Sistema de Gestão do Espaço Aéreo
Sistemas Operativos 2
Relatório meta 1
2020/2021

Francisco Amaral 2018019123
João Valente 2017016033

Estruturas de dados

Estrutura do espaço aéreo, que tem todas as informações do controlador:

```
typedef struct {
    AirPlane* airPlanes;
    int nAirPlanes;
    int maxAirPlanes; // max de avioes
    Airport* airports;
    int nAirports;
    int maxAirports;
    //Variaveis de controlo:
    BOOL endThreadReceiveInfo; //1 caso seja para terminar a thread
    HANDLE hMutex;
    HANDLE hEvent;
    HANDLE hEvent2;
    HANDLE hEvent3;
    Map* mapMemory;
    AirPlane* airPlaneMemory;
    HANDLE aerialMapObjMap;
    HANDLE airPlaneObjMap;
} AerialSpace;
```

Esta estrutura tem o array de aviões, de aeroportos, variáveis de controlos de eventos e mutexes, e variáveis essenciais para o uso de memória partilhada.

Estrutura do avião:

```
typedef struct {
    DWORD id; //id do processo em si DWORD
    time_t tm;
    Coordenates coordenates;
    Airport airportDestiny;
    TCHAR InitialAirport[100];
    BOOL flying; //Se está a voar ou não
    int capacity; //Numero máximo de passajeiros do avião
    int speed;  //Velocidade máxima do avião (posições por segundo)
    //Variaveis de pedidos
    BOOL refreshData; //Pedido para atualizar dados
    BOOL startingLife;  //Pedido no inicio do programa
    BOOL boarding;      //Pedido para iniciar viagem
    BOOL requestFlying;
    BOOL requestDestiny;
    BOOL exit;          //Caso saia do programa
    //Variavel de resposta
    BOOL answer;
} AirPlane;
```

Esta estrutura, além de ter a informação do avião (id, coordenadas, destino e aeroporto que se encontra, velocidade, capacidade máxima), tem as variáveis que funcionam como pedido ao controlador, assim como uma variável de resposta. A variável ™ tem como objetivo controlar o ping feito de 3s em 3s.

Estrutura do aeroporto:

```
typedef struct {
    Coordenates coordenates;
    TCHAR name[100];
    Passanger passengers[20];
    int nPassengers;
} Airport;
```

Um aeroporto tem umas coordenadas, um nome e um conjunto de passageiros.

Estrutura de controlo do mapa:

```
typedef struct {
    Coordenates coordenates;
    int id;
} AerialAirplane;

typedef struct {
    AerialAirplane posBusy[5];
    int tam;
} Map;
```

Tem um array de posições ocupadas, onde cada posição tem o id do avião correspondente.

Estrutura de dados do programa avião:

```
typedef struct {
    AirPlane airplane;
    //HANDLES
    HANDLE hMapFileCom;
    HANDLE hMapFileMap;
    HANDLE hMutexMemory; //Mutexe para controlar os copyMemories
    AirPlane* memory;
    Map* memoryMap;
    HANDLE hEvent;
    HANDLE hEvent2;
    HANDLE hMutex;
    // int finish;
} Data;
```

Além do avião em si, tem todas as variáveis de controlo de eventos, mutexes e memória partilhada.

Comunicação

Toda a comunicação é feita em base de pedidos, especificamente nas variáveis requestFlying e requestDestiny.

O controlador tem a thread **waitingAirplaneInfo** que recebe dados do avião, através de um **CopyMemory**. Este copy memory envia sempre uma estrutura do tipo AirPlane. Dependendo do tipo de pedido, esta thread atua de forma diferente,fazendo as suas operações e de seguida, enviando uma resposta para o avião, também através de um CopyMemory. Toda a comunicação tem mecanismos de sincronização, como mutexes antes de dar copymemory, e também eventos, para garantir que a informação só é acedida quando devido.

Threads

Control:

O programa controlador tem 3 threads, uma que está sempre à espera de informação dos passageiros (ainda não implementada nesta meta), outra à espera de informação dos aviões, e uma terceira que vai atualizando os dados do mapa consoante os aviões que se encontram a voar.

Razões de implementação

O programa verifica se é o único programa desse tipo a correr, tentando abrir um semáforo, caso o consiga abrir, é porque existe uma outra instância desse programa. Os pedidos são feitos com as variáveis descritas anteriormente, sendo que alguns desses pedidos recebem uma resposta.

Funcionalidades implementadas

Control

Funcionalidade	**Estado**
Criar aeroporto	Implementado
Mostrar aeroporto	Implementado
Mostrar aviões	Implementado
Aceita novos aviões	Implementado
Escreve/obtém valores do registry	Implementado
Verifica unicidade do programa	Implementado
Estabelece conexão com aviões(escrita e leitura)	Implementado
Verificação da atividade dos aviões(timeout de 3s)	Por implementar

Controlo da quantidade de aviões	Implementado

Avião

Funcionalidade	**Estado**
Recebe input através de argumentos	Implementado
Estabelece conexão com control(escrita e leitura)	Implementado
Calcula posicionamento(voo)	Implementado
Verifica “colisões” entre aviões	Implementado
Ligação com a DLL	Implementado
Buffer Circular	Por implementar