Departamento de Engenharia Informática e de Sistemas
Instituto Superior de Engenharia de Coimbra
Licenciatura em Engenharia Informática

[Sistemas Operativos]
2020/2021

Sistema Champions

2016013407 – Paul Ionut Bob

Índice

Introdução

Este documento tem como objetivo relatar as principais implementações efetuadas ao longo do trabalho pratico da cadeira de Sistemas Operativos.

É importante referir que este trabalho prático foi realizado ao longo de vários meses, em simultâneo com a assimilação de conhecimento proveniente tanto das aulas teóricas como das práticas. Posto isto, algumas implementações foram feitas com base no conhecimento adquirido até à altura em questão e certamente que algumas delas seriam repensadas e implementadas de forma diferente no presente.

Sendo eu o único contribuidor neste projeto e tendo as dimensões que tem, eventualmente existem algumas partes de código que deixaram de ser utilizadas e não foram apagados por lapso. Alguns excertos de código foram inspirados nas aulas práticas do professor Luís Santos, outros foram retirados da internet, tendo sempre o cuidado de entender toda a lógica por trás do funcionamento desses excertos. O restante código (quase a totalidade do projeto) é da minha autoria.

Estrutura e Organização

Visto que os header files sofreram alterações desde a entrega da última meta, segue a apresentação de todos os ficheiros que constituem tanto o árbitro como o cliente.

- arbitro.h
  Contém alguns defines de variáveis *default* utilizadas pelo árbitro e a declaração de funções utilizadas pelo mesmo.

- auxiliar.h
  Contém a declaração de funções auxiliares tanto ao árbitro como ao cliente.

- cliente.h
  Contém declarações de funções que são utilizadas apenas pelo cliente.

- comandos.h
  Contém a declaração das funções utilizadas pelo árbitro para interpretar comandos, tanto do administrador como comandos provenientes dos jogadores.

- communication.h
  Contém alguns defines e declarações de estruturas + funções utilizadas para estabelecer e efetuar comunicações entre árbitro <-> jogador.

- memDinamica.h
  Contém declaração de funções utilizadas para manipular memória dinâmica sob a forma de listas ligadas.

- structs.h
  Contém as estruturas utilizadas para guardar informações acerca do estado do servidor e dos jogadores / jogos.

- threads.h
  Contém a declaração das threads.

Cada header file tem o seu próprio file.c que contém o código de todas as funções declaradas.

- Estruturas de comunicação

```
//pedido ao arbitro pelo cliente
typedef struct MensagemClienteParaServidor {
        Jogador dados;
        char mensagem[MSG_BUFFER];
} PedidoJogador;

//resposta do arbitro ao cliente                          //Erros
typedef struct MensagemServidorParaCliente {            // -1 já existe esse nome
        char mensagem[MSG_BUFFER];                      // -2 campeonato full
        int erro;                                       // -3 nome banido
        int espera;  //segundos até novo campeonato     // -4 Campeonato acabo.
} RespostaArbitro;                                      // -5 campeonato a decorrer
```

- Estruturas de dados

```
typedef struct game Jogo, *pJogo;
struct game{
        char nomeJogo[PATH_MAX];
        char path[PATH_MAX];
        pJogo prox;
};

typedef struct cliente Jogador, *pJogador;  //estrutura Jogador
struct cliente{
        char nome[MAX_NAME];        //nome do jogador
        pid_t pidJogador;           //pid do jogador
        int score;                  //pontuação do jogador
        pJogador prox;              //ponteiro para criação de listas ligadas
        pJogo jogoAtribuido;        //ponteiro para o jogo atribuido
        pid_t pidJogo;              //pid jogo atribuido
        int clientpipe_fd;          //pipe do jogador
        int arbitroJogo_fd[2];      //pipe arbitro->jogo
        pthread_t threadResponde;   //thread que trata do jogadro
        int estado;             //0 - Conectado e em jogo ativo
};                              //-1 conectado enquanto decorre cameponato
                                // 1 - Suspenso
                                // 2 - conectado sem jogo ativo
                                // 3 - campeonato acabou nao consegue comunicar mais
                                // 4 - saiu
                                // 5 - eliminei
```

```
typedef struct servidor arbitro;
struct servidor{
   pid_t pid;                          //Pid arbitro
   int MAXPLAYER;                      //Limite de jogadores
   long DURACAO;                       //Duração campeonato
   long ESPERA;                        //Tempo de espera por jogadores
   char GAMEDIR[PATH_MAX];             //Diretoria de jogos
   pJogador listaJogadores;            //criaçaõ da lista ligada de jogadores
   pJogo    listaJogos;                //lista ligada de jogos
   pthread_t threadCampeonato;         //thread que gere o campeonato
   struct timeval  tv1, tv3;           //Calcular tempos, até final campeonato etc...
   pid_t avisaFim[30];                 //pid's pendentes para registar
   int quantos;                        //quantos estao pendetes
   int flagCampeonato;                 //0 - Ainda nao começou
};                                     // 1 - A decorrer
                                       // 2 - Acabou
                                       // 3 - tempo de espera a decorrer
```

Implementação

Para conseguirmos lançar devidamente o árbitro será necessário declarar pelo menos uma variável de ambiente. Essa declaração é feita através do script defvars.sh e é executada pela linha de comando 'source defvars.sh'. Para o lançamento do árbitro deve ser utilizada uma das seguintes sintaxes:

```
Sintaxes:  ./arbitro
           ./arbitro -d <segundos>
           ./arbitro -e <segundos>
           ./arbitro -d <segundos> -e <segundos>
           ./arbitro -e <segundos> -d <segundos>
Argumentos: -d --> duração do campeonato.
            -e --> tempo de espera por jogadores.
Caso inicie o arbitro sem argumentos ele inicia com os valores default!
 -> Espera default:  60
 -> Duração default: 120
```

Após um lançamento efetuado com sucesso o árbitro ficará à espera de jogadores. Pode verificar os comandos possíveis para o administrador através do comando 'help'.

```
Comandos disponíveis:
players - Listar jogadores em jogo (nome e jogo atribuído).
games - Listar jogos disponíveis.
k<nome> - Remover um jogador do campeonato. EX: "krui" – remove jogador "rui"
s<nome> - Suspender a comunicação entre jogador e jogo.
r<nome> - Retomar a comunicação entre jogador e jogo.
end - Encerrar o campeonato imediatamente.
exit - Sair, encerrando o árbitro.

Comandos extra:
start  - Interrompe o tempo de espera e começa o campeonato!
change <argumento> <valor>
        d  -> altera duracao do campeonato para 'valor' segundos.
        e  -> altera tempo de espera por jogadores para 'valor' segundos.
show   <argumento>
        d  -> mostra duracao do campeonato.
        e  -> mostra tempo de espera por jogadores.
        t  -> mostra tempo restante do campeonato atual.
```

Estes comandos estarão sempre disponíveis, alguns deles sendo executáveis em certas situações e noutras não. O feedback ao administrador deixa bastante claro se o comando foi devidamente executado ou se existiu algum problema na interpretação/execução deste.

Lançamento do árbitro

O árbitro ao ser executado abre um named pipe (arbitro.pipe) que será utilizado pelos clientes para mandar pedidos. Quando o cliente é lançado manda um pedido de adesão por esse pipe e se tudo correr sem problemas será estabelecida uma ligação de comunicação entre o árbitro <-> cliente.

Ao ser estabelecida essa comunicação o árbitro abre dois pipes e associa-os ao cliente em questão. Um desses pipes é named pipe (cliente-pid.pipe), possibilitado assim a comunicação árbitro -> cliente. O outro pipe é um pipe anónimo, que irá possibilitar a comunicação cliente -> árbitro -> jogo, quando o campeonato começar.

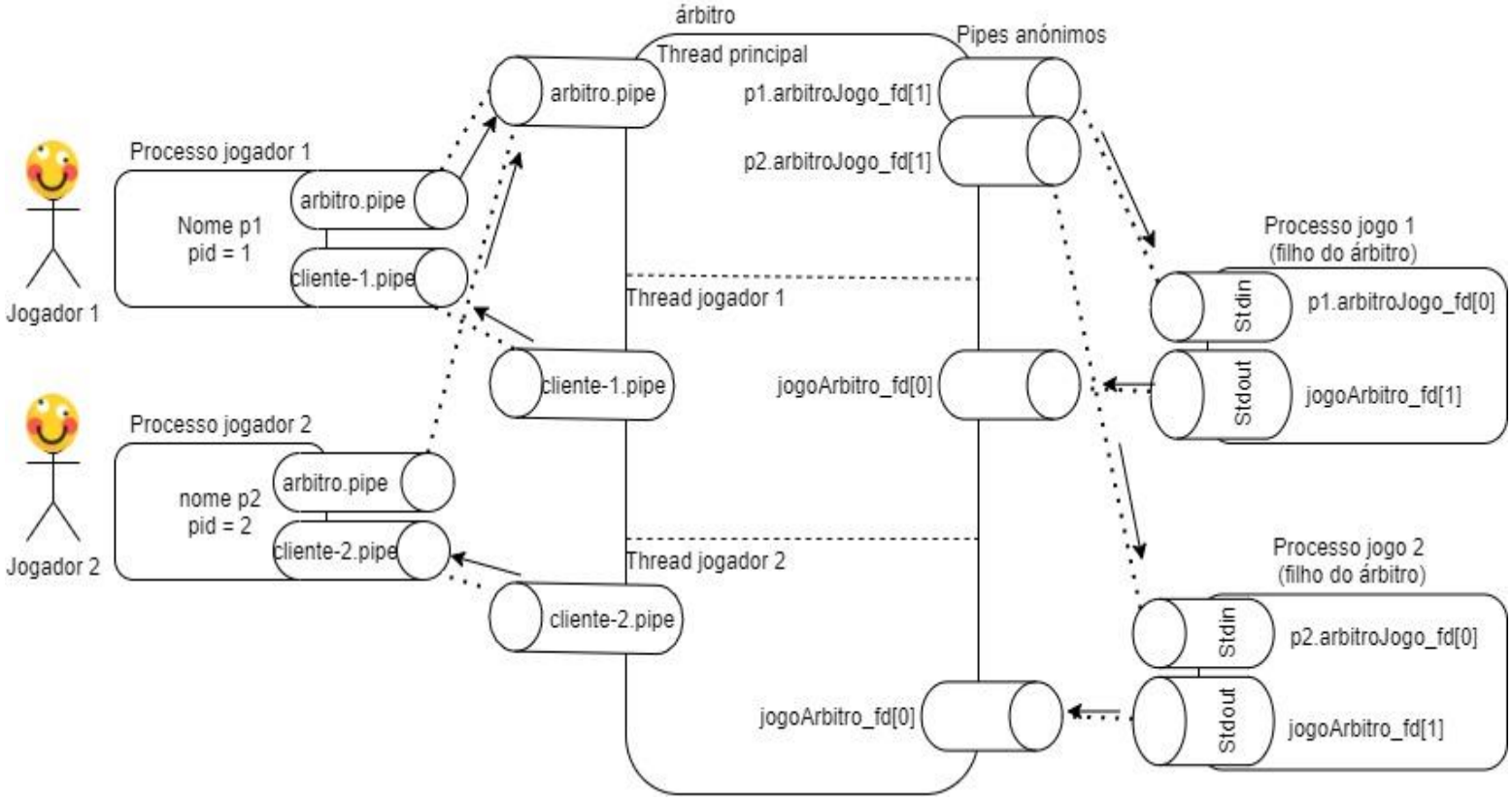
Este processo repete-se até ser atingido o limite mínimo de jogadores (2).

Lançamento do gestor do campeonato

Ao ser atingido o número mínimo de jogadores, é lançada a threadCampeonato que gere o campeonato. A thread faz uma difusão para todos os clientes indicando que o campeonato está prestes a começar dentro de alguns segundos e faz um sleep de 'tempo de espera' segundos. Nesta fase ainda são aceites novos jogadores. Quando a threadCampeonato "acorda" volta a fazer uma difusão avisando todos os jogadores ativos do começo do campeonato.

Lançamento do campeonato

Após isso, para cada jogador é lançada uma thread que trata das comunicações entre jogo -> árbitro -> jogador. Após todas as threads serem lançadas a threadCampeonato volta a fazer sleep de 'duração do campeonato' segundos, podendo esse sono ser interrompido por um sinal caso o administrador decida encerrar sessão ou todos os clientes tenham abandonado o campeonato ficando apenas um ativo. Outros clientes que tentarem conectar serão avisados do estado atual do campeonato e é dada uma escolha ao cliente de esperar para ser inscrito no próximo campeonato ou abandonar. Através dum esquema gráfico a comunicação completa entre processos é apresentada toda esta arquitetura de comunicação.



Este diagrama demostra a comunicação durante um campeonato entre 2 jogadores. Fora do campeonato a thread principal responde diretamente para o pipe do jogador em questão sem fazer redireccionamentos para os jogos, o mesmo se aplica quando o jogador está suspenso ou executa os comandos '#mygame' e '#quit'.

Tentei proteger o servidor de todos os possíveis erros que me lembrei, tais como, jogos com mal funcionamento, conexões durante o campeonato, saída aquando o decorrer do campeonato (tanto através do comando exit com ctrl+c) etc… Quando é recebido o exit status de um jogo antes do que é devido esse jogador é automaticamente suspenso e tanto o administrador como o jogador são informados que existe um problema com o jogo. Isso também acontece caso seja detetado algum broken pipe por todo o processo de comunicação. Qualquer tentativa de conexão durante o campeonato é aceite e posta em fila de espera, o jogador é informado do tempo de espera e caso o decida esperar até o atual campeonato acabar então será inscrito no próximo campeonato a decorrer. Se o comando exit ocorrer durante um campeonato então é feita uma limpeza de forma ordeira aos processos filhos, os jogadores são informados da pontuação acumulada até ao instante de desconexão do árbitro, são informados dessa desconexão e consequentemente também encerram o processo. A combinação

de teclas ctrl + c simula o comando exit utilizando uma estratégia que envolve um siglongjmp. Esta estratégia foi aprendida através de fóruns online e visto que nunca a abordamos nas aulas não sei até que ponto é 100% segura. Dos meus ensaios nunca houve nenhum problema com ela.

Caso a mesma combinação seja efetuada no cliente, este simula um pedido do comando '#quit', para o árbitro é irrelevante se o jogador utilizou a combinação ctrl + c ou se executou o comando '#quit'.

Em ambas as situações, tanto os árbitros com o cliente saem de forma ordeira, limpando memória dinâmica e named pipes utilizados.

Final do campeonato

Pode chegar-se a este ponto de várias formas, duração acabou, jogadores abandonaram, árbitro encerra sessão ou decide acabar o campeonato. Qualquer das situações é tratada de forma similar, é feita uma difusão de sinais aos jogadores ativos e aos jogos dos mesmos, é recolhido o exit status de cada jogo e a threadCampeonato constrói uma string com os vencedores. Além de fazer uma difusão para todos os clientes avisando dos jogadores que obtiveram a maior pontuação, cada jogador é avisado da sua própria pontuação. Do lado do árbitro é feita uma limpeza geral, tanto aos file descriptros como à memória dinâmica ocupada pelos jogadores. Do lado do cliente, este é perguntado se deseja inscrever-se no próximo campeonato. Caso decida participar é pedido novamente um nickname. A partir deste ponto todo o processo é semelhante ao descrito anteriormente.

Conclusão

Partes da implementação foram propositalmente omitidas para não tornar o relatório desnecessariamente longo e maçador. Foi descrito por alto todo o processo de comunicação e o funcionamento de um campeonato. O código está minimamente comentado e está segmentado por vários ficheiros para, na minha opinião, facilitar a sua leitura e interpretação.

Com a realização deste projeto tive a oportunidade de ter uma interação ativa com um ambiente Unix que descobri ser muito mais vasto do que aquilo que tinha ideia antes de frequentar a cadeira de SO. A partir desta interação aprendi várias estratégias de comunicação entre processos e o mais importante, programar com threads que é algo muito útil e prático!