



Trabalho Prático no âmbito da cadeira Redes de Comunicação

**Meta Final**

# Sistema de turmas com uso dos protocolos TCP/UDP e multicast.

**Realizado por:**
**Miguel Curto Castela - uc2022212972**
**Francisco Lapa Silva - uc2022213583**

# Índice

## Conteúdo



Notas:

- Como alternativa ao Netcat, foi feita uma aplicação cliente UDP (class_admin).
- Um makefile foi incluído no servidor e em cada PC para facilitar a compilação (make server; make client; make admin).

# 1. Funcionalidades do sistema

## 1.1 class_server.c

**main**

- É feito o controlo dos argumentos de entrada, ou seja, as *ports* das turmas e o ficheiro de configurações com as informações dos *users*.
- São validados as *ports* e os argumentos iniciais.
- O SIGINT é redirecionado para controlar o encerramento (fechar o *main* e as conexões TCP e UDP, desligar o sistema, terminar os processos filho e libertar recursos partilhados). As funções usadas neste efeito são as system_shutdown, close_main e close_tcp
- Um semáforo é usado para controlar o acesso a cada memória partilhada e dois ao ficheiro de configuração.
- É criada a memória partilhada (com create_shared_memory) para armazenar informações de turmas e PIDs dos processos filhos.
- Inicializa semáforos para sincronizar o acesso aos recursos partilhados e controla as conexões UDP e TCP.
- São criadas as *threads* para as conexões TCP e UDP (com o handle_tcp e handle_udp).

**funções**

O servidor cria e gere *threads* separadas para lidar com conexões TCP (os utilizadores) e UDP (os administradores), aceita novas conexões TCP e cria um processo para cada cliente. Cada um desses processos filho controlam a ligação com os clientes (através de process_client_tcp) e interpreta, responde e formata as *requests* dos clientes, (através do handle_requests_tcp e handle_usercursor) incluindo o login, logout, listagem de turmas, inscrição em turmas e envio de mensagens. A *thread* UDP controla a ligação UDP (através de process_admin_udp) e interpreta, responde e formata os *requests* dos administradores (através de handle_request_udp e handle_usercursor), incluindo o login, logout, listagem de comandos, adicionar, remover, listar utilizadores e desligar o servidor.

## 1.2 file_manager.c

- Controla e valida as informações dos utilizadores guardados nos ficheiros de configuração.
- São utilizados semáforos para a sincronização de acesso ao ficheiro.

**funções**

As principais funções incluem file_checkintegrity (chamada no *main* localizado no início do programa, no processo de validação dos *inputs*), que verifica a integridade do ficheiro de configuração. file_finduser, que procura um utilizador verificando o nome de utilizador e a palavra-passe específica no ficheiro, e devolvendo o tipo (utilizado para login). file_adduser, que adiciona um novo utilizador ao ficheiro, validando previamente se este já existe e se o tipo é válido, e file_removeuser, que remove um utilizador do ficheiro.
Adicionalmente, a função file_listusers lista, de forma formatada, todos os utilizadores presentes no ficheiro.

## 1.3 class_struct.c

- Uma *struct* de turma tem os campos **name, size, subscribed, subscribed_names, mutilcast_addr, udp_socket**

**funções**

Uma *struct* turma é criada e inicializada com a função create_classstruct. Esta valida também se a estrutura já está ocupada e se o tamanho da turma é válido. A função destroy_classstruct verifica que a turma não está vazia e por fim reinicia os campos da *struct*. A função addsub_classstruct tenta adicionar um aluno a uma turma, verificando primeiro se há espaço e se este já está inscrito. A função sendmsg_classstruct escreve

para o endereço multicast da turma (**multicast_addr**) a mensagem dada. Cada chamada desta função abre e configura um novo *socket* com o dado endereço, encerrando este após a mensagem ser enviada.

### 1.4 Commands_server.c

- Todas as funções descritas neste ficheiro são chamadas pelas funções handle_request_tcp e handle_request_udp presentes no ficheiro **Class_server.c**.
- Todas estas funções recebem os inputs necessários para funcionarem e um ponteiro ***response** para onde vão escrever (num ***buffer***) a informação a enviar ao utilizador.

**funções**

| login | create_class | list_users |
|---|---|---|
| Tenta autenticar um utilizador com o nome e password dados, devolvendo o tipo, se as credenciais estiverem corretas | Cria uma turma com o nome e tamanho dados. Atribui-lhe o próximo endereço multicast disponivel | Lista todos os utilizadores e os seus tipos a partir do ficheiro de configuração. |
| list_classes | list_subscribe | add_user |
| Lista todas as aulas existentes. | Lista todas as aulas em que o utilizador está inscrito. | Adiciona um novo utilizador ao ficheiro de configuração, com as credenciais dadas. |
| subscribe_class | send_message | *list_cmds_udp* |
| Inscreve o utilizador na turma com o nome dado. | Envia uma mensagem para o endereço **multicast** da turma dada. | Lista todos os comandos UDP disponíveis. |
| logout | del_user | list_commands_tcp |
| Dá reset à struct ***User***, o que retira as permissões de User ao cliente | Apaga um utilizador, com o nome dado, do ficheiro de configuração. | Lista todos os comandos TCP disponíveis. |

### 1.5 class_client.c

**main**

- Faz o controlo de elementos de entrada (IP do servidor e port TCP a que está ligado).
- Verifica que o IP é válido (a partir do gethostbyname()) e que a *port* é válido (entre 1024-65535).
- Abre o **socket** e tenta ligar ao endereço e port especificados no servidor.
- Após entrar, fica num *loop* para lidar com a comunicação com o servidor, para registar a entrada do aluno e ler mensagens deste (através dos **buffers** para entrada e saída de mensagens).
- Sempre que é recebida uma mensagem TCP do servidor, o cliente verifica se é uma mensagem especial antes de escrever para o ecrã e esperar resposta do utilizador (mensagens especiais :"+!SERVER-CL0SING!+-"-> quebra o *loop* e fecha o cliente, o servidor também fecha "-+!L0G0UT!+-" -> o servidor deu ordem de saída a todos os grupos multicast em que este utilizador estava previamente e espera por nova mensagem do servidor antes de continuar "-+!MULT1C4ST!+-XXX.XXX.XXX.XXX" -> o servidor ordena que este cliente se junte ao grupo multicast do endereço dado e espera nova mensagem do servidor antes de continuar). Isto é feito para que o servidor possa controlar algumas funções do cliente.

**funções**

Garante que o cliente envia uma mensagem de saída ao servidor e fecha o **socket** antes de sair através da handle_sigint. A função *muliticast_listener ouve e espera, em paralelo do *main*, a partir de *threads*, por mensagens multicast de um dado IP, escrevendo-as no terminal.

### 1.6 Class_admin.c

**main**

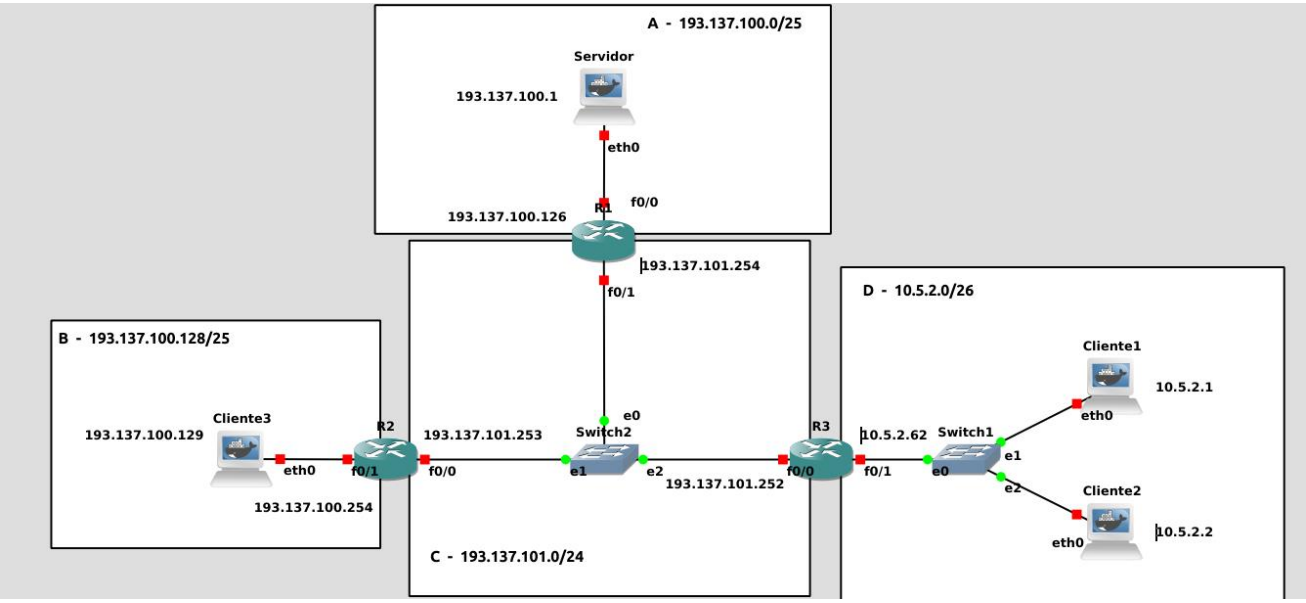
- Faz o controlo de elementos de entrada (IP do servidor e port UDP a que está ligado).

- Cria um *socket* UDP e inicializa a estrutura **server_address** (com os campos **tipo de ligação,** endereço **IP** e **port do servidor**)
- Redireciona o SIGINT para assegurar que o *socket* é fechado corretamente.
- Envia uma mensagem de "HELLO" ao servidor para iniciar a comunicação, depois entra no *loop* de esperar e enviar pedidos do utilizador e receber e escrever resposta do servidor.

## 2. Comandos de utilizador

| ADMIN | | COMANDOS EXCLUSIVOS A ALUNOS | |
|---|---|---|---|
| **ADD USER <user_name> <password> <type>:** | Adiciona um novo utilizador (**type** pode ser aluno, professor ou admin) | **SUBSCRIBE_CLASS <class_name>:** | Inscreve o utilizador numa turma. |
| **LIST:** | Lista todos os utilizadores do sistema | **CREATE_CLASS <class_name> <size>:** | Permite que um professor crie uma turma. |
| **DEL <username>:** | Apaga um utilizador | **LIST_SUBSCRIBED** | Lista todas as classes às quais o utilizador está inscrito. |
| | | **COMANDOS EXCLUSIVOS A PROFESSORES** | |
| **QUIT_SERVER:** | Fecha a sessão de admin e ordena o encerramento do servidor | **SEND <class_name> <message>:** | Permite que um professor envie uma mensagem para uma turma. |
| | | **LIST_CLASSES** | Lista todas as turmas disponíveis. |
| | | **COMANDOS DE PROFESSORES E ALUNOS** | |
| **LOGOUT:** | Dá logout ao admin atual. | Faz logout do utilizador. | |
| **LOGIN <user_name> <password>:** | Autêntica o administrador | Autêntica o utilizador e ordena-o a juntar-se aos grupos multicast das turmas em que ele já está inscrito | |
| **HELP** : | Lista os comandos disponíveis para o admin | Lista os comandos disponíveis para os professores ou alunos | |

## 3. Configuração de rede com o GNS3



## 3.1 Comandos de configuração usados para os dispositivos

### Cliente 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
        up echo nameserver 192.168.0.1 > /etc/resolv.conf
```

### Servidor

```
SERVIDOR:
# 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
        up echo nameserver 192.168.0.1 > /etc/resolv.conf
```

### Cliente 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
        up echo nameserver 192.168.0.1 > /etc/resolv.conf
```

### Router 1

```
R1#config terminal

R1(config-if)# ip multicast-routing

R1(config)#  interface FastEthernet0/0
R1(config-if)#  ip address 193.137.100.126 255.255.255.128
R1(config-if)#  no shutdown
R1(config-if)# ip pim sparse-dense-mode
R1(config-if)#  exit

R1(config)#  interface FastEthernet0/1
R1(config-if)#  ip address 193.137.101.254 255.255.255.0
R1(config-if)#  no shutdown
R1(config-if)# ip pim sparse-dense-mode
R1(config-if)#  exit

R1(config)#  ip route 193.137.100.128 255.255.255.128 193.137.101.253
R1(config)#  ip route 10.5.2.0 255.255.255.192 193.137.101.252

R1#  copy running-config startup-config
R1#  write memory
```

### Cliente 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
        up echo nameserver 192.168.0.1 > /etc/resolv.conf
```

### Router 2

```
R2#  config terminal

R1(config-if)# ip multicast-routing

R2(config)#  interface fastEthernet0/0
R2(config-if)#  ip address 193.137.101.253 255.255.255.0
R2(config-if)#  no shutdown
R1(config-if)# ip pim sparse-dense-mode
R2(config-if)#  exit

R2(config)#  interface fastEthernet0/1
R2(config-if)#  ip address 193.137.100.254 255.255.255.128
R2(config-if)#  no shutdown
R1(config-if)# ip pim sparse-dense-mode
R2(config-if)#  exit

R2(config)#  ip route 10.5.2.0 255.255.255.192 193.137.101.252
R2(config)#  ip route 193.137.100.0 255.255.255.128 193.137.101.254
R2(config)# exit

R2# copy running-config startup-config
R2# write memory
```

### Router 3

```
R3#  config terminal

R1(config-if)# ip multicast-routing

R3(config)#  access-list 1 permit 10.5.2.0  0.0.0.63
R3(config)#  ip nat inside source list 1 interface FastEthernet0/1 overload

R3(config)#  interface FastEthernet0/1
R3(config-if)#  ip address 10.5.2.62 255.255.255.192
R3(config-if)#  ip nat inside
R3(config-if)#  no shutdown
R1(config-if)# ip pim sparse-dense-mode
R3(config-if)#  exit

R3(config)#  interface fastEthernet0/0
R3(config-if)#  ip address 193.137.101.252 255.255.255.0
R3(config-if)#  ip nat outside
R3(config-if)#  no shutdown
R1(config-if)# ip pim sparse-dense-mode
R3(config-if)#  exit


R3(config)#  ip route 193.137.100.0 255.255.255.128 193.137.101.254
R3(config)#  ip route 193.137.100.128 255.255.255.128 193.137.101.253
R3(config)#  exit


R3#  copy running-config startup-config
R3#  write memory
```