
Licenciatura em Engenharia Informática

Regime Pós-Laboral

# Sistemas Operativos II

Primeira Meta

**2020/2021**

Liliana Faustino 2017012944

Paul Ionut Bob 2016013407

# Introdução

O objetivo deste relatório é explicar as principais decisões no que toca à arquitectura e implementação do sistema Control. Pretende-se que com uma leitura breve seja possível perceber qual foi o pensamento por detrás de cada decisão e como a mesma foi concretizada.

Devido a ter sido um trabalho desenvolvido ao longo dos últimos tempos e que com uma evolução de conhecimento constante, existem decisões que na altura pareceram a melhor opção mas que se fossem feitas hoje não seriam as mesmas. Ao longo deste relatório estes pormenores são explicados e também é apresentada a razão pela qual esta questão não foi alterada.

O formato deste relatório irá respeitar o fluxo do controlador e irá seguir o processo utilizado por todo o sistema para se organizar.

# Estruturas de Dados

## Estruturas Partilhadas

```
struct Aviao {
	int maxCapacity, velocity, planeID, index;
	TCHAR actualAirport[NAMESIZE];
	TCHAR destinAirport[NAMESIZE];
	TCHAR departureAirport[NAMESIZE];
	Coordinate initial, current, final;
	HANDLE heartbeatTimer;
};
```

Existe uma estrutura do tipo “Avião” que tem os dados que o controlador necessita saber para que o sistema funcione.

```
enum messageType { Register, Departure, Arrive, Heartbeat };
typedef struct protocolo Protocol, * pProtocol;
struct protocolo {
	enum messageType type;
	int planeID;
	int index;
};
```

O protocolo de comunicação é necessário para o produtor/consumidor (explicado mais abaixo neste relatório).

```
struct produtorConsumidor {
	int in, out;
	Protocol buffer[DIM_BUFFER];
};
```

Estrutura onde é gerida a informação do produtor/consumidor.

```
struct aeroporto {
	TCHAR name[NAMESIZE];
	int coordinates[2]; //sendo coordinates[0] o x
};
```

A informação dos aeroportos também é partilhada em memória (para que o avião tenha acesso às coordenadas do mesmo).

```
struct dados {
	HANDLE objMap;
	HANDLE objAirports;
	HANDLE objPlanes;
	HANDLE objProducerConsumer;
	HANDLE airportsMutex;
	HANDLE mapMutex;
	HANDLE producerConsumerThread;
	HANDLE itemsSemaphore;
	HANDLE emptiesSemaphore;
	HANDLE controlPlanes;
	pProducerConsumer producerConsumer;
	pMap map;
	pAirport airports;
	pPlane planes;
	int maxAirports  , nrAirports;
	int maxAirplanes , nrAirplanes;
};
```

Estrutura da informação necessária ao funcionamento do acesso à melhoria partilhada, criação de vectores de informação, e todos os mecanismos de sincronização necessários: Mutex, semáforos, etc.

## Estruturas do Avião

```
struct Dados {
	pPlane plane;
	int ongoingTrip;
	HANDLE tripThread;
	HANDLE heartbeatThread;
	CRITICAL_SECTION criticalSection;
	HANDLE controlSemaphore;
	HANDLE producerMutex;
	HANDLE emptiesSemaphore;
	HANDLE itemsSemaphore;
	HANDLE objProducerConsumer;
	HANDLE objAirports;
	HANDLE objPlanes;
```

```
	pPlane planes;
	int maxAirports;
	pAirport airports;
	pProducerConsumer producerConsumer;
};
```

Estrutura onde é guardada toda a informação que só é necessária ao processo do avião, nomeadamente, a critical section que controla se o avião está em andamento ou não, as características do avião, e todos os mecanismos de comunicação com o controlador.

# Implementação

## Criação de Aeroportos

O controlador começa por criar uma zona de memória partilhada com um vetor de aeroportos. Esse vetor mais tarde poderá ser consultado pelos aviões para a execução de algumas tarefas. Através do comando “criar [nomeAeroporto] [coordenadaX] [coordenadaY]” é possível a criação de um aeroporto. Caso exista algum aeroporto num raio de 10 posições o controlador não permite a criação do aeroporto, fazendo esta verificação diretamente no mapa. As coordenadas possíveis para a criação de aeroportos pertencem ao intervalo [0,999] tanto para a coordenada X como para a coordenada Y.

## Registo dos Aviões

Após o lançamento de um avião pelo comando “plane [Lotação] [velocidade em distância/segundos] [Aeroporto em que se deseja registar]” o próprio software verifica um semáforo de controle e caso exista lugares avança, caso contrário fica a espera até que haja uma vaga disponível. O avião acede à memória partilhada dos aeroportos e percorre o vetor à procura de uma correspondência através do nome com o aeroporto no qual se deseja registar. Optamos por esta implementação não apenas pela sua simplicidade mas também porque achamos inconveniente o controlador estar a receber inúmeros pedidos de registo inválidos dos aviões… O avião ainda acede a um vetor de aviões , também este disponível em memória partilhada e procura um “lugar vazio” para se registar, quando o encontra preenche esse lugar no array e guarda o index tal como um ponteiro para essa zona de memória partilhada. A partir desse momento qualquer modificação no avião será efetuada através desse ponteiro diretamente na zona de memória partilhada.

## Atualização do Mapa

O mapa do controlador está numa matriz de 1000x1000 onde são representados os espaços vazios (0), os aviões em voo (1) e os aeroportos (2). A atualização do mapa é feita pelos aviões, a cada vez que se deslocam para uma célula que esteja vazia. Esta atualização é feita com o acesso à memória partilhada que ao instanciar o controlador é criada.

Tanto os aviões têm poderes de leitura/escrita (para serem capazes de verificar a ocupação e para se registarem nas células) como o controlador (para criarem os aeroportos).
Toda a atualização do mapa é feita com recurso a um Mutex chamado “mapMutex” que controla o acesso ao mapa (quer de leitura, quer de escrita). Cada avião/aeroporto só irá aceder à memória para ler/escrever quando mais ninguém o estiver a fazer.

# Comunicação Produtor/Consumidor

O Controlador é o consumidor deste paradigma. Para que essa comunicação fosse possível, foi definido um protocolo que contém um tipo de mensagem (messageType), o avião que está a enviar a mensagem (planeID) e a posição do mesmo no vetor de aviões (index).
Existem 4 tipos de mensagens possíveis: O registo de um novo avião, o levantamento de voo, a chegada ao aeroporto e o heartbeat. Este último será analisado no tópico seguinte deste relatório.
Relativamente ao registo, o avião quando arranca tenta registar-se no controlador. Este registo só é feito se existir espaço no vetor de aviões e é controlado pelo semáforo controlSemaphore. A partir do momento em que existe espaço, esse avião é registrado.
No levantamento de voo e na chegada ao aeroporto, o controlador recebe uma mensagem a avisar de que foi realizada esta viagem. Essa mensagem é apresentada no ecrã.
Ao longo de toda a viagem, é possível saber onde estão os aviões. Optámos por não estar sempre a enviar para o consumidor a posição por uma questão de desempenho e de limpeza das mensagens.

# Heartbeat

O tipo de mensagem heartbeat foi desenhado para responder à necessidade de o controlador considerar que o avião terminou se não receber novidades em 3 segundos. Desta forma, temos uma Thread (hearbeat) que de 2,5 em 2,5 segundos envia uma mensagem deste tipo ao controlador.
Do lado do controlador, temos um Waitable Timer associado a cada avião ao qual é feito reset cada vez que recebe uma mensagem do tipo heartbeat. Temos também uma thread por avião “parada” à espera que o waitable timer dispare. Se não receber nenhuma mensagem do tipo heartbeat em 3 segundos, a thread desbloqueia e vai remover da memória o registo do avião e libertar o espaço no vetor.

# Manual de Utilização

## Geral

1. O sistema deve ser compilado na versão x86. Deverá garantir que a DLL se encontra na pasta de Debug.

## Controlador

1. Iniciar o executável;
2. Criar 2 ou mais aeroportos através do comando:
    - criar [nome do aeroporto] [coordenada x] [coordenada y]
3. Para listar os aeroportos disponíveis e quais os aviões registados, basta fazer o comando:
    - listar
4. Para encerrar o sistema utiliza o comando:
    - exit
5. Para ajudar o utilizador e ter toda a informação necessária para usar o sistema, basta executar o comando:
    - help

## Avião

1. Para iniciar o programa, é necessário lançá-lo pela linha de comandos, utilizando o seguinte comando:
    - Plane [Lotação] [Velocidade em distância/segundos] [Aeroporto em que se deseja registar]
        i. O aeroporto deve respeitar o formato de string
2. Definir o destino (também em formato string) através do comando
    - destino [nome do aeroporto]
3. Iniciar a deslocação através do comando
    - iniciar
4. Para encerrar o avião utiliza o comando:
    - exit