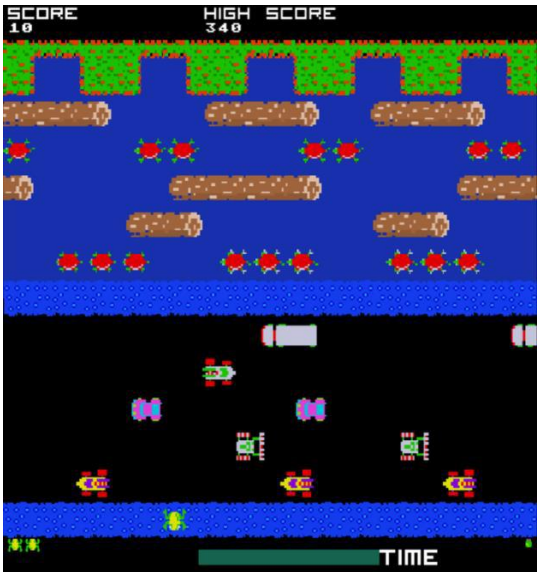
Instituto Superior
de Engenharia
Politécnico de Coimbra

Trabalho Prático - Frogger

Sistemas Operativos II

2022/2023



Docente: José Luís Guerra da Rocha Nunes

Beatriz Maia - 2020128841

João Santos – 2020136093

Licenciatura em Engenharia Informática

13 de maio de 2023

Índice

1. Introdução

O trabalho prático consiste na implementação do jogo "Frogger" em modo multiutilizador, com um máximo de dois jogadores. O jogo consiste em ajudar um sapo a atravessar uma estrada movimentada.

O trabalho prático foi realizado em linguagem C, usando a API Win32 do Windows.

2. Arquitetura Geral

A aplicação Servidor é responsável por gerir todo o funcionamento do jogo. Sem este, nem o operador nem o cliente funcionam, logo, é necessário que exista um e apenas um servidor a "correr" antes de iniciar qualquer outro programa.

A comunicação entre o servidor e o operador é feita exclusivamente por memória partilhada, ou seja, a verificação do conteúdo das células e a sua atualização é feita de forma direta na memória partilhada, enquanto a restante comunicação do operador para o servidor é feita através do paradigma produtor/consumidor (Buffer Circular).

2.1. Sincronização

2.1.1. Mutexes

Para garantir que apenas existe apenas uma instância da aplicação servidor a executar foi utilizado o mutex com o nome "checkFroggerServerMutex", sendo este também usado para verificar se já existe uma instância do servidor a executar quando for executada a aplicação operador. Caso não exista, este é terminado. Caso se tentem executar vários servidores, apenas o primeiro terá sucesso e os outros serão terminados.

De forma a garantir a coerência dos dados dos jogos a decorrer, tanto no Servidor como na aplicação Operador, é utilizado o mutex "updateFroggerMapMutex".

Existe ainda um mutex com o nome "mutexCircularOperadorToServer" que é atribuído no Buffer Circular utilizado na comunicação de comandos entre a aplicação Servidor e Operador, tendo como principal objetivo permitir apenas a leitura ou escrita de mensagens no Buffer Circular num determinado momento.

2.1.2. Semáforos

O semáforo "shareGameMapSemaphore" é usado na aplicação Servidor de forma a informar a thread que envia as alterações ao Operador que ocorreu uma alteração em um ou mais jogos que estão a decorrer.

Na comunicação entre a aplicação Servidor e Operador através de um Buffer Circular existem dois semáforos que indicam quantas posições do Buffer Circular estão disponíveis para escrita e quantas que já foram escritas e ainda não foram lidas.

Para isso são utilizados os semáforos:

- “semaforoWriteCircularOperadorToServidor”
- “semaforoReadCirculaOperadorToServidor”

2.1.3. Eventos

O evento “closeAllEvent” tem como objetivo informar athread que existe em todas as aplicações a decorrer, sejam elas Servidor ou Operador, que devem terminar a sua execução.

É utilizado o evento “updateFroggerMapEvent”, com o objetivo de informar todas as aplicações Operador que estão a ser executadas que existe uma alteração em um ou mais mapas de jogo.

Para a aplicação Servidor controlar se existem jogos atualmente a decorrer é utilizado o evento “anyInGameMatchesEvent”, ficando a thread responsável pelo decorrer dos jogos à espera de que este evento esteja sinalizado.

2.2. Comunicação

2.2.1. Memória Partilhada

De forma a partilhar todos os dados sobre os vários jogos que podem estar a decorrer em simultâneo é utilizada a memória partilhada. Para isto são partilhadas as diversas informações sobre os jogos (Estrutura SharedGame) entre as aplicações Servidor e Operador.

2.2.2. Buffer Circular

Para a passagem de comandos da aplicação Operador para a aplicação Servidor é utilizado um Buffer Circular. Este Buffer Circular tem dez posições, sendo cada uma destas do tipo BufferCell que contém uma string com a mensagem. Na estrutura principal do BufferCircular existem ainda duas posições, sendo estas referentes à posição atual do ponteiro de escrita e a posição atual do ponteiro de leitura.

2.2.3. Named Pipes

Para que o cliente comunique com o servidor são utilizados dois named pipes. Um dos named pipes é referente ao tabuleiro e ao jogo em si, tais como, os elementos (frogger, carros, etc), a pontuação, o tempo e o nível. O outro named pipe é para as mensagens enviadas pelo cliente para o servidor de forma a movimentar o frogger, resetar a sua posição, etc.

3. Uso da DLL

Usámos a DLL de forma explicita, de forma que o sistema usasse menos recursos (reduzir a duplicação de código), e vários ficheiros usassem a mesma DLL, assim, beneficiarão da atualização ou correção da DLL.

Com a forma explicita, não é necessário nenhum **.lib** e assim a **dll** é libertada quando não é mais precisa, libertando recursos ao sistema.

Na **dll** é incluída a verificação se a aplicação servidor já está em execução ou não, e a criação dos mecanismos de sincronização.

4. Estruturas de Dados

4.1. Estrutura SharedGame

```
typedef struct {
    HANDLE hMapFileGame[MAX_PLAYERS];

    GameData* gameData[MAX_PLAYERS];

    GameSettings gameSettings;

    HANDLE hEventCloseAll;

    HANDLE hReadWriteMutexUpdateGame;

    HANDLE hEventUpdateGame;
    HANDLE hSemaforoSendGameUpdate;
    HANDLE hEventInGame;

    // BUFFER CIRCULAR MONITOR PARA SERVER
    HANDLE hMapFileBufferCircular_OperadorServidor;
    HANDLE hMutexBufferCircular_OperadorServidor;

    HANDLE hSemaforoReadBufferCircular_OperadorServidor;
    HANDLE hSemaforoWriteBufferCircular_OperadorServidor;

    BufferCircular* bufferCircular_OperadorServidor;


}SharedGame;
```

Esta estrutura é utilizada para os mecanismos de sincronização (Eventos, Mutex, Semáforos) e de comunicação entre servidor e operador.

4.2. Estruturas GameData e GameSettings

```
typedef struct {
    TCHAR nickname[MAX_TAM];
    DWORD idPlayer;

    BOOL inGame;
    BOOL pausedGame;

    BOOL won;
    BOOL lost;

    BOOL endGame;

    DWORD tipoJogo;

    DWORD level;

    DWORD points;
    DWORD secondsInGame;

    DWORD nLines;
    DWORD nColumns;

    DWORD nFaixasRodagem;
    DWORD velocidadeInicialCarros;

    DWORD carrosPerLane;
    DWORD carrosInGame;

    DWORD carrosStop;
    DWORD bitMap;

    Frogger frogger;
    Car cars[MAX_CARS_PER_LANE * 8];    // 8 x 8 é o número máximo de carros possivel num mapa de jogo (8 máx por linha) e (8 linhas de estrada)

    DWORD gameMap[10][20];  // MAX 10 por 20

    // VERIFICAR SE PAROU DE MEXER DURANTE 10 SEGUNDOS
    int inactiveSeconds;
}GameData;
```

Esta estrutura é responsável pelos dados de todo o jogo para um jogador (Cada jogador tem a sua estrutura GameData com os dados referentes ao mesmo): se o jogo está em pausa, pontuação, tempo decorrido, as linhas e colunas do mapa, o número de faixas de rodagem e a velocidade inicial dos carros, carros por cada faixa de rodagem, número de carros no mapa inteiro e uma variável referente aos segundos que os carros devem estar parados (se for 0 os carros estão em movimento). Um variável do tipo estrutura "Frogger" e outra do tipo "Car" que vão ser descritas mais à frente e um array bidimensional que é preenchido com DWORD's mais tarde processado pelo Operador e Cliente para apresentar na tela o caracter (imagem no caso do cliente) referente à posição linha coluna.

```
typedef struct {
    DWORD nLines;
    DWORD nColumns;

}GameSettings;
```

Esta estrutura representa as definições associadas ao jogo, tais como o número de linhas e colunas.

4.3. Estrutura BufferCell e BufferCircular

```
#define MAX_TAM_BUFFER 10
typedef struct{
    TCHAR message[MAX_TAM];
}BufferCell;

typedef struct {
    DWORD wP;
    DWORD rP;

    BufferCell buffer[MAX_TAM_BUFFER];
}BufferCircular;
```

Estas estruturas são responsáveis para que o buffer circule entre o monitor e o servidor, para tal existe a mensagem a circular e a posição do buffer para escrita e leitura.

4.4. Estrutura ThreadsServer, ThreadsCliente e ThreadsOperador

```
#define N_THREADS_SERVER 4
typedef struct {
    HANDLE hThreads[N_THREADS_SERVER];
    HANDLE hEventCloseAllThreads;
}ThreadsServer;

typedef struct {
    DadosPipe hPipes[MAX_PLAYERS];

    HANDLE hEvents[MAX_PLAYERS];

    HANDLE hMutex;

    int* deveContinuar;
} ThreadsCliente;

#define N_THREADS_OPERADOR 2
typedef struct {
    HANDLE hThreads[N_THREADS_OPERADOR];
    HANDLE hEventCloseAllThread;
}ThreadsOperador;
```

Estas estruturas são inerentes às threads das 3 entidades (servidor, operador e cliente). As estruturas ThreadsServer e ThreadsOperador ambas possuem um array de threads enquanto que a estrutura ThreadsCliente possui um array de "DadosPipe" (descrito mais à frente) com 2 posições (1 para cada jogador). Tem também uma variável correspondente a se o cliente deveContinuar a receber informação pelo named pipe.

4.5. Estrutura DadosPipe

```
typedef struct {
    HANDLE hInstancia;

    OVERLAPPED overlap; //estrutura overlapped para uso asincrono

    BOOL active; //para ver se já tem cliente ou não
} DadosPipe;
```

Esta estrutura serve para criar os named pipes entre o cliente e o servidor.

4.6. Estrutura PartilhaJogoServidorCliente

```
typedef struct {
    DadosPipe hPipes[MAX_PLAYERS];

    GameData* gameData[MAX_PLAYERS];

    HANDLE hEvents[MAX_PLAYERS];

    HANDLE hReadWriteMutexUpdateGame;

    HANDLE hEventUpdateGame;

    int deveContinuar;

} PartilhaJogoServidorCliente;
```

Esta estrutura serve para que o cliente e o servidor troquem mensagens através de um named pipe, para o tabuleiro (sendo que estão todos os dados agregados a este, tais como a pontuação, tempo, elementos, etc).

4.7. Estrutura Frogger e Car

```
typedef struct {
    DWORD x;
    DWORD y;
}Frogger;

typedef struct {
    long x;
    long y;
    long direction;
}Car;
```

As estruturas Frogger e Car são referentes ao elementos móveis do jogo. Ambos têm variáveis que definem as suas coordenadas no mapa. A estrutura Car tem ainda uma variável referente à direção que cada carro vai ter (os carros na mesma linha têm a mesma direção).

4.7. Estrutura threadDataDrawMap

```
typedef struct {
    HANDLE hMutex;
    int* xBitmap;
    int* yBitmap;
    HWND hWnd;
    BITMAP bmp;
    HDC bmpDC;
    HDC* memDC;

    GameData gameData;
} threadDataDrawMap;
```

Esta estrutura serve para a interface gráfica, sendo esta feita no cliente.

5. Manual de Utilização

5.1 Servidor

Comandos	Descrição
start	Começar o jogo
exit	Terminar todas as aplicações em execução
pause	Pausar o jogo atual
resume	Resumir o jogo atual
restart	Recomeçar o nível atual

5.2. Operador

Comandos	Descrição
<co (linha) (coluna)>	Colocar um obstáculo numa certa linha e coluna do mapa de jogo
<md (linha)>	Mudar a direção dos carros presentes numa certa linha (faixa de rodagem)
<stop (segundos)>	Parar o movimento de todos os carros durante um período de tempo limitado

(COMANDOS APENAS DISPONÍVEIS DURANTE O JOGO)

5.3. Cliente

Comandos	Descrição
Setas ou Botão esquerdo do rato*	Movimentar o frogger para uma zona adjacente à posição atual
Botão direito do rato	Reposicionar o frogger na partida
Passar o rato pelo frogger	Mostrar informação do número de vitórias

**O frogger apenas se movimenta (usando o botão esquerdo do rato) quando é escolhida uma posição adjacente à atual*

(No menu é possível escolher o tipo de bitmaps do jogo, visualizar os autores e sair da aplicação)

6. Requisitos Implementos

Descrição funcionalidade/ requisito	Implementado	Não Implementado	Implementada Parcialmente
Jogo Individual	X		
Jogo Multijogador		X	
Evolução de Níveis	X		
Suspender e retomar jogo	X		
Reiniciar o Jogo	X		
Encerrar o sistema	X		
Parar o movimento dos carros	X		
Inserir obstáculos	X		
Inverter o sentido da marcha de uma faixa de rodagem	X		
Movimento com teclado	X		
Movimento com rato	X		
DLL	X		
Escolha de Bitmaps	X		
Regressar com o sapo à partida com o botão direito do rato	X		
Informação do número de vitórias com o rato por cima do sapo	X		
Comandos	X		
Memória Partilhada	X		
Named pipes	X		
Registry	X		
Outros	X		

7. Decisões tomadas

Mudança de Nível:

- No nível 1 em cada faixa de rodagem só existe 1 carro e esse número é incrementado uma unidade a cada nível até ao máximo de 8 carros, do nível 8 em diante.

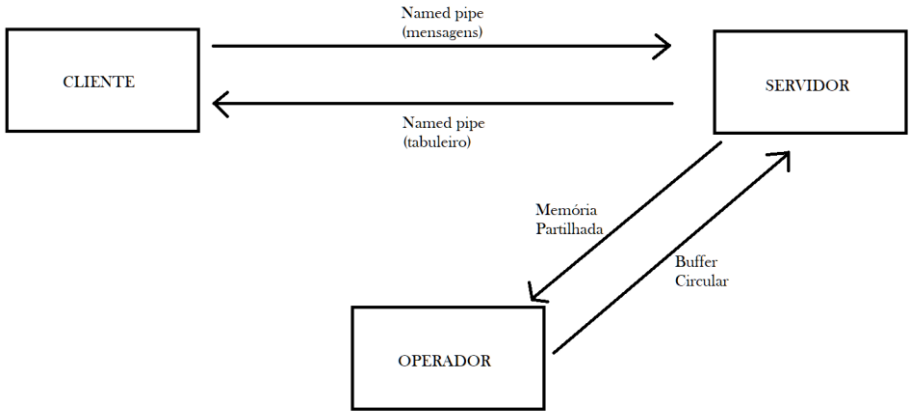
- A velocidade dos carros aumenta para o dobro no nível 5 em diante.

- O tempo disponível de cada nível é de 60 segundos no nível 1 e decrementado em 5 por cada nível sendo o mínimo, seja o nível qual for, de 20 segundos para tentar ganhar.

Voltar à base (botão do lado direito):

- Não referindo no enunciado se o Frogger voltaria à posição inicial ou a uma posição aleatória da base optámos por fazer com que este volte à posição inicial.

8. Diagrama com os mecanismos de comunicação e de sincronização



O cliente envia informações ao servidor através de um named pipe exclusivo para mensagens, do servidor para o cliente através de um named pipe exclusivo para o envio da estrutura responsável por armazenar dados do jogo e o mapa em si.
Do operador para o servidor usa-se um buffer circular para o envio de comandos que alteram o mapa de jogo e do servidor para o operador usa-se memória partilhada para que o operador tenha acesso aos dados e ao mapa de jogo. Não existe comunicação entre cliente e operador.

9. Conclusão

Este relatório foi elaborado como parte da cadeira de Sistemas Operativos 2 e concluímos que, por meio do uso de named pipes e mecanismos de sincronização, como a memória partilhada entre o servidor e o monitor, podemos criar interação entre duas entidades e aplicar conceitos de arquitetura e desenvolvimento do jogo clássico FROGGER.

Esse trabalho foi de grande importância para aprofundar o nosso conhecimento nesse tema, pois permitiu-nos compreender melhor a criação e utilização de estruturas de dados, implementação de comandos, mecanismos de comunicação e sincronização, uso de informações do Registry e criação de interfaces gráficas.