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

Engenharia Informática

Sistemas Operativos

Relatório

Trabalho Prático – Meta 2

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: 13/12/2020

Índice

1. 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.

1.1. Constantes

```
#define DEFAULT_GAMEDIR "~/Documents/SO/SO_CHAMPION/Games/"
#define DEFAULT_MAXPLAYER 10

#define MIN_CHAMP_DURATION 60000
#define MAX_CHAMP_DURATION 600000

#define MIN_WAITING_DURATION 30000
#define MAX_WAITING_DURATION 120000

#define MAX_MAXPLAYER 30
```

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

1.2. Estruturas de dados

```
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:

- *Referee*, que armazena a informação geral do árbitro;
- *Player*, que armazena a informação geral de cada jogador autenticado, e o *file descriptor* do seu *named pipe*;
- *Game*, que armazena a informação geral de um jogo;
- *AvailableGames*, que armazena todos os jogos encontrados no diretório escolhido;
- *ThreadHandles*, que armazena os *handles* para cada *thread*.

2. 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.

2.1. 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*.

3. RNGGuess (Jogo)

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 logo do jogo. Esta aplicação atualmente não tem qualquer constante nem estruturas de dados.

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

4. 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 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).