Sistemas Operativos II

Sistema de Gestão do Espaço Aéreo

2020 - 2021

TheForgotten
BrunoTeixeira1996

Licenciatura de Engenharia Informática
13 de junho de 2021

Conteúdo

1 Introdução

O trabalho prático consiste na implementação de vários programas que, no seu conjunto, simulam um sistema de gestão de aeroportos, espaço aéreo, passageiros e aviões que nele circulam.

O trabalho prático foi concretizado em linguagem C usando a API do Windows chamada de Win32. Para esta segunda meta foi criada uma GUI para o **Control**, mantendo o paradigma de consola para as restante aplicações.

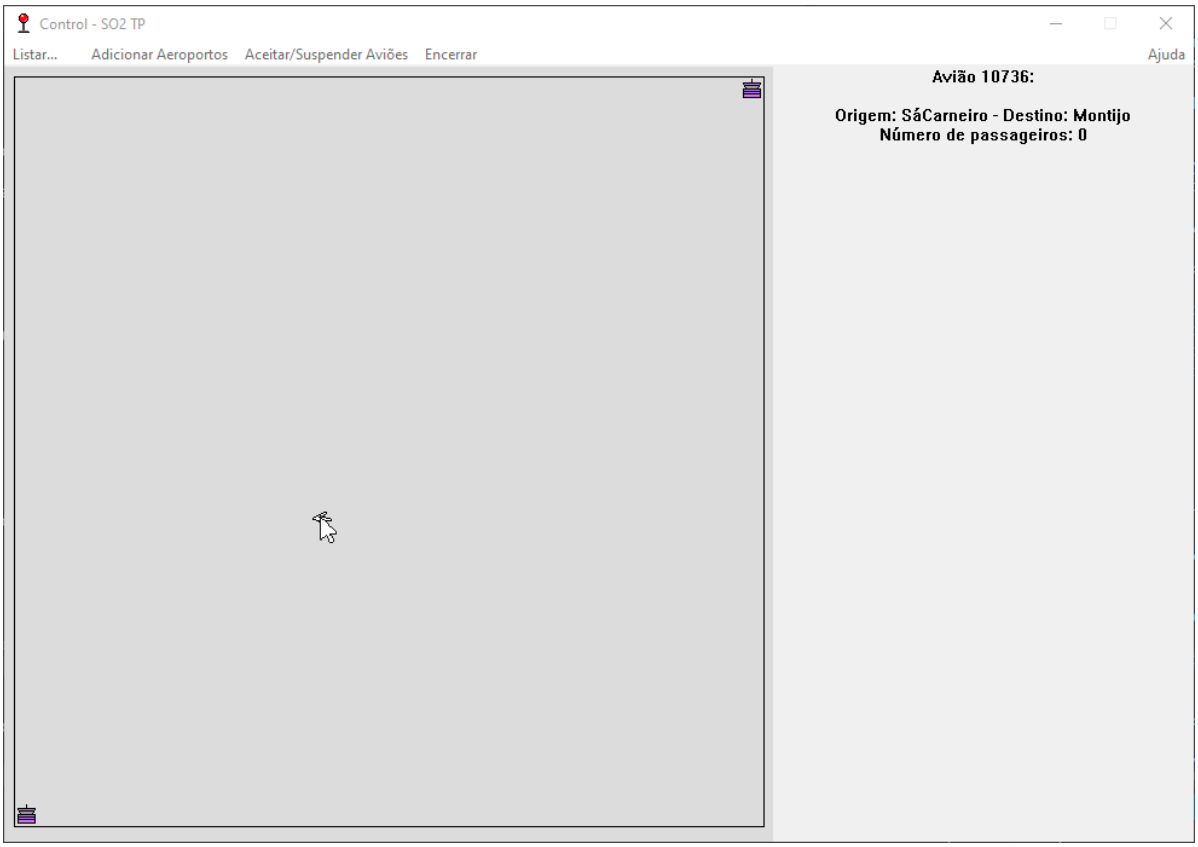

Figura 1: Control - Hover num Avião

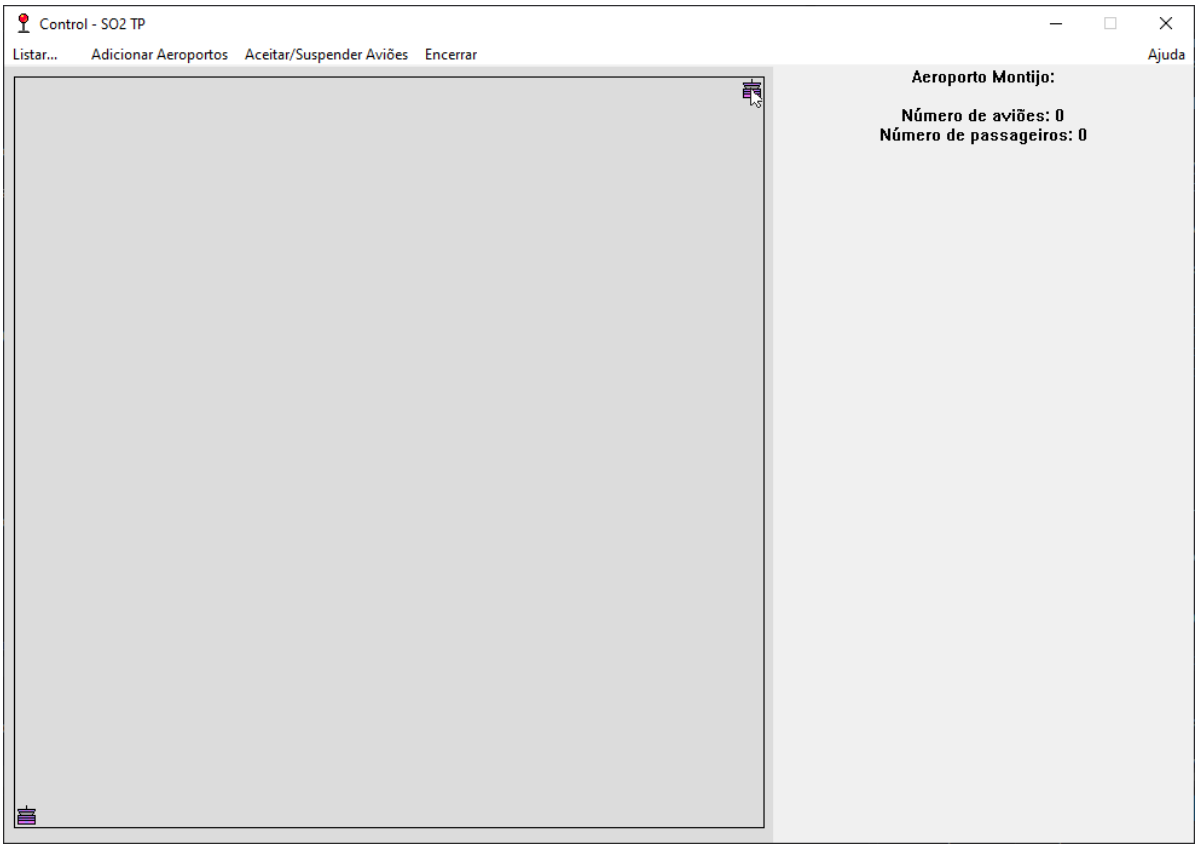

Figura 2: Control - Clique num Aeroporto

2 Correções Feitas Após a Primeira Meta

Agora toda a comunicação do avião com o Control é única e exclusivamente feita por buffer circular, ao contrário do que era feito na 1ª meta.

Os aeroportos já não se encontram no Registry. Em vez disso, existe um bloco de memória partilhada onde os aviões acedem aos aeroportos que existem.

Agora o avião assinala um evento para escrever a sua estrutura atualizada no buffer circular quando se move, ao contrário do que fazia antes que escrevia diretamente na memória partilhada das coordenadas.

A memória partilhada que era usada para escrever aviões e ver que coordenadas estavam ou não ocupadas é agora exclusivamente para as coordenadas, os aviões apenas leêm o bloco de memória, não lhe fazendo alterações. Este bloco é atualizado apenas pelo Control, uma vez que o Control é que contém essas coordenadas localmente.

3 Arquitetura Geral

A aplicação **Control** é a responsável por gerir todo o funcionamento do espaço aéreo. Sem esta, nem os aviões nem os passageiros conseguem funcionar. Deste modo, esta deve ser sempre aberta

antes dos outros.

Depois é só inicializar quantos aviões desejar, estes comunicarão com o **Control** por buffer circular usando o paradigma produtor/consumidor, sendo o avião considerado o produtor e o controlador o consumidor.

O avião irá avisar o **Control** momentaneamente de que este se encontra ligado e a funcionar normalmente, usando uma thread para isso.

Depois o processo de verificação de posições ocupadas por parte dos aviões, é feita também com memória partilhada, uma vez que o controlador vai atualizando a memória partilhada com as coordenadas ocupadas, o avião gera umas novas coordenadas com a **DLL** disponibilizada e depois vai verificar nessa memória partilhada se essas coordenadas estão ou não livres, caso não estejam o mesmo volta a gerar outras coordenadas sempre usando a **DLL** para esse efeito.

Os passageiros comunicam com o **Control** usando named pipes. Os passageiros nunca comunicam diretamente com os aviões, nem vice-versa. Podemos abrir quantos passageiros desejarmos, indicando por argumento o nome, onde está, para onde quer ir e, opcionalmente, quanto tempo (em segundos) fica à espera para embarcar.

4 Mecanismos de Sincronização

Para garantir que apenas existe uma instância da aplicação **Control** é usado um semáforo com nome. Para gerir dados partilhados entre threads do mesmo processo são usadas secções críticas. No entanto para gerir dados entre processos diferentes usamos um mutex com nome.

O **Control** garante também que apenas são permitidos aviões consoante o número máximo definido no início da execução. Este controlo é feito usando um semáforo que é criado no início da execução do **Control**, semáforo este que apenas tem lugar igual ao número máximo de aviões permitidos.

O mecanismo de **ping** do avião em relação ao **Control** funciona com eventos com nome. É criado um evento com um nome genérico baseado no ID de processo do avião, e depois é aberto no lado do **Control**. Este fica 3 segundos à espera do evento ser assinalado e, no caso de não ser assinalado, deixa de dar atenção ao avião e assume que deixou de existir.

No paradigma de produtor/consumidor, uma vez que existem **N** produtores e apenas um consumidor, usamos dois semáforos, um de leitura e outro de escrita, usamos também algumas secções criticas dentro dos processos e usamos um mutex apenas para o produtor, de modo a garantirmos um correto funcionamento do paradigma.

Dentro do **Control**, de modo a informar as threads de passageiro que há um avião há procura de passageiros para embarcar, são usados eventos e, de modo a diminuir o tráfego no acesso à informação sobre os aviões, são usados eventos com nomes genéricos. A cada instância que se liga ao **named pipe** é atribuida uma thread que lida com a leitura e escrita no mesmo.

5 Uso da DLL

Na nossa implementação, decidimos usar a **DLL** de forma explícita por acharmos que traz mais vantagens em relação à implícita.

O código não é muito mais difícil na explícita e, para além disso, com este tipo de implementação é muito mais fácil fazer mudanças na **DLL**. Também nos agradou o facto de não ser preciso fazer alterações nas definições de projeto no Visual Studio.

6 Estruturas de Dados

6.1 Struct COORDINATES

```
struct COORDINATES_STRUCT{
    int x, y;
};
```

Código 1: Struct COORDINATES

Estrutura responsável pelas coordenadas.

6.2 Struct SHARED_COORDINATES

```
struct SHARED_COORDINATES_STRUCT {
   int x, y;
   unsigned int maxAirplanes;
  };
```

Código 2: Struct SHARED_COORDINATES

Estrutura utilizada para o uso da memória partilhada para a verificação das coordenadas.

6.3 Struct AIRPLANE

```
struct AIRPLANE_STRUCT {
     DWORD id;
       unsigned int capacity, velocity;
       coordinatesStruct currentCoordinates; // coordenadas atuais
       coordinatesStruct destinationCoordinates; // coordenadas do destino
       TCHAR destAirport[STR_SIZE], srcAirport[STR_SIZE];
       BOOL stopped; // está parado ?
};
```

Código 3: Struct AIRPLANE

Esta é a estrutura associada aos aviões. Cada avião vai estar associado a uma estrutura destas.

6.4 Struct AIRPORT

```
struct AIRPLANE_STRUCT {
        TCHAR name[STR_SIZE];

         coordinatesStruct coordinates;

         unsigned int maxAirports;

         unsigned int currentPassengers;
};
```

Código 4: Struct AIRPORT

Esta é a estrutura associada aos aeroportos. Cada aeroporto vai estar associado a uma estrutura destas.

6.5 Struct PASSENGER

```
struct AIRPLANE_STRUCT {
        TCHAR name[STR_SIZE];
         DWORD id, airplaneId;

         TCHAR destAirport[STR_SIZE], srcAirport[STR_SIZE];

         unsigned int waitToBoard;

         BOOL stopped;
};
```

Código 5: Struct PASSENGER

Esta é a estrutura associada aos aeroportos. Cada aeroporto vai estar associado a uma estrutura destas.

6.6 Struct SHAREDMEM

```
struct SHAREDMEM_STRUCT {
 int nProducers;
 int writeIndex; // Próxima posição de escrita
 int readIndex; // Próxima posição de leitura

 airplane buffer[CIRCULAR_BUFFER_SIZE]; // Buffer circular em si (array de
     estruturas)
};
```

Código 6: Struct SHAREDMEM_STRUCT

Estrutura utilizada para o uso da memória partilhada usando o paradigma produtor/consumidor.

7 Controlador Aéreo

7.1 Funcionalidades principais

7.1.1 Controlo da informação de aeroportos e aviões

Toda a informação relativamente aos aeroportos, aviões e passageiros está guardada localmente no **Control** para que fique mais fácil manipular todas as estruturas de dados. O **Control** guarda também um array de handles usado na thread de avisos dos aviões, assim como um array de estruturas dos **Pings**.

Na nossa interface representamos o mapa em escala, ou seja, representamos o campo 1000x1000 num espaço de 600x600. Esta escala pode facilmente ser alterada.

7.1.2 Criação de aeroportos

Usando o botão "Adicionar Aeroportos"do menu superior podemos adicionar aeroportos. Este botão irá abrir uma **dialog box** onde podemos inserir todos os dados necessários à criação do mesmo. A informação recolhida é então tratada e passada para dentro da função *caeroporto*, que irá tratar de adicionar o aeroporto ao array.

7.1.3 Atualização das posições dos aviões

O **Control** tem uma thread responsável por interagir com a memória partilhada (**coordinatesThread**), memória esta que é usada pelo avião para verificar a ocupação de posições.

Inicialmente o **Control** preenche a memória partilhada com um array de estruturas do tipo **SHARED_COORDINATES**, tendo estas estruturas a informação do número de aviões existentes e as coordenadas ocupadas.

Depois, em curtos intervalos de tempos, vai atualizando este espaço de memória, mantendo sempre neste espaço apenas as coordenadas ocupadas. O número de aviões existentes também vai sendo atualizado.

8 Aviões

O programa **Airplanes** representa cada instância de um avião distinto. Este programa é lançado pelo utilizador e faz várias viagens.

8.1 Funcionalidades principais

8.1.1 Lançamento

No início da execução, é pedido ao utilizador a lotação, a velocidade em posições por segundo e o aeroporto inicial onde se encontra.

Depois de serem validados todos os parâmetros introduzidos pelo utilizador, o avião vai ao buffer circular que está em memória partilhada com o **Control** e escreve lá a sua estrutura, como uma espécie de inscrição.

8.1.2 Movimentação no espaço aéreo

O avião antes de iniciar a movimentação gera umas coordenadas usando a **DLL** e logo de seguida vê se o **Mutex** partilhado entre ele e o **Control** está livre, caso esteja significa que a memória partilhada ja contém as coordenadas ocupadas atualizadas. De seguida o avião acede à memória partilhada e verifica se as coordenadas geradas pela **DLL** são ou não válidas, caso sejam, o mesmo aciona um evento que irá pedir a uma thread interna para escrever a sua struct de novo no buffer circular e avança uma posição, caso não sejam o avião volta a chamar a **DLL** até encontrar umas coordenadas válidas. Se chegar ao fim a **DLL** retorna 0, sendo assim o mesmo aciona o evento mencionado anteriormente e muda o seu estado **hasArrivedAtDestination** para **TRUE**.

Uma vez que a **DLL** fornece coordenadas seguidas (1,1 ou 2,2, ...) tivemos de implementar uma solução para isto. A solução encontrada foi, no processo de chamada à **DLL**, as coordenadas do aeroporto de destino foram alteradas de maneira aleatória, somando um valor entre 0 e 1000 às mesmas (valores corrigidos na seguinte iteração), forçando o avião a ir por um caminho alternativo nunca entrando em colisão contra outro avião.

9 Passageiros

O programa **Passengers** representa cada instância de um passageiro distinto. Este programa é lançado pelo utilizador e faz apenas uma viagem.

9.1 Funcionalidades principais

9.1.1 Lançamento

Ao iniciar o programa, deve ser indicado por argumento o nome do passageiro, donde parte, para onde quer ir e quanto tempo fica à espera para embarcar, sendo este último opcional.

Após isto, ele apenas serve para imprimir informação, não voltando a receber mais dados do utilizador.

10 Manual de Utilização

10.1 Control

Para receber ajuda quanto à utilização desta aplicação basta usar a opção "Ajuda"no menu.

10.2 Airplanes

Comando	Modo de uso	Descrição
destino	destino <nome_aeroporto>	Define o destino do avião
embarcar	embarcar	Avisa o Control que pretende que passageiros embarquem
inicia	inicia	Inicia a viagem
terminar	terminar	Desliga o programa

10.3 Passageiros

Esta aplicação não aceita qualquer tipo de interação do utilizador depois de estar aberta. Para isso, deve ser aberta com argumentos, usando o template *app.exe <nome><aeroporto origem><aeroporto destino><tempo espera (opcional)>*, onde *nome* é o nome do passageiro em particular, *aeroporto origem* é o aeroporto onde este inicialmente se encontra, *aeroporto destino* o aeroporto para onde este se pretende dirigir e o *tempo espera* o tempo máximo que este fica a aguardar por embarcar, podendo não ser mencionado ou ser dado o valor '0' e, se assim for, fica à espera infinitamente.

10.4 Reg

Reg é uma pequena aplicação feita com o intuito de facilitar a criação das chaves no **registry** do Windows.

Esta começa por perguntar o que pretendemos que seja o número máximo de aviões e o que pretendemos que seja o número máximo de aeroportos. Em seguida cria as chaves e os atributos valor no **registry**, tendo a chave o path "HKEY_CURRENT_USER\SOFTWARE\SO2-TP\MaximumValues"e sendo os atributos valor "MaxAirplanes"e "MaxAirports".

11 Tabela de Funcionalidades Implementadas

ID	Descrição da Funcionalidade / Requisito	Estado
0	Interface Gráfica para Control	Implementada
1	Mouseover do Rato em Cima de um Avião em Voo Mostra Informação Adicional	Implementada
2	Click do rato em cima de um aeroporto mostra informação adicional	Implementada
3	Menu e Dialogbox na Interface Gráfica	Implementada
4	Visualização do Espaço Aéreo	Implementada
5	Uso de AlphaBlend (não requisito)	Implementada
6	Criação de BMPs 16x16 para aeroporto, avião e ícone do Control	Implementada
7	Buffer Circular (Comunicação Control <- Aviões)	Implementada
8	Movimentação dos Aviões (com recurso à DLL)	Implementada
9	Ping dos Aviões ao Control	Implementada
10	Não Colisão dos Aviões	Implementada
11	Passageiros	Implementada
12	Comunicação Control <->Passageiros com Named Pipes	Implementada
13	Tirar Valores Máximos do Registry	Implementada
14	Garantir que Existe Apenas 1 Control	Implementada
15	Aeroportos (possuir nome único e serem únicos num raio (quadrado) de 10)	Implementada

Temos confiança que implementamos todo o que era requisitado e ainda mais algumas coisas que achamos que davam *flavor* ao produto final.

12 Conclusão

Ao longo deste trabalho, deparámo-nos e fomos resolvendo vários desafios que não esperávamos ter, tratando-se de uma excelente oportunidade para consolidação de matéria das aulas teóricas e práticas de Sistemas Operativos 2. Este permitiu-nos colocar em prática conceitos importantes sobre a arquitetura, criação e desenvolvimento de programas para sistemas NT, dando-nos uma visão para os vários detalhes dessas atividades.

13 Anexos

Lista de Figuras

Pedaços de Código