DEPARTAMENTO DE ENGENHARIA INFORMÁTICA E DE SISTEMAS
INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

Engenharia Informática

Sistemas Operativos

Relatório

Trabalho Prático – Meta 3

Ano Letivo: 2020/2021
Curso: Engenharia Informática
Autores: Ricardo Pereira – 2015015815, Pedro Ruivo - 2015016289
Disciplina: Sistemas Operativos
Turma: P1
Professores: João Durães, Jorge Rodrigues
Data de Submissão: 09/02/2021

Índice

1. Introdução

Este projeto foi realizado no âmbito da disciplina de Sistemas Operativos e o tema abordado é sobre um campeonato em que vários jogadores competem entre si, ao interagir com diversos jogos. O projeto foi desenvolvido em C num ambiente UNIX.

O projeto está dividido em três tipos de programas: o programa do árbitro, jogador, e os diversos jogos. O árbitro é responsável por gerir todas os jogadores e os jogos, incluindo a comunicação entre ambos. Adicionalmente, o árbitro recebe comandos do seu utilizador. O jogador apenas efetua o registo da entrada e joga quando um campeonato é iniciado.

Todos os requisitos solicitados foram implementados pelos autores.

2. Referee (Árbitro)

Na meta 1, o árbitro apenas prepara a sua aplicação. Onde absorve os parâmetros recebidos pela consola e armazena na sua estrutura de dados, esta funcionalidade não está totalmente testada, mas foi desenvolvida de modo a que a ordem não seja importante, mas a aplicação será encerrada (corretamente) se os parâmetros não forem fornecidos como esperado. Juntamente com os parâmetros, esta aplicação também obtém variáveis de ambiente. Se não conseguir obter as variáveis de ambiente, esta irá usar os valores pré-definidos no seu ficheiro *header*.

Na meta 2, o árbitro já está preparado para ler comandos do teclado, receber e enviar pedidos de cada jogador. Adicionalmente, este já tem a funcionalidade para dar *handle* ao sinal SIGUSR1.

Na meta 3, o árbitro tem posse de apenas um *named pipe*, e em adição às restantes metas, completa os restantes comandos não desenvolvidos, e a criação dos jogos para cada jogador.

Para o campeonato, foi necessário um sistema de *countdown*, para isso, foi usado um *mutex*, aproveitando a sua *api*, permitiu que a aplicação ficasse bloqueada no *mutex* durante ***X*** segundos.

A questão da comunicação será detalhada no capítulo 6.

2.1. Constantes

```
#define DEBUG 1 //0 to remove debug messages

#define DEFAULT_GAMEDIR "../Execs/Games/"
#define DEFAULT_MAXPLAYER 10
#define DEFAULT_MINPLAYERS_START 2

#define MIN_CHAMP_DURATION 60
#define MAX_CHAMP_DURATION 600

#define MIN_WAITING_DURATION 30
#define MAX_WAITING_DURATION 120

#define MAX_MAXPLAYER 30
```

Estas constantes são autoexplicativas e usadas para evitar falhas no uso da aplicação.

2.2. Estruturas de dados e *Enums*

```
typedef enum ChampionshipState ChampionshipState;

typedef struct GameProc GameProc;
typedef struct Game Game;
typedef struct Player Player;
typedef struct Referee Referee;
typedef struct AvailableGames AvailableGames;
typedef struct ThreadHandles ThreadHandles;

typedef struct Application Application;
```

A estrutura de dados principal, *Application* é usada para armazenar a informação necessária para o funcionamento da aplicação. Esta contem outras estruturas, sendo estas:

- ChampionshipState, *enum*, usado para identificar o estado do campeonato a ser *broadcasted* aos jogadores.
- *GameProc*, que armazena a informação de jogos;
- *Game*, que armazena a informação geral de um jogo;
- *Player*, que armazena a informação geral de cada jogador autenticado, e os *files descriptors* dos seus *named pipes*, leitura e escrita;
- *Referee*, que armazena a informação geral do árbitro;
- *AvailableGames*, que armazena todos os jogos encontrados no diretório escolhido;
- *ThreadHandles*, que armazena os *handles* para cada *thread*.

3. Player (Jogador)

O jogador é a aplicação que não tem muito desenvolvimento na meta 1. Esta aplicação apenas prepara a sua aplicação e pede o nome de utilizador ao jogador.

Na meta 2, o jogador já pode efetuar o login, e enviar comandos ao árbitro ou *input* direcionado ao jogo. Adicionalmente, este já tem a funcionalidade para dar *handle* aos sinais SIGUSR1 e SIGINT.

Nota: O *input* direcionado ao jogo só é absorvido pelo árbitro, não é enviado ao jogo.

Na meta 3, o *input* direcionado ao jogo já é absorvido pelo árbitro, e enviado ao jogo. Adicionalmente, nesta meta, foi implementado um novo sistema de comunicação, em que cada jogador tem 2 *named pipes*, leitura e escrita. A questão da comunicação será detalhada no capítulo 5.

3.1. Estruturas de dados

```
typedef struct Player Player;

typedef struct NamedPipeHandles NamedPipeHandles;
typedef struct ThreadHandles ThreadHandles;
typedef struct MutexHandles MutexHandles;

typedef struct Application Application;
```

A estrutura de dados principal, *Application* é usada para armazenar a informação necessária para o funcionamento da aplicação. Esta contem outras estruturas, sendo estas:

- *Player*, que armazena a informação geral do jogador;
- *NamedPipeHandles*, que armazena os *handles* para cada *named pipe*;
- *ThreadHandles*, que armazena os *handles* para cada *thread*;
- *MutexHandles*, que armazena os *handles* para cada *mutex*.

4. Jogos

4.1. RNGGuess

Na meta 1, e para o primeiro jogo, foi desenvolvido um simples jogo de adivinhar um número randomizado. Este jogo tem uma pequena introdução, desenvolvimento, e armazena a pontuação do jogador ao longo do jogo.

Na meta 2 foi apenas introduzida a funcionalidade para dar *handle* ao sinal SIGUSR1.

Na meta 3, o sistema de *stdin* e *stdout* foi adaptado, de modo a que se possa interagir quando é criado de forma independente, ou quando é criado por outros processos.

4.2. Arithmetic

Jogo de resolver expressões aritméticas simples e randomizadas. Este jogo tem uma pequena introdução, desenvolvimento, e armazena a pontuação do jogador ao longo do jogo.

Esta aplicação implementa os mesmos sistemas complexos que o RNGGuess.

4.3. Translation

Jogo de traduzir palavras de Inglês para Português ou ao contrário. Este jogo tem uma pequena introdução, desenvolvimento, e armazena a pontuação do jogador ao longo do jogo.

Esta aplicação implementa os mesmos sistemas complexos que o RNGGuess.

4.4. DinoTrivia (Original)

Jogo de responder perguntas de trívia sobre dinossauros. Estas perguntas são randomizadas, limitada pela lista definida, juntamente com a pergunta, as respostas possíveis também são apresentadas, deixando o jogador decidir qual delas é a correta. Este jogo tem uma pequena introdução, desenvolvimento, e armazena a pontuação do jogador ao longo do jogo.

Esta aplicação implementa os mesmos sistemas complexos que o RNGGuess.

5. Utils e Makefile

Adicionalmente, foi criado um ficheiro com o objetivo de ser uma fonte de funcionalidades úteis para todas as aplicações. Com este ficheiro, o desenvolvimento do projeto deverá ser mais dinâmico, simples e espera-se evitar bugs simples.

O *makefile* compila todo o código e com as dependências necessárias. Os seguintes comandos são possíveis:

- Make – executa a primeira *tag* (no caso atual será o *all*);
- Make all – compila tudo;
- Make full – compila e executa tudo;
- Make fullReferee – compila e executa o árbitro;
- Make fullPlayer – compila e executa o jogador;
- Make fullGames – compila e executa os jogos;
- Make cleanObj – remove os ficheiros compilados;
- Os restantes comandos podem ser chamados, mas não são necessários, visto que os de cima executam tudo o que é necessário.

Na meta 2, foram adicionadas algumas pequenas funcionalidades ao Utils. Adicionalmente, foi criado um ficheiro *header* "CCommunication" com o objetivo de partilhar as estruturas de dados e constantes de comunicação entre o *Player* (Cliente) e *Referee* (Servidor).

6. Comunicação na aplicação

6.1. Named pipes

Named pipes são apenas usados para a comunicação entre o árbitro e jogadores.

6.1.1. Árbitro

Entrada: Serve apenas para aceitar os jogadores, no sucesso, passa a apenas os *named pipes* específicos do cliente.

6.1.2. Jogador

Leitura: Recebe mensagens do árbitro
Escrita: Envia mensagens para o árbitro

6.2. Pipes

Pipes são apenas usados para a comunicação entre o árbitro e jogos. Existem 2 pipes, um que trata da comunicação do Árbitro para o Jogo (**A2J**) e outro do Jogo para o árbitro (**J2A**).

6.2.1. Árbitro

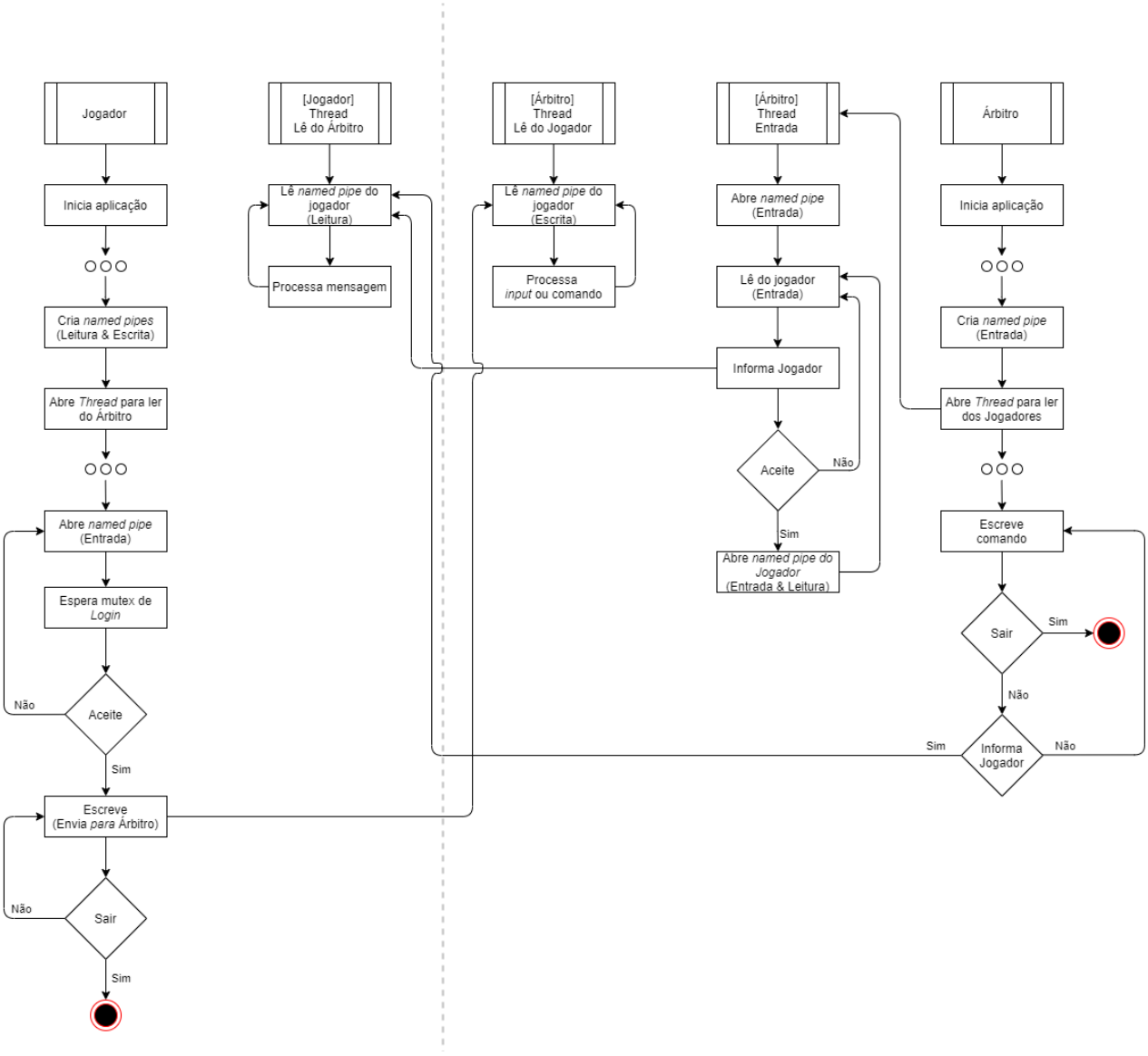
PipeJ2A - Leitura: Recebe mensagens do jogo (reencaminha para o jogador)

PipeA2J - Escrita: Envia mensagens ao jogo (vindo do jogador)

6.2.2. Jogo

PipeA2J - Leitura: Recebe mensagens do árbitro (vindo do jogador)
PipeJ2A - Escrita: Envia mensagens ao árbitro (reencaminha para o jogador)

6.3. Diagrama de Named Pipes



7. Estrutura do projeto

7.1. Jogos

Os jogos seguem uma estrutura bastante simples em que todos a seguem. De início é definido o comportamento do sinal *SIGUSR1* e de seguida, é apresentada uma mensagem de apresentação. Depois deste *setup* começa o jogo, em que cada jogo irá randomizar o seu fator, e esperar pela resposta do jogador. Quando o jogo recebe o sinal *SIGUSR1*, este irá fechar a aplicação, e retorna, no seu *exit status*, a pontuação obtida no decorrer do jogo.

7.2. Árbitro e Jogador

A estruturação destes são idênticos, em que englobam uma estrutura *Application* (diferente em cada um), e esta estrutura é preparada no começo da aplicação, que será usada para as restantes funcionalidades da aplicação. Em conjunto, é definido o comportamento do sinal *SIGUSR1*.

Após o *setup* ficar totalmente finalizado, estes programas ficam bloqueados à espera de *input* dos seus utilizadores.

A funcionalidade é separada por ficheiros, sendo estes:

- Thread – Apenas contém estruturas e funções a serem executadas por *threads*.
- Service – Código responsável para o funcionamento principal de cada programa.
- Normal – Apenas contém o *main* onde são chamadas as funções de *setup* e o ciclo de *stdin*.
    - Nota: “Normal” refere-se ao ficheiro com o nome do programa

7.3. Outros

Para auxílio e de modo a evitar duplicação de informação, foi criado um ficheiro *Utils*, que fornece constantes, estruturas, e funcionalidades gerais à aplicação toda. Juntamente com este ficheiro, existe também um outro ficheiro *header*, *Communication*, que fornece constantes e estruturas para uso múltiplo e focadas na comunicação da aplicação.

8. Funcionalidades desenvolvidas

8.1. Árbitro

✓	Apenas uma instância
✓	Quando termina, informa os jogadores
✓	Comunicação apropriada
✓	• Recebe pedidos de participação de jogadores
✓	• Recebe comandos de jogadores, para o árbitro
✓	• Recebe mensagens de jogadores, para o seu jogo (a ser reencaminhado)
✓	• Recebe mensagens de jogos, para o seu cliente (a ser reencaminhado)
✓	• Envia mensagens aos jogos, vindo do jogador
✓	• Envia mensagens aos jogadores, vindas do jogo
✓	• Envia mensagens aos jogadores, respostas de pedidos ou informações.
✓	Administrador interage com o Árbitro com comandos
✓	• players
✓	• games
✓	• k[username]
✓	• s[username]
✓	• r[username]
✓	• end
✓	• exit
✓	Lê variáveis de ambiente (cria caminho de jogo se não existe)
✓	Procura jogos (nomes começados por “g_”) a partir da diretoria fornecida
✓	Recebe argumentos (ordem indiferente)

8.2. Jogador

✓	Login
✓	• Nome único
✓	• Bloqueado quando campeonato está em progresso
✓	Interage com o Árbitro com comandos
✓	• #mygame
✓	• #quit
✓	Interage com o Jogo indiretamente
✓	• Enviado para o Árbitro
✓	• Árbitro processa e envia para jogo caso este exista e esteja a decorrer
✓	Funcionamento do sinal SIGUSR1, ALT + C.

8.3. Jogos

✓	Possível interagir quando abertos de forma independente
✓	Possível interagir quando abertos por processos
✓	Desenvolver 4 jogos (incluindo 1 original)
✓	• Arithmetic
✓	• RNGGuess
✓	• Translation
✓	• DinoTrivia

9. Conclusão

Os autores conseguiram desenvolver todas as funcionalidades solicitadas no enunciado.

Durante o desenvolvimento do projeto foram enfrentadas diversas dificuldades e obstáculos, sendo que todos foram superados. O projeto está funcional de acordo com os requisitos do enunciado.

Durante este projeto os autores passaram por uma linha de aprendizagem sendo que desenvolveram mais as suas capacidades de programar, estruturar e arquiteturas cliente-servidor num ambiente C Unix e conseguiram, assim, adquirir mais experiência e conhecimento em relação à programação com a linguagem C e ao ambiente Unix utilizado.