

Projeto *IscteFlight* (Parte 3)

O presente trabalho visa aplicar os conhecimentos adquiridos durante as aulas de Sistemas Operativos e será composto por três partes. O objetivo é desenvolver os diferentes aspetos e funcionalidades da plataforma **IscteFlight**. Iremos procurar minimizar as interdependências entre partes do trabalho.

Este enunciado detalha **apenas** as funcionalidades que devem ser implementadas na parte 3 do trabalho.

Este trabalho está publicado na sua [versão 3](#). O [Anexo D: Histórico de Versões](#) apresenta as alterações efetuadas.

A plataforma **IscteFlight** é um sistema de gestão de reservas de voos totalmente automatizada. O seu objetivo é facilitar a interação entre os passageiros e os serviços oferecidos por uma companhia aérea.

O propósito desta terceira parte é lidar com a [escolha de lugares na viagem de avião](#).



Os alunos não deverão usar a função **printf** (não será analisado para efeitos de avaliação, a não ser que seja explicitamente indicado) para as respostas diretas às alíneas. Deverão sim usar as macros **so_success** (para as respostas de sucesso) e **so_error** (para as respostas de insucesso), ver [Anexo A: Macros de suporte ao trabalho](#) no final do enunciado. A função **printf** pode, no entanto, ser usada para as interações com o utilizador.

Não serão aceites trabalhos em scripts "*from scratch*", mas sim apenas aqueles usando os *templates* fornecidos.

Essa *baseline* para o trabalho encontra-se no Tigre, na diretoria [/home/so/trabalho-2023-2024/parte-3](#):

- **EXECUTE, a partir da sua diretoria local de projeto, e sem mudar os nomes, o seguinte comando:**

```
$ cp -rn /home/so/trabalho-2023-2024/parte-3 .
```

Procedimento de entrega e submissão do trabalho

O trabalho de SO será realizado individualmente, logo sem recurso a grupos.

A entrega da Parte 3 do trabalho será realizada através da criação de um ficheiro ZIP cujo nome é o nº do aluno, e.g., “a<nºaluno>-parte-3.zip” (ATENÇÃO: não serão aceites ficheiros RAR, 7Z ou outro formato) onde estarão todos os ficheiros criados. Estes serão apenas os ficheiros de código, ou seja, na parte 3, apenas os ficheiros (*.c *.h).

Cada um dos módulos será desenvolvido com base nos ficheiros fornecidos, e que estão na diretoria do Tigre “/home/so/trabalho-2023-2024/parte-3”, e deverá incluir nos comentários iniciais um “relatório” indicando a descrição do módulo e explicação do mesmo (poderá ser muito reduzida se o código tiver comentários descriptivos).

Para criar o ficheiro ZIP, use, no Tigre, o comando **zip a<nº aluno>-parte-3.zip <ficheiros>**, por exemplo:

```
$ zip $USER-part-3.zip *.c *.h
```

O ficheiro ZIP deverá depois ser transferido do Tigre para a sua área local (Windows/Linux/Mac) via SFTP, para depois ser submetido via Moodle (ver no Moodle a KB Basics / Criar ficheiro ZIP para submeter trabalho no Moodle).

Antes de submeter, por favor valide que o ficheiro ZIP não inclui diretórios ou ficheiros extra indesejados.

A entrega desta parte do trabalho deverá ser feita por via eletrónica, através do Moodle:

- Moodle da UC Sistemas Operativos, selecione a opção sub-menu “Quizzes & Assignments”.
- Selecione o link “Submit SO Assignment 2023-2024 Part 3”.
- Dentro do formulário, selecione o botão “Enviar trabalho” e anexe o seu ficheiro **.zip** (a forma mais fácil é simplesmente fazer via “drag-and-drop”) e selecione o botão “Guardar alterações”. Poderá depois, mais tarde, ressubmeter o seu trabalho as vezes que desejar, enquanto estiver dentro do prazo para entrega do trabalho. Para isso, na mesma opção, pressione o botão “Editar submissão”, selecione o ficheiro, e depois o botão “Apagar”, sendo que depois pode arrastar o novo ficheiro **.zip** e pressionar “Guardar alterações”. **Apenas a última submissão será contabilizada**. Certifique-se que a submissão foi concluída, e que esta última versão tem todas as alterações que deseja entregar, dado que os docentes apenas considerarão esta última submissão.
- Avisamos que a hora *deadline* acontece sempre poucos **minutos antes da meia-noite**, pelo que se urge a que os alunos não esperem por essa hora final para entregar e o façam antes, idealmente um dia antes, ou no pior dos casos, pelo menos uma hora antes. **Não serão consideradas válidas as entregas realizadas por e-mail**. Poderão testar a entrega nos dias anteriores para perceber se há algum problema com a entrega, sendo que, **apenas a última submissão conta**.

Política em caso de fraude

O trabalho corresponde ao esforço individual de cada aluno. São consideradas fraudes as seguintes situações: Trabalho parcialmente copiado, facilitar a cópia através da partilha de ficheiros, ou utilizar material alheio sem referir a fonte.

Em caso de deteção de fraude, os trabalhos em questão não serão avaliados, sendo enviados à Comissão Pedagógica da escola (ISTA) ou ao Conselho Pedagógico do ISCTE, consoante a gravidade da situação, que decidirão a sanção a aplicar aos alunos envolvidos. Serão utilizadas as ferramentas *Moss* e *SafeAssign* para deteção automática de cópias.

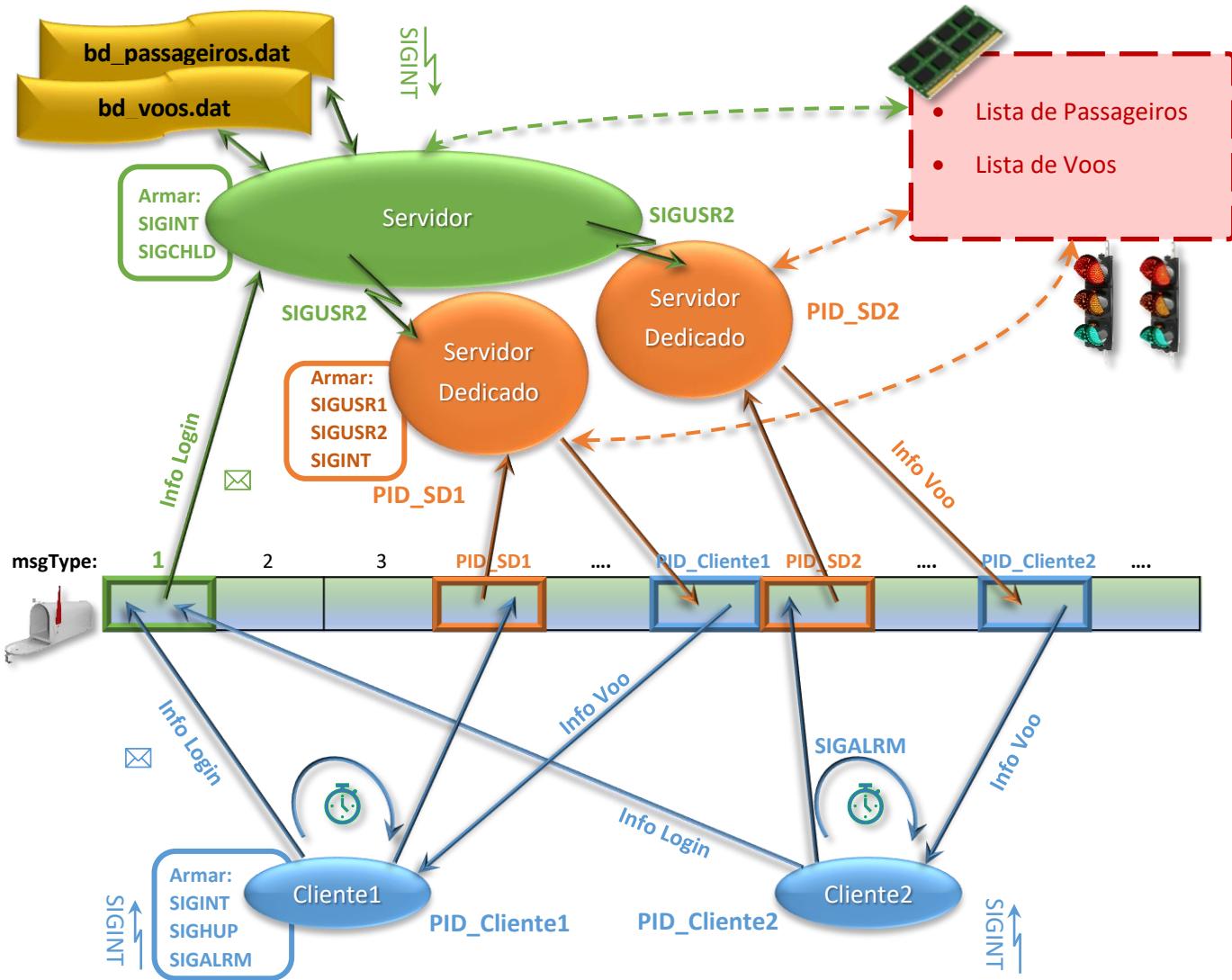
Recorda-se ainda que o Anexo I do Código de Conduta Académica, publicado a 25 de janeiro de 2016 em Diário da República, 2ª Série, nº 16, indica no seu ponto 2 que quando um trabalho ou outro elemento de avaliação apresentar um nível de coincidência elevado com outros trabalhos (percentagem de coincidência com outras fontes reportada no relatório que o referido software produz), cabe ao docente da UC, orientador ou a qualquer elemento do júri, após a análise qualitativa desse relatório, e em caso de se confirmar a suspeita de plágio, desencadear o respetivo procedimento disciplinar, de acordo com o Regulamento Disciplinar de Discentes do ISCTE - Instituto Universitário de Lisboa, aprovado pela deliberação nº 2246/2010, de 6 de dezembro.

O ponto 2.1 desse mesmo anexo indica ainda que no âmbito do Regulamento Disciplinar de Discentes do ISCTE-IUL, são definidas as sanções disciplinares aplicáveis e os seus efeitos, podendo estas variar entre a advertência e a interdição da frequência de atividades escolares no ISCTE-IUL até cinco anos.

Parte 3 – Processos, Sinais e IPC

Data de entrega: 25 de maio de 2024

Nesta parte do trabalho, será implementado um modelo simplificado de uma viagem de avião através da plataforma **IscteFlight**, baseado em comunicação entre processos, utilizando a linguagem de programação C. Considere o seguinte diagrama, que apresenta uma visão geral da arquitetura pretendida:



Pretende-se, nesta fase, simular uma viagem de avião na companhia **IscteFlight**. Assim, teremos dois módulos – **Cliente** e **Servidor**. Os conhecimentos que se pretende que os alunos sejam avaliados com este trabalho são:

- Utilização de memórias partilhadas IPC (**shmget**, **shmat**, **shmdt**);
- Concorrência entre processos usando semáforos IPC (**semget**, **semctl**, **semop**);
- Comunicação usando filas de mensagem IPC (**msgget**, **msgsnd**, **msgrcv**).

NOTA: Nenhum dos CICLOS referidos neste enunciado deverá ser realizado usando recursividade, apenas iterativos.

Os alunos têm acesso aos *templates* dos módulos **servidor.c** e **cliente.c** para realizarem o trabalho. Em ambos os módulos, os alunos **não deverão alterar a função main()**, apenas compreender o que faz. Deverão, sim, completar **as funções seguintes à função main()**, nos locais onde está claramente assinalado “// Substituir este comentário pelo código da função a ser implementado pelo aluno”, por forma a cumprir as especificações seguintes.

Algumas definições básicas e tipos de dados para interoperabilidade entre **Cliente** e **Servidor** estão no ficheiro **common.h**:

```
typedef struct {
    int nif;                                // Número de contribuinte do passageiro
    char senha[20];                           // Senha do passageiro
    char nome[56];                            // Nome do passageiro
    char nrVoo[8];                            // Número do voo escolhido
    int pidCliente;                          // PID do processo Cliente
    int pidServidorDedicado;                // PID do processo Servidor Dedicado
    int lugarEscolhido;                      // Indicação de qual foi o lugar escolhido
} CheckIn;

typedef struct {
    char nrVoo[8];                            // Número do voo escolhido
    char origem[24];                           // Origem do voo
    char destino[24];                          // Destino do voo
    int lugares[MAX_SEATS];                  // Lista de Lugares disponíveis e ocupados do voo
} Voo;

typedef struct {
    long msgType;                            // Tipo da Mensagem
    struct {
        CheckIn infoCheckIn;                // Informação sobre o CheckIn
        Voo infoVoo;                        // Informação sobre um Voo
    } msgData;                               // Dados da Mensagem
} MsgContent;

typedef struct {                                // Base de dados de Negócio, em Memória Partilhada
    CheckIn listClients[MAX_PASSENGERS]; // Lista de passageiros
    Voo listFlights[MAX_FLIGHTS];       // Lista de voos dos passageiros
} DadosServidor;
```

Para além disso, teremos definições adicionais, no ficheiro **defines.h**, nomeadamente a definição de **IPC_KEY**, que deverá ser alterada para ter o valor do nº do aluno, seguindo a regra utilizada nas últimas aulas práticas de SO.

ATENÇÃO: Lembra-se que os processos definidos neste enunciado, nomeadamente o **Servidor** e os **Servidores Dedicados**, atuam de forma concorrente, acedendo a recursos partilhados. Os alunos deverão cuidar para que sejam **definidos e utilizados os mecanismos de exclusão mútua no acesso aos mesmos**, sendo que essa exclusão deverá ser feita pelo menor tempo possível (dentro dos limites razoáveis, claro), e sem nunca permitir que haja esperas ativas.

1. Módulo: servidor.c

O módulo **Servidor**, responsável pelo processamento da autenticação e da gestão de voos, está dividido em duas partes, **um Servidor** (pai) e **zero ou mais Servidores Dedicados** (filhos). Este módulo realiza as seguintes tarefas:

S1 Abre/Cria a *Shared Memory* (SHM) IPC do projeto, que tem a KEY **IPC_KEY** definida em **defines.h** (terminando o **Servidor** em caso de erro). Para tal, realiza as seguintes operações:

S1.1 Valida se os ficheiros **bd_passageiros.dat** e **bd_voos.dat** (ver [Anexo B: Manipulação de Ficheiros Binários](#)) existem na diretoria local, e se temos permissão de leitura e escrita nos ficheiros. Se tal não acontecer em algum deles, dá **so_error** (i.e., **so_error("S1.1", "")**), e retorna erro (-1). Caso contrário, dá **so_success** (i.e., **so_success("S1.1", "")**).

S1.2 Abre a SHM com KEY **IPC_KEY**. Se não tiver sucesso, dá **so_error**. Caso contrário, dá **so_success**, e:

S1.2.1 Coloca **database** a apontar para essa SHM. Se conseguir, dá **so_success "<shmId>"**, e retorna a **shmId**. Senão, se abriu a SHM mas não anexou **database**, dá **so_error**, e retorna erro (-1).

S1.3 Valida se o erro na abertura da SHM foi devido a não existir ainda nenhuma SHM criada com essa KEY (testando o valor da variável **errno**). Se o problema não foi esse (mas outro qualquer), então dá **so_error**, e retorna erro (-1). Caso contrário (o problema foi não haver SHM criada), dá **so_success**.

S1.4 Cria uma SHM (em modo exclusivo) com a KEY **IPC_KEY** definida em **defines.h** e com o tamanho para conter as duas listas do **Servidor**: uma de passageiros (**CheckIn**), que comporta um máximo de **MAX_PASSENGERS** elementos, e outra de voos (**Voos**), que comporta um máximo de **MAX_FLIGHTS** elementos. Em caso de sucesso na criação da SHM, coloca a variável **database** a apontar para essa SHM. Se não houver erro em nenhuma das duas operações, dá **so_success "<shmId>"**. Senão, dá **so_error**, e retorna erro (-1).

S1.5 Inicia a Lista de Passageiros, preenchendo em todos os elementos o campo **nif=PASSENGER_NOT_FOUND** ("Limpa" a lista de passageiros), e a Lista de Voos, preenchendo em todos os elementos o campo **nrVoo=FLIGHT_NOT_FOUND** ("Limpa" a lista de voos). No final, dá **so_success**.

S1.6 Lê o ficheiro **bd_passageiros.dat** e preenche a lista de passageiros na SHM com a informação dos passageiros, preenchendo os campos **pidCliente** e **pidServidorDedicado** com o valor **PID_INVALID**. Em caso de qualquer erro, dá **so_error**, e retorna erro (-1). Caso contrário, dá **so_success**.

S1.7 Lê o ficheiro **bd_voos.dat** e preenche a lista de voos na memória partilhada com a informação dos voos. Em caso de qualquer erro, dá **so_error**, e retorna erro (-1). Caso contrário, dá **so_success** e retorna a **shmId**.

S2 Cria a *Message Queue* (MSG) do projeto, que tem a KEY **IPC_KEY**, realizando as seguintes operações:

S2.1 Se a MSG já existir, deve apagá-la. Em caso de erro na remoção, dá **so_error**, e retorna erro (-1). Caso contrário, dá **so_success**.

S2.2 Cria a *Message Queue* com a KEY **IPC_KEY**. Em caso de qualquer erro, dá **so_error**, e retorna erro (-1). Caso contrário, dá **so_success "<msgId>"** e retorna a **msgId**.

S3 Cria um grupo de semáforos (SEM) que tem a KEY **IPC_KEY**, realizando as seguintes operações:

S3.1 Se o SEM já existir, deve apagá-lo. Em caso de erro na remoção, dá **so_error**, e retorna erro (-1). Caso contrário, dá **so_success**.

S3.2 Cria um grupo de 3 (três) semáforos com a KEY **IPC_KEY**. Em caso de qualquer erro, dá **so_error**, e retorna erro (-1). Caso contrário, dá **so_success "<semId>"**.

S3.3 Inicia os dois primeiros semáforos (**SEM_PASSAGEIROS** e **SEM_VOOS**) com o valor **1** para que possam trabalhar em modo “exclusão mútua”, e o terceiro **SEM_NR_SRVDEDICADOS** com o valor **0**. Em caso de erro, dá **so_error**, e retorna erro (-1). Caso contrário, dá **so_success** e retorna o **semId**.

- S4** Arma e trata os sinais **SIGINT** (ver **S8**) e **SIGCHLD** (ver **S9**). Em caso de erro a armar os sinais, dá **so_error**, e retorna erro (-1). Caso contrário, dá **so_success**, e retorna sucesso (0).
- S5** **CICLO1:** Lê da MSG uma mensagem do tipo **MSGTYPE_LOGIN** para **clientRequest**. Se receber um sinal (e.g., **SIGCHLD**), retorna o valor **CICLO1_CONTINUE** (2) para que **main()** recomece automaticamente o **CICLO1** no passo **S5** (ler novo pedido). Caso contrário, se houve outro erro qualquer na leitura, dá **so_error**, e retorna erro (-1). Caso contrário, se não houver erros, dá **so_success** "<nif> <senha> <pidCliente>", e retorna sucesso (0).
- S6** Cria um processo filho (**fork**) **Servidor Dedicado**. Se houver erro, dá **so_error**, e retorna erro (-1). Caso contrário, o **Servidor Dedicado** (filho, com o PID **pidServidorDedicado**) dá **so_success** "Servidor Dedicado: Nasci", e retorna 0 para que **main()** siga automaticamente para o passo **SD10**, enquanto o **Servidor** (pai) dá **so_success** "Servidor: Inicie SD <pidServidorDedicado>", incrementa a variável **nrServidoresDedicados**, e retorna um valor > 0, para que **main()** recomece automaticamente o **CICLO1** no passo **S5** (ler novo pedido).
- S7** **Encerramento do Servidor:** Dá **so_success** "Servidor: Start Shutdown", e realiza as seguintes ações:
- S7.1** Verifica se existe SHM aberta e apontada pela variável **database**. Se alguma destas condições não acontecer, dá **so_error** e segue para o passo **S7.5**. Caso contrário, dá **so_success**.
- S7.2** **CICLO2:** Percorre a lista de passageiros de **database**. Por cada passageiro que tenha um **Servidor Dedicado** associado (significando que está num processo de início de voo, ou voo a decorrer), ou seja, tiver o campo **pidServidorDedicado** > 0, então envia ao PID (Process ID) desse processo **Servidor Dedicado** o sinal **SIGUSR2** para que termine, dá **so_success** "Servidor: Shutdown SD <pidServidorDedicado>", e recomeça o CICLO2 no passo **S7.2**, até ter terminado a lista de passageiros.
- S7.3** Aguarda que TODOS os **Servidores Dedicados** terminem (usando um algoritmo do tipo “barreira” como ensinado na aula SO-T08), para garantir que não haverá mais alterações às listas de passageiros nem de voos. Depois de todos eles terminarem, dá **so_success**.
- S7.4** Reescreve os ficheiros **bd_passageiros.dat** e **bd_voos.dat** com a informação das listas de passageiros e de voos da memória partilhada. Em caso de qualquer erro, dá **so_error**. Caso contrário, dá **so_success**.
- S7.5** Apaga todos os elementos IPC (SHM, SEM e MSG) que existam com a KEY **IPC_KEY**. Ignora quaisquer erros nas operações de remoção destes elementos, e, no final, dá **so_success** "Servidor: End Shutdown", e termina o processo **Servidor**.
- S8** O sinal armado **SIGINT** serve para o dono do aeroporto iniciar o processo de encerramento do **Servidor**, usando o atalho com as teclas <**CTRL+C**>. Se receber esse sinal (do utilizador via Shell), dá **so_success**, e segue para o passo **S7** (encerramento do **Servidor**).
- S9** O sinal armado **SIGCHLD** serve para que o **Servidor** seja alertado quando um dos seus filhos **Servidor Dedicado** terminar. Se o **Servidor** receber esse sinal, identifica o PID do **Servidor Dedicado** que terminou (**wait**), e dá **so_success** "Servidor: Confirmo fim de SD <pidServidorDedicado>", e retorna.
- SD10** O novo processo **Servidor Dedicado** (filho) arma os sinais **SIGUSR1** (ver **SD19**), **SIGUSR2** (ver **SD20**) e **SIGINT** (programa-o para ignorar este sinal). Em caso de erro, dá **so_error**, e retorna erro (-1). Caso contrário, dá **so_success**, e retorna sucesso (0).
- SD11** Inicia a variável **indexClient** a -1 (índice da lista de passageiros de **database**), e, de seguida, faz as seguintes operações:
- SD11.1** **CICLO3:** Percorre a lista de passageiros de **database**: Se chegou ao final da lista sem encontrar o passageiro com o **NIF** do pedido **clientRequest**, dá **so_error** "Cliente <NIF>: não encontrado", e retorna erro (-1).
- SD11.2** Para cada passageiro da lista, valida se o seu campo **nif** é igual ao campo **nif** passado no pedido **clientRequest**. Se **NÃO forem iguais**, recomeça o CICLO3 no passo **SD11.1**.

SD11.3 Compara o campo **senha** do passageiro com o do pedido **clientRequest**. Se forem iguais, dá **so_success <indexClient>**, e retorna **indexClient**. Se NÃO forem iguais, dá **so_error "Cliente <NIF>: Senha errada"**, e retorna erro (-1).

SD12 Inicia a variável **indexFlight** a **-1** (índice da lista de voos de **database**). Se o passo **SD11** concluiu com sucesso (validado em **main()**), faz as seguintes operações:

SD12.1 CICLO4: Percorre a lista de voos de **database**: Se chegou ao final dessa lista sem encontrar o **nrVoo** do passageiro **indexClient**, dá **so_error "Voo <nrVoo>: não encontrado"**, e retorna erro (-1).

SD12.2 Para cada voo da lista, valida se o seu campo **nrVoo** é igual ao campo **nrVoo** do passageiro **indexClient**. Se forem iguais, dá **so_success <indexFlight>**, e retorna **indexFlight**. Se NÃO forem iguais, recomeça o CICLO4 no passo SD12.1.

SD13 Se os passos **SD11** e **SD12** tiveram sucesso (validado em **main()**), atualiza a informação de **database**. **Dica:** atenção que, enquanto as atividades deste passo estão a ser executadas, outro processo **Cliente** com um **pidCliente** diferente deste, associado a outro **Servidor Dedicado**, também com um **pidServidorDedicado** diferente deste, pode também estar a tentar fazer check-in deste mesmo NIF de passageiro no sistema...

SD13.1 Dá **so_success "Start Check-in: <NIF> <pidCliente>"**.

SD13.2 Valida se, no elemento **indexClient** da lista de passageiros, **pidCliente** é **PID_INVALID**, e **lugarEscolhido** é **EMPTY_SEAT**. Se não forem, dá **so_error "Cliente <NIF>: Já fez check-in"**, retorna erro (-1).

SD13.3 Aguarda (**sleep**) 4 segundos (“tempo de processo burocrático 😊”). Cuidado que durante este período, outro **Servidor Dedicado** poderá estar também a tentar fazer o mesmo processo para este NIF!

SD13.4 Preenche o campo **pidCliente** com a informação registada no pedido **clientRequest**, e o campo **pidServidorDedicado** com o PID do processo **Servidor Dedicado**.

SD13.5 Dá **so_success "End Check-in: <NIF> <pidCliente>"**, e retorna sucesso (0).

SD14 CICLO5: Resposta **Cliente**: Cria uma mensagem com tipo de mensagem o PID do **Cliente** que enviou **clientRequest**:

SD14.1 Se houve qualquer erro nas validações dos passos **SD11**, **SD12**, ou **SD13**, então dá **so_error**, e preenche na mensagem o campo **pidServidorDedicado=PID_INVALID**. Caso contrário, dá **so_success**, e:

- Preenche na mensagem o campo **pidServidorDedicado** com o PID do processo **Servidor Dedicado**.
- Preenche na mensagem o campo **lugarEscolhido=EMPTY_SEAT**, e toda a estrutura **infoVoo** com o elemento **indexFlight** da lista de voos. **Informação:** O campo **lugares** é um array de inteiros, com cada índice a representar um lugar no avião (se a entrada nesse array tiver o valor **EMPTY_SEAT**, então o lugar está disponível para ser selecionado; caso contrário, o lugar está ocupado).

SD14.2 Envia a mensagem para a MSG. Se houver erro no envio, dá **so_error**, e retorna erro (-1). Caso contrário, dá **so_success**, e retorna sucesso (0).

SD15 Se os pontos anteriores tiveram sucesso (validado em **main()**), lê da MSG uma mensagem com tipo igual ao PID do **Servidor Dedicado** para a variável **clientRequest**. Se houver erro na leitura, dá **so_error**, e retorna erro (-1). Caso contrário, dá **so_success "<NIF> <lugarEscolhido> <pidCliente>"**, e retorna sucesso (0).

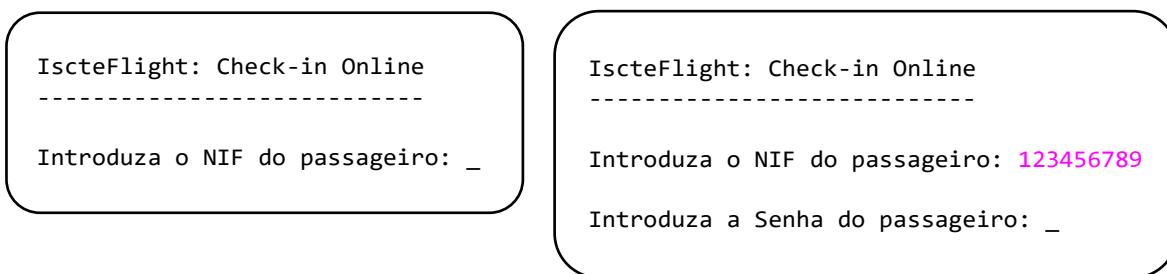
SD16 Preenche a informação do lugar na base de dados apontada por **database**. **Dica:** atenção que, enquanto as atividades deste passo estão a ser executadas, outro **Cliente** (invocado por outro passageiro com um **NIF diferente** deste, e que também vai viajar neste voo), pode também ter escolhido o mesmo lugar deste voo...

- SD16.1** Dá **so_success** "Start Reserva lugar: <NrVoo> <NIF> <lugarEscolhido>".
- SD16.2** Valida se, na **database**, no elemento da lista de voos com índice **indexFlight**, o **lugarEscolhido** no pedido **clientRequest** está disponível (**EMPTY_SEAT**) no campo **lugares**. Se não estiver, dá **so_error** "Cliente <NIF>: Lugar já estava ocupado", e retorna erro (-1) para que **main()** recomece o **CICLO5** em **SD14**.
- SD16.3** Aguarda (**sleep**) 4 segundos ("tempo de processo burocrático 😊"). Cuidado que durante este período, outro **Cliente** com um NIF diferente pode também ter escolhido ocupar o mesmo lugar deste voo...
- SD16.4** Preenche, na lista de voos de **database**, no seu elemento **indexFlight**, o campo **lugares[lugarEscolhido]** deste array com o valor do **NIF** do passageiro, registado no pedido **clientRequest**.
- SD16.5** Preenche, na lista de passageiros de **database**, no seu elemento **indexClient**, o campo **lugarEscolhido** com o valor **lugarEscolhido**, registado no pedido **clientRequest**. Nota: esta alteração em particular já não tem problemas de concorrência, dado que quaisquer outros processos **Cliente** que pretendessem alterar este campo ficaram barrados no ponto **SD13.2**.
- SD16.6** Dá **so_success** "End Reserva lugar: <NrVoo> <NIF> <lugarEscolhido>", retorna sucesso (0).
- SD17** **Confirmação de lugar:** Cria uma mensagem com tipo de mensagem o PID do **Cliente** que enviou **clientRequest**: preenche na mensagem o campo **pidServidorDedicado** com o PID do processo **Servidor Dedicado**. Preenche na mensagem o campo **lugarEscolhido** com o valor escolhido pelo utilizador, para confirmar que o mesmo foi aceite. Preenche também os campos **origem** e **destino**. Envia a mensagem para a MSG. Se houver erro no envio, dá **so_error**, e retorna erro (-1). Caso contrário, dá **so_success**, e retorna sucesso (0). Em ambos os casos, **main()** avança para o passo **SD18** para terminar o **Servidor Dedicado**.
- SD18** **Encerramento do Servidor Dedicado:** Atualiza a informação do passageiro na base de dados apontada por **database**, no elemento da lista de passageiros correspondente ao elemento **indexClient** (se este for >= 0): atualiza os campos **pidCliente** e **pidServidorDedicado** com o valor **PID_INVALID**. Dá **so_success** e termina o **Servidor Dedicado**.
- SD19** O sinal armado **SIGUSR1** serve para que o **Servidor Dedicado** seja alertado que o **Cliente** quer terminar (desistir). Se o **Servidor Dedicado** receber esse sinal, dá **so_success** "SD: Recebi pedido do Cliente para terminar", e avança para o passo **SD18** para terminar o **Servidor Dedicado**.
- SD20** O sinal armado **SIGUSR2** serve para que o **Servidor Dedicado** seja alertado que o **Servidor** principal quer terminar. Se o **Servidor Dedicado** receber esse sinal, dá **so_success** "SD: Recebi pedido do Servidor para terminar". Se na mensagem **clientRequest** o campo **pidCliente** <> **PID_INVALID**, então envia um sinal **SIGHUP** ao processo **Cliente** com esse PID para avisá-lo que vai terminar, e avança para o passo **SD18** para terminar o **Servidor Dedicado**.

2. Módulo: cliente.c

O módulo **Cliente** é responsável pela interação com o utilizador. Após o login do utilizador, este poderá realizar atividades durante o tempo da sessão. Assim, definem-se as seguintes tarefas a desenvolver:

- C1 Abre a *Message Queue* (MSG) do projeto, que tem a KEY **IPC_KEY**. Deve assumir que a fila de mensagens já foi criada pelo **Servidor**. Em caso de erro, dá **so_error**, e retorna erro (-1). Senão, dá **so_success** "<msgId>", e retorna sucesso a msgId.
- C2 Arma e trata os sinais **SIGHUP** (ver C10), **SIGINT** (ver C10), e **SIGALRM** (ver C11). Em caso de qualquer erro a armar os sinais, dá **so_error**, e retorna erro (-1). Caso contrário, dá **so_success**, e retorna sucesso (0).
- C3 Pede ao utilizador que preencha os dados referentes à sua autenticação (**NIF** e **Senha**), dados que serão preenchidos diretamente na mensagem **clientRequest**. Em caso de erro (i.e., se introduzir um NIF que não seja um número com um máximo de 9 dígitos), dá **so_error**, e retorna erro (-1). Caso contrário, dá **so_success**, e retorna sucesso (0).



- C4 A mensagem **clientRequest** será do tipo **MSGTYPE_LOGIN** com as informações do utilizador, preenchendo também o campo **pidCliente** com o PID do processo **Cliente**, e **pidServidorDedicado=PID_INVALID**. Os outros campos de **clientRequest** não precisam ser preenchidos. Envia a mensagem para a MSG. Em caso de erro, dá **so_error**, e retorna erro (-1). Senão, dá **so_success** "<nif> <senha> <pidCliente>", e retorna sucesso (0).
- C5 **CICLO6:** Configura um alarme com o valor de **MAX_ESPERA** segundos (ver C11), e dá **so_success** "Espera resposta em <MAX_ESPERA> segundos".
- C6 Lê da MSG uma mensagem com tipo igual ao PID do **Cliente** para a variável **clientRequest**. Se houver erro na leitura, dá **so_error**, e retorna erro (-1). Caso contrário, se não houver erro na leitura, dá **so_success** "<NIF> <lugarEscolhido> <pidCliente>", e retorna sucesso (0).
- C7 Trata mensagem do **Servidor Dedicado**:
 - C7.1 “Desliga” o alarme configurado em C5.
 - C7.2 Se, na variável **clientRequest**, o campo **pidServidorDedicado** for **PID_INVALID**, significa que houve algum problema na operação de login (e.g., NIF inexistente, Senha errada, NIF já registado, Voo inexistente, entre outros). Dá **so_error**, e termina o **Cliente**.
 - C7.3 Se, na variável **clientRequest**, o campo **lugarEscolhido** tiver um valor diferente de **EMPTY_SEAT**, significa que o utilizador já conseguiu reservar um lugar. Dá **so_success** "Reserva concluída: <origem> <destino> <lugarEscolhido>", e termina o **Cliente**.

- C7.4 O campo **lugarEscolhido** tem o valor **EMPTY_SEAT**, significa que o **Servidor Dedicado** está a pedir para que o utilizador escolha um novo lugar:
 - C7.4.1 Se é a primeira vez que está a realizar esta operação, dá **so_success**. Caso contrário (significa que já tentou anteriormente), dá **so_error** "Erro na reserva de lugar".

- C7.4.2** Mostra ao utilizador a lista de lugares disponíveis: Percorre o array do campo **lugares**, e lista todas as posições do array cujo valor seja **EMPTY_SEAT**. Para simplificar, considere que os lugares têm valores entre **0** e **MAX_SEATS-1**.
- C7.4.3** Pede ao utilizador que escolha o lugar pretendido. Em caso de erro (i.e., se introduzir um lugar que não seja um número entre **0** e **MAX_SEATS-1**), dá **so_error**, e retorna erro (-1). Caso contrário, dá **so_success "⟨lugarEscolhido⟩"**, e retorna o valor de **lugarEscolhido**.

```
IscteFlight: Voo FR5102
-----
Lugares disponíveis: 0, 1, 3, 4, 5, 10
Introduza o lugar que deseja reservar: _
```

- C8** Cria uma mensagem **clientRequest** com o tipo igual ao campo **pidServidorDedicado** (PID do **Servidor Dedicado**), preenchendo a informação de **nif** e do **lugarEscolhido**, preenchendo também o campo **pidCliente** com o PID do próprio processo **Cliente**. Os outros campos de **clientRequest** não precisam ser preenchidos. Envia a mensagem para a MSG. Se houver erro no envio, dá **so_error**, e retorna erro (-1), para que main() siga para terminar o Cliente em **C9**. Senão, dá **so_success "⟨nif⟩ <lugarEscolhido⟩ <pidCliente⟩"**, e retorna sucesso (0) para que main() recomece o CICLO6 em **C6**.
- C9** **Encerramento do Cliente:** Valida se a mensagem **clientRequest** tem um campo **pidServidorDedicado** cujo valor é **PID_INVALID**. Se o valor for esse, dá **so_error**. Caso contrário, dá **so_success** e envia um sinal **SIGUSR1** para o processo **pidServidorDedicado** (PID do **Servidor Dedicado**). Em ambos os casos, termina o **Cliente**.
- C10** O sinal armado **SIGHUP** serve para o **Servidor Dedicado** indicar que vai terminar. Se o **Cliente** receber esse sinal, dá **so_success "Check-in concluído sem sucesso"**, e termina o **Cliente**.
- C11** O sinal armado **SIGINT** serve para que o utilizador possa cancelar o pedido do lado do **Cliente**, usando o atalho **<CTRL+C>**. Se receber esse sinal (do utilizador via Shell), o **Cliente** dá **so_success "Cliente: Shutdown"**, e segue para terminar o Cliente em **C9**.
- C12** O sinal armado **SIGNALRM** serve para que, se o **Cliente** esperou mais do que **MAX_ESPERA** segundos sem resposta, o **Cliente** dá **so_error "Cliente: Timeout"**, e segue para terminar o Cliente em **C9**.

Anexo A: Macros de suporte ao trabalho

Macros fornecidas, no ficheiro `/home/so/utils/include/so_utils.h`, com Mensagens de sucesso, erro e debug:

Mensagens de output com Erro (com exemplos): Macro `so_error(<Passo>, <Mensagem>, [...])`

A sintaxe dos argumentos `<Mensagem>, [...]` é semelhante à de `printf("<string formatação>, [...])`.

Esta macro, tem como output no STDOUT:

```
"@ERROR {<Passo>} [<Mensagem>, [...]]"
```

Exemplos de invocação:

- Em **S1**, O ficheiro **bd_passageiros.dat** não existe:
 - `so_error("S1", "");` // Ainda que não haja mensagem de erro, tem de ter pelo menos 2 strings
- Em **C11**, O **Cliente** deu Timeout:
 - `so_error("C11", "Cliente: Timeout");`

Mensagens de output com Sucesso (com exemplos): Macro `so_success(<Passo>, <Mensagem>, [...])`

A sintaxe dos argumentos `<Mensagem>, [...]` é semelhante à de `printf("<string formatação>, [...])`.

Esta macro, tem como output no STDOUT:

```
"@SUCCESS {<Passo>} [<Mensagem>, [...]]"
```

Exemplos de invocação:

- Em **S4**, o **Servidor** armou corretamente os sinais SIGINT e SIGCHLD:
 - `so_success("S4", "");` // Ainda que não haja mensagem de sucesso, tem de ter pelo menos 2 strings
- Em **C5**, o **Cliente** indica que iniciou o período de espera:
 - `so_success("C5", "Espera resposta em %d segundos", MAX_ESPERA);`
- Em **SD11.2**, o **Servidor Dedicado** indica que descobriu a entrada do cliente no ficheiro **bd_passageiros.dat**:
 - `so_success("SD11.2", "%d", indexClient);`
- Em **S5**, o **Servidor** leu um pedido do **Cliente** na forma de um elemento **Error! Reference source not found.**:
 - `so_success("S5", "%d %s %d", request.nif, request.senha, request.pidCliente);`

Mensagens de Debug: Apesar de não ser necessário, disponibilizou-se também uma macro para as mensagens de debug dos scripts, dado que será muito útil aos alunos: **Macro `so_debug(<Mensagem>, [...])`**

A sintaxe dos argumentos `<Mensagem>, [...]` é semelhante à de `printf("<string formatação>, [...])`.

Esta macro, tem como output no STDOUT:

```
"@DEBUG:<Source file>:<line>:<function>: [<Mensagem>, [...]]"
```

Exemplos de invocação:

- Em **SD11.1**, para ver os valores da entrada atual lida da BD:
 - `so_debug("Entrada atual BD: NIF: %d; Senha: %s", itemDB.nif, itemDB.senha);`
- Em **S7.2**, simplesmente para indicar um teste de passagem por uma parte do código:
 - `so_debug("Passei por aqui");`

Tem a vantagem de que mostra sempre as mensagens de debug (não precisa sequer ser nunca apagado). Quando quiser que essas mensagens de debug não apareçam, simplesmente tire o comentário da seguinte linha do programa:

```
// #define SO_HIDE_DEBUG      // Uncomment this line to hide all @DEBUG statements
```

E, assim, não precisam de apagar as invocações à macro `so_debug()`, mantendo os vossos programas intocados.

Anexo B: Manipulação de Ficheiros Binários

Neste trabalho são armazenadas informações num ficheiro em formato binário, **bd_passageiros.dat**. Não é fácil visualizar este ficheiro usando a aplicação **cat**. Uma das formas sugeridas de analisar estes ficheiros é usando as aplicações **hexdump** ou **xxd**. No entanto, para facilitar esta tarefa, foi fornecido um script que ajuda a visualizar os conteúdos deste ficheiro ([ver no Moodle a KB Basics / Visualizar o conteúdo de um ficheiro binário](#)), que estão de acordo com a estrutura **CheckIn**:

```
typedef struct {
    int nif;                                // NIF do passageiro
    char senha[20];                          // Senha do passageiro
    char nome[56];                           // Nome do passageiro
    char nrVoo[8];                            // ID do voo reservado pelo passageiro
    int pidCliente;                          // PID do processo Cliente
    int pidServidorDedicado;                // PID do processo Servidor Dedicado
    int lugarEscolhido;                      // Indicação de qual foi o Lugar escolhido
} CheckIn;
```

Assim, o comando:

```
$ ./so_show_binary_checkin_file.sh bd_passageiros.dat
```

Mostra o resultado:

This program displays on STDOUT a binary file with one or more elements of type "CheckIn".							
nif	senha	nome	nrVoo	pidCliente	pidServidorDedicado	lugarEscolhido	
225608358	34erdfcv	Fabio Cardoso	FR2101	-1	-1	-1	
275067109	12qwazx	David Gabriel	FR5102	-1	-1	-1	
213654377	09polkmm	Nuno Garrido	PR3201	-1	-1	-1	
212345678	123qwaeasd	Carlos Coutinho	PT6202	-1	-1	-1	
298765432	098poilkj	Jorge Rafael	FR2301	-1	-1	-1	
245678912	1qa2ws3ed	Filipa Prudencio	PT5000	-1	-1	-1	
231564897	0ok9ij8uh	Ricardo Fonseca	PT5030	-1	-1	-1	
258147369	321ewqdsa	Paulo Jose Pereira	FR2101	-1	-1	-1	
247158369	890uiojkl	Miguel Tavares	FR5102	-1	-1	-1	
269358147	213wqesad	David Fernandes	PR3201	-1	-1	-1	
289456123	980iuokjl	Paulo Jorge Pereira	XY9999	1	-1	10	

O mesmo é aplicável para a estrutura **Voo**:

```
typedef struct {
    char nrVoo[8];                            // Número do voo escolhido
    char origem[24];                          // Origem do voo
    char destino[24];                         // Destino do voo
    int lugares[MAX_SEATS];                  // Lista de Lugares disponíveis e ocupados do voo
} Voo;
```

Assim, o comando:

```
$ ./so_show_binary_voo_file.sh bd_voos.dat
```

Mostra o resultado:

This program displays on STDOUT a binary file with one or more elements of type "Voo".													
nrVoo	origem	destino	lugares[0]	lugares[1]	lugares[2]	lugares[3]	lugares[4]	lugares[5]	lugares[6]	lugares[7]	lugares[8]	lugares[9]	lugares[10]
FR2101	Lisboa	Dubai	-1	512345601	-1	512345603	-1	512345605	-1	512345607	-1	512345609	-1
FR5102	Dubai	Bangkok	512345610	-1	512345612	-1	512345614	-1	512345616	-1	512345618	512345619	-1
PR3201	Londres	Nova York	512345620	512345621	-1	512345623	512345624	-1	-1	512345627	512345628	-1	269358147
PT6202	Toquio	Singapura	-1	512345631	512345632	-1	-1	512345635	512345636	-1	-1	512345639	512345630
FR2301	Dubai	Sydney	-1	-1	512345642	512345643	512345644	-1	-1	512345647	-1	-1	512345640
PT5000	Sydney	Los Angeles	512345650	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
PT5030	Lisboa	Terceira	-1	512345661	512345662	512345663	512345664	512345665	512345666	512345667	512345668	512345669	512345660

Anexo C: Script Validador do Trabalho

Como anunciado nas aulas, está disponível para os alunos um script de validação dos trabalhos, para os alunos terem uma noção dos critérios de avaliação utilizados.

Passos para realizar a validação do seu trabalho:

- Garanta que o seu trabalho (i.e., os scripts *.c *.h) está localizado numa diretoria local da sua área. Para os efeitos de exemplo para esta demonstração, assumiremos que essa diretoria terá o nome **parte-3** (mas poderá ser outra qualquer).
- Posicione-se nessa diretoria **parte-3** da sua área:
\$ cd parte-3
- Dê o comando seguinte e valide que está mesmo na diretoria correta:
\$ pwd
- Dê o comando seguinte, e confirme que todos os ficheiros *.c *.h do seu trabalho estão mesmo nessa diretoria, que esses ficheiros têm permissão de execução, e que nessa diretoria existe uma subdiretoria chamada **so_2023_trab3_validator**:
\$ ls -l
- Vá para a subdiretoria onde está o validador:
\$ cd so_2023_trab3_validator
- E, finalmente, dentro dessa diretoria, execute o script de validação do seu trabalho:
\$./so_2023_trab3_validator.py ..
- Resta agora verificar quais dos seus testes “passam” (✓) e quais “chumbam” (✗).
- Faça as alterações para correção dos seus scripts.
- Sempre que quiser voltar a fazer nova validação, basta novamente posicionar-se na subdiretoria **so_2023_trab3_validator** e correr o script de validação como demonstrado acima.

A aplicação **so_2023_trab3_validator.py** tem algumas opções que podem ser úteis aos alunos:

- Se fizer **so_2023_trab3_validator.py -h** (ou **--help**), pode ver as várias opções.
- Se usar a opção **-d** (ou **--debug**), pode visualizar as mensagens que colocou no código usando **so_debug**, **so_success**, e **so_error**. Caso contrário essas mensagens serão omitidas.
- Se usar a opção **-e** (ou **--stoponerror**), o validador irá parar assim que encontrar o primeiro erro, o que pode ser prático para não ter um output muito extenso.
- Se usar a opção **-s** (ou **--server**), o validador apenas irá validar o módulo servidor (ou seja, não irá validar o módulo cliente).
- se usar a opção **-c** (ou **--client**), o validador apenas irá validar o módulo cliente (ou seja, não irá validar o módulo servidor).
- se usar as duas opções **-cs** (ou **--client --server**), nenhum dos dois módulos, cliente nem servidor, será validado.

Anexo D: Histórico de Versões

Versão 1: publicada em 2024-05-09

- Primeira versão (original) do trabalho.

Versão 2: publicada em 2024-05-12

- **S1.2, S2.1, S3.1, S5**: Melhoria muito ligeira na descrição destes passos
- **S3**: Em vez de 2, cria 3 semáforos, sendo o terceiro, **SEM_NR_SRVDEDICADOS**, destinado a fazer uma operação de semáforo do tipo “barreira” em **S7.3** (ver slides da aula SO-T08).

Versão 3: publicada em 2024-05-13

- **S9** não pode decrementar a variável **nrServidoresDedicados**, senão a barreira em **S7.3** deixa de funcionar. O modo de funcionamento deste processo é o seguinte:
 - Cada vez que um **Servidor Dedicado** nasce, a variável **nrServidoresDedicados** é incrementada.
 - Cada vez que esse **Servidor Dedicado** termina, acrescenta uma unidade ao semáforo, sem decrementar a variável **nrServidoresDedicados**.
 - Quando o **Servidor** termina, decrementa **nrServidoresDedicados** unidades ao semáforo, fazendo assim a “barreira” que apenas liberta o Servidor quando todos os **Servidores Dedicados** terminarem.
- **SD11, SD12, SD13, SD14.1 e SD16**: Melhoria na descrição destes passos, nomeadamente **SD16.4** não fazia update aos campos corretos (**listFlights[indexFlight].lugares[lugarEscolhido]** e **listClients[indexClient].lugarEscolhido**)
- **C7.4.1**: Foi criada, no template **cliente.c** uma variável global **nrTentativasEscolhaLugar**, para ajudar a validação no nº de tentativas (eventualmente falhadas) na escolha de um lugar no voo.