INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

INSTITUTO POLITÉCNICO DE COIMBRA

Licenciatura em Engenharia Informática 2º Ano – 2º Semestre 2022/2023

Frogger Game

Rafael Couto Nº 2019142454

Rafaela Carvalho Nº 2019127935

COIMBRA

18 de junho de 2023

Índice

Introdução

A elaboração deste trabalho prático consiste na implementação do jogo "*Frogger*". Todos os intervenientes serão processos a correr e os programas existentes terão recurso a interfaces gráfica ou consola.

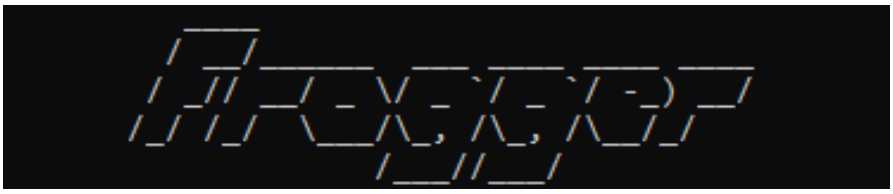
O trabalho será desenvolvido em linguagem C para Windows API (Win32) que visa o desenvolvimento de aplicações 32-bit.

O jogo "*Frogger*" baseia-se em conduzir um sapo de um ponto de origem para um ponto de destino, com obstáculos como carros em movimento e outros possíveis bloqueios, numa determinada área que será o denominado "*board*" ou tabuleiro neste trabalho prático.

Será seccionado em três projetos essenciais sendo eles, servidor, responsável pelo controlo de jogo, operador, responsável pela monitorização do jogo, e sapo, responsável pela jogabilidade.

Deste modo, pretende-se desenvolver conhecimentos em relação a *named pipes* e *threads*, dando continuação a matérias anteriormente lecionadas na unidade curricular de Sistemas Operativos I. São também implementados métodos com recorrência a memória partilhada e ao *Windows registry*, que se trata de uma base de dados que gere recursos e guarda configurações para as aplicações do sistema operativo Windows.

Relativamente ao *registry*, este é usado no trabalho para guardar valores de variáveis de jogo de modo que seja possível reutilizar valores de argumentos sem necessidade de estar recorrentemente a introduzi-los cada vez que é iniciado um jogo.



Implementação

Estruturas de Dados

De modo que existisse bom funcionamento do sistema, foram criadas três estruturas de dados referentes à comunicação, ao jogo e ao controlo de dados.

Para que fosse possível comunicar entre o monitor e o servidor, recorreu-se a uma estrutura para o "buffer circular" responsável apenas pela passagem do comando de um lado a outro. Esta estrutura está incluída dentro da estrutura de jogo que carrega todas as informações necessárias para a inicialização do tabuleiro e carrega os valores dos argumentos necessários ao início do jogo.

Num cenário de encapsulamento foi então criado uma última estrutura que controlo de dados que engloba a estrutura de dados de jogo e "*handles*" relativos a ficheiros de leitura, escrita, "mutexes" e semáforos, responsáveis pela transferência de informação.

Em relação ao que foi desenvolvido para a meta um foram acrescentadas algumas estruturas responsáveis por fazer a comunicação do cliente para o servidor, que é enviada pelos *named pipes*.

Servidor

No ficheiro *server.c* foi implementado os semáforos relativos ao controlo de instâncias dos programas servidor. São verificados os valores presentes para os argumentos de início de jogo e a leitura e ou registo, ou não, do *registry*. Também é no *server.c* que são inicializados o tabuleiro e a função responsável pela gestão de comandos. É também inicializada a *thread* responsável pela leitura dos comandos vindos do cliente com recurso a *circular buffer*.

De modo que fosse possível visualizar as alterações requisitadas pelo operador, a interface foi desenhada com um "operator log" e o leitor de comandos do servidor.

Todas as verificações necessárias de jogo após receção de um comando do cliente foram igualmente implementadas no servidor, presente numa *thread.* De igual modo, permitindo que um jogador se conecte à partida e que o servidor inicie o jogo, existe uma thread responsável pela aceitação dos mesmos.

As restantes modificações face ao apresentado na meta 1 do trabalho prático são funções auxiliares aos restantes processos existentes, colocações de cursor, divisão de comandos em argumentos, etc.

Operator

No ficheiro *operator.c* inicializamos os semáforos necessários à comunicação com o servidor e é mapeado o bloco de memória para o espaço de endereçamento do processo criado. Posto isto, lançamos a *thread* responsável pela gestão dos comandos e aguardamos em ciclo infinito até que haja ordem de saída. Está em escuta permanente.

De momento a interface é igualmente desenhada como o servidor e deste modo permite que tanto o mapa como o leitor de comandos não se envolvam e permita tanto o refrescamento do mapa em tempo real como da leitura de comandos no ecrã, dada a colocação estratégica do cursor na posição de impressão para remoção de conflitos e efeito "*flickering*".

Para que tal fosse possível, houve a adição de uma *Critical Section* de modo que fosse possível escrever numa dada zona do ecrã sem que haja mistura de "*impressões*".

Relativamente à particular necessidade de receber o comando de encerramento do servidor, optou-se por uma escolha mais sensata em que este encerramento é despoletado por um evento, desbloqueando a *thread* em escuta.

Frog

O cliente, por outro lado, trata-se de uma GUI capaz de representar o jogo e permitir ao cliente efetuar jogadas.

Foi implementada a criação da janela, bem como a colocação dos elementos inerentes à necessidade do jogo. São igualmente verificados os comandos e permite mover o sapo, porém sem verificações.

Na implementação deste trabalho este foi um dos aspetos em falta, dado que há a impressão do mapa, dos carros e a existência do sapo que é amovível, porém algum problema na comunicação através dos *named pipes* não nos permitiu fazer uma leitura da memória existente e, portanto, o jogo não é refrescado em tempo real.

Funcionalidades Implementadas

Relativamente às funcionalidades do sistema atualmente implementadas, estão presente a possibilidade de comunicação entre o programa monitor e o programa servidor, comandos básicos, com funções básicas de um programa, como, comando de saída, impressão de tabuleiro, limpeza de ecrã e comando de ajuda.

São chamados através da introdução dos comandos, "*stop*", "*obstacle*", "invert", "*clear*" e "*help*", "*exit*", respetivamente.

Funcionalidades	**Implementada**	**Parcialmente**	**Não Implementada**

Mecanismo(s) de comunicação e sincronização com o programa operator.	X		
Utilizar a informação que se encontra definida no Registry.	X		
Mapa com uma solução para o problema (ligação entre o ponto de origem e destino)	X		
Implementação do movimento dos carros no mapa.	X		
Visualizar o(s) mapa(s) de jogo em tempo real no operator.	X		
Comunicação com o utilizador e permite a receção e envio de informação de/para o server.			X
Possibilidade de leitura de comandos e apresentação do mapa em tempo real no programa operator.	X		
Apresentação do mapa de jogo no programa frog.		X	
Verificações de comandos enviados pelo programa frog para o servidor.		X	
Possibilidade de movimento do sapo no programa frog.		X	
Implementação da Dynamic-Link Library			X

Anexos

- server.c
- server.h
- operator.c
- operator.h
- frog.c
- frog.h
- registry.c
- SO2 – 2023 – TP - Relatório.pdf

Conclusão

A elaboração deste projeto permitiu consolidar conceitos abordados em sala de aula e explorar outros tantos para resolução de problemas.

Relativamente ao que era pedido, existiram alguns aspetos em falha, nomeadamente a comunicação exigida para o programa frog, pelo que a implementação trabalho não está para o objetivo máximo. Igualmente, a Dynamic Link Library também não foi implementada por escassez de tempo em busca de ver solucionados outros programas que considerámos mais importantes.

Ainda assim, relativamente à aplicação de conceitos de comunicação e partilha de dados foram altamente compreendidos e bem aplicados, fosse o uso de memória partilhada, double buffer, eventos etc.

Concluindo, foi possível sim colocar em prática assuntos estudados durante o semestre e com a compreensão deseja, porém pecou na organização do tempo para que fosse possível realizar um trabalho mais robusto.