, logramos descubrir bastante tarde que teníamos mal puesto un puerto, lo que nos llevó a muchas complicaciones.

Con respecto a la implementación del socket ipv6 hubo un problema que encontramos al compilar en pampero que logramos resolver.

Además la principal dificultad encontrada a la hora de desarrollar este trabajo fue la falta de tiempo y la cantidad de otros trabajos que tuvimos que entregar para estas fechas.

Y más allá de todas las dificultades encontradas logramos superar las adversidades y generamos un resultado acorde a nuestras expectativas, cumpliendo con todos los requisitos obligatorios.

6. Posibles extensiones

La principal posible extensión que se le podría aplicar al proyectos es una capa wrapper que recopile estadísticas del servidor.

Además una posible expansión relativamente sencilla sería vincular el sistema de usuarios a una base de datos no volátil. Esto abriría la puerta a implementar comandos que permitan a los usuarios configurar sus propias cuentas.

Otra área de extensión interesante sería el control de acceso para el administrador. Dentro de este ámbito, se podrían desarrollar diversas funcionalidades, como el bloqueo de IPs o de usuarios, la implementación de políticas de contraseñas más sólidas, la asignación de roles con diferentes niveles de permisos, entre otras opciones.

Una adición adicional que podría enriquecer la funcionalidad del servidor es la integración de SMTP. Esto habilitaría a los usuarios para enviar y recibir correos electrónicos a través de internet. Con esta ampliación, el servidor completaría todo el flujo de trabajo del cliente, desde la gestión de buzones hasta el transporte de los correos electrónicos.

7. Conclusiones

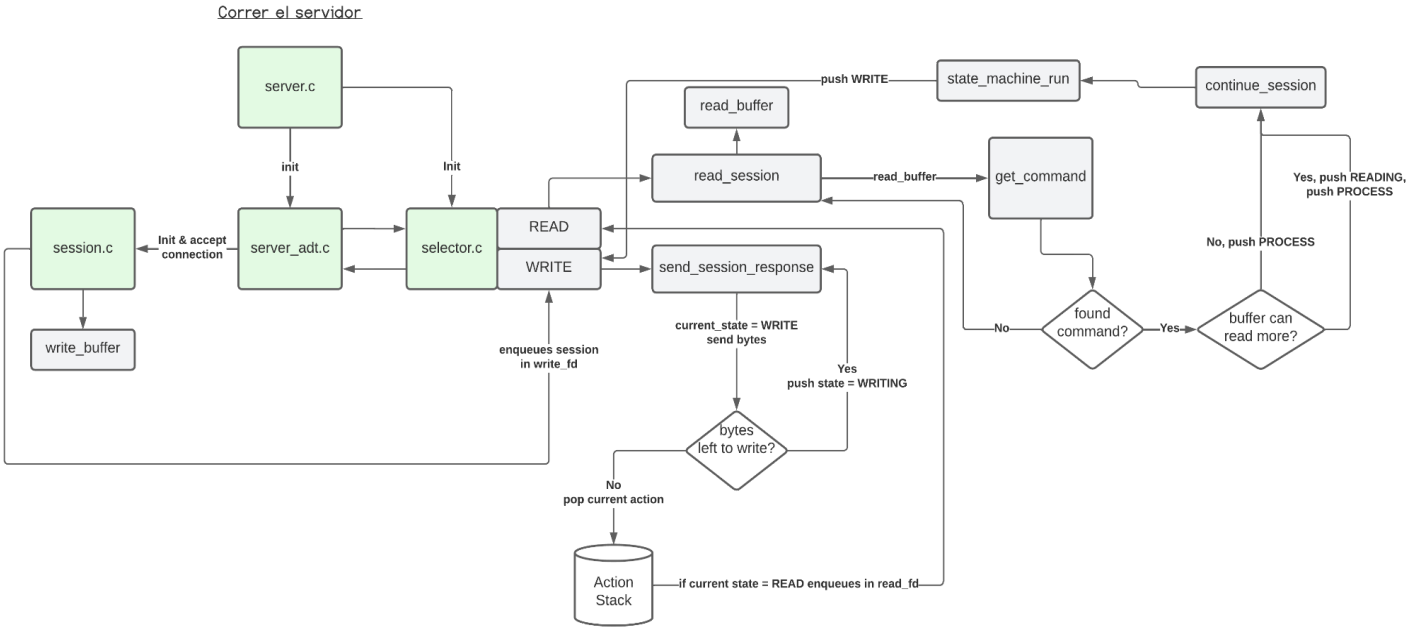
En resumen, este proyecto representó un desafío significativo que puso a prueba nuestras habilidades como desarrolladores. Implementar nuestro primer servidor y lograr su interacción exitosa con Mail User Agents resultó gratificante.

Consideramos que podríamos haber optimizado nuestra administración del tiempo, comenzando incluso pocos días antes de la entrega, así y todo logramos obtener un resultado acorde a lo esperado, aunque nos quedamos con ganas de incluir algunas funcionalidades específicas que podrían haber mejorado tanto la experiencia del cliente como la performance del servidor desarrollado.

Decidimos optar por una distribución un poco distinta a la que estamos acostumbrados, y decidimos utilizar, con el objetivo de optimizar el tiempo, mucho la técnica de pair programming o incluso programar al mismo tiempo desde la extensión *liveshare*.

Este proyecto nos brindó una valiosa lección sobre cómo abordar nuevas tecnologías y comprender el código ajeno, al tener que comenzar en base a los parches dados por la cátedra. Enfrentamos momentos de estancamiento y aprendimos a mantener la calma, colaborar entre nosotros para resolver dudas y obstáculos. Consideramos que este desafío culminó en uno de los mayores aprendizajes en nuestra carrera hasta ahora, proporcionándonos habilidades para enfrentar situaciones adversas con mayor confianza y eficacia.

8. Diagrama


Correr el servidor
server.c
init
Init
session.c
Init & accept connection
server_adt.c
selector.c
READ
WRITE
write_buffer
enqueues session in write_fd
push WRITE
state_machine_run
continue_session
read_buffer
read_session
read_buffer
get_command
Yes, push READING, push PROCESS
No, push PROCESS
No
found command?
Yes
buffer can read more?
current_state = WRITE send bytes
send_session_response
Yes push state = WRITING
bytes left to write?
No pop current action
Action Stack
if current state = READ enqueues in read_fd