Instituto Superior de Engenharia de Coimbra

Licenciatura em Engenharia Informática

Sistemas Operativos

2023/2024

Trabalho Prático
Jogo de Labirinto Multijogador
Meta 1

Docente:

João António Pereira Almeida Durães

Discentes:

Francisco Costa Quelhas – a2020144121@isec.pt

Miguel Lograd Tavares – a2018011325@isec.pt

Índice

1. Introdução

O trabalho prático de Sistemas Operativos propõe a criação de um jogo de labirinto multijogador. Os jogadores têm como objetivo movimentar-se de um ponto inicial até ao ponto final em labirintos dinâmicos, onde obstáculos vão surgindo aleatoriamente. À medida que os jogadores concluem labirintos, avançam para níveis mais complexos. Este trabalho é concretizado em linguagem C para a plataforma Unix (Linux).

2. Estruturas de Dados/Headers

Neste trabalho prático, as estruturas de dados foram divididas pelos respetivos headers pertencentes, sendo estes do motor e do jogoUI. Esta divisão facilitou a organização deste projeto.

```
 GNU nano 7.2
#ifndef MOTOR_H
#define MOTOR_H

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <ncurses.h>
#include <time.h>
#include <sys/types.h>
#include <signal.h>
#include "jogoUI.h"

static int child_exit_status;

typedef struct {
    Jogador jogadores[5];
    int nivel;
    int maze[16][40];
} Jogo;

#endif
```

```
 GNU nano 7.2
#ifndef JOGOUI_H
#define JOGOUI_H

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>
#include <sys/wait.h>
#include <ncurses.h>

#define MAX_NOME 50

typedef struct {
    char nome[MAX_NOME];
    int pos_x;
    int pos_y;
    int pontuacao;
    bool isPlaying;
} Jogador;

#endif
```

3. Ficheiro Makefile

O ficheiro makefile possui os targets de compilação “all”, “jogoUI”, “motor”, “bot” e “clean”. Cada um destes tem a sua função sendo que é possível a compilação de todos os programas, apenas do jogoUI, do motor, do bot e também a eliminação de todos os ficheiros temporários de apoio, respetivamente. Além disso, ainda é utilizada também a biblioteca ‘ncurses’.

```
all: jogoUI motor bot

jogoUI: jogoUI.c
	gcc jogoUI.c -o jogoUI -lncurses

motor: motor.c
	gcc motor.c -o motor -lncurses

bot: bot.c
	gcc bot.c -o bot -lncurses

clean:
	rm jogoUI motor bot
```

4. Comunicação Bot - Motor

Nesta primeira meta foi requerido que o bot imprimisse periodicamente no seu stdout duas coordenadas e respetiva duração para inserção de pedras dentro de um labirinto por parte do programa motor. O programa motor captura esta informação e determina em que posição e durante quanto tempo a pedra ficará no labirinto. Isto significa que o bot apenas envia a informação e o motor faz tudo o resto. Para isto a função lancarBot() foi criada com um pipe de forma a comunicar entre processos, foi utilizada a função fork() para criar um novo processo. No processo filho, o código fecha a saída padrão, redireciona para a gravação do pipe (‘descriptor[1]’) e fecha os descritores de pipes não necessários, executando em seguida o programa bot com alguns argumentos, usando o execl. Já no processo pai, um loop é executado, até que a variável ‘child_exit_status’ seja diferente de zero ou ocorra um erro na leitura do pipe. O sinal de interrupção (‘SIGINT’) é manipulado pela função ‘handle_sigint’.

```c
void lancarBot() {
    printf("Lançar bot\n");
    char string[100][MAX_MSG_LEN];
    int descriptor[2];
    child_exit_status = 0;

    if (pipe(descriptor) == -1) {
        perror("Erro ao criar pipe");
        return;
    }

    int id = fork();

    if (id == 0) { // Filho
        close(STDOUT_FILENO);
        dup(descriptor[1]);
        close(descriptor[0]);
        close(descriptor[1]);

        execl("./bot", "bot", "1", "10", (char *) NULL);
        perror("Erro ao lançar bot");
        exit(1);
    } else { // Pai
        close(descriptor[1]);
        signal(SIGINT, handle_sigint);

        int i = 0;

        while (!child_exit_status) {
            ssize_t count = read(descriptor[0], string[i], sizeof(string[i]));

            if (count == -1) {
                perror("Erro na leitura do pipe");
                break;
            } else if (count == 0) {
                break;
            }

            printf("RECEBI: %s", string[i]);
            ++i;
        }

        close(descriptor[0]);

        int status;
        wait(&status);
    }
}
```

5. Variáveis Ambiente

Para a execução do sistema como desejado são necessárias as variáveis ambientes INSCRICAO, de modo a que no final de um período de tempo o jogo comece com o número de jogadores registados. A variável NPLAYERS guarda a quantidade de jogadores inscritos durante esse tempo, este terá um número mínimo para que o jogo comece. Além disso, o primeiro nível demora uma quantidade de segundos, indicada na variável DURACAO, sendo que cada nível seguinte demora menos tendo em conta a variável DECREMENTO.
A implementação destas foram efetuadas através de um script com export’s num outro ficheiro (‘env.sh’).

```c
void VariaveisAmbiente() {

    char *env_INSCRICAO = getenv("INSCRICAO");
    char *env_NPLAYERS = getenv("NPLAYERS");
    char *env_DURACAO = getenv("DURACAO");
    char *env_DECREMENTO = getenv("DECREMENTO");

    printf("INSCRICAO: %s\n", env_INSCRICAO);
    printf("NPLAYERS: %s\n", env_NPLAYERS);
    printf("DURACAO: %s\n", env_DURACAO);
    printf("DECREMENTO: %s\n", env_DECREMENTO);
}
```

```
 GNU nano 7.2                                  env.sh
export INSCRICAO=60
export NPLAYERS=3
export DURACAO=300
export DECREMENTO=30
```