Redes de Computadores

1st Semester, P2, 2021/2022
LEIC Alameda

Programming using the *Sockets* interface

"RC Centralized Messaging"

1. Introduction

The goal of this project is to develop a prototype of a centralized messaging service.

The development of the project requires implementing a *Directory Server* (*DS*) and a *User Application* (*User*). The *DS* server and the various *User* application instances are intended to operate on different machines connected to the Internet.
The *DS* will be running on a machine with known IP address and ports.

The interface with the *User* application uses the keyboard to provide a set of commands that control the actions to take:

- User registration management. Each user is identified by a user ID `UID`, the 5-digit IST student number, and a password `pass`, consisting of 8 alphanumerical characters, restricted to letters and numbers.
- User access management. After a successful login, using an existing `UID`, the user can interact with the directory server, to manage group membership and view or post messages to groups.
- Group membership management. The user can ask to be subscribed or unsubscribed from groups, create new groups, and select the group that becomes the active group. Each group is characterized by a group ID `GID`, a 2-digit number (01 – 99), initialized with value `01` and sequentially increased by the *DS* server when creating new groups, and a group name `GName`, limited to a total of 24 alphanumerical characters (plus '-', and '_').
- Message handling. The user can select to post a message to the active `GID` group, or to view the messages posted to that group.
  Each message within each group is identified by a message ID `MID`, a 4-digit number initialized with value `0001` and sequentially increased by the *DS* server when accepting new messages for the group.
  Messages are composed by a text and optionally also a file. A text message `text` contains `Tsize` characters, with `Tsize` limited to a maximum of 240 characters.
  If a file is included in the message, it is characterized by the following information:
  - a filename `Fname`, limited to a maximum of 24 alphanumerical characters (and '-', '_' and '.'), including the separating dot and the 3-letter extension. Example: "*nnn…nnnn.xxx*";
  - the file size `Fsize`, in bytes. The `Fsize` field can have at most 10 digits.
  - the contents of the selected file (`data`).

The goal of the application is to provide a messaging platform in which users can subscribe to the groups of their interest, and post messages to those groups, as well as to read the messages that have been published in those groups.

For the implementation, the application layer protocols operate according to the client-server paradigm, using the transport layer services made available by the socket interface. The communication protocols supporting this application are specified in the following.

2. Project Specification

2.1 User Application (*User*)

The program implementing the user application (*User*) should be invoked using:

`./user [-n DSIP] [-p DSport]`,

where:

`DSIP` this is the IP address of the machine where the directory server (*DS*) runs. This is an optional argument. If this argument is omitted, the *DS* should be running on the same machine.

`DSport` this is the well-known port (TCP and UDP) where the *DS* server accepts requests. This is an optional argument. If omitted, it assumes the value 58000+GN, where GN is the group number.

Once the *User* program is running, it can perform different types of actions: (i) registration of a new user ID `UID`; (ii) login to interact with the directory server using an existing `UID`; (iii) management of the user's group membership and selection of the active group; and (iv) viewing or posting messages to groups.

The commands related to the **registration** of a new user ID are:

- ***reg*** `UID pass` – following this command the *User* application sends a message to the *DS* server, using the UDP protocol, asking to register a new user, sending its identification `UID` and a selected password `pass`.
  The result of the *DS* registration request should be displayed.
- ***unregister*** `UID pass` or ***unr*** `UID pass` – the *User* application sends a message to the *DS* server, using the UDP protocol, asking to unregister the user with identification `UID` and password `pass`. The *DS* server should unsubscribe this user from all groups in which it was subscribed. The result of the unregister request should be displayed.

The commands related to **user identification** and **session termination** are:

- ***login*** `UID pass` – with this command the *User* application sends a message in UDP to the *DS* to validate the user credentials: `UID` and `pass`. The result of the *DS* validation should be displayed to the user.
  The *User* application memorizes the `UID` and `pass` in usage.
- ***logout*** – the *User* application (locally) forgets the credentials of the previously logged in user. A new login command, with different credentials, can then be issued.
- ***showuid*** or ***su*** – following this command the *User* application locally displays the `UID` of the user that is logged in.

- ***exit*** – the *User* application terminates, after making that all TCP connections are closed.

The commands related to **group management** are listed below.

- ***groups*** or ***gl*** – following this command the *User* application sends the *DS* server a message in UDP asking for the list of available groups. The reply should be displayed as a list of group IDs (*GID*) and names (*GName*).

These following group management commands can only be issued after a user has logged in:

- ***subscribe*** *GID GName* or ***s*** *GID GName* – following this command the *User* application sends the *DS* server a message in UDP, including the user's *UID*, asking to subscribe the desired group, identified by its *GID* and *GName*. If *GID = 0* this corresponds to a request to create and subscribe to a new group named *GName*. The confirmation of successful subscription (or not) should be displayed.
- ***unsubscribe*** *GID* or ***u*** *GID* – following this command the *User* application sends the *DS* server a message in UDP, including the user's *UID*, asking to unsubscribe group *GID*. The confirmation of success (or not) should be displayed.
- ***my_groups*** or ***mgl*** – following this command the *User* application sends the *DS* server a message in UDP, including the user's *UID*, asking the list of groups to which this user has already subscribed. The reply should be displayed as a list of group IDs and names.
- ***select*** *GID* or ***sag*** *GID* – following this command the *User* application locally memorizes *GID* as the ID of the active group. Subsequent ***ulist***, ***post*** and ***retrieve*** messaging commands refer to this *GID*.
- ***showgid*** or ***sg*** – following this command the *User* application locally displays the *GID* of the selected group.
- ***ulist*** or ***ul*** – following this command the *User* application sends the *DS* server a message in TCP asking for the list of user *UID*s that are subscribed to the currently subscribed group *GID*.

The commands related to **messaging** are listed below. These commands can only be issued after a user has logged in and an active group *GID* has been selected.

- ***post*** *"text" [Fname]* – following this command the *User* establishes a TCP session with the *DS* server and sends a message containing text (between " "), and possibly also a file with name *Fname*.
  The confirmation of success (or not) should be displayed, including the posted message's ID *MID*. The TCP connection is then closed.
- ***retrieve*** *MID* or ***r*** *MID* – following this command the *User* establishes a TCP session with the *DS* server and sends a message asking to receive up to 20 messages, starting with the one with identifier *MID*, for the active group *GID*. The *DS* server only sends messages that include at least an author *UID* and text – any incomplete messages are omitted.
  After receiving the messages, the *User* application sends the DS a confirmation and then closes the TCP session. The reply should be displayed as a numbered list of text messages and, if available, the associated filenames and respective sizes.

Only one messaging command can be issued at a given time.
The result of each interaction with the *DS* should be displayed to the user.

2.3 *Directory Server* (*DS*)

The program implementing the Directory Server (*DS*) is invoked with the command:

`./DS [-p DSport] [-v],`

where:

DSport is the well-known port where the *DS* server accepts requests, both in UDP and TCP. This is an optional argument. If omitted, it assumes the value 58000+GN, where GN is the number of the group.

The *DS* makes available two server applications, both with well-known port *DSport*, one in UDP, to answer configuration messages, and the other in TCP, to answer messaging requests, both originating in the *User* application.
If the *-v* option is set when invoking the program, it operates in *verbose* mode, meaning that the *DS* server outputs to the screen a short description of the received requests (*UID*, *GID*) and the IP and port originating those requests.
Each received request should start being processed once it is received.

3. Communication Protocols Specification

The control and messaging protocols to be implemented are described in this section.
For both communication protocols *UID* is always sent using 5 digits, *GID* with 2 digits and *MID* with 4 digits.

3.1 *User–DS* Control Protocol (in UDP)

The interaction between the user application (*User*) and the directory server (*DS*) for user and group management operations is supported by the UDP protocol.
The request and reply protocol messages to consider are:

a) REG *UID pass*
Following the *reg* command, the *User* application sends the user ID *UID* and the selected password *pass* for registration at the *DS* server.

b) RRG *status*
In reply to a REG request the *DS* server sends a status message. If the REG request was successful (valid *UID* and *pass*) the *status* is OK; if the *UID* was already registered the *status* is DUP; if the registration is not accepted for some other reason (e.g. too many users already registered) the *status* is NOK.

c) UNR *UID pass*
Following the *unregister* command, the *User* application asks the *DS* to unregister the user with *UID* and whose password is *pass*.

d) RUN *status*
In reply to a UNR request the *DS* server sends a status message. If the UNR request was successful (valid *UID* and *pass*) the *status* is OK; if the unregister request is not accepted (e.g., invalid *UID*, incorrect *pass*) the *status* is NOK.

e) LOG *UID pass*
Following the login command, the *User* application sends the *DS* server a message with the user's *UID* and password *pass* for validation.

f) RLO *status*
In reply to a LOG request the *DS* server sends a status message. If the *UID* and *pass* are valid the *status* is OK; otherwise the *status* is NOK.

g) OUT *UID pass*
Following the logout command, the *User* application sends the *DS* server a message with the user's *UID* and password *pass* for validation.

h) ROU *status*
In reply to a OUT request the *DS* server sends a status message. If the *UID* and *pass* are valid the *status* is OK; otherwise the *status* is NOK.

i) GLS
Following the *groups* command, the *User* application sends the *DS* a request asking for the list of existing groups.

j) RGL *N[GID GName MID]**
In reply to a GLS request the *DS* server sends the list of *N* available groups, including for each one the group identifier *GID*, the group name *GName* and the number *MID* of the last message available for that group, or *0000* if no messages are available.
In case no groups are available the reply is RGL 0.

k) GSR *UID GID GName*
Following the *subscribe* command, the *User* application sends the *DS* a request asking for user *UID* to join the group with *GID* and *GName*. If *GID = 00* this corresponds to a request to create a new group with name *GName*.

l) RGS *status*
In reply to a GSR request the *DS* server sends a status message. If the *UID*, *GID* and *GName* are valid the *status* is OK; if a new group was created (and subscribed to) the *status* is NEW *GID*; if the *UID* is invalid the *status* is E_USR; if the *GID* is invalid the *status* is E_GRP; if the *Gname* is invalid the *status* is E_GNAME; if a new group could not be created (already 99 groups exist) the *status* is E_FULL; otherwise the *status* is NOK.

m) GUR *UID GID*
Following the *unsubscribe* command, the *User* application sends the *DS* a request asking for user *UID* to unsubscribe the group *GID*.

n) RGU *status*
In reply to a GUR request the *DS* server sends a status message. If the *UID* and *GID* are valid the *status* is OK; if the *UID* is invalid the *status* is E_USR; if the *GID* is invalid the *status* is E_GRP; otherwise the *status* is NOK.

o) GLM UID
Following the *my_groups* command, the *User* application sends the *DS* a request asking for the list of groups to which user *UID* has already subscribed.

p) RGM *N[GID GName MID]**
In reply to a GLM request the *DS* server sends the list of *N* groups the user *UID* has subscribed, including for each one the group identifier *GID*, the group name *GName* and the number *MID* of the last message available for that group, or *0000* if no messages are available.
In case no groups have been subscribed by *UID* the reply is RGM 0. If the *UID* is invalid or there is no login for that user *N* is replaced by the string E_USR.

If an unexpected protocol message is received, the reply is ERR.
In the above messages the separation between any two items consists of a single space.
Each request or reply message ends with the character "\n".

3.2 *User–DS* Messaging Protocol (in TCP)

The interaction between the user application (*User*) and the directory server (*DS*) for messaging operations is supported by the TCP protocol.
The request and reply protocol messages to consider are:

a) ULS *GID*
Following the *ulist* command, the *User* application opens a TCP connection with the *DS* server and asks for the list of users subscribed to group *GID*.

b) RUL *status [GName [UID]*]*
In reply to a ULS request the *DS* server sends a status message that will also contain the list of users subscribed to the selected group, if any.
If the RUL request was successful the *status* is OK, followed by the group name *Gname*. The *status* is *NOK* if the group does not exist.
The list of *UID*s is sent by the *DS* server and displayed by the *User* application.
After receiving the reply message, the *User* application closes the TCP connection with the *DS*.

c) `PST UID GID Tsize text [Fname Fsize data]`
Following the *post* command, the *User* application opens a TCP connection with the *DS* server to send a text message and optionally also a file.
The text message *text* contains *Tsize* characters.
If a file is being posted the following information is sent:

- the filename *Fname*;
- the file size *Fsize*, in bytes;
- the contents of the selected file (*data*).

d) `RPT status`
In reply to a PST request the *DS* server sends a status message. If the PST request was successful the *status* is the number of the message MID, otherwise the *status* is NOK.
The *post* success (or not) is displayed by the *User* application.
After receiving the reply message, the *User* application closes the TCP connection with the *DS*.

e) `RTV UID GID MID`
Following the *retrieve* command, the *User* application opens a TCP connection with the *DS* server to receive up to 20 messages, starting from te message with the identifier *MID*, of group *GID*.

f) `RRT status [N[ MID UID Tsize text[ / Fname Fsize data]]*]`
In reply to a RTV request the *DS* server sends a status message that will also contain the messages being sent, if any.
If the RTV request was successful the *status* is OK. The *status* is *EOF* if there are no messages available (which would correspond to *N = 0*), and the *status* is *NOK* if there is any other problem with the RTV request.
If there are messages to send (*N > 0*), the *DS* server reply includes the following components per each message:

- the ID of the message *MID*;
- the ID of the user who posted the message *UID*;
- the text message size *Tsize*, in bytes;
- the text message contents *text*;
- if a file is associated with this message the following information is also sent:
  - the filename *Fname*;
  - the file size *Fsize*, in bytes;
  - the contents of the selected file (*data*).

The text messages and the name of any associated files received (which are locally stored) are displayed by the *User* application.
After receiving the reply message, the *User* application closes the TCP connection with the *DS*.

If an unexpected protocol message is received, the reply will be ERR.
In the above messages the separation between any two items consists of a single space.
Each request or reply message ends with the character "\n".

4. Development

4.1 Development and test environment

Make sure your code compiles and executes correctly in the development environment available in the lab *LT5*.

4.2 Programming

The operation of your program, developed in *C* or *C++*, may need to use the following set of system calls:

- Reading user information into the application: `fgets()`;
- Manipulation of strings: `sscanf()`, `sprintf()`;
- UDP client management: `socket()`, `close()`;
- UDP server management: `socket()`, `bind()`, `close()`;
- UDP communication: `sendto()`, `recvfrom()`;
- TCP client management: `socket()`, `connect()`, `close()`;
- TCP server management: `socket()`, `bind()`, `listen()`, `accept()`, `close()`;
- TCP communication: `write()`, `read()`;
- Multiple inputs multiplexing: `select()`.

4.3 Implementation notes

Developed code should be adequately structured and commented.
The `read()` and `write()` system calls may read and write, respectively, a smaller number of bytes than solicited – you need to ensure that your implementation still works correctly.
Both the client and server processes should terminate gracefully at least in the following failure situations:

- wrong protocol messages received from the corresponding peer entity;
- error conditions from the system calls.

5 Bibliography

- W. Richard Stevens, Unix Network Programming: Networking APIs: Sockets and XTI (Volume 1), 2nd edition, Prentice-Hall PTR, 1998, ISBN 0-13-490012-X, chap. 5.
- D. E. Comer, Computer Networks and Internets, 2nd edition, Prentice Hall, Inc, 1999, ISBN 0-13-084222-2, chap. 24.
- Michael J. Donahoo, Kenneth L. Calvert, TCP/IP Sockets in C: Practical Guide for Programmers, Morgan Kaufmann, ISBN 1558608265, 2000
- On-line manual, `man` command
- Code Complete - http://www.cc2e.com/
- http://developerweb.net/viewforum.php?id=70

6 Project Submission

6.1 Code

The project submission should include the source code of the programs implementing the *User* and the *DS server*, as well as the corresponding *Makefile*.
The makefile should compile the code and place the executables in the current directory.

6.2 Auxiliary Files

Together with the project submission you should also include any auxiliary files needed for the project operation together with a *readme.txt* file.

6.3 Submission

The project submission is done by e-mail to the lab teacher, **no later than January 7, 2022, at 23:59 PM**.
You should create a single `zip` archive containing all the source code, makefile and all auxiliary files required for executing the project. The archive should be prepared to be opened to the current directory and compiled with the command `make`.
The name of the archive should follow the format: **`proj_"group number".zip`**

7 Open Issues

You are encouraged to think about how to extend this protocol in order to make it more generic.
For instance how would the communications protocol need to be changed to allow the creation of private groups? Or to allow moderated posting to a group? How to delete a group? Or how could an alert system be implemented to informs the user which groups have unread messages?