2025

Relatório Sistemas Operativos 2

LICENCIATURA ENGENHARIA INFORMÁTICA

MARTIM OLIVEIRA Nº2022132041
RICARDO DUARTE Nº2022137878

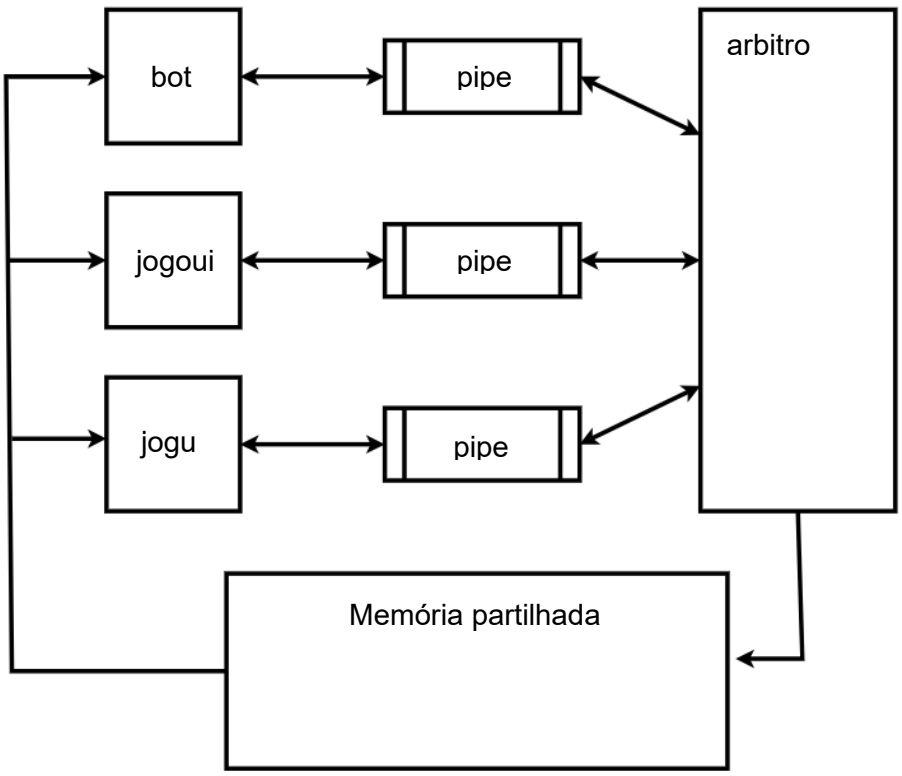
Índice

Introdução

O relatório descreve o desenvolvimento de um trabalho prático no âmbito da unidade curricular de Sistemas Operativos 2, que consiste na implementação de um jogo de palavras. Embora o foco principal não seja o jogo em si, mas sim na correta aplicação dos conceitos e mecanismos estudados na disciplina, a construção deste projeto permitiu explorar de forma prática a comunicação entre processos, gestão de concorrência, sincronização e controlo de execução num ambiente multi-processo.

A lógica do jogo baseia-se na identificação de palavras válidas a partir de letras apresentadas aleatoriamente no ecrã, a uma cadência constante. Os jogadores interagem com o sistema através de comandos na consola, podendo formar palavras, consultar a pontuação, visualizar outros jogadores ou abandonar o jogo. O jogo suporta até 20 jogadores simultâneos e é gerido de forma automática, sendo necessário a presença de dois ou mais jogadores e terminando quando resta apenas um.

Este relatório descreve as principais funcionalidades implementadas, as decisões técnicas tomadas e os desafios enfrentados durante o desenvolvimento do projeto, destacando a aplicação prática dos conceitos teóricos abordados ao longo da disciplina.



Programas

Para a elaboração do jogo, efetuamos o desenvolvimento de 3 programas, nomeadamente “arbitro”, “bot” e “jogoui”, cada um com características distintas.

O programa “arbitro” é responsável pela gestão do jogo, controlando a lógica principal. O “jogoui” corresponde ao cliente utilizado pelos jogadores humanos, servindo de interface para interação com o jogo. Por fim, o “bot” representa um jogador automático que interage com o jogo, imitando o comportamento de um utilizador

Estruturas de dados

```c
typedef struct _BufferCell {
    TCHAR  letra;
} BufferCell;

typedef struct _SharedMem {
    unsigned int c;   //
    unsigned int wP;  // posicao do buffer escrita
    unsigned int rP;  // posicao do buffer leitura
    BufferCell buffer[BUFFER_SIZE]; // buffer
    char users[NUSERS][25]; // array de strings para os nomes dos jogadores
    int pontuacao[NUSERS]; // array de inteiros para as pontuações dos jogadores
    int nusers;
} SharedMem;

typedef struct _ControlData {
    unsigned int shutdown;  // flag "continua". 0 = continua, 1 = deve terminar
    HANDLE hMapFile;        // ficheiro de memoria
    SharedMem* sharedMem;   // memoria partilhada
    HANDLE hMutex;          // mutex
    HANDLE hEvent;          // evento para leitura sincronizada
    HANDLE hEvSai;          // evento para desbloquear o gets
    HANDLE hPipe[NUSERS];   // array de handles para os pipes de cada jogador
    unsigned int nPipes; // maximo de pipes
    TCHAR username[26];         // Novo campo para guardar username do cliente
    TCHAR lastCommand[256];     // Último comando lido
} ControlData;

typedef struct _PipeMsg {
    HANDLE hPipe;
    TCHAR buff[256];
    BOOL isUsernameInvalid;
    TCHAR username[26];
} PipeMsg;
```

```c
typedef struct _BufferCell {
    TCHAR  letra;
} BufferCell;

typedef struct _SharedMem {
    unsigned int c;   //
    unsigned int wP;  // posicao do buffer para a escrita
    unsigned int rP;  // posicao do buffer para a escrita
    BufferCell buffer[BUFFER_SIZE]; // buffer
    char users[NUSERS][25]; // array de strings para os nomes dos jogadores
    int pontuacao[NUSERS]; // array de inteiros para as pontuações dos jogadores
    int nusers;
} SharedMem;

typedef struct _ControlData {
    unsigned int shutdown;  // flag "continua". 0 = continua, 1 = deve terminar
    HANDLE hMapFile;        // ficheiro de memoria
    SharedMem* sharedMem;   // memoria partilhada
    HANDLE hMutex;          // mutex
    HANDLE hEvent;          // evento para leitura sincronizada
    HANDLE hEvSai;          // evento para desbloquear o gets
    HANDLE hPipe[NUSERS];   // array de handles para os pipes de cada jogador
    unsigned int nPipes; // maximo de letras
    TCHAR dicionario[3][BUFFER_SIZE + 1]; // dicionario de palavras, maximo 3 palavras
} ControlData;

typedef struct _PipeMsg {
    HANDLE hPipe;
    TCHAR buff[256];
    BOOL isUsernameInvalid;
    TCHAR username[26];
} PipeMsg;
```

Aqui estão as estruturas usadas no árbitro e no jogoui, respectivamente, são bastante similares, mas as estruturas de controle têm diferenças cruciais, as estruturas de dados do bot são exatamente iguais às do jogoui com exceção da estrutura extra que tem para se organizar.

```c
typedef struct _BotData {
    ControlData* cdata;     // ponteiro para os dados de controlo
    unsigned int velocidade; // de palavras que o bot pode enviar
} BotData;
```

Arbitro

O “arbitro”, como o nome indica é quem controla o jogo, sendo assim, o programa mais importante.

Este é único, sendo assim, este garante que, caso alguém tente adicionar outro árbitro seja impossível. Para isso, criamos um mutex, este não permite a criação de outro programa arbitro, caso já seja existente na máquina

```
HANDLE hSingle_instance = CreateMutex(NULL, FALSE, TEXT("Global\\ARBITRO_UNICO"));
if (hSingle_instance == NULL) {
    _tprintf(TEXT("Erro a criar mutex de instancia unica (%d)\n"), GetLastError());
    return 1;
}
```

Após criar este mutex pode então prosseguir para o funcionamento do jogo, começa por chamar a função “initMemAndSync” que cria e inicializa a memória partilhada, chama uma função que vai ao registry buscar os valores para o número máximo de letras e o ritmo e também inicializa alguns campos da estrutura de controle que vai ser usada ao longo do programa, é também criado um evento que é usado para alertar os clientes que tem letras novas na memória partilhada.

```
BOOL initMemAndSync(ControlData* cdata)
{

    //ir buscar os valores ao registry
    int maxLetras = 0, ritmo = 0;
    if (!readOrCreateRegistryValues(&maxLetras, &ritmo)) { ... }
    _tprintf(TEXT("MAXLETRAS: %d | RITMO: %d\n"), maxLetras, ritmo);


    //Criar ou abrir um ficheiro para mapear em memoria
    BOOL firstProcess = FALSE;

    cdata->hMapFile = OpenFileMapping(FILE_MAP_ALL_ACCESS, FALSE, SHM_NAME);//se correr bem ja existe
    if (cdata->hMapFile == NULL) { ... }
    if (cdata->hMapFile == NULL) { ... }

    //Maps a view of a file mapping into the address space of a calling process
    cdata->sharedMem = (SharedMem*)MapViewOfFile(cdata->hMapFile,//aquilo que pretendemos mapear
        FILE_MAP_ALL_ACCESS, //permissoes de acesso, tipo de acesso
        0,//USAR 0 SE O FICHEIRO FOR MENOR QUE 4GB
        0,//de onde começamos a mapear
        sizeof(SharedMem)); //tamanho max
```

De volta no main, é criada a thread que vai tratar de todos os comandos inseridos pelo árbitro, comandos estes como listar jogadores, excluir jogadores e criar bots com um nome inserido pelo árbitro.

```
DWORD WINAPI keyboardThread(LPVOID p) {
    ControlData* cdata = (ControlData*)p;
    TCHAR command[100];

    do {
        if (_getts_s(command, 100) == NULL) {

            break;
        }

        // Remove the & since cdata is already a pointer
        tratarComando(command, cdata);
    } while (cdata->shutdown == 0);

    return 0;  // Return 0 instead of NULL for DWORD
}
```

Finalmente entramos no ciclo principal do programa em que são criados tantos named pipes e threads para tratar da comunicação quantos clientes se juntarem ao jogo, são primeiro verificados os nomes inseridos pelos clientes para certificar que não há jogadores duplicados, o jogo efetivamente só começa quando se juntarem pelo menos dois jogadores.

Quando se juntarem jogadores suficientes é lançada a thread que vai escrever na memória partilhada e ativar o evento para os clientes.

```
if (startgame == 0 && cdata.sharedMem->nusers >= 2) { //se ainda não tiver começado e tiver 2 ou mais users, começa
    _tprintf(_T("Iniciando jogo com %d jogadores...\n"), cdata.sharedMem->nusers);
    startgame = 1;
    hThread = CreateThread(NULL, 0, enviaLetras, &cdata, 0, NULL);

    if (hThread == NULL) {
        _tprintf(TEXT("Error creating thread (%d)\n"), GetLastError());
        saidaordeira(&cdata, hThread, hSingle_instance);
        return 1;
    }
}
```

O árbitro permanece neste ciclo enquanto não houver sinal de shutdown, quando este for acionado, sai do ciclo, espera que a thread do teclado termine e fecha o programa.

Jogoui

O jogoui é bastante mais simples, recebe o seu nome por argumento e imediatamente tenta ter conexão com o pipe para tantar fazer login no jogo, se for bem-sucedido prossegue com o jogo, se não fecha imediatamente.

```
ZeroMemory(&loginMsg, sizeof(PipeMsg));
_tcscpy_s(loginMsg.username, 26, cdata.username);
_tcscpy_s(loginMsg.buff, 256, _T("login"));

_tprintf(_T("Enviando login...\n"));
if (!WriteFile(hPipe, &loginMsg, sizeof(PipeMsg), &bytesWritten, NULL)) {
    _tprintf(_T("[ERRO] ao enviar login! Error: %d\n"), GetLastError());
    CloseHandle(hPipe);
    return -1;
}

_tprintf(_T("Aguardando resposta de login...\n"));
if (!ReadFile(hPipe, &loginResponse, sizeof(PipeMsg), &bytesRead, NULL)) {
    _tprintf(_T("[ERRO] ao ler resposta de login! Error: %d\n"), GetLastError());
    CloseHandle(hPipe);
    return -1;
}

if (loginResponse.isUsernameInvalid) {
    _tprintf(_T("[ERRO] Nome de usuário inválido ou já em uso\n"));
    CloseHandle(hPipe);
    return -1;
}

_tprintf(_T("Login bem-sucedido como %s\n"), cdata.username);
```

Depois de um login bem-sucedido é lançada a thread recebePipe que vai receber respostas a comandos inseridos por si, vindos do árbitro, tem, tal como o árbitro uma função initMemAndSync que inicializa a memoria partilhada e algumas partes da estrutura de controle geral.

```
// Inicia thread para receber respostas
HANDLE hReceiveThread = CreateThread(NULL, 0, recebePipe, &cdata, 0, NULL);
if (hReceiveThread == NULL) {
    _tprintf(_T("Erro ao criar thread de recepção\n"));
    CloseHandle(hPipe);
    return -1;
}

// Inicializa memória partilhada
if (!initMemAndSync(&cdata)) {
    _tprintf(TEXT("Erro ao inicializar memória compartilhada\n"));
    CloseHandle(hPipe);
    CloseHandle(hReceiveThread);
    exit(-1);
}
```

Lança também logo de seguida a thread comunica que vai ler, em loop, o teclado, verificar se foi inserido um comando ou uma tentativa de adivinhar palavra, e envia o que tiver sido inserido (comando ou palavra) para o árbitro, este lê e responde apropriadamente.

Finalmente é aberto o evento para leitura da memória partilhada e é lançada a thread que a vai ler e apresentar no ecrã sempre que houver alterações.

```
hEvent = OpenEvent(EVENT_ALL_ACCESS, FALSE, EVENT_NAME);
if (hEvent == NULL) {
    _tprintf_s(_T("Erro ao abrir evento: %ld\n"), GetLastError());
    exit(-1);
}
cdata.hEvent = hEvent;

hThread = CreateThread(NULL, 0, lerMemPart, &cdata, 0, NULL);
if (hThread == NULL) {
    _tprintf(TEXT("Erro ao criar thread de leitura da memória partilhada (%d)\n"), GetLastError());
    CloseHandle(hPipe);
    CloseHandle(hReceiveThread);
    exit(-1);
}
```

Bot

Como dito anteriormente o programa bot é semelhante ao programa jogoui, no entanto este não qualquer interação com o utilizador, sendo “lançado” e funcionando sozinho. Este quando é executado necessita de dois argumentos adicionais, o seu “nome” e a velocidade que deve enviar as palavras.

A comunicação com o programa árbitro é igual ao jogoui, ou seja, sabe as letras através da memória partilhada e envia palavras através de named pipes.

A diferença deste programa para o jogoui, é que este efetua o lançamento das palavras automaticamente, para isso, este sabe o dicionário de palavras possíveis e envia uma dessas palavras.

```
DWORD WINAPI enviaPalavras(LPVOID param) {
    BotData* bdata = (BotData*)param;
    PipeMsg msg;
    DWORD bytesWritten;

    // Lista de palavras que o bot vai enviar
    const TCHAR* palavras[] = {
        _T("carro"),
        _T("mae"),
        _T("pai")
    };
    int numPalavras = sizeof(palavras) / sizeof(palavras[0]);

    // Intervalo entre envios (em milissegundos)


    _tcscpy_s(msg.username, 26, bdata->cdata->username);

    while (bdata->cdata->shutdown == 0) {
        // Seleciona uma palavra aleatória da lista
        int idx = rand() % numPalavras;
        _tcscpy_s(msg.buff, 256, palavras[idx]);

        _tprintf(_T("Enviando palavra: %s\n"), msg.buff);
        if (!WriteFile(bdata->cdata->hPipe[0], &msg, sizeof(PipeMsg), &bytesWritten, NULL)) {
            _tprintf(_T("[ERRO] ao enviar palavra! Error: %d\n"), GetLastError());
            break;
        }
        Sleep(1000 * bdata->velocidade);

    }
    return 0;
}
```

Tabela de Conteúdos

ID	Descrição funcionalidade / requisito	Estado
1	As letras são apresentadas de forma aleatória, a uma cadência fixa	implementado
2	Trocar a letra mais antiga	implementado
3	Sistema de pontos	Implementado
4	Remover letras do array	Implementado
5	Limite users	Implementado
6	Número mínimo para começar	Implementado
7	Verificação de users	Implementado
8	Informação entre processos/programas	Implementado
9	Diferença entre comando e palavras	Implementado
10	Informação de jogo/participantes	Implementado
11	Manipulação de utilizadores	Implementado
12	Velocidade de jogo	Implementado
13	painel	Não Implementado
14	Uso registry	Implementado
15	Named pipdes	Implementado
16	Threads	Implementado
17	Mutex	Implementado
18	Memória Partilhada	Implementado

O painel não foi implementado, devido a sua complexidade e tempo necessário, para efetuar testes, corrigir bugs, entre outros problemas adjacentes

Conclusão

A realização deste projeto permitiu aplicar, diversos conceitos fundamentais da unidade curricular de Sistemas Operativos 2, como memória partilhada, sincronização com mutexes e eventos, comunicação interprocessos através de named pipes e utilização de threads.

Apesar de o foco não ter sido na complexidade do jogo em si, foi possível desenvolver uma aplicação funcional e estável, com suporte para múltiplos jogadores e interações em tempo real. A separação em três programas distintos (“arbitro”, “jogoui” e “bot”) evidenciou a importância da coordenação entre processos e a gestão rigorosa da partilha de recursos.

Em suma, este trabalho constituiu uma oportunidade valiosa para consolidar conhecimentos sobre os mecanismos de baixo nível do sistema operativo Windows.