Redes de Computadores

1st Semestre, P2, 2023/2024
LEIC Alameda

Programming using the *Sockets* interface

"RC Auction"

1. Introduction

The goal of this project is to implement a simple auction platform. Users can open (host) an auction to sell some asset, and close it, as well list ongoing auctions and make bids.

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

The interface, using the keyboard, allows the *User* application to control the actions to take:

- 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 *AS* receives a login request it will inform of a successful or incorrect login attempt or, in case the `UID` was not present at the *AS*, register a new *user*.
- Open a new auction. The *User* application sends a message to the *AS* asking to open a new auction, providing a short description name (represented with up to 10 alphanumerical characters), an image (or other document file) of the asset to sell, the start value (represented with up to 6 digits), and the duration of the auction in seconds (represented with up to 5 digits). In reply, the *AS* informs if the request was successful, and the assigned auction identifier, `AID`, a 3-digit number.
- Close ongoing auction. The *User* application sends a message to the *AS* asking to close an ongoing auction, which had been started by the logged in *user*. The *AS* will reply informing whether the auction was successfully closed, cancelling the sale, or if the auction time had already ended.
- List auctions started by this user (*myauctions*). The *User* application asks the *AS* for a list of the auctions started by the logged in *user*. The *AS* will reply with the requested list, or an information that the *user* has not started any active auction.
- List auctions for which this user made a bid (*mybids*). The *User* application asks the *AS* for a list of the auctions in which the logged in *user* has placed a bid. The *AS* will reply with the requested list, or an information that the *user* has not made a bid in any of the currently active auctions.
- List all auctions. The *User* application asks the *AS* for a list of auctions. The *AS* will reply with the requested list, or an information that no auctions were yet started.
- Show asset. For an ongoing auction, the *User* application asks the *AS* to send the file associated with the asset in sale for specified auction. In reply, the *AS* sends the required file, or an error message. The file is stored, and its name and the directory of storage are displayed to the *user*. Filenames 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 is limited to 10 MB, represented using a maximum of 8 digits.

- Bid. The *User* application asks the *AS* to place a bid, with the indicated value, for the selected auction. The *AS* will reply reporting the result of the bid: accepted, refused (if `value` is not larger than the previous highest bid), or informing that the auction is no longer active.
- Show record. The *user* asks the *AS* about the record of an auction. The *AS* will reply with information about the auction, including its name, start value, start time and duration, followed by a description of the received bids, including the bidder ID, the value and time of the bid, as well as an indication if the auction was finished and when.
- Logout. The user can ask the *AS* server to terminate the interaction (logout).
- Unregister. The user can ask the *AS* server to unregister this *user*.
- 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 auction platform (*User*) is invoked using:

`./user [-n ASIP] [-p ASport],`

where:

ASIP is the IP address of the machine where the auction server (*AS*) runs. This is an optional argument. If this argument is omitted, the *AS* should be running on the same machine.

ASport is the well-known port (TCP and UDP) where the *AS* 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 auction, as well as check the status of the auctions started by this *user* application. The *User* application can ask for a list of currently active auctions, and then for a specific auction it can ask that the auction data be shown, to see the status of the bidding process, or to make a new bid. In the first login of a *user* its ID, *UID*, is used to register this *user* in *AS* server. The *user* also has the possibility to logout, unregister or exit the *User* application. After unregistering, a *user* may register again, issuing again a login command but defining a new password. All data existing in the server about a *user* from previous registrations is preserved, except for the password.

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 *AS*, using the UDP protocol, confirm the ID, *UID*, and *password* of this *user*, or register it if this *UID* is not present in the *AS* database.
  The result of the request should be displayed: successful login, incorrect login attempt, or new *user* registered.
- ***logout*** – the *User* application sends a message to the *AS*, 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.
- ***unregister*** – the *User* application sends a message to the *AS*, 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.
- ***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 *AS*.

- ***open*** *name asset_fname start_value timeactive* – the *User* application establishes a TCP session with the *AS* and sends a message asking to open a new auction, whose short description name is *name*, providing an image of the asset to sell, stored in the file *asset_fname*, indicating the start value

for the auction, *start_value*, and the duration of the auction, *timeactive*. In reply, the *AS* sends a message indicating whether the request was successful, and the assigned auction identifier, *AID*, which should be displayed to the *User*. After receiving the reply from the *AS*, the *User* closes the TCP connection.

- **close** *AID* – the *User* application sends a message to the *AS*, using the TCP protocol, asking to close an ongoing auction, with identifier *AID*, that had been started by the logged in *user*.
  The *AS* will reply informing whether the auction was successfully closed, cancelling the sale, or if the auction time had already ended. This information should be displayed to the *User*. After receiving the reply from the *AS*, the *User* closes the TCP connection.
- **myauctions** or **ma** – the *User* application sends a message to the *AS*, using the UDP protocol, asking for a list of the auctions started by the logged in *user*, or auctions in which the logged in *user* has placed a bid.
  The *AS* will reply with the requested list, or an information that the *user* is not involved in any of the currently active auctions. This information should be displayed to the *User*.
- **mybids** or **mb** – the *User* application sends a message to the *AS*, using the UDP protocol, asking for a list of the auctions for which the logged in *user* has placed a bid.
  The *AS* will reply with the requested list, or an information that the *user* has no active auction bids. This information should be displayed to the *User*.

- **list** or **l** – the *User* application sends a message to the *AS*, using the UDP protocol, asking for a list of the currently active auctions.
  The *AS* will reply with the requested list, or an information that no auctions are currently active. This information should be displayed to the *User*.
- **show_asset** *AID* or **sa** *AID* – the *User* establishes a TCP session with the *AS* and sends a message asking to receive the image file of the asset in sale for auction number *AID*.
  In reply, the *AS* sends the required file, or an error message. The file is stored and its name and the directory of storage are displayed to the *User*. After receiving the reply from the *AS*, the *user* closes the TCP connection.
- **bid** *AID value* or **b** *AID value* – the *User* application sends a message to the *AS*, using the TCP protocol, asking to place a bid for auction *AID* of value *value*.
  The *AS* will reply reporting the result of the bid: accepted, refused (if *value* is not larger than the previous highest bid), or informing that the auction is no longer active. The user is not allowed to bid in an auction hosted by him. This information should be displayed to the *User*. After receiving the reply from the *AS*, the *User* closes the TCP connection.
- **show_record** *AID* or **sr** *AID* – the *User* application sends a message to the *AS*, using the UDP protocol, asking to see the record of auction *AID*.
  The *AS* will reply with the auction details, including the list of received bids and information if the auction is already closed. 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 *AS* should be displayed to the *user*.

2.2 *Auction Server* (*AS*)

The program implementing the Auction Server (*AS*) is invoked with the command:

`./AS [-p ASport] [-v],`

where:

ASport is the well-known port where the *AS* 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 *AS* makes available two server applications, both with well-known port *ASport*, one in UDP, used for managing the auction and the bids, and the other in TCP, used to transfer the files with asset images to the *User* application.
If the *–v* option is set when invoking the program, it operates in *verbose* mode, meaning that the *AS* 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 auction framework are described in this section. For the communication protocols *UID* is always sent using 6 digits.

3.1 *User–AS* Protocol (in UDP)

Part of the interaction between the user application (*User*) and the auction server (*AS*) 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 *AS* that user *UID*, with password *password*, wants to login, to be able to participate in auctions.

b) RLI *status*
In reply to a LIN request the *AS* checks if a user with ID *UID* is already registered. If so, the password *password* is checked, and for a correct match the *AS* 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 *AS* 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 *AS* that the currently logged in user, with ID *UID*, wants to logout.

d) RLO *status*
In reply to a LOU request the *AS* 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.

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

f) RUR *status*
In reply to a UNR request the *AS* server checks if a user with ID *UID* is registered. If so, the *user* can be unregistered 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.

g) LMA *UID*
Following the ***myauctions*** command the *User* sends the *AS* a request to list the auctions started by user *UID*.

h) RMA *status[AID state]**
In reply to a LMA request the *AS* reply *status* is NOK if user UID has no ongoing auctions. If the user is not logged in the reply *status* is NLG. If there are ongoing auctions for user UID the reply *status* is OK and a list of the identifiers *AID* and *state* for all ongoing auctions started by this user,

separated by single spaces, is sent by the *AS*. *state* takes value *1* if the auction is active, or *0* otherwise.

i) LMB *UID*
Following the ***mybids*** command the *User* sends the *AS* a request to list the auctions for which the user *UID* has made bids.

j) RMB *status[AID state]**
In reply to a LMB request the *AS* reply *status* is NOK if user UID has no ongoing bids. If the user is not logged in the reply *status* is NLG. If there are ongoing bids for user UID the reply *status* is OK and a list of the identifiers *AID* and *state* for all ongoing auctions for which this user has placed bids, separated by single spaces, is sent by the *AS*. *state* takes value *1* if the auction is active, or *0* otherwise.

k) LST
Following the ***list*** command the *User* sends the *AS* an auction list request.

l) RLS *status[AID state]**
In reply to a LST request the *AS* reply *status* is NOK if no auction was yet started. If there are ongoing auctions the reply *status* is OK and a list of the identifiers *AID* and *state* for all auctions, separated by single spaces, is sent by the *AS*. *state* takes value *1* if the auction is active, or *0* otherwise.

m) SRC *AID*
Following the ***show_record*** command the *User* sends the *AS* a request for the record of auction *AID*.

n) RRC *status [host_UID auction_name asset_fname start_value start_date-time timeactive]*
*[B bidder_UID bid_value bid_date-time bid_sec_time]**
[E end_date-time end_sec_time]
In reply to a SRC request the *AS* reply *status* is NOK if the auction *AID* does not exist. Otherwise the reply *status* is OK followed by information about the ID *host_UID* of the user that started the auction, the auction name *auction_name* and the name of the file *asset_fname* with information about the item being sold, the minimum bid value *start_value*, and the start date and time *start_date-time* of the auction in the format YYYY-MM-DD HH:MM:SS (19 bytes), as well as the duration of the auction *timeactive* in seconds (represented using 6 digits).
If this auction has received bids then a description of each bid is presented in a separate line starting with B and including: the ID of the user that place this bid *bidder_UID*, the bid value *bid_value*, the bid date and time *bid_date-time* in the format YYYY-MM-DD HH:MM:SS (19 bytes), as well as the number of seconds elapsed since the beginning of the auction until the bid was made *bid_sec_time* (represented using 6 digits).
In case the auction is already closed there is one last line added to the reply including the date and time of the auction closing *end_date-time* in the format YYYY-MM-DD HH:MM:SS (19 bytes), as well as the number of seconds elapsed since the beginning of the auction until the bid was made *end_sec_time*.

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.
*[field]** means that several values of *field* can be included, separated by a 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 is ERR.

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

The interaction between the player application (*User*) and the auction server (*AS*) 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) OPA *UID password name start_value timeactive Fname Fsize Fdata*
Following the ***open*** command the *User* application opens a TCP connection with the *AS* and asks to open a new auction. The information sent includes:
- a short description name (a single word): *name*
- the minimum selling value for the asset: *start_value*
- the duration of the auction in minutes: *timeactive*
- the filename where an image of the asst to be sold is included: *Fname*
- the file size in bytes: *Fsize*
- the contents of the selected file: *Fdata*.

b) ROA *status [AID]*
In reply to a OPA request the *AS* replies with *status* = NOK if the auction could not be started. If the user was not logged in the reply *status* is NLG. Otherwise the *AS* replies with *status* = OK, and sends a unique auction identifier *AID*.
A local copy of the asset file is stored using the filename *Fname*.
After receiving the reply message, the *User* closes the TCP connection with the *AS*.

c) CLS *UID password AID*
Following the ***close*** command the *User* application opens a TCP connection with the *AS* and sends a request to close the auction with identifier *AID*, which had been opened by the logged in user, whose ID is *UID*.

d) RCL *status*
In reply to a CLS request the *AS* replies informing whether it was able to close auction *AID*. The reply *status* is OK, if auction *AID* was ongoing, it was started by user *UID*, and could be successfully closed by the *AS*. If the user was not logged in the reply *status* is NLG. The *status* is EAU, if the auction *AID* does not exist. *status* is EOW, if the auction is not owned by user *UID*, and *status* is END, if auction *AID* owned by user *UID* has already finished.
After receiving the reply message, the *User* closes the TCP connection with the *AS*.

e) SAS *AID*
Following the ***show_asset*** command the *User* application opens a TCP connection with the *AS* and asks to receive the image illustrating the asset for sale in auction *AID*.

f) RSA *status [Fname Fsize Fdata]*
In reply to a SAS request the *AS* replies with *status* = OK and sends a file containing the image illustrative of the asset for sale. The information sent includes:
- 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, or some other problem, the *AS* replies with *status* = NOK.
After receiving the reply message, the *User* closes the TCP connection with the *AS*.

g) BID *UID* password *AID value*
Following the ***bid*** command the *User* application opens a TCP connection with the *AS* and sends the *AS* a request to place a bid, with value *value*, for auction *AID*.

h) RBD *status*
In reply to a BID request the *AS* reply *status* is NOK if auction *AID* is not active. If the user was not logged in the reply *status* is NLG. If auction *AID* is ongoing the reply *status* is ACC if the bid was accepted. The reply *status* is REF if the bid was refused because a larger bid has already been placed previously. The reply *status* is ILG if the *user* tries to make a bid in an auction hosted by himself.
After receiving the reply message, the *User* closes the TCP connection with the *AS*.

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".

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 *AS 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 15, 2023, 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.