UNIVERSIDADE DE ÉVORA

Redes de Computadores - Trabalho Prático - Relatório

Trabalho realizado por:

Rui Roque nº 42720

Tomás Dias nº 42784

Implementação

Estruturas e funções base

Antes da implementação das funcionalidades pedidas, foram elaboradas as seguintes estruturas de dados e funções:

- **struct client**: Estrutura que define um cliente. Tem como atributos **i** (corresponde ao socket do cliente), **user** (indica se o cliente está conectado ao servidor com um nome definido), **auth** (indica se o cliente já se encontra autenticado no caso de se tratar de um utilizador registado), **nickname**, **password**, **role** (indica se o cliente é não registado, registado ou operador) e **reg** (utilizada para verificar se um cliente é ou não um utilizador registado).

- **client clients[]**: Lista com dimensão fixa (possível de ser ajustada a partir do define **NUM_CLIENTS**) que guarda os clientes conectados ao servidor.

- **char buffer[]**: Lista de dimensão 512 bytes que guarda o input do cliente (comando e parâmetros).

- **client search_client()**: Função que retorna um determinado cliente da lista de clientes. Recebe como argumentos a lista dos clientes (**clients[]**) e o socket do cliente (**i**).

- **client update()**: Função que atualiza a lista dos clientes. Recebe como argumentos a lista de clientes (**clients[]**) e o cliente atualizado (client) e retorna a lista de clientes atualizada.

- **client set_user()**: Função que atualiza o atributo **user** de um determinado cliente. Recebe como argumentos o cliente (**client**) e uma string (**c**) que caso contenha "**NICK**" atualiza **user** para o valor **1**. Caso contrário, atualiza para o valor **0**. Retorna o cliente atualizado.

- **client set_auth()**: Função que atualiza o atributo **auth** de um determinado cliente. Recebe como argumentos o cliente (**client**) e uma string (**c**) que caso contenha "**PASS**" atualiza **auth** para o valor **1**. Caso contrário, atualiza para o valor **0**. Retorna o cliente atualizado.

- **client set_registed()**: Função que atualiza o atributo **reg** de um determinado cliente. Recebe como argumentos o cliente (**client**) e um inteiro (**r**) que caso seja igual a **0** atualiza **reg** para o valor **0**. Caso contrário, atualiza para o valor **1**. Retorna o cliente atualizado.

- **client set_role()**: Função que atualiza o atributo **role** de um determinado cliente. Recebe como argumentos o cliente (**client**) e um inteiro (**r**) que caso seja igual a 0 atualiza **role** para o valor **"regs"**. Caso contrário, atualiza para o valor **"oper"**. Retorna o cliente atualizado.

Toda a informação sobre utilizadores registados (nickname, password e role) foi guardada no ficheiro **server_data.txt**.

Comandos de utilizador

NICK

Para além do uso de algumas das funções descritas anteriormente, esta funcionalidade utiliza também as seguintes funções auxiliares:

- **ascii()**: Função que verifica se o nome introduzido pelo cliente é composto apenas por letras ASCII. Recebe como argumentos o buffer (**nickname**) e retorna **1** se o nome apenas contém letras ASCII e **0** caso contrário.

- **check_nickname()**: Função que verifica se o nome introduzido pelo cliente já pertence a outro cliente conectado ao servidor. Recebe como argumentos a lista de clientes (**clients[]**) e o buffer (**buffer[]**) e retorna **1** se o nome está disponível e **0** caso contrário.

- **check_spaces()**: Função que verifica se o nome introduzido pelo cliente contém espaços. Recebe como argumento o buffer (**buffer[]**) e retorna **1** se o nome contém espaços e **0** caso contrário.

- **check_regs()**: Função que verifica se o nome introduzido pelo cliente está no ficheiro **server_data.txt**. Recebe como argumento o buffer (**buffer[]**) e retorna **1** se o nome consta na lista e **0** caso contrário.

- **check_regs2()**: Função que verifica se o nome introduzido pelo cliente é igual ao nome anteriormente introduzido pelo mesmo quando feito o registo. Esta verificação é feita no ficheiro **server_data.txt**. Recebe como argumento o nome antigo (**buffer[]**) e retorna **1** se o nome for igual e **0** caso contrário.

- **change_nick()**: Função que atualiza o nome com o qual o cliente se registou para o novo nome no ficheiro **server_data.txt**. Recebe como argumentos o nome antigo (**buffer[]**) e o novo nome (**new_nick[]**) e retorna **1** se a atualização foi bem-sucedida e **0** caso contrário.

MSSG

Para além do uso de algumas das funções descritas anteriormente, esta funcionalidade utiliza também a seguinte função auxiliar:

- **empty()**: Função que verifica se a mensagem introduzida pelo cliente é constituída apenas por espaços. Recebe como argumento o buffer (**buffer[]**) e retorna **1** se a mensagem apenas contém espaços e **0** caso contrário.

Foi também implementada a funcionalidade **EXIT** que permite ao cliente sair corretamente do servidor.

Comandos de utilizador registado

PASS

Para além do uso de algumas das funções descritas anteriormente, esta funcionalidade utiliza também as seguintes funções auxiliares:

- **authenticate()**: Função que verifica se a password introduzida pelo cliente é aquela com a qual este se encontra registado. É utilizado para esta verificação o ficheiro **server_data.txt**. Recebe como argumentos o cliente (**client**) e o buffer (**buffer[]**) e retorna **1** se a password é a correta e **0** caso contrário.

- **role()**: Função que verifica qual a role do cliente registado. É utilizado para esta verificação o ficheiro **server_data.txt**. Recebe como argumentos o cliente (**client**) e o buffer (**buffer[]**) e retorna **0** se o cliente é apenas um utilizador registado e **1** se o cliente é um operador.

Foi também implementada a funcionalidade **WHOS** e **LIST** (esta meramente descritiva pois apenas foi implementado o canal **default**).

Comandos de operador

KICK

Para além do uso de algumas das funções descritas anteriormente, esta funcionalidade utiliza também a seguinte função auxiliar:

- **kick_regs()**: Função que elimina as informações de um utilizador registado do ficheiro **server_data.txt**. Recebe como argumento o buffer (**buffer[]**) e retorna **1** se a eliminação foi bem-sucedida e **0** caso contrário.

REGS

Para além do uso de algumas das funções descritas anteriormente, esta funcionalidade utiliza também a seguinte função auxiliar:

- **regs_user()**: Função que adiciona informações de um cliente não registado ao ficheiro **server_data.txt**. Recebe como argumento o buffer (**buffer[]**) e retorna **1** se a adição foi bem-sucedida e **0** caso contrário.

OPER

Para além do uso de algumas das funções descritas anteriormente, esta funcionalidade utiliza também a seguinte função auxiliar:

- **oper_user()**: Função que atualiza a **role** de um utilizador registado para "**oper**" no ficheiro **server_data.txt**. Recebe como argumento o buffer (**buffer[]**) e retorna **1** se a atualização foi bem-sucedida e **0** caso contrário.

QUIT

Para além do uso de algumas das funções descritas anteriormente, esta funcionalidade utiliza também a seguinte função auxiliar:

- **quit_oper()**: Função que atualiza a role de um operador para "**regs**" no ficheiro **server_data.txt**, passando a ser apenas um utilizador registado. Recebe como argumento o cliente operador (**client**) e retorna **1** se a atualização foi bem-sucedida e **0** caso contrário.

Função main() do server.c

Foi utilizado como base o **server_echo_select.c** implementado durante as aulas práticas para a construção da **main**, realçando-se a utilização das funções **socket()**, **setsockopt()**, **bind()**, **listen()**, **select()**, **accept()**, **send()** e **recv()**. Estas duas últimas foram muito importantes para a troca de mensagens entre servidor e cliente.

A utilização das estruturas de dados **fd_set** e **struct sockaddr_in** também foram fundamentais para a implementação do servidor.

Para o funcionamento dos comandos foram utilizados, para além das funções descritas na secção anterior, ciclos e condições.

Função main() do client.c

A implementação da **main** teve como base o uso das funções **socket()**, **connect()**, **send()** e **read()**. Estas duas últimas, tal como em server.c, foram as responsáveis pela troca de mensagens entre cliente e servidor.

Também foi utilizada a estrutura de dados **struct sockaddr_in**.

Para o funcionamento da **main** foram igualmente utilizados ciclos e condições, destacando-se o ciclo para a leitura de comandos e as condições correspondentes às respostas do servidor.

Notas importantes e problemas encontrados

Para a compilação e teste do código devem ser usados os comandos:

- **gcc -o server server.c** para o server.c
- **gcc -o client client.c** para o client.c

Deve se executar em primeiro lugar o servidor e depois o cliente(s).

A interface do cliente é constituída por linhas de comando identificadas pelo símbolo “> ” .

Para um cliente se desligar do servidor corretamente deve-se sempre utilizar o comando **EXIT**.

Após definido o nome do utilizador através do comando **NICK**, é aconselhável “saltar” sempre uma linha de código (através da tecla **enter**) antes da execução de qualquer outro comando de modo a apresentar possíveis mensagens pendentes por parte do servidor (mensagens relacionadas com novos utilizadores, mensagens enviadas por outros utilizadores, etc.).

O ficheiro **server_data.txt** deve separar os valores dos atributos nickname, password e role por um espaço e acrescentar um espaço a seguir ao último valor, neste caso, a role. Deve ainda ser acrescentado uma linha em branco após a linha que contém a informação do último utilizador registado. Má formatação deste ficheiro pode originar erros no programa.

Não foi possível implementar os restantes canais, existindo apenas o canal default. A correção de erros também não foi implementada.