# Relatório do Trabalho Prático
# de
# Sistemas Operativos

**2ºAno - 1ºSemestre - 2020/2021 - Meta Final**


**Trabalho Realizado por:**
Daniel Moreira Ribeiro - 2017013425
João Miguel Neto Lopes - 2017010484

# Índice

# Introdução

Este trabalho consistia em que fosse desenvolvido um programa que poderia ter vários utilizadores ao mesmo tempo, que iriam jogar um campeonato gerido por um árbitro que faz a ligação dos utilizadores com os jogos , sendo que no final do campeonato seria atribuída uma pontuação a cada utilizador.

# Ficheiros

## MakeFile

As mudanças neste ficheiro não mudaram muito ao longo do tempo mas nesta versão final removemos a linha que apagaria o jogadores.bin optamos por esta mudança pois desistimos de usar ficheiros para armazenar os jogadores , optando agora por um array.

Dito isto a constituição manteve se e pode executar os seguintes comandos

- make all: que irá fazer a compilação de todos os ficheiros '*.c'(sendo eles o cliente.c arbitro.c e o jogo que é g_001.c) ,caso já tenhamos feito alguma compilação ele só irá compilar caso tenha havido mudanças no ficheiro.
    - cliente: cria o executável compilando o cliente.o
        - cliente.o: cria os objeto do cliente.c
    - árbitro: cria o executável compilando o arbitro.o
        - arbitro.o: cria os objeto do arbitro.c
    - jogo
        - jogos/g_001: cria o executável compilando o g_001.o
            - jogos/g_001.o: cria os objeto do g_001.c
- make clean_o: apaga todos os ficheiros '.o'
- make clean: apaga todos os executáveis e ficheiros '.o'

## Cliente

O cliente consegue conectar-se ao servidor e mandar um nome para validação. Se o nome já tiver sido usado, será recusado e será pedido um novo username até o nome ser aceito ou fechar a sua interface com o comando shutdown.Consegue também ver qual o jogo atribuído e sair do campeonato .

### Cliente.h

Inclui as bibliotecas stdio.h, getopt.h, ctype.h,stdlib.h, signal.h, fcntl.h, sys/types.h, sys/stat.h ,unistd.h string.h e a inclusão das estruturas.h e inicializa a estrutura JOGADORES

### Cliente.c

Definimos várias variáveis globais e flags (l.2 a l.14)

Temos várias funções MSG de INFO(l.103) ou de ERRO(l.17 a l.45)

No main começamos por inicializar 2 threads, de seguida faz validação de argumentos,configuração de sinais para posterior uso seja como exemplo fechar em segurança,depois criamos o fifo do cliente e do árbitro e de seguida abrimos o do árbitro para escrita e o do cliente para escrita e leitura, depois criamos a thread threadTestaConexaoComArbitro que testa a conexão com o árbitro.

Depois de estar conectado ao árbitro é lhe perguntado que username quer ter ele decide e se o árbitro validar ele avança se não será lhe pedido outro nome, pois este já existia. Depois entra num ciclo de envio de comandos para o árbitro , que irá continuar até o campeonato acabar ou levar um SIGUSR2 que o retira do campeonato ou o comando shutdown que fecha o programa fora deste ciclo temos ainda uma thread para caso se perca a conexão com o árbitro ele procure durante 30 seg ,quando esse tempo acabar ele morre.

## Árbitro

### Árbitro.h

Inclui algumas bibliotecas o valor default do MAXPLAYERS e do GAMEDIR e ainda algumas variáveis constantes.

### Árbitro.c

Definimos várias variáveis globais e flags (l.2 a l.30)

Temos várias funções MSG de ERRO(l.31 a l.98), de INFO(l.99 a l.211), de UI(l.213 a l.247) e de AJUDA (l.229 a l.234)

Possuímos uma função para limpar a lista dos jogadores ao findar um campeonato.(l.236 a l.246)

Possuímos um handler para o SIGINT para que se possa terminar o árbitro em segurança.(l.248 a l.259)

Possuímos uma função que atualiza os jogos Disponíveis no GAMEDIR (l.266 a l.271)

Possuímos uma função que verifica se o nome do jogador está a ser utilizado (l.279 a l.286 )

Possuímos uma função que expulsa (dá kick) a um jogador indicado no argumento da mesma(l.288 a l.301)

Possuímos o handler de um alarme que faz countdown para o início do campeonato/ final do lobby(l.303 a l.315)

Possuímos uma função que verifica qual a posição do jogador na lista correspondente aos jogadores logados(l.317a l.324)

Possuímos um handler de um alarme que faz countdown para o fim do campeonato(l.326 a l.338)

Possuímos uma função que encontra o jogo atribuído ao jogador em questão(l.340 a l.351)

Possuímos uma função que devolve um jogo random para que futuramente possa ser associado ao jogador(l.353 a l.356)

Possuímos uma função responsável pelo atendimento dos comandos clientes (que corre em thread) em que se faz a verificação da validade do id do jogador e se distinguem as mensagens vindas do cliente (verificação de username / comandos / mensagens para o jogo)(l.358 a l.517)

Possuímos uma função que faz a gestão do campeonato nomeadamente a ativação de certas flags e dos alarmes já mencionados neste documento (que corre em thread) (l.520 a l.536)

Possuímos um menu administrador (que corre em thread) que é responsável pelos comandos que provém do administrador do árbitro(l.539 a l.582)

A função main faz as verificações necessárias para que se possa iniciar o campeonato de forma segura com as variáveis de ambiente e os argumentos requisitados pela aplicação. Aqui encontra-se também a iniciação dos FIFOs e a chamada das threads da aplicação.(l.584 a l.714)

## Jogos

Temos ao todo 3 jogos sendo dois que não existiam nas metas anteriores.

### jogos.h

Inclui as bibliotecas stdio.h,stdlib.h,time.h,signal.h e string.h e a inclusão das estruturas.h

### g_001.c

Implementação de um jogo em que o utilizador tem de adivinhar o distrito a que aquele concelho pertence ,exemplo: se o programa “perguntar” em que distrito

fica Condeixa a resposta correta será Coimbra,o programa dá o feedback caso o utilizador tenha respondido certo ou errado e no final atribui uma pontuação via exit().O jogo pode ser terminado com o signal SIGUSR1;

### g_002.c

Implementação do jogo par ou ímpar o jogador escolhe par ou ímpar e de seguida um número se o seu número e o número gerado pelo jogo derem a escolha(par , impar) ganha e no final atribui uma pontuação via exit().O jogo pode ser terminado com o signal SIGUSR1;

### g_003.c

Implementação do jogo que o jogador tem de acertar a soma de dois números no final atribui uma pontuação via exit().O jogo pode ser terminado com o signal SIGUSR1;

## Estruturas

### estruturas.h

- **JOGADOR:**
  - pid_cliente - id do processo do cliente
  - nomeID - nome único de utilizador
  - validID - Serve de validação para o nomeID
  - cmd - input que o jogador pode enviar.
- **ARBITRO:**
  - msg - input que o árbitro possa enviar
- **JOGO:**
  - nomeJogo - nome do jogo associado ao jogador
  - pid_jogo - id do processo do jogo
  - msg - input que o jogo possa enviar
- **CAMPEONATO:**
  - pid_cliente - id do processo do cliente
  - nomeJogador - nome único de utilizador
  - nomeJogo - nome do jogo associado ao jogador
  - pid_jogo - id do processo do jogo
  - pontuacao- pontuação do jogador
- **Criação de FIFOS:**
  - FIFO_ARBITRO “/tmp/fifo_arbitro”.

    - FIFO_CLIENTE “/tmp/fifo_cliente_%d” %d será preenchida pelo pid do cliente.
- #define vários “exit” para sabermos qual foi o erro.