

TW-Mailer – Basic Version

Work in groups of 2 and write a socket-based client-server application in C/C++ to send and receive messages like an internal mail-server.

The client connects to the server and communicates through a stream socket connection in a proprietary plain-text format delimited by new-line or "dot + new-line" (see detail-description).

Usage Client:

```
./twmailer-client <ip> <port>
```

IP and port define the address of the server application.

Usage Server:

```
./twmailer-server <port> <mail-spool-directoryname>
```

The port describes the listening port of the server; the mail-spool-directory the folder to persist the incoming and outgoing messages and meta-data. The server is here designed as iterative server (no simultaneous request handling). Hereby the server responds to the following commands:

- SEND: client sends a message to the server.
- LIST: lists all messages of a specific user.
- READ: display a specific message of a specific user.
- DEL: removes a specific message.
- QUIT: logout the client.

Hints

- Sent messages for each user should be persisted in the mail-spool-directory.
- There are no restrictions about the design / structure of the message store.
 - e.g.: 1 user with 1 file storing all related messages.
 - e.g.: 1 user with 1 subfolder storing each message in 1 file inside the folder.
 - ...
- Consider to use “readline()”-function (see: tcpip_linux-prog-details.pdf)
- We define a username for the sender and receiver as a max. 8 chars (a-z, 0-9)
 - e.g.: kienboec, nimm, bergerw, if12b345, if98a765
- Code quality and compliance to the principles of C-programming is part of the grading.
- Keep an eye on
 - memory management (memory leaks, buffer overflows),
 - child-process management (zombie-process),
 - input validation and
 - error handling (with meaningful error messages)
- Comment, structure and indent your code properly.

Protocol specification

SEND

```
SEND\n<Sender>\n<Receiver>\n<Subject (max. 80 chars)>\n<message (multi-line; no length restrictions)>\n.\n
```

- The final dot ends the command.
- The server always responds with either "OK\n" or "ERR\n".

LIST

```
LIST\n<Username>\n
```

- The server responds with

```
<count of messages of the user (0 if no message or user unknown)>\n<subject 1>\n<subject 2>\n...<subject N>\n
```

READ

```
READ\n<Username>\n<Message-Number>\n
```

- The server responds with

```
OK\n<complete message content (as defined in SEND)>\n
```

- or

```
ERR\n
```

DEL

```
DEL\n<Username>\n<Message-Number>\n
```

- The server responds with

```
OK\n
```

- or

```
ERR\n
```

QUIT

QUIT\n

- The server does not respond.

Deliverables

- Hand-in
 - The commented code for the client and the server code
 - Makefile for the targets “all” and “clean”
 - Executables
 - Description of (1 page; pdf)
 - the client and server architecture,
 - used technologies
 - development strategy and needed adoptions

The first part of the project will be graded in a moodle workshop, where students feedback other students in a peer-review format.

Consider:

- everyone in the group hands in (including an unfinished state)
- everyone is assigned 3 submissions for a review
- give sufficient written feedback (spot checks by lecturers)
- don't be too strict and try to help

Marking System (5 Points)

- Your grading consists of your submission (received feedback) and your feedback given
- You give between 0 and 4 points for each review, detailed instructions are available in the Moodle workshop
- A weighted sum of all your reviews given and received will be calculated by Moodle after the workshop finished and you will receive max 5 points for this activity.