Sistemas Operativos II

Relatório 1ªMeta – Frogger Game

Trabalho realizado por

Pedro Rodrigues Jorge – a2021142041

Daniel Ferreira Rodrigues – a2021142013

Conteúdo

1. Introdução

No âmbito da unidade curricular de Sistemas Operativos II foi proposto o desenvolvimento de um jogo semelhante ao do conhecido "Frogger game". Durante a leitura deste relatório, será possível encontrar informações sobre a comunicação dos dois processos desenvolvidos, "Operador" e "Servidor", sendo também referida a DLL utilizada para gerir a comunicação de ambos os programas.

Neste relatório, serão abordados os detalhes técnicos da arquitetura atualmente desenvolvida, destacando algumas funcionalidades e destaques principais do desenvolvimento atualmente realizado, como o mecanismo de comunicação entre o servidor e o operador bem como as estruturas desenvolvidas.

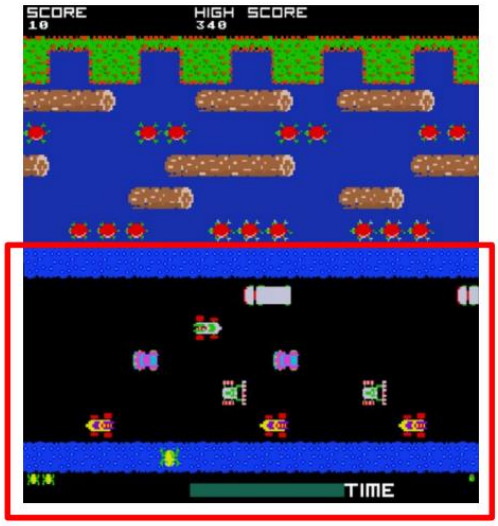


Figura 1-> Estado pretendido no final do projeto

Figura 2-> Estado Atual do Operador

2. Comunicação Servidor / Operador / Sincronização

Toda a comunicação realizada entre o operador e o servidor foi realizada através do mecanismo de memoria partilhada. Antes de resumir o funcionamento desta implementação no projeto iremos falar um pouco do funcionamento do operador e do servidor.

- O Servidor é o programa responsável por realizar todas as operações e funcionamentos do estado atual do jogo, como por exemplo o movimento dos carros e o posicionamento inicial dos sapos na linha de partida. Para além deste funcionamento o servidor é o responsável por enviar qualquer informação exclusiva aos restantes programas do sistema, como os operadores e os programas sapos, sendo assim possível gerir o contexto do sistema a ser desenvolvido, enviando eventos e informações aos restantes processos, como por exemplo sempre que o comando "exit 1" é introduzido, o servidor sinaliza um evento (ShutDownEvent) aos restantes programas, "dizendo" para estes encerrarem de imediato (de forma correta).

- O Operador é o programa que interage com o servidor recebendo através da memoria partilhada o estado atual do jogo, mostrando no ecrã. Este operador permite diversas instâncias de funcionamento em simultâneo, sendo importante garantir a sincronização entre as mesmas, visto que estes poderão enviar comandos ao servidor ao mesmo tempo. De modo que os comandos não fiquem sobrepostos e que as mensagens cheguem corretamente ao servidor, foi implementada a técnica do Buffer Circular lecionada durante as aulas, que permite através de diversos mecanismos de sincronização (semáforos, mutex) organizar assim a listagem de comandos escritos por diversas instâncias de operadores, sendo assim possível gerir o fluxo de mensagens entre o servidor e o operador.

Mecanismo de memória partilhada

As memórias partilhadas desenvolvidas consistem em duas partes, a primeira parte corresponde à memória associada a todo o gerenciamento dos comandos escritos pelos operadores, sendo rececionadas pelo servidor na thread (cmd_receiver) onde o servidor irá correr as funções da DLL desenvolvida para o mecanismo de comentação a ser documentado, e toma uma iniciativa para o comando introduzido por um determinado utilizador, por exemplo: caso o operador introduza o comando "dir 1", este comando será adicionado ao buffer através da função expecifica da DLL implementada no projeto e assim da mesma forma o servidor irá recebe-lo e irá chamar a função (invert_orientation) que irá alterar a direção de todos os carros do estado atual do jogo.

A segunda parte monitorizada pela DLL de modo a utilizar o mecanismo da memoria partilhada no projeto é a parte onde o servidor envia o estado atual do jogo para o operador, este comportamento é realizado na thread "game Manager"

do servidor onde irá ser chamada uma função da DLL que irá copiar uma estrutura do tipo "game" com todas as informações do jogo, como o mapa de jogo, a pontuação dos sapos e outros aspetos relevantes para a visualização do estado do jogo em tempo real, na tela do operador. De modo que a cópia dos dados seja corretamente realizada, utilizamos um mecanismo de sincronização mutex (SharedMutex) que basicamente sincroniza o acesso aos dados, não sendo permitido mexer neste tipo de informação enquanto esta está a ser alterada noutro local do programa.

Nota: A principal razão que nos fez decidir realizar esta divisão da memória partilhada (1ªParte – Comandos / 2ªParte - Jogo), foi o facto da organização interna dos dados, visto que a informação do Jogo é uma estrutura e esta tem dados mais sensíveis achamos por bem criar uma memória só para o tratamento dos comandos, deixando assim a estrutura do projeto, mais simples, organizado.

Em seguida está um esquema do mecanismo de comunicação entre o servidor e os operadores:

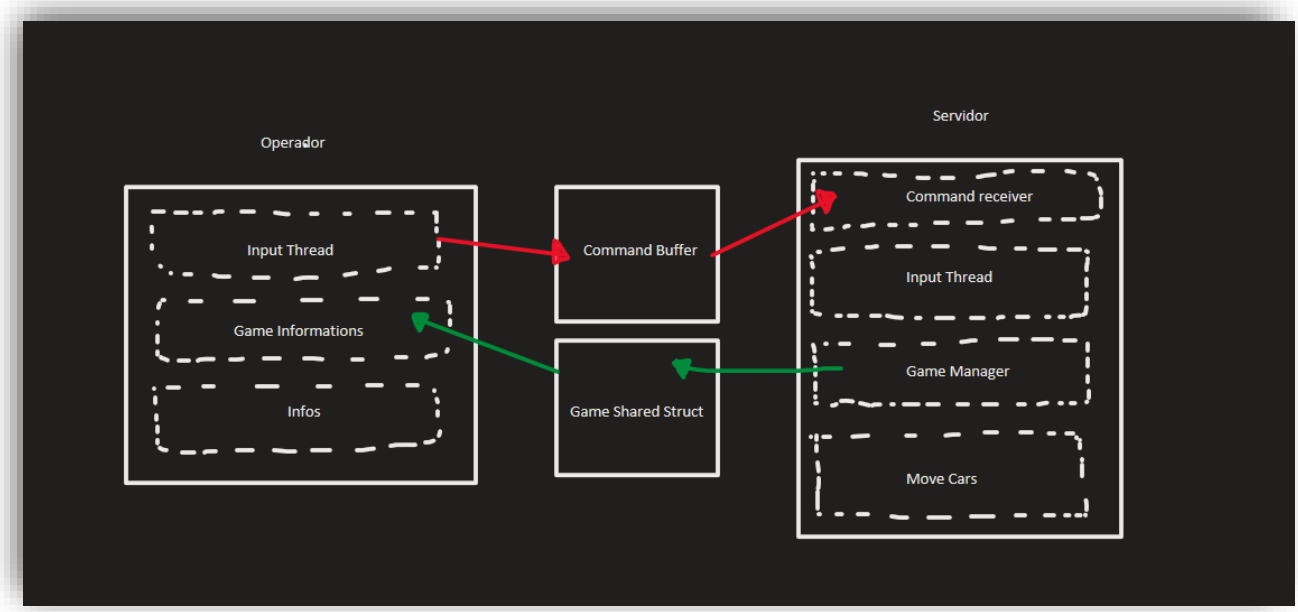


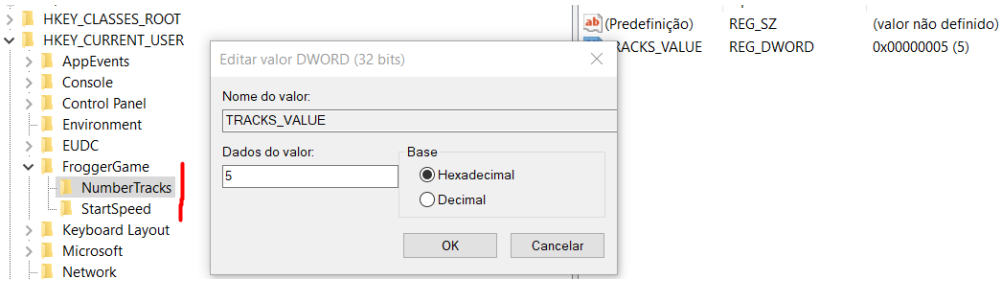
Figura 3 Esquema da comunicação do servidor com o operador

3. Aspetos relevantes da utilização do código desenvolvido

Temos 5 estruturas: frog(que armazena os pontos e as coordenadas dos sapos no jogo), vehicles(com as coordenadas e a direção do veículo 1 direita, 2 esquerda), buffer(estrutura de comunicação usada no buffer circular, com os cursores na posição de leitura e de escrita, e com a estrutura bufferCircular), bufferCircular(onde armazena os comandos), game(utilizada para armazenar os dados do jogo, velocidade de cada pista, o nº de estradas, com as estruturas dos sapos e dos carros, modo de jogo, número de carros por pista, e a tabela de visualização).

```c
typedef struct frog {
    int x, y;
    int points; // se -1 perdeu o jogo e é desconectado
}frog;

typedef struct vehicles {
    int x, y;
    int orientation; // 1-> direita para a esquerda, 2-> esquerda para a direita
} vehicles;

typedef struct bufferCircular {
    TCHAR cmd[100];
}bufferCircular;

typedef struct buffer {
    int pRead;
    int pWrite;
    bufferCircular buffer[BUFFER_SIZE];
}buffer;

typedef struct game {

    frog frogs[2];          //Sapos (Clientes)
    vehicles cars[8][8]; // veiculos[pos na faxa][o id do carro]
    int mode;               // 1-> Individual 2-> Competição
    INT num_tracks;         // Numero de estradas
    INT vehicle_speed;     // Velocidade dos carros
    int track_speed[8];
    int n_cars_per_track; // Random de numero de carros por track
    wchar_t table[H_GAME][W_GAME]; // Tabela de vizualização

}game;
```

Assim que o server é iniciado, é feito uma verificação no caminho da variável N_TRACKS, se for bem-sucedida então armazena os dados do número de estradas convertendo para inteiro o valor registado em TRACKS_VALUE, caso seja má sucedida então o user escreve o número de estradas que deseja e cria uma key registry. Após isso é feito o mesmo processo para a velocidade dos carros. Resumidamente é feito uma leitura nas constantes TRACKS_VALUE e SPEED_VALUE, caso não exista é criada uma nova, e escreve um valor novo.



Cada pista pode ter velocidades diferentes, para isso optámos por então criar x threads com o valor lido da key que armazena o x número de pistas, facilitando o controlo de velocidade de pistas especificas. Cada thread usa a função DWORD WINAPI move_cars(LPVOID lpParam), que trata da movimentação dos veículos com auxilio do Sleep().

```
Sleep(p->gameData->track_speed[p->track] * 200);
```

Os comandos do operador são enviados por memória partilhada do operador e são tratados pelo servidor, com auxílio da DLL.

No server usamos uma thread para que o server possa atender aos comandos que o user que está a utilizar o server, enquanto outros processos são atendidos em simultâneo como o processo do jogo.

Lista dos Comandos:

Comando	Programa	Função
dir 1	Operador	Inverte o sentido (apenas troca a direção 1 ou 2)
Object 1	Operador	Insere um objeto numa linha e coluna aleatória
Stopcars 1	Operador	Suspende as threads do jogo.
Resume 1	Operador	Retoma a execução das threads.
Exit 1	Operador	Fecha operador.
Tracks %d	Server	Altera o número de estradas no jogo e regista na key registry.
Vspeed %d	Server	Altera a velocidade das viaturas e regista na key registry.
List 1	Server	Apresenta o número de estradas e a velocidade.
Pause 1	Server	Suspende as threads do jogo.
Resume 1	Server	Retoma a execução das threads.
Restart 1	Server	Chama a função de inicio de jogo por padrão(FillGameDefaults(game* g)).

Exit 1	Server	Fecha Server.