

1 2 9 0



UNIVERSIDADE D
COIMBRA


Faculty of Sciences and Technology

COMMUNICATION NETWORKS - BACHELOR'S DEGREE IN
INFORMATICS ENGINEERING

ONLINE CLASSES SYSTEM USING TCP/UDP STACK

Project by:

Nuno Batista  2022216127

Miguel Martins  2022213951

May, 2024

Table of Contents

1 - class_server.c.....	2
1.1 - TCP.....	2
1.2 - UDP.....	2
1.3 - structures_creation.c.....	2
2 - class_client.c.....	2
2.1 Process flow and universal commands.....	2
2.2 Student only commands.....	3
2.3 Professor only commands.....	3
2.4 Multicast.....	3
3 - class_admin.c.....	4
4 - GNS3 Network Configuration.....	4
4.1 Network Topology and Addresses.....	4
4.2 Configuration Commands.....	5

Notes:

- As an alternative to Netcat, an UDP client application was made.
- A makefile was included in the server and in every PC for easy compilation.

1 - class_server.c

The program starts by setting up the SIGINT signal override, which will orderly close every socket, disconnecting every client and cleaning the shared memory segment. It will then verify if both ports are valid values, and if the configuration file is set up as expected. The classes' shared memory is then created, storing a struct of type `Classes`, this contains the current amount of classes and an array of `ClassInfo` structs, this struct in turn, stores the information about a given class: the class name, total capacity, number of students enrolled, the multicast address, the UDP socket, and an array of the enrolled users. The shared memory semaphore is then initialized, and two threads are created, one of them listens to a TCP socket for professor and student commands and the other one listens to an UDP socket for admin commands.

1.1 - TCP

The *handle_tcp* thread creates the socket with `SOCK_STREAM`, binds it to the server address and port and starts listening for incoming connections. After accepting the connection a new process is created to handle the client.

The *process_client* process is responsible for reading the clients messages until it disconnects, making sure the client is removed from the subscribed classes when disconnected. The requests made from the client will be handled in the *interpret_client_command* function, which will parse the command accordingly, verifying if the request is valid. Upon validation, either sends an error message or serves the request by performing a variety of operations in the shared memory, said operations can be executed by multiple processes at the same time, hence, the need for a binary semaphore.

1.2 - UDP

The *handle_udp* thread creates the socket with `SOCK_DGRAM`, binds it to the server address and port. Since UDP is connectionless, unlike TCP, there is no need for a new process to be created or connection to be made, and simply handles the request with *interpret_admin_command*, which works similarly to *interpret_client_command*. In this program, the only type of user allowed to use the UDP socket is the system administrator. As already mentioned, this protocol is connectionless, therefore, we cannot store information about an admin's current session, this means that in order to prevent the access to admin commands before providing an admin's credentials, those commands are only executed once the *admin_logged_in* flag is activated.

2 - class_client.c

2.1 - Process flow and universal commands

The program starts by handling the SIGINT signal override, which will drop the client's multicast groups (unsubscribing every class), close the sockets and exit the main process. It will then check for the port validity, reset the `server_address` structure, and set the servers' ip address and port (passed as an argument). The socket is then created and the client connected to the server, entering the main work cycle: The user types a command, the client sends it to the server and then waits for a response.

The client has a list of possible commands to submit and if the given command is not in the list, the server gives a response acknowledging that. The user can log into the server as a professor or as a student, noting that some commands can be accessed by both users and some can not.

List of commands accessible to both users:

- LOGIN <username> <password> : Verifies if the username and password are registered in the list, and logs the user in according to the role assigned to the user. Since only students and professors can use the TCP client, if the user trying to log in is an Admin (only uses UDP), the server will reject it, sending the according message.
- LIST_CLASSES : Waits for permission to access the Classes shared memory, then prints the name of every listed class.

2.2 - Student only commands:

- LIST_SUBSCRIBED : Accesses and displays the logged in student's subscribed classes.
- SUBSCRIBE_CLASS <class_name> : Waits for permission to access the Classes' shared memory. The request to subscribe to a class is rejected if either the class is full or the student is already enrolled in it. If accepted, the user is then added to the classes' subscribed user array, and the class added to the user's subscribed classes' array. The multicast details will be discussed further ahead.

2.3 - Professor only commands:

- CREATE_CLASS <class_name> <capacity> : Assigns the next available multicast IP and waits for permission to access the Classes' shared memory. Then checks if the class to be added already exists and if the maximum number of classes has been reached. The multicast socket is then created and opened and the class is then added to the Classes' shared memory, with all the info.
- SEND <class_name> <message> : Waits for permission to access the Classes' shared memory. Checks if the class exists, and sends the message to the class' UDP socket.

2.4 - Multicast

When a student enrolls in a given class for the first time, the *join_multicast_group* function is called, this function opens a reusable UDP socket and binds it to the class' multicast address, provided by the server. After making sure the multicast membership has been properly added, the *receive_multicast_messages* thread is created. This thread's work cycle consists in continuously listening to the newly added multicast address, displaying every received message to the client. Moreover, every time a multicast group is subscribed to, its respective socket and *ip_mreq* structure are added to a global array exclusive to that students process memory, ensuring easy access to those variables, which will be used before the system termination to clean up the resources and drop the multicast membership.

To configure the multicast in the GNS3 routers, we activated the multicast routing option with the **ip multicast-routing** command in each router and the protocol-independent multicast (PIM) option in each interface with the **ip pim sparse-dense-mode** command.

3 - class_admin.c

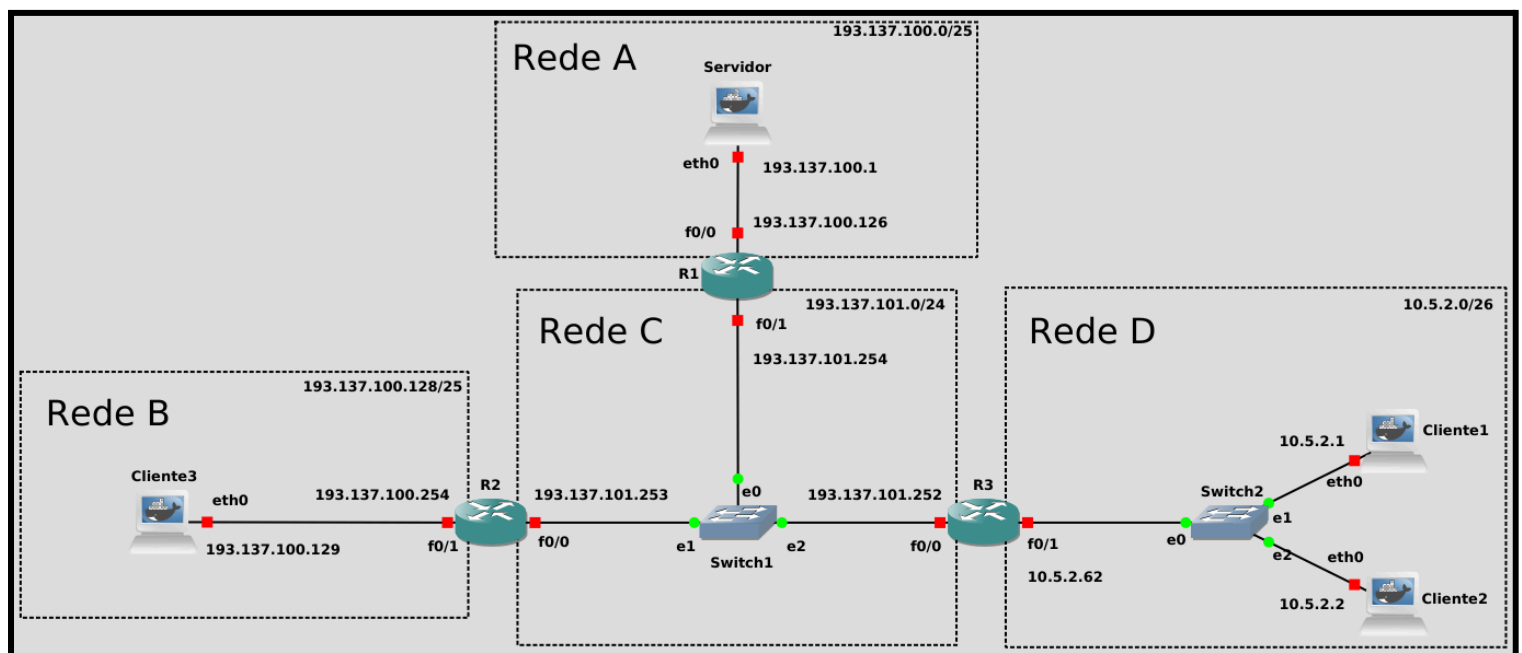
The program starts by handling the SIGINT signal override, which will close the client's socket. It will then check for the port's validity, reset the server_address structure and set the servers' ip address and port (passed as an argument). The client's socket is then created and the program enters the main work cycle: The user types a command, the client sends it to the server and waits for a response.

The client has a list of possible commands to submit and if the given command is not in the list, the server gives a response acknowledging that. Needless to say, only users with the administrator role can login.

- LOGIN <username> <password> : Verifies if the username and password are registered in the list and logs the user if its assigned role is Admin. Since only admin users use UDP, this client only works with admin users and the server rejects the rest, sending a message accordingly.
- ADD_USER <username> <password> <role> : Check if the user to add already belongs to the list of registered users, or the maximum number of users has already been reached. If not, add the new user to the registered_users array, and its info added to the configuration file.
- DEL <username> : Check if the user is registered. Removes the user from both the registered_users array and rewrites the configuration file without the removed user. Sends a message to the server according to the action taken.
- LIST : Displays the complete information about every registered user: name, password and role.
- QUIT_SERVER : Cleans up the server's resources and disconnects clients ensuring a clean termination

4 - GNS3 Network Configuration

4.1 - Network Topology and Addresses



4.2 - Configuration Commands

Router 1

```
conf t
ip multicast-routing
interface FastEthernet0/0
ip address 193.137.100.126 255.255.255.128
ip pim sparse-dense-mode
no shutdown
exit

interface FastEthernet0/1
ip address 193.137.101.254 255.255.255.0
ip pim sparse-dense-mode
no shutdown
exit

ip route 193.137.100.128 255.255.255.128
193.137.101.253

end
copy running-config startup-config
```

Router 2

```
conf t
ip multicast-routing
interface FastEthernet0/0
ip address 193.137.101.253 255.255.255.0
ip pim sparse-dense-mode
no shutdown
exit

interface FastEthernet0/1
ip address 193.137.100.254 255.255.255.128
ip pim sparse-dense-mode
no shutdown
exit

ip route 193.137.100.0 255.255.255.128
193.137.101.254

end
copy running-config startup-config
```

Client 1:

```
# Static config for eth0
auto eth0
iface eth0 inet static
    address 10.5.2.1
    netmask 255.255.255.192
    gateway 10.5.2.62
```

Client 2:

```
# Static config for eth0
auto eth0
iface eth0 inet static
    address 10.5.2.2
```

```
netmask 255.255.255.192
gateway 10.5.2.62
```

Router 3

```
conf t
ip multicast-routing
interface FastEthernet0/0
ip address 193.137.101.252 255.255.255.0
ip pim sparse-dense-mode
no shutdown
exit

interface FastEthernet0/1
ip address 10.5.2.62 255.255.255.192
ip pim sparse-dense-mode
no shutdown
exit
```

```
access-list 1 permit 10.5.2.0 0.0.0.63
ip nat inside source list 1 interface
FastEthernet0/0 overload
```

```
interface FastEthernet0/1
ip nat inside
exit
```

```
interface FastEthernet0/0
ip nat outside
exit
```

```
ip route 193.137.100.128 255.255.255.128
193.137.101.253
ip route 193.137.100.0 255.255.255.128
193.137.101.254
```

```
end
copy running-config startup-config
```

Server:

```
# Static config for eth0
auto eth0
iface eth0 inet static
    address 193.137.100.1
    netmask 255.255.255.128
    gateway 193.137.100.126
```

Client 3:

```
# Static config for eth0
auto eth0
iface eth0 inet static
    address 193.137.100.129
    netmask 255.255.255.128
    gateway 193.137.100.254
```