Relatório do Trabalho Prático
Redes de Computadores

Pedro Claudino (nº39870)
Duarte Anastácio (nº40090)
Ludgero Teixeira (nº41348)

Professor: Pedro Patinho

Julho de 2021

1 Introdução

Para a realização deste trabalho foi nos apresentado o seguinte problema.

A empresa UltraSecure, Lda. pretende utilizar um sistema de comunicação para os membros da sua equipa, mas não pode confiar em software de terceiros.
Assim sendo, precisa de implementar "in-house" um software deste tipo, garantindo que não há possibilidade de falhas de segurança nesta aplicação.

Para ajudar a UltraSecure na implementação deste software, deve-se implementar as seguintes componentes:

Sistema de chat

- Comunicação em tempo real 1-1: um cliente envia mensagens para outro cliente, que as recebe em tempo real.
- Comunicação em tempo real 1-todos: um cliente envia mensagens para um canal comum, e todos os outros clientes recebem as mensagens em tempo real.
- Comunicação em diferido 1-1: um cliente envia mensagens para um outro cliente que não está online, e este receberá as mensagens mais tarde, quando se ligar ao servidor.

Sistema de transferência de ficheiros

- Transmissão de ficheiros nos três métodos referidos anteriormente: 1-1 e 1-todos em tempo real e 1-1 em diferido.

Sistema de jogo

- Jogo do galo (jogo escolhido por nós para implementar)
- Batalha Naval
- Xadrez

2 Abordagem

O objetivo deste trabalho é o desenvolvimento de um sistema de comunicação (servidor/cliente), sendo que optamos por implementar o mesmo num protocolo TCP.

O servidor foi implementado de maneira a gerir vários clientes em simultâneo usando **select** e posuindo vários canais como:

- **Default:** Canal geral para as pessoas que não têm um 'Username' associado a elas.
- **Geral:** Canal geral para as pessoas que têm um 'Username' associado a elas.
- **Privado:** Canal de comunicação privado de 1-1 entre 2 clientes.
- **Galo:** Canal onde 2 clientes podem jogar em tempo real o jogo do galo.

Os clientes funcionam em dois modos:

- Clientes com um username, onde podem comunicar num canal geral e mandar mensagens em privado para outros clientes registados.
- Clientes sem username, onde apenas funcionam no canal 'default'.

Um cliente com um username inicia no canal preferido **Geral**, e um cliente que não possua um username inicia no canal preferido **Default**

Como não foi pedido, não é realizado qualquer tipo de autenticação de um cliente, sendo que 'confiamos' que um determinado cliente diz quem é realmente ser.

Um cliente pode:

- Mandar mensagens para um canal geral 'Default' (caso não tenham um username).
- Mandar mensagens para o canal 'Geral' (caso possua um username)
- Mandar mensagens privadas para outros clientes (caso possua um username)
- Enviar ficheiros para outros clientes (caso possua um username)
- Jogar em conjunto com outro jogador o jogo do galo (caso possua um username)

Para a implementação do sistema de jogo entre 2 clientes, optamos pelo jogo do galo visto que já tinha sido anteriormente implementado por nos num trabalho anterior.

3 Persistência dos dados

Para um melhor desempelho achamos que os dados dos clientes deveriam ser guardados de forma persistente.

O servidor tem um sistema que guarda os utilizadores registados (com um Username) em disco num ficheiro chamado "2hashtable.dat". A escrita e a navegação pelo ficheiro é feita como numa *hashtable* e a chave para o índice de uma posição é feita a partir do nome do cliente.

4 Início do servidor

O primeiro passo no início do servidor é a inicialização dos 4 canais principais e um canal reservado para os clientes sem nome definido, a função que realiza essas inicializações chama-se ***inicialize_canals.***

De seguida é feito os passos para o início do servidor no porto TCP 5555, realizado na função ***setup_server***. Nesta função é criado o *socket* do servidor assim como definir o socket para permitir múltiplas conexões, *bind* o *socket* para o localhost e porto 5555 e prepará-lo para aceitar novas conexões.

Por último, limpa a lista de descritores dos *sockets* e adiciona o *socket* do servidor à lista.

5 Ciclo infinito do servidor

O ciclo infinito do funcionamento do servidor consiste em:

1. Limpar a lista dos descritores dos *sockets*;
2. Adicionar novamente todos os *sockets* do sistema à lista des descritores;
3. Esperar até algum *socket* dum cliente se tornar ativo;
4. Ver se existe novos clientes e adicioná-los ao canal dedicado aos clientes sem nome;
5. Ver se existe novos pedidos nos canais.

6 Canal

Cada canal está implementado como uma *linkedlist* em que os seus nós são os utilizadores presentes nesse canal. Por isso, para a realização de novos pedidos é percorrido todos os nós (utilizadores) da *linkedlist*(canal) à procura do utilizador que fez o pedido.

6.1 Realização de pedidos num canal

Se durante o passo referido no último parágrafo for encontrado um *socket* com um pedido então o servidor recebe esse pedido. Contudo se o número de bytes lido for 0, isso significa que o utilizador saiu do servidor. Para eliminar esse utilizador do canal onde estava presente, é preciso fechar o seu *socket* e removê-lo do canal onde estava. Se o número de bytes lidos não for 0 é realizado o pedido.

De seguida é visto que tipo de pedido corresponde e este é realizado, no final é enviado uma resposta para o utilizador que fez o pedido e também, se for o caso, enviar para os clientes de um canal ou para todos os clientes no servidor. O envio da resposta ao utilizador é feito pela função ***send_response*** implementado no ficheiro ***aux_functions***.

Os diferentes pedidos implementados foram:

- **USERNAME**, implementado na função ***USERNAME_ request***, atribui ou muda o nome do cliente.

  O procedimento do pedido é feito na seguinte forma:

  1. Recolher o nome que está no argumento;
  2. Ver se o nome é válido, ou seja, se:
     - não é vazio;
     - tem caracteres senão letras ASCII e algarismos;
     - já está em uso por algum cliente.
  3. Ver se o cliente tem já o nome definido:
     - Caso não tiver o nome definido é um novo utilizador e assim é definido a mensagem a informar do novo utilizador, inserido o novo utilizador no canal "default"como utilizador não registado e por fim removido do antigo canal dos novos utilizadores.

- **MSSG**, implementado na função ***MSSG_ request***, envia uma mensagem para o canal ativo.

  O procedimento do pedido é feito na seguinte forma:

  1. Recolher a mensagem que está no argumento;
  2. Ver se é válida, ou seja, se:
     - não é vazio;
     - é maior que 512 bytes.
  3. Construir a mensagem a enviar.

- **JOIN**, implementado na função ***JOIN_ request***, mudança de canal ativo.

  O procedimento do pedido é feito na seguinte forma:

  1. Recolher o nome que está no argumento;
  2. Definir e enviar mensagem de saída para o canal ativo;
  3. Sair do canal ativo;
  4. Inserir o cliente no novo canal.

FILE, implementado na função***FILE_ request***, envia uma mensagem para o canal ativo.
O procedimento do pedido é feito na seguinte forma:

1. Recolher o nome que está no argumento;
2. Ver existe erros, ou seja, se:
   - o ficheiro não existe;
3. Definir e enviar mensagem de saída para o canal ativo;

GALO, implementado na função ***GALO_request***

O procedimento do pedido é feito na seguinte forma:

1. Recolher o nome que está no argumento;
2. Manda o tabuleiro ao cliente e pede a jogada;
3. Sempre que uma mensagem vier do canal 'GALO', o servidor envia a jogada e o tabuleiro atualizado para o adversário.

7 Cliente

O primeiro passo para a entrada do cliente é ler a informação sobre o servidor no localhost no porto TCP 5555. De seguida é criado o *socket* do cliente e conecta com o servidor. É ainda criado uma *thread* com o objetivo de estar sempre a receber informação do servidor e a mostrar ao cliente.

Por fim o cliente estará num loop em que, caso o cliente escreva um pedido, este ser sempre enviado. Foi ainda implementado uma funcionalidade para sair do loop, e também do servidor, que consiste na escrita unicamente do caractere 'E'.

8 GCC comandos

Compilação cliente.c : > gcc -Wall -pthread client.c -o client
Compilação server.c : > gcc -Wall canal.c aux_functions.c server.c hashtable.c -o server
Execução cliente.c : > ./client
Execução server.c : > ./server
A compilação e execução foi feita no sistema UNIX - LINUX Ubuntu 20.04.2 LTS e em Macos Catalina 10.15.7 .

9 Conclusão

Em suma, embora a implementação total das funcionalidades do trabalho não tenham sido implementadas com sucesso, como a funcionalidade de transferência de ficheiros, que embora implementada, não funciona como esperado, visto que o servidor lê o ficheiro mas não consegue escrever no mesmo, a funcionalidade de chat em tempo real 1-1 em diferido visto que não conseguimos armazenar as mensagem de um cliente que se encontre offline no momento em que lhe são enviadas mensagens para mais tarde que o mesmo se ligar poder vê-las, achamos que a realização deste trabalho teve um balanço positivo, pois embora perante algumas dificuldades, procuramos sempre meter em prática maior parte do conhecimento por nós adquirido tanto nas componentes práticas como teóricas desta unidade curricular.