# Special Project

## Implementation of a POP3 server, a management interface and a companion application

Subject:

(20242Q) 72.07 - Protocolos de Comunicación

Professors:

Pedro Martín Valerio Ramos
Hugo Javier Stupenengo Fause
Marcelo Fabio Garberoglio
Roberto Oscar Axt Roberto Oscar
Sebastian Kulesz
Thomas Mizrahi

Members:

Lonegro Gurfinkel, Lucas 63.738 - llonegrogurfinkel@itba.edu.ar
Noceda, Tobías Martín 63.422 - tnoceda@itba.edu.ar
Raiti, Tomás
Ghigliani, Franco

# 1. Index



# 2. Protocols and Applications Developed

The presented project implements a non-blocking TCP POP3 server along with a manager system protocol to control the server settings (MSMP) without needing to restart it, and analyze the system's statistics. This tool can be interfaced via net-cat or with the companion manager application developed. A more detailed description of this protocol is available Appendix 9.1, the protocol's RFC.

## 2.1. POP3 Server Module

### Purpose

The server acts as a mail access agent, enabling users to retrieve email from their inboxes stored on a mail server.

Supports:

- Concurrent client connections.
- Standard POP3 commands for authentication, mailbox management, and message retrieval.
- Dynamic server configuration and monitoring capabilities.

### Features

#### Concurrency:

- Handles at least 500 simultaneous client connections using a non-blocking I/O model for scalability.
- Supports both IPv4 and IPv6 connections.

#### Authentication:

- Implements the USER and PASS commands for username/password authentication.
- Fully compliant with the security and error-handling requirements of POP3.

#### Pipelining:

- Enables the use of multiple commands in a single transmission, reducing latency and improving performance (as per RFC 2449).

#### Email Processing:

- Integrates external programs for processing email content. For example:
- Anti-spam tools like SpamAssassin.
- Anti-malware tools like RenAttach.
- External programs communicate via standard input and output, ensuring modularity.

#### Metrics Collection:

- Tracks server operation through metrics such as:
    - Total historical connections.
    - Concurrent connections.
    - Total bytes transferred.

#### Access Logs:

- Maintains logs for each user session, allowing administrators to trace user activity if needed.

#### Dynamic Configuration:

- Supports runtime modifications without server restarts.

## 2.2. Configuration and Monitoring Client (MSMP)

### Purpose

A command-line client application is implemented to allow administrators to:

- Monitor server metrics in real-time.
- Modify server configuration dynamically.

#### Features

##### Configuration Commands:

- Change runtime parameters such as the maildir or the transformation program.
- Example: `SET maildir ./this/is/the/maildir`.

##### Monitoring Commands:

- Query metrics like maximum active connections in history, total connections, user session logs and bytes transferred.
- Example: `STAT`.

# 3. Difficulties encountered

## 3.1. Non-Blocking Requirement

One of the core requirements of this project is to achieve a non-blocking management of user requests and responses in TCP sessions. Throughout the project, this requirement elicited behaviour and models that triggered unwanted errors; clobbering of session information, . To solve this, a custom poll loop was created that handles both passive and active sockets, as well as triggers events for each of them.

# 4. Stress-tests

The server was capable of withholding over a thousand connections concurrently. No degradation was noticed on the server performance, with the exception of a DDoS protection system automatically rejecting the requests when the server was tested on Pampero. We also managed to have multiple MUAs clients connected on the server and it was able to withstand it.

# 5. Limitations

## 5.1. Connection Limit

The server's implementation assigns a single process to the supervision of all read and write operations relating to sockets. While this reduces overheads produced by the creation of multiple processes, it incurs an increasing burden as the number of connections - and, thus, file descriptors - managed by this process increases. As a result, the number of active connections allowed by the server at one time has been limited to 500.

## 5.2. Logging

As serializable logs are not a requirement of the project, logging was designed to take place in a memory allocated Abstract-Data-Type. As a result, the structure that stores and manages these logs is stored in a single process' heap and is only accessible to this process. This limitation required limiting read and write access of the logging structure to the main process, increasing complexity.

# 6. Possible extensions

## 6.1. Multiprocessing, Multithreading

Management of multiple file descriptors, particularly in a task that is I/O bound, greatly benefits from the use of several processes and threads. Possible approaches to these optimizations include offloading the management of connections to a child process when a certain number is reached, and managing read and write operations in separate threads, using signals to detect completion.

## 6.2. Logging Serialization

Serializing server logs would help resolve limitations related to the management of multi-process interactions with the logging structure, as logs could be regularly serialized, allowing other processes to read log files without requiring the use of shared memories or bespoke Inter-Process Communication mechanisms. More importantly, errors that cause server crashes could be examined via logs persisted in files and logging could be maintained across several lifetimes of the server module.

# 7. Conclusions

The tools and techniques learnt in class were applied to follow and implement an existing protocol, as well as design and implement another protocol in a relatively short period of time. The latter is particularly notable, as it highlights the focus on simplicity and conciseness that many communication protocols achieve. Though some difficulties were encountered in the duration of this project, they were both overcome and leveraged into proposals for future extensions.

# 8. Installation guide

Requirements:

- make

- gcc

Run *make* in a shell located in the project route. To initialize the POP3 server run the *server* executable created in the *./dist* directory. To initialize the manager companion application run the *manager* executable created in the *./dist* directory. Refer to appendices 9.2 and 9.3 for information on configuring flags for initialization.

# 9. Appendix

## 9.1. RFC for the Server Management Interface

November 26, 2024

RFC for Mail Server Management Protocol

Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Table of Contents



1. Introduction

This document describes a text-based connection oriented protocol for setting configuration values and retrieving statistics from a POP3 server. The protocol allows clients to set and get configuration values and obtain server statistics.

2. Generalizations

Initialy, the server host SHOULD start the MSMP server by listening on TCP port 4321.

All server responses MUST start with:

+OK

in case of possitive results. Or:

-ERR

in case of negative results.

Messages providing more information MAY be appended to the end of the response.

All responses MUST NOT exceed 512 characters in total and MUST be terminated with CRLF.





3. Start of connection

On connection to the server, the server MUST send an initial positive message indicating the server's version.

In response the client MUST authenticate themselves in the following 2 steps:

Firstly, the client sends this message:

USER <username>

To which the server will always answer positively. And then the client sends:

PASS <password>

If the user and password are correct, the server MUST send a positive response.
If the user does not exist or the password is not correct, the server SHOULD send a negative response.

Note: The server SHOULD NOT send different responses to when the user does not exists and when the password is not correct. This may be a security vulnerability.

Possible responses:
+OK Logged in

-ERR Invalid credentials

Example:
S: +OK MSMP1 server ready
C: USER user1
S: +OK
C: PASS pass1
S: -ERR Invalid credentials
C: USER user1
S: +OK
C: PASS pass2
S: +OK Logged in

4. ADD

In order to add or update a user, the client MUST send the following command:

ADD <username> <password>

If the user with the given username already exists, the password SHOULD be updated.
Otherwise, if the user does not exist and there is space for a new user, the server MUST create the new user and its maildir and send a positive response.
If the user does not exist and there is no more space for users, the server SHOULD send a negative response.



Note: usernames SHOULD be limited to 40 characters, as it is the limit set by POP3.

Possible responses:
+OK Added user
+OK Updated password
-ERR Unsafe username
-ERR User limit reached

Example:
C: ADD user1 pass1
S: +OK Updated password
C: ADD user2 pass2
S: +OK Added user
C: ADD 41_character_long_username_____ pass1
S: -ERR Unsafe username
C: ADD user3 pass3

S: -ERR User limit reached

5. DELE

In order to delete a user, the client MUST send the following command:

DELE <username>

If the user with the given username exists, the server MUST send a positive response.
Otherwise, if the user does not exist or the user is currently logged in, the server SHOULD send a negative response.

Possible responses:
+OK Deleted user
-ERR User does not exist
-ERR User is currently logged in

Example:
C: DELE user3
S: -ERR user does not exist
C: DELE user1
S: -ERR User is currently logged in
C: DELE user2
S: +OK Deleted user



6. LIST

To obtain a list of all users, the client MUST send the following command:

LIST

If there are no users, the server MAY send a negative response.

Possible responses:
+OK
<username 1>
<username 2>
...

Example:
C: LIST
S: +OK
S: user1

S: user1

The client MAY add a parameter in order to obtain a specific user:

LIST <username>

The server MAY append additional information to the response.
If the specified user does not exist, the server SHOULD send a
negative response.

Possible responses:
+OK
<username> <log count>
-ERR User does not exist

Example:
C: LIST user1
S: +OK
S: user1 69 logs



7. SET

To set a configuration value the client MUST send the following command:

SET <key> <value>

Valid keys MAY be:
- maildir
- transformer

Keys MUST be limited to 40 characters case sensitive.
Values MUST be limited to 100 characters and MUST NOT contain spaces.

If the key is valid, the server MUST send a positive response.
If the key is invalid, the server MUST send a negative response.

Possible responses:
+OK Changed value
-ERR Invalid key

Example:
C: SET this_is_an_invalid_key ./maildir
S: -ERR Invalid key
C: SET maildir ./maildir
S: +OK Changed value

C: SET transformer wc
S: +OK Changed value

8. GET

To retrieve a configuration value, the client MUST send the following command:

GET <key>

Valid keys MAY be:
- maildir
- transformer

If the key is valid, the server MUST send a positive response.
If the key is invalid, the server MUST send a negative response.

Possible responses:
+OK <value>
-ERR Invalid key

Example:
C: GET this_is_an_invalid_key
S: -ERR Invalid key
C: GET maildir
S: +OK ./maildir
C: GET transformer
S: +OK wc





9. STAT

To retrieve server statistics, the client MUST send the following command:

STAT

The statistics MAY include:
- Total number of historical connections
- Maximum number of simultaneous connections
- Total number of bytes transferred

Possible responses:
+OK
Total connections: <connection count>
Historical maximum traffic: <max traffic>

Total transferred bytes: <total bytes>

Example:
C: STAT
S: +OK
S: Total connections: 156
S: Historical maximum traffic: 201
S: Total transferred bytes: 640028

10. LOGS

To retrieve a list of access logs for a user, the client MUST send the following command:

LOGS <username>

If a user with the given username does not exist, the server SHOULD send a negative resopnse.

Possible responses:
+OK
<log entries>
-ERR Invalid user

Example:
C: LOGS user123
S: +OK
S: 2023-10-01 12:00:00 IP: 192.168.1.1
S: 2023-10-02 14:30:00 IP: 192.168.1.2

11. Conclusion

This protocol provides a simple and effective way to set configuration values, retrieve configuration values, and obtain server statistics.

Mail Server Management Protocol [Page 6]

## 9.2. Execution Flags for the POP3 Server

| **Flag** | **Description** |
|---|---|
| -h | Prints the help menu and terminates |
| -l <POP3 addr> | Establishes the address to serve the POP3 server. Listens on all interfaces by defect. |
| -L <conf addr> | Establishes the address to serve the management service. By default listens on the loopback address. |

| | |
|---|---|
| -p <POP3 port> | Incoming port for POP3 connections. By default is port 110. |
| -P <conf port> | Incoming port for management connections. By default is port 4321. |
| -u <name>:<pass> | List of users and passwords recognized by the server. Maximum value is 10. |
| -a <name>:<pass> | List of admin users and passwords. Maximum value is 4. |
| -t <cmd> | Sets a filter/transformer program for output. Defect filter program is cat. |
| -d <dir> | Specifies the directory for Maildirs. By defect is./dist/mail |
| -v | Prints version information and terminates. |

## 9.3. Execution Flags for the Companion Management Application

| Flag | Description |
|---|---|
| -P | Sets the port of the configuration server. |
| -L | Sets the address to the configuration server |