[Sistemas Operativos]
2016/2017

Jogo de Futebol

Trabalho Prático de SO

Rafael Henriques 21250203
Rui Teixeira 21250189

Índice

1. Introdução

Neste relatório, consta as conclusões, esboços e raciocínios utilizados na estruturação e execução do trabalho prático de SO. Com base na matéria lecionada nas aulas de Sistemas Operativos e fóruns, como por exemplo http://stackoverflow.com/, o grupo tentou escrever o programa por forma a aplicar os conceitos lecionados.

Por não ser um programa grande e preocuparmo-nos mais em aplicar os conceitos de SO, algumas das funções são muito extensas.

2. Interface Gráfica

Neste tópico é mostrado um pouco da iteração com o programa. Os seguintes exemplos mostram a GUI com descrição, em 2.1. do Cliente e em 2.2. do Servidor.

2.1. Cliente

A interação do Cliente é apresentada através de uma GUI, desenvolvida com ncurses. Está divida em duas janelas, a do lado esquerdo, a verde, representa o campo do jogo e do lado direito, a janela de interação via comandos e logs.

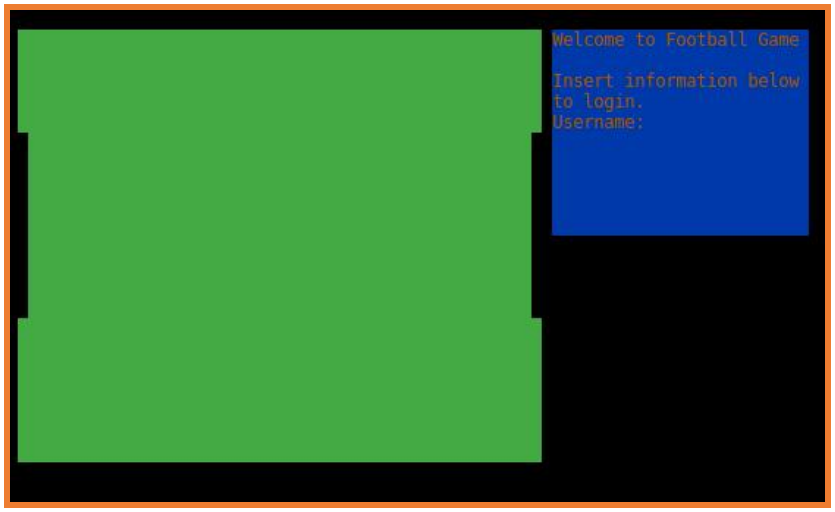

Figura 1 – Status da Janela Inicial após ligar o Client

- Ao correr o Cliente no terminal com o comando *./client* é mostrado como output a figura acima.
- O campo a verde é onde o jogo irá decorrer, os jogadores azuis do lado esquerdo e os vermelhos do lado direito.

- Na janela azul é onde o jogador faz o login, escolhe jogador, equipa e recebe alguns outputs do servidor.

Figura 2 – Jogo a Decorrer

- Para mover o jogador tem de utilizar as teclas:
  - W – Cima;
  - S – Baixo;
  - A – Esquerda;
  - D – Direita;
  - Q – Esquerda Cima;
  - E – Direita Cima;
  - Z – Esquerda Baixo;
  - X – Direita Baixo.
- Caso queira sair do jogo de futebol, em que está a participar:
  - K – Sair do Jogo (continua no Cliente);
  - SPACE – Rematar para a baliza do adversário;
  - SPACE + <número do jogador da equipa> - passar a bola;
- Para passar a bola clique no número dos respetivos jogadores da equipa, que estão indicados na GUI. Se quiser rematar clique tecla 0.
- Na janela azul de comandos pode escrever:
  - Introduzir ID e password válido para estabelecer ligação com o servidor;
  - Para jogar escreva:
    jogar <número equipa 0 ou 1> <número jogador da equipa>;
  - Para sair do jogo escreva:
    sair

2.2. Servidor

A interação com o servidor é pelo terminal, onde respeita os comandos do enunciado do trabalho prático.

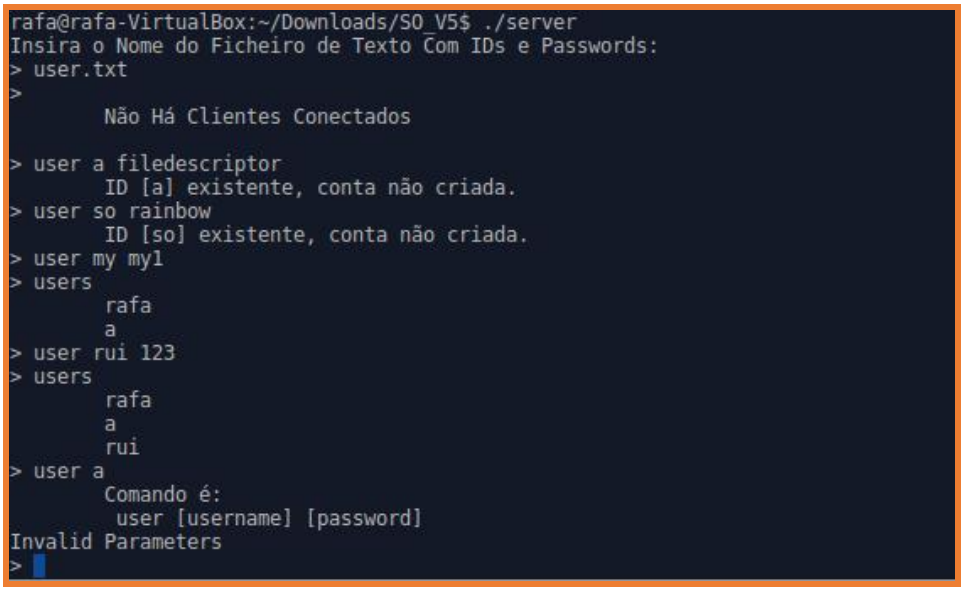

Figura 3 – Server

- Para correr o Servidor no terminal é necessário escrever *./Server* no terminal.
- É necessário adicionar o ficheiro de texto que contém os ids e password, antes de iniciar o servidor. (Exemplo: Ficheiro chama-se *user.txt*)
- O servidor está pronto a receber comandos conforme o enunciado. Em caso de comando inválido ou sem sucesso, o programa irá retornar um Output para melhor interação.

3. Estruturação dos Dados

Tentou-se utilizar os conceitos de Sistemas Operativos, o mais correto possível. O trabalho tem 4 ficheiros: server, serverH, client e useful. O ficheiro useful é um conjunto de estruturas e declarações que o Cliente e o Servidor têm em comum.

Nos subtópicos seguintes irá ser esclarecido cada um deles.

3.1. Comunicação Client-Server

Para estabelecer ligação cliente-server são utilizados Named Pipes (FIFO) e Sinais presentes no Sistema Operativos Unix.

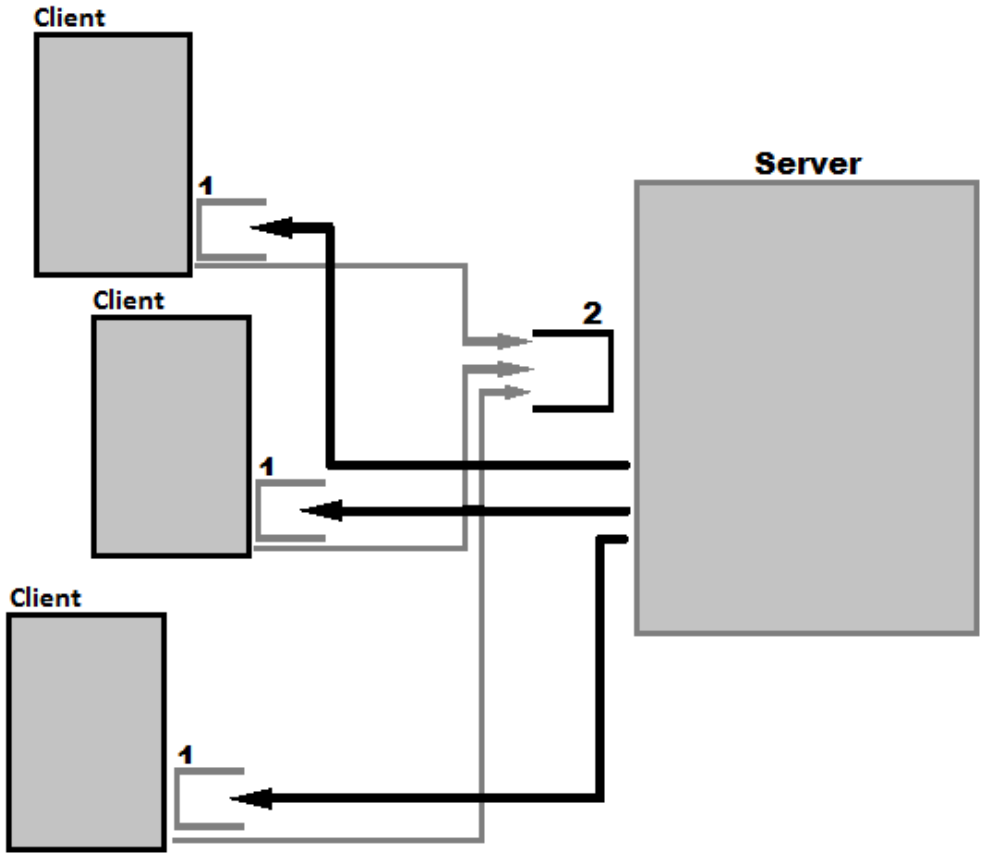

Figura 4 – Esquema de ligação entre Clientes e Servidor.
1 – Client FIFO; 2 – Server FIFO

O Servidor tem apenas um Pipe (2) em que recebe toda a informação dos clientes. Esta informação é distinguida através do Nome do Pipe que identifica o Cliente. O servidor pode enviar um sinal SIGUSR1 através do tratamento de sinais sigqueue/sigaction. Não era necessário ser deste modo, mas ao fazer isto aplicamos a matéria das Aulas.

Cada cliente tem apenas um Pipe (1), tal como o Servidor, onde recebe informação deste. Como o Servidor tem sempre o mesmo nome do Pipe e apenas recebe deste, não é necessário distinguir o emissor, pois é sempre o mesmo Servidor.

3.2. Cliente

O programa do Cliente é constituído por 2 ficheiros, sendo um dele o ficheiro *useful.h*, comum entre o Cliente e o Servidor. Os outro ficheiro é *server.c*.

3.2.1. Named Pipes

O programa do cliente está estruturado para ter apenas um Named Pipe. A informação é recebida e enviada respeitando a estrutura presente no

ficheiro comum entre o cliente e o servidor. Uma explicação mais detalhada deste ponto é apresentada no tópico 3.4.

3.2.2. Sinais

O cliente atende a três tipo de sinais. SIGINT e SIGHUP sendo que os dois fecham o cliente. O Named Pipe é fechado, apagado e a Thread de atualização do campo terminada. Esta medida foi tomada para garantir a eliminação do Named Pipe.

O cliente também pode receber um sinal SIGUSR1, enviado pelo servidor através do modelo *sigqueue-sigaction*, caso o servidor seja encerrado através de um sinal. Achamos que era a forma mais eficiente, pois se o servidor é fechado de forma assíncrona, logo a forma mais rápida de fechar os clientes seria enviar-lhes igualmente sinais.

3.2.3. Threads

O cliente tem apenas uma Thread. Esta é responsável pela atualização do campo de futebol, para que o cliente tenha uma perceção dinâmica do que ocorre no jogo.

O programa principal trata da validação dos comandos e teclas do jogador, enviando-as para o Servidor.

3.3. Servidor

O programa do Servidor é constituído por 3 ficheiros, sendo um deles o ficheiro *useful.h*, comum entre o Cliente e o Servidor. Os outros ficheiros são o *server.c* e *serverH.h*.

3.3.1. Named Pipes

O programa do servidor está estruturado para ter apenas um Named Pipe. A informação é recebida e enviada respeitando a estrutura presente no ficheiro comum entre o cliente e o servidor. Uma explicação mais detalhada deste ponto é apresentada no tópico 3.4.

3.3.2. Sinais

O servidor atende a três tipos de sinais. SIGINT e SIGHUP e SIGUSR1, ambos chamam a mesma função. Esta medida foi tomada por precaução, caso o servidor seja terminado com Ctrl + C ou o terminal diretamente fechado. No caso do SIGURS1 é um requisito do trabalho prático.

Caso o servidor recebe um destes sinais, enviará um sinal SIGUSR1 para o cliente.

3.3.3. Threads

O servidor tem um número considerável de Threads, sendo este responsável por processar toda a lógica do jogo, validações das ações dos jogadores, assim como processamento da IA dos BOTS, logins de clientes e comandos de Controlo de Jogo.

Os tipos de Threads a poderem ser geradas pelo servidor são:

- BOTS Defesas;
- BOTS Atacantes;
- BOTS Guarda-Redes;
- Leitura e Resposta da informação dos Clientes;
- Processamento das Jogadas Efetuadas pelos Clientes e pelos BOTS;
- Duas Threads que acertam os timings dos jogadores (defesas 4 em 4, Guarda-Redes e Avançados 3 em 3);

O programa principal trada da validação dos parâmetros de Controlo de Jogo enviados pelo Administrador.

3.4. Dados Partilhados

Para este trabalho, por forma a garantir que as estruturas e dados comuns entre o cliente e servidor estão corretos, utilizou-se um header file *useful.h*.

O ficheiro contém defines de todos os valores a configurar e as estruturas comuns entre o cliente e o servidor. A struct tipo é a estrutura utilizada nas comunicações cliente-servidor através dos Named Pipes. Essa estrutura contém duas variáveis, a primeira é um valor inteiro do tipo de estrutura a ser enviada e a segunda é uma Union que contém as estruturas possíveis a ser recebidas, pelo cliente.

A cada possível estrutura faz corresponder um valor inteiro. As estruturas são as seguintes:

- Se o jogador conectou-se ou não, sendo uma resposta do servidor ao pedido de Login;
- Os dados enviados pelo Cliente para proceder ao login (id, password e identificação do pipe para receber respostas);
- Estrutura de atualizar o jogo, que contém duas estruturas uma sobre os jogadores e outra com informações sobre a bola;
- Jogadas (ações);
- Pedido do cliente para jogar, identificando qual o jogador e equipa a que pretende controlar;
- Resposta do servidor ao pedido do cliente de controlar determinado jogador;

- Quando um jogador sai do jogo por Cartão Vermelho enviado pelo servidor;
- Resultado do Jogo atualizado.

Esta é a informação estipulada que pode ser transmitida entre Cliente-Servidor, para este Projeto.

4. Algoritmos

Para conseguir criar uma "mini" Inteligência Artificial para os BOTS e para a mecânica de Jogo foram criadas algumas funções que devolvem:

- O número do Jogador mais perto (da sua equipa);
- Distância relativamente ao jogador mais perto;
- 2 Arrays com as 8 Coordenadas da vizinhança;
- Distância entre dois pontos;
- Devolve a jogada para a Coordenada do turno seguinte, seguindo uma trajetória. Por exemplo o jogador quer ir de (2,3) para (6,3),
  a função devolve a jogada para a coordenada (3,3), caso seja válida;

Para os BOTS escolherem a jogada, há uma estrutura no programa que é apenas necessário indicar a jogada, passe ou remate sem ser necessário proceder a essas verificações na thread dos BOTS.

5. Makefile

Na figura seguinte, está presente o código do Makefile que foi utilizado para compilar o projeto. Consideramos que esta foi das partes mais interessadas deste trabalho, pois mostrou-nos que não necessitamos de "depender de um botão ou shortcut de teclado" num IDE, para compilar e gerar os executáveis dos nossos projetos.

```
all: useful client ServerH server

useful: useful.h
	gcc useful.h -o useful

client: client.c
	gcc client.c -o client -pthread -lncurses -lm

ServerH: ServerH.h
	 gcc ServerH.h -o ServerH

server: server.c
	gcc server.c -o server -pthread -lm
```

Figura 5 – Makefile

Um Makefile muito simples e específico para o projeto desenvolvido, é o que está a ser utilizado.

Tentamos utilizar outros scripts mais genéricos, que compilassem qualquer quantidade de ficheiro de um projeto c dentro de uma pasta. Vimos que para compilar vários ficheiros num executável, era necessário gerar o código objeto de cada um. Depois de gerado os códigos objetos, procedia-se à Linkagem entre eles por forma a gerar um executável. Chegamos a dividir o cliente e o servidor em 2 pastas e cada um deles com vários ficheiros .c mas a maioria dos Makefiles que utilizamos não compilavam corretamente. Tentamos Makefiles genéricos e específicos para este projeto.

Acabamos por deixar apenas um file .c para o server e um .c para o cliente e abandonamos a ideia do Makefile genérico.

Link principais que recorremos:

https://www.cs.bu.edu/teaching/cpp/writing-makefiles/
https://www.gnu.org/software/make/manual/make.html

6. Conclusões

O Trabalho Prático foi desenvolvido por um grupo duas pessoas, sendo que, pessoalmente, consideramos que tivemos pouco tempo para o desenvolver.

Quando demos inicio ao Trabalho Prático, era para o desenvolvermos utilizado a framework do Qt Creator para criação de uma GUI. Como tínhamos de aprender alguns conceitos de Event-Driven Programming decidimos não o fazer.

Verificamos que para construir uma inteligência artificial, mesmo para programas 2D, é uma tarefa trabalhosa. Aplicados apenas um conjunto de condições e ciclos para os jogadores mostrarem um pouco de dinamismo e não tornar o jogo monótono. O código tem funções muitos grandes podendo ser repartido em muitas mais funções, podendo reutilizá-las e tornar o código mais intuitivo.

Relativamente à comunicação Cliente – Servidor, e da informação transmitida pelos Named Pipes:

- Estrutura que contém um inteiro com um valor, e consoante esse valor tem uma Union em que umas das suas Estruturas é a transmitida achamos que era a solução mais apropriada para não criar mais que um Named Pipe.
- Envia para os Clientes sempre uma estrutura com o mesmo tamanho que é máximo (a estrutura que ocupa mais bytes da Union).

A quantidade de Threads é razoável para este Projeto e não houve necessidade de utilizar Mutex. Foram as que pensamos por forma a realizar o Trabalho Prático.

Em suma, o Trabalho Prático desenvolvido é a solução que propomos para o problema proposto, com base no tempo que dispusemos e do nosso conhecimento até à data.