TÉCNICO LISBOA

Redes de Computadores

1st Semestre, P2, 2025/2026
LEIC Alameda

Programming using the *Sockets* interface

"Event Reservation"

Date: 16 NOV 2025

1. Introduction

The goal of this project is to implement an event reservation platform. Users can create an event, cancel and close it, as well as list ongoing events and make reservations.

The development of the project requires implementing an *Event-reservation Server* (*ES*) and a *User Application* (*User*). The *ES* and multiple *User* application instances are intended to operate simultaneously on different machines connected to the Internet.
The *ES* will be running on a machine with known IP address and port.

The *User* can be used for two purposes: i) event management (create, close, list own events); or ii) event reservation (list existing events, reserve seats, list own reservations). Any registered user can do both activities.

The *User* application interface allows controlling the actions to take, via keyboard commands:

- Login (*login*). Each user is identified by a *user ID*, `UID`, a 6-digit IST student number, and a *password* composed of 8 alphanumeric (only letters and digits) characters. When the *ES* receives a login request it will inform of a successful or incorrect login attempt or, in case the `UID` was not yet present at the *ES*, it registers a new user.
- Create a new Event (*create*). The *User* application sends a message to the *ES* asking to create a new event, providing a short event name (represented with up to 10 alphanumerical characters), a file (an image or other type of file) describing the event, the event date and time (represented as dd-mm-yyyy hh:mm), and the number of available seats/attendees (represented with number, from a minimum of 10 to a maximum of 999). In reply, the *ES* informs if the request was successful, and the assigned event identifier, `EID`, a 3-digit number.
- Close an event (*close*). The *User* application sends a message to the *ES* asking to close an ongoing event owned by the logged in user, meaning that no more reservations are accepted. The *ES* will reply informing whether the event was successfully closed, the event date already passed, or if the event is already sold-out.
- List events started by this user (*myevents*). The *User* application asks the *ES* for a list and booking status of the events created by the logged in user. The *ES* will reply with the requested list or inform the user that there are no active events for this user.
- List all events (*list*). The *User* application asks the *ES* for a list of all events. The *ES* will reply with the requested list, or an indication that no active events are available.
- Show event (*show*). The *user* asks the *ES* about the description of a selected event, and its reservation status. The *ES* will reply with information about the event, including its name, date and time, the number of available seats, and the number of reservations done, as well as the file describing the event.

- Reserve an event (*reserve*). The *User* application asks the *ES* to make a reservation for *n* people to attend a selected event. The *ES* will reply reporting the result of the reservation: accepted or refused (if `n` is larger than remaining number of available seats) or informing that the event is no longer accepting reservations (event was previously closed by the owner), or an error if the event ID does not exist.
- List reservations made by this user (*myreservations*). The *User* application asks the *ES* for a list of the events in which they have reservations. The *ES* will reply with the requested list or inform the user that there is no reservation for active events.
- Change Password (*changePass*). The user can ask the *ES* server to change the password of the logged in *user*. In the reply, the *ES* reports if the request was successful or not.
- Unregister (*unregister*). The user can ask the *ES* server to unregister this *user*.
- Logout (*logout*). The user can ask the *ES* server to terminate the interaction.
- Exit (*exit*). The user can ask to exit the *User* application. If a user is still logged in, then the *User* application informs the user that the logout command should be executed first.

The implementation uses the application layer protocols operating according to the client-server paradigm, using the transport layer services made available by the socket interface. The applications to develop and the supporting communication protocols are specified in the following.

2. Project Specification

2.1 User Application (*User*)

The program implementing the *users* of the Event Reservation platform (*User*) is invoked using:

`./user [-n ESIP] [-p ESport]`,

where:

ESIP is the IP address of the machine where the Event Reservation Server (*ES*) runs. This is an optional argument. If this argument is omitted, the *ES* should be running on the same machine.

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

Once the *User* application is running, it can open or close an event, as well as check the status of the events started by this *user* application. The *User* application can also ask for a list of currently active events, and then for a specific event it can ask that the specific data of this event be shown, to see the status of the reservation process, or to make a new reservation. In the first login of a user its ID, *UID*, is used to register this user in *ES* server. The *user* also has the possibility to logout, change password, unregister or exit the *User* application.

The commands supported by the *User* interface (using the keyboard for input) are:

- ***login*** *UID password* – following this command the *User* application sends a message to the *ES*, using the UDP protocol. The *ES* confirms the ID, *UID*, and *password* of this user, or registers it if this *UID* is not present in the *ES* database.
  The result of the request should be displayed: successful login, incorrect login attempt, or new user registered.
- ***changePass*** *oldPassword newPassword* – the *User* application establishes a TCP session with the *ES* and sends a message asking to change the password to a new password.
  The result of the request should be displayed: successful password change, unknown *user*, user not logged In or incorrect password. After receiving the reply from the *ES*, the *User* closes the TCP connection.
- ***unregister*** – the *User* application sends a message to the *ES*, using the UDP protocol, asking to unregister the currently logged in user. A logout operation is also performed.
  The result of the request should be displayed: successful unregister, unknown user, or incorrect unregister attempt.
- ***logout*** – the *User* application sends a message to the *ES*, using the UDP protocol, asking to logout the currently logged in user, with ID *UID*.
  The result of the request should be displayed: successful logout, unknown *user*, or *user* not logged in.
- ***exit*** – this is a request to exit the *User* application. If a user is still logged in the *User* application should inform the user to first execute the *logout* command. Otherwise, the application can be terminated. *This is a local command, not involving communication with the ES.*

- ***create*** *name event_fname event_date num_attendees* – the *User* application establishes a TCP session with the *ES* and sends a message asking to create a new event, whose short description name is *name*, providing the file describing the event, stored in the file *event_fname*, indicating the date and time (dd-mm-yyyy hh:mm) of the event, *event_date*, and the number of people who can attend the event, *num_attendees*. the file *event_fname* should exist in the same folder.
  In reply, the *ES* sends a message indicating whether the request was successful, and the assigned event identifier, *EID*, which should be displayed to the user. After receiving the reply from the *ES*, the *User* closes the TCP connection.
- ***close*** *EID* – the *User* application establishes a TCP session with the *ES* and sends a message to the *ES*, asking to close an ongoing event, with identifier *EID*, that had been created by the logged in user.
  The *ES* will reply informing whether the event was successfully closed, stopping the acceptance of more reservations. Before closing an event, the *ES* checks the status of the event. If the date of the event had already passed, the *ES* sends that the event has already expired. If the event has been fully reserved, the *ES* should inform the *User* that this event is sold-out. This information should be displayed to the user. After receiving the reply from the *ES*, the *User* closes the TCP connection.
- ***myevents*** or ***mye*** – the *User* application sends a message to the *ES*, using the UDP protocol, asking for a list of the events created by the logged in user and their booking status.
  The *ES* will reply with the requested list, or an information that the user has not created any events. This information should be displayed to the user.
- ***list*** the *User* application establishes a TCP session with the *ES* and sends a message asking for a list of available events.
  The *ES* will reply with the requested list, or an information that no events has yet been created. This information should be displayed to the *User*.
  After receiving the reply from the *ES*, the *User* closes the TCP connection.
- ***show*** *EID* the *User* application establishes a TCP session with the *ES* and sends a message asking to see the record of event *EID*, and to receive the file describing the event.
  The *ES* will reply with the event's details, including the event name, number of total seats available, in addition to how many seats have been reserved, as well as the required file, or an error message. The file should be stored in the local drive of the *User* and its name and the directory of storage are displayed, together with the information if the event is already closed or sold-out. After receiving the reply from the *ES*, the *User* closes the TCP connection.
- ***reserve*** *EID value* – the *User* application establishes a TCP session with the *ES* and sends a message to the *ES*, asking to reserve seats in the event *EID* and the number of seats to be reserved is *value*.
  The *ES* will reply reporting the result of the reservation: accepted, refused (if *value* is larger than available seats, the *ES* also informs about the number of seats still available for the event), or informing that the event is no longer active. This information in the reply should be displayed to the *User*. After receiving the reply from the *ES*, the *User* closes the TCP connection.
- ***myreservations*** or ***myr*** – the *User* application sends a message to the *ES*, using the UDP protocol, asking for a list of the events for which the logged in user

has made a reservation.
The *ES* will reply with the requested list, or an information that the user has not made any reservation. This information should be displayed to the user.

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

2.2 *Event-reservation Server* (*ES*)

The program implementing the Event-reservation Server (*ES*) is invoked with the command:

`./ES [-p ESport] [-v],`

where:

ESport is the well-known port where the *ES* 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 *ES* basically runs two server applications, both with well-known port *ESport*, one in UDP, used for managing users and listing users' activities, and the other in TCP, used to transfer the files with more information to the *User* application, and managing events and reservations.
If the *–v* option is set when invoking the program, it operates in *verbose* mode, meaning that the *ES* outputs to the screen a short description of the received requests (*UID*, type of request) and the IP and port originating those requests.
Each received request should start being processed once it is received.

3. Communication Protocols Specification

The communication protocols used to implement the event reservation platform are described in this section. For the communication protocols *UID* is always sent using 6 digits, and *EID* is always sent using 3 digits.

3.1 *User–ES* Protocol (in UDP)

Part of the interaction between the user application (*User*) and the Event-reservation Server (*ES*) is supported by the UDP protocol. The related request and reply protocol messages to consider are:

a) LIN *UID password*
Following the ***login*** command the *User* application informs the *ES* that user *UID*, with password *password*, wants to login, to use the application.

b) RLI *status*
In reply to a LIN request the *ES* checks if a user with ID *UID* is already registered. If so, the password *password* is checked, and for a correct match the *ES* takes note that the *user* is now logged in and replies with *status* = OK. For an incorrect match the reply *status* is NOK. If the *user* was not yet registered, the *ES* registers and logs in this *user*, storing the corresponding UID and password, and the reply *status* is REG.

c) LOU *UID password*
Following the ***logout*** command the *User* application informs the *ES* that the currently logged in user, with ID *UID*, wants to logout.

d) RLO *status*
In reply to a LOU request the *ES* checks if a user with ID *UID* is logged in. If so, the *user* is logged out and the reply *status* is OK. If the user was not logged in the reply *status* is NOK. If the *user* was not registered the reply *status* is UNR. If the password is incorrect the reply *status* is WRP.

e) UNR *UID password*
Following the ***unregister*** command the *User* informs the *ES* that the currently logged in user, with ID *UID*, wants to unregister.

f) RUR *status*
In reply to a UNR request the *ES* checks if a user with ID *UID* is registered. If so, the user can be unregistered and reply *status* is OK. If the user was not logged in the reply *status* is NOK. If the user was not registered the reply *status* is UNR. If the password is incorrect the reply *status* is WRP.

g) LME *UID password*
Following the ***myevents*** command the *User* sends the *ES* a request to list the events created by user *UID*.

h) RME *status[EID state]**
In reply to a LME request, the *ES* reply *status* is NOK if user UID has not created any events. If the user is not logged in, the reply *status* is NLG. If the user UID has created events, the reply *status* is OK and a list of the identifiers *EID* and *state* for all events started by this user, separated by single spaces, is sent by the *ES*. *state* takes value *1* if the event is still accepting reservation, *0* if the

event was in the past, *2* if the event is still in the future but sold out, and *3* if the event was closed by the user *UID*. If the password is incorrect the reply *status* is WRP.

i) LMR *UID password*
Following the ***myreservations*** command the *User* sends the *ES* a request to list the reservations made by user *UID*.

j) RMR *status[EID date value]**
In reply to a LMR request, the *ES* reply *status* is NOK if user *UID* has not made any reservation. If the user is not logged in, the reply *status* is NLG. If the user UID has made some reservations, the reply *status* is OK and a list of the identifiers *EID* along with the *date* and *time* of the reservation (format: dd-mm-yyyy hh:mm:ss), and the number of places reserved (*value*) by user UID. *value* is an integer between 1 and 999. If the password is incorrect the reply *status* is WRP.

The *User* application should always display information about the received replies in a human friendly format.
In the above messages the separation between any two items consists of a single space.
** [field]* means that *field* is optional.

Each request or reply message ends with the character "\n".

For replies including the *status* field it takes the value ERR when the syntax of the request message was incorrect or when the parameter values take invalid values.

If an unexpected protocol message is received, the reply is ERR.

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

The interaction between the player application (*User*) and the event server (*ES*) for messaging related to the transfer of files is supported by the TCP protocol.
The related request and reply protocol messages to consider are:

a) `CRE` *`UID password name event_date attendance_size Fname Fsize Fdata`*
Following the ***`create`*** command the *User* application opens a TCP connection with the *ES* and asks to create a new event. The information sent includes:
- a short description name (a single word): *`name (format: maximum of 10 characters – alphanumeric only)`*
- the date and time of the event: *`event_date (format: dd-mm-yyyy hh:mm)`*
- the number of places to book: *`attendance_size (10 → 999);`*
- the filename where a description of the event is included: *`Fname`*
- the file size in bytes: *`Fsize`* (maximum size is 10 MB (10.10^6 B))
- the contents of the selected file: *`Fdata`*.

b) `RCE` *`status [EID]`*
In reply to a `CRE` request the *ES* replies with *`status`* = `NOK` if the event could not be created. If the user was not logged in, the reply *`status`* is `NLG`. If the password is incorrect the reply *`status`* is `WRP`. Otherwise the *ES* replies with *`status`* = `OK`, and sends a unique event identifier *`EID`*. *`EID`* is a 3-digit number from 001 till 999.
A local copy of the file is stored using the filename *`Fname`* at the server side.
After receiving the reply message, the *User* closes the TCP connection with the *ES*.

c) `CLS UID password EID`
Following the ***`close`*** command the *User* application opens a TCP connection with the *ES* and asks to close the event *`EID`*.

d) `RCL status`
In reply to a `CLS` request the *ES* replies informing whether it was able to close the event *`EID`*. The reply *`status`* is `OK`, if event *`EID`* was open, it was created by user *`UID`* and could be successfully closed by the *ES*. The reply *`status`* is `NOK`, if the user *`UID`* does not exist or if the password is incorrect. If the user was not logged in, the reply *`status`* is `NLG`. The *`status`* is `NOE`, if the event *`EID`* does not exist. The *`status`* is `EOW`, if the event is not created by user *`UID`*, and *`status`* is `SLD`, if event *`EID`*, created by user *`UID`*, is already sold out. The status *`status`* is `PST`, if the event was already in the past (date is before current date). The status *`status`* is `CLO`, if the event was previously closed by user *`UID`*.
After receiving the reply message, the *User* closes the TCP connection with the *ES*.

e) `LST`
Following the ***`list`*** command the *User* opens a TCP connection with the *ES* and sends the *ES* a request to send the list of events.

f) `RLS status[ EID name state event_date ]*`
In reply to a `LST` request, the *ES* reply *`status`* is `NOK` if no event was created. Otherwise, the reply *`status`* is `OK` and a list of the identifiers *`EID, name,`*

state and *date* for all events, separated by single spaces, is sent by the *ES*. *state* takes value *1* if the event is in the future and is not sold out, *0* if the event is in the past, *2* if the event is in the future but is sold out, or *3* if the event was closed by the user *UID*. *Event_date* is the date of the event in the format: *dd-mm-yyyy hh:mm*. The server ES sends "\n" as an indication of the end of the list. The *User* continues receiving the list, till it receives "\n", then the *User* closes the TCP connection with the *ES*.

g) SED *EID*

Following the ***show*** command the *User* application opens a TCP connection with the *ES* and asks to receive the file describing the details of the event *EID* (format of *EID* is 3-digit number from 001 till 999).

h) RSE *status [UID name event_date attendance_size Seats_reserved Fname Fsize Fdata]*

In reply to a SED request the *ES* replies with *status* = OK and sends a file containing the description of the event. The information sent includes:

- the owner of the event *UID*;
- the event name *name*;
- event date *event_date (format: dd-mm-yyyy hh:mm)*;
- the total number of places in the event: *attendance_size (10 → 999)*;
- the number of seats reserved *Seats_reserved*;
- the filename *Fname*;
- the file size *Fsize*, in bytes;
- the contents of the selected file (*Fdata*).

The file is locally stored using the filename *Fname*.

The *User* displays the name and size of the stored file.

If there is no file to be sent, event does not exist, or some other problem, the *ES* replies with *status* = NOK.

After receiving the reply message, the *User* closes the TCP connection with the *ES*.

i) RID UID password EID people

Following the ***reserve*** command the *User* application opens a TCP connection with the *ES* and asks to make a reservation, with number of places *people*, for event *EID*. *EID* is a 3-digit number from 001 till 999. *people* is digit number between 1 and 999.

j) RRI status [n_seats]*

In reply to an RID request the *ES* reply *status* is NOK if event *EID* is not active. If the user was not logged in the reply *status* is NLG. If event *EID* is open, the reply *status* is ACC if the reservation can be fulfilled. The reply *status* is CLS, if the event is closed. The reply *status* is SLD, if the event is sold out. The reply *status* is REJ, if the reservation was rejected because the number of requested places *people* is larger than the number of remaining places. In this case, the reply also includes the number of remaining seats to reserve (*n_seats*).

The reply *status* is PST, if the event's date has passed. If the password is incorrect the reply *status* is WRP.
After receiving the reply message, the *User* closes the TCP connection with the *ES*.

k) CPS UID oldPassword newPassword
Following the ***changePass*** command the User application opens a TCP connection with the ES and sends a request to change the existing password (oldPassword) to a new password (newPassword), for user *UID*.

l) RCP status
In reply to a CPS request, the *ES* first checks the user UID status. If the user was not logged in, the reply *status* is NLG. If the password is not correct, the reply *status* is NOK. If the user does not exist, the reply *status* is NID. If the user was logged in and the oldPassword matches the saved password, the reply *status* is OK and the *ES* modifies the saved password to the newPassword.
After receiving the reply message, the *User* closes the TCP connection with the *ES*.

The filenames *Fname*, are limited to a total of 24 alphanumerical characters (plus '-', '_' and '.'), including the separating dot and the 3-letter extension: "*nnn…nnnn.xxx*".
The file size *Fsize* is limited to 10 MB (10.10^6 B), being transmitted using a maximum of 8 digits.

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".

For replies including the *status* field it takes the value ERR when the syntax of the request message was incorrect or when the parameter values take invalid values.

If an unexpected protocol message is received, the reply will be ERR.

4. Development

4.1 Development and test environment

Make sure your code compiles and executes correctly in the development environment available in the labs *LT4* or *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.

The client and the server processes should not terminate abruptly in failure situations, such as these:

- wrong protocol messages received by the server: an error message should be returned, as specified by the protocol;
- wrong protocol messages received by the client: the interaction with the server is not continued and the user is informed;
- error conditions from system calls: the programs should not terminate abruptly, avoiding cases of "segmentation fault" or "core dump".

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 *ES server*, as well as the corresponding *Makefile* and the *auto-avaliação* excel file.
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 December 19, 2025, at 23:59 PM**.
You should create a single `zip` archive containing all the source code, makefile, all auxiliary files required for executing the project and the *auto-avaliação* excel file. 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.