

Projeto *IscteFlight* (Parte 2)

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 2 do trabalho.

Este trabalho está publicado na sua [versão 4](#). 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 segunda parte é lidar com o procedimento de **check-in eletrónico de voos**.



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

```
#define MAX_ESPERA      5      // Tempo máximo de espera por parte do Cliente

typedef struct {
    int nif;                  // NIF do passageiro
    char senha[40];           // Senha do passageiro
    char nome[60];            // 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
} CheckIn;
```

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-2**:

- **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-2 .

Procedimento de entrega e submissão do trabalho

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

A entrega da Parte 2 do trabalho será realizada através da criação de um ficheiro ZIP cujo nome é o nº do aluno, e.g., “a<nºaluno>-parte-2.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 2, 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-2”, 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-2.zip <ficheiros>**, por exemplo:

```
$ zip $USER-part-2.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 2”.
- 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.

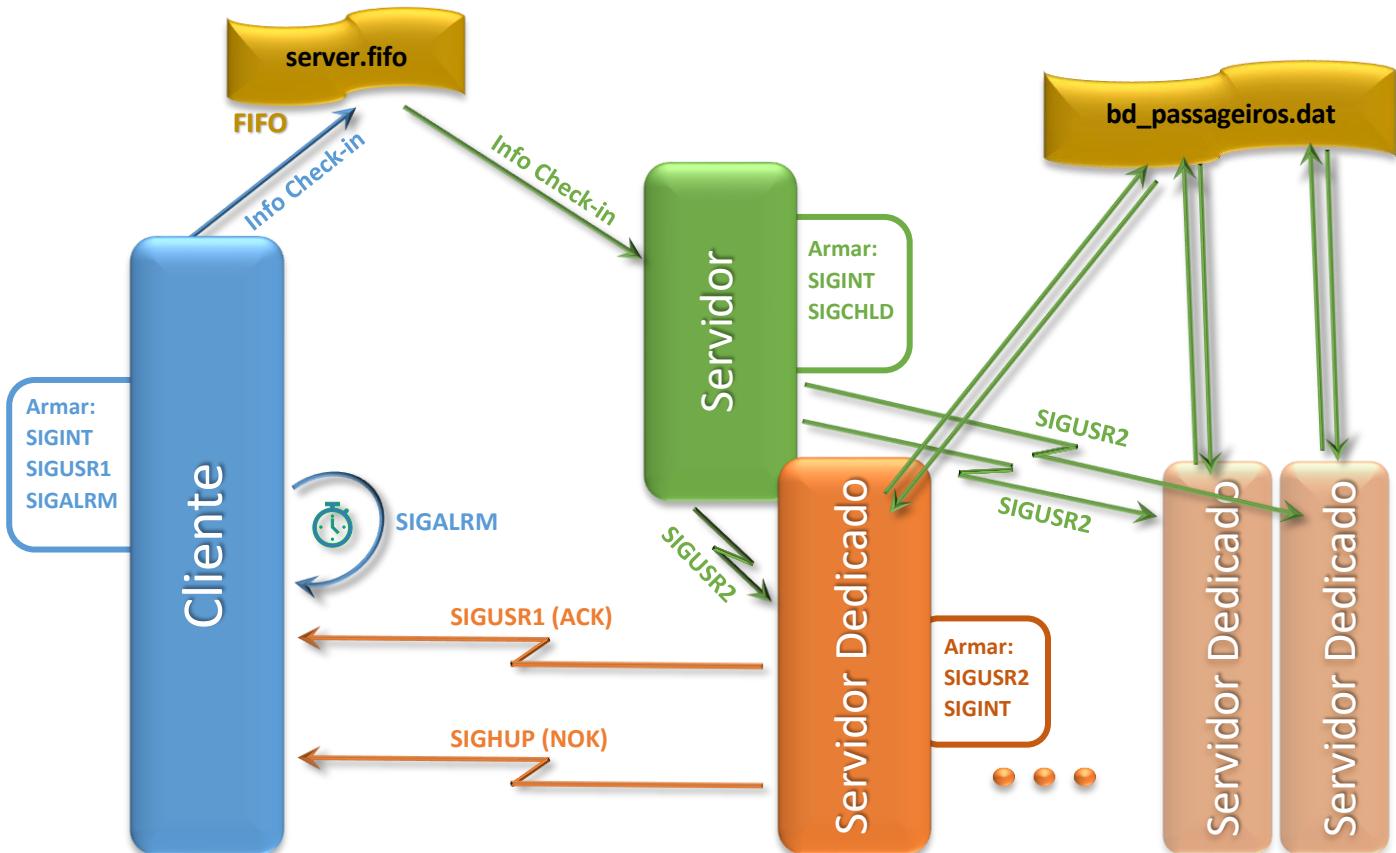
Parte 2 – Processos e Sinais

Data de entrega: 21 de abril de 2024

Nesta parte do trabalho, o objetivo é criar um sistema de modelo cliente-servidor, utilizando a linguagem de programação C, para interação entre o utilizador e a plataforma, neste caso, **para realizar o serviço de check-in online**.

Atenção: Apesar dos vários ficheiros necessários para a realização do trabalho serem fornecidos na diretoria do Tigre “`/home/so/trabalho-2023-2024/parte-2`”, assume-se que, para a sua execução, os scripts e todos os ficheiros de input e de output estarão todos **sempre** presentes na mesma diretoria, que não deve estar *hard-coded*, ou seja, os Shell scripts entregues devem correr em qualquer diretoria.

Considere o seguinte diagrama, que apresenta uma visão geral da arquitetura pretendida:



Pretende-se, nesta fase, simular um servidor de sessões de check-in de passageiros na aplicação **IsciteFlight**. Assim, teremos dois módulos – **Cliente** e **Servidor**. Os conhecimentos que se pretende que os alunos sejam avaliados com este trabalho são:

- Criação de processos e concorrência entre processos (**fork** e **wait**).
- Interacção entre processos usando sinais (**kill**, **signal** e **alarm**).
- Manuseamento de ficheiros de acesso sequencial (**so_fgets**, **fprintf**) e acesso direto (**fread**, **fwrite**, **fseek**).
- Interacção com o utilizador (**printf**, **so_gets**, **so_geti**).
- Comunicação síncrona usando Named Pipes ou FIFOs (**S_ISFIFO**, **stat**).

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.

1. Módulo: servidor.c

O módulo **Servidor** é responsável pelo processamento do check-in dos passageiros. Está dividido em duas partes, um **Servidor** (pai) e **zero ou mais Servidores Dedicados** (filhos). Este módulo realiza as seguintes tarefas:

- S1** Valida se o ficheiro **bd_passageiros.dat** (ver [Anexo B: Manipulação de Ficheiros Binários](#)) existe na diretoria local, e se temos permissão de leitura e escrita no ficheiro. Se tal não acontecer, dá **so_error** (i.e., `so_error("S1", "")`) e termina o Servidor. Caso contrário, dá **so_success** (i.e., `so_success("S1", "")`).
- S2** Cria o ficheiro (apagando-o se já existia) com organização FIFO (*named pipe*) do **Servidor**, de nome **server fifo**, na diretoria local. Se houver erro na operação, dá **so_error** e termina o Servidor. Caso contrário, dá **so_success**.
- S3** Arma e trata os sinais **SIGINT** (ver **S6**) e **SIGCHLD** (ver **S8**). Em caso de erro a armar os sinais, dá **so_error**, e segue para o passo S7 (encerramento do **Servidor**). Caso contrário, dá **so_success**.
- S4** **CICLO1:** Abre o FIFO do **Servidor**, de nome **server fifo**, para leitura. Se receber um sinal (e.g. **SIGCHLD**), recomeça o CICLO1 no passo S4. Senão, se houver erro, dá **so_error**, e segue para o passo S7 (encerramento do **Servidor**).
 - S4.1** Lê as informações **nif**, **senha** e **pidCliente** (escritas em modo de texto, ver passo **C5**) para a variável **clientRequest**, e fecha o FIFO. Se, entretanto, receber um sinal (e.g. **SIGCHLD**), recomeça o CICLO1 no passo S4. Caso contrário, se houver qualquer erro na leitura do FIFO, ou se o formato dos dados lidos não estiver correto (e.g., **nif <= 0** ou **pidCliente <= 0**), dá **so_error**, e segue para o passo S7 (encerramento do **Servidor**). Caso contrário (se não houver erros), dá **so_success "⟨nif⟩ <senha> <pidCliente⟩"**.
- S5** Cria um processo filho (**fork**) **Servidor Dedicado**. Se houver erro na criação do processo, dá **so_error**, e segue para o passo S7 (encerramento do **Servidor**). Senão, o **Servidor Dedicado** (que tem o PID **pidServidorDedicado**) segue para o passo SD9, enquanto o processo **Servidor** (pai) dá **so_success "Servidor: Inicieie SD <pidServidorDedicado>"**, e recomeça o CICLO1 (main()) no passo S4 (ou seja, volta a aguardar um novo pedido).
- S6** O sinal armado **SIGINT** serve para o dono da agência de viagens iniciar o processo de encerramento do **Servidor**, usando o atalho com as teclas **<CTRL+C>**. Se receber esse sinal (do utilizador via Shell), o **Servidor** dá **so_success "Servidor: Start Shutdown"**, e depois realiza as seguintes ações:
 - S6.1** Abre o ficheiro **bd_passageiros.dat** para leitura. Em caso de erro na abertura do ficheiro, dá **so_error**, e segue para o passo S7 (encerramento do **Servidor**). Caso contrário, dá **so_success**.
 - S6.2** **CICLO2:** Lê do ficheiro **bd_passageiros.dat** (escrito em modo acesso direto) um elemento do tipo **CheckIn**. Se houver erro na leitura do ficheiro, dá **so_error** e segue para o passo S7 (encerramento do **Servidor**). Senão, se chegou ao final do ficheiro **bd_passageiros.dat**, dá **so_success**, e segue para o passo S7 (encerramento).
 - S6.3** Se o elemento **CheckIn** lido do ficheiro **bd_passageiros.dat** tiver o campo **pidServidorDedicado > 0**, então envia ao PID (Process ID) desse processo **Servidor Dedicado** o sinal **SIGUSR2** para que termine, e dá **so_success "Servidor: Shutdown SD <pidServidorDedicado>"**.
 - S6.4** Recomeça o CICLO2 no passo S6.2.
- S7** **Encerramento do Servidor:** Remove o FIFO do **Servidor**, de nome **server fifo**, da diretoria local. Em caso de erro, dá **so_error**. Caso contrário, dá **so_success "Servidor: End Shutdown"**. Em ambos os casos, termina o processo Servidor.
- S8** 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.
- SD9** O novo processo **Servidor Dedicado** (filho) arma os sinais **SIGUSR2** (ver **SD14**) e **SIGINT** (programa-o para ignorar este sinal). Em caso de erro, dá **so_error** e termina o Servidor Dedicado. Caso contrário, dá **so_success**.

SD10 Abre o ficheiro **bd_passageiros.dat** para leitura. Em caso de erro na abertura do ficheiro, dá **so_error** e termina o **Servidor Dedicado**. Caso contrário, dá **so_success**, inicia uma variável **indexClient** a 0 (índice do elemento **CheckIn** corrente lido do ficheiro), e faz as seguintes operações:

SD10.1 CICLO3: Lê do ficheiro **bd_passageiros.dat** (escrito em modo acesso direto) um elemento **itemDB** do tipo **CheckIn**, e incrementa a variável **indexClient**. Em caso de erro na leitura do ficheiro, dá **so_error** e termina o **Servidor Dedicado**. Se chegou ao final do ficheiro **bd_passageiros.dat** sem encontrar o **Cliente** do pedido **clientRequest**, dá **so_error** "Cliente <NIF>: não encontrado", envia o sinal **SIGHUP** (indicando erro) ao processo **Cliente**, e termina o **Servidor Dedicado**.

SD10.2 Valida se o campo **NIF** passado no pedido **clientRequest** corresponde ao do elemento **CheckIn** do ficheiro. Se **NÃO forem iguais**, recomeça o CICLO3 no passo SD10.1, lendo mais um elemento do ficheiro.

SD10.3 Valida se o campo **Senha** passado no pedido **clientRequest** corresponde ao do elemento **CheckIn** do ficheiro. Se **forem iguais**, dá **so_success** "<indexClient>", fecha o ficheiro **bd_passageiros.dat**, e sai do CICLO3 para o passo SD11. Se **NÃO forem iguais**, dá **so_error** "Cliente <NIF>: Senha errada", envia o sinal **SIGHUP** (indicando erro) ao processo **Cliente**, e termina o **Servidor Dedicado**.

SD11 Modifica a base de dados correspondente ao elemento **itemDB** para **fazer Check-in**, com as seguintes operações:

SD11.1 Preenche os campos **Nome** e **NrVoo** do pedido **clientRequest** com os valores do elemento **itemDB** do ficheiro correspondente ao elemento **indexClient**. Preenche também o campo **pidServidorDedicado** com o PID do próprio processo **Servidor Dedicado**, completando assim os dados da estrutura **CheckIn** do pedido **clientRequest**. Dá **so_success** "<Nome> <NrVoo> <pidServidorDedicado>".

SD11.2 Abre o ficheiro **bd_passageiros.dat** para leitura e escrita ("r+"). Em caso de erro, dá **so_error**, envia o sinal **SIGHUP** ao processo **Cliente**, e termina o **Servidor Dedicado**. Caso contrário, dá **so_success**.

SD11.3 Posiciona o ficheiro (**fseek**) para o início do elemento correspondente a **indexClient**. Em caso de erro, dá **so_error**, envia **SIGHUP** ao **Cliente**, e termina o **Servidor Dedicado**. Caso contrário, dá **so_success**.

SD11.4 Escreve no ficheiro (em modo de acesso direto), na posição atual (ou seja, **indexClient**), o elemento **clientRequest**, atualizando assim o elemento correspondente a este **Cliente** no ficheiro de base de dados, e fecha o ficheiro **bd_passageiros.dat**. Em caso de erro, dá **so_error**, envia o sinal **SIGHUP** (indicando erro) ao processo **Cliente**, e termina o **Servidor Dedicado**. Caso contrário, dá **so_success**.

SD12 Define um tempo aleatório entre 1 e **MAX_ESPERA** ("tempo de processo burocrático 😊"). Dá **so_success** <tempo> e aguarda (**sleep**) esse tempo. Envia ao **Cliente** o sinal **SIGUSR1** indicando que o **check-in** foi concluído.

SD13 Modifica a base de dados correspondente ao elemento **itemDB**, para terminar a sessão do **Servidor Dedicado**:

SD13.1 Preenche os campos **clientRequest.pidCliente=-1** e **clientRequest.pidServidorDedicado=-1**. Abre o ficheiro **bd_passageiros.dat** na diretoria local para leitura e escrita ("r+"). Em caso de erro na abertura do ficheiro, dá **so_error**, e termina o **Servidor Dedicado**. Caso contrário, dá **so_success**.

SD13.2 Posiciona o apontador do ficheiro (**fseek**) para o início do elemento **itemDB** correspondente a **indexClient**. Em caso de erro, dá **so_error**, e termina o **Servidor Dedicado**. Caso contrário, dá **so_success**.

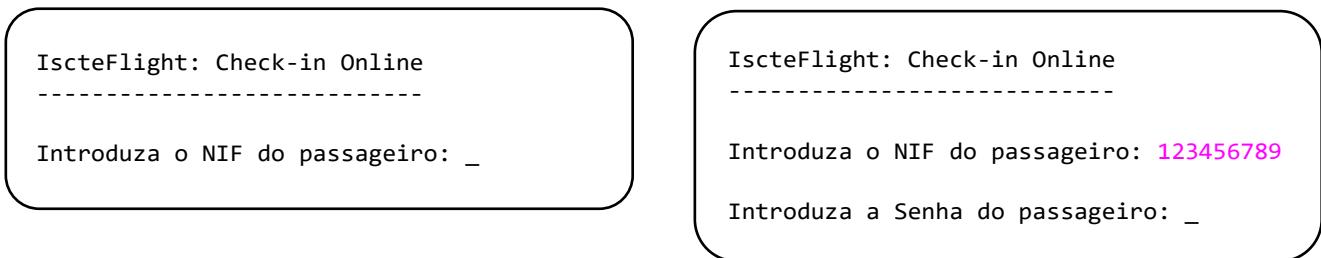
SD13.3 Escreve no ficheiro (em modo de acesso direto), na posição atual (ou seja, **indexClient**), o elemento **clientRequest**, "limpando" assim os dados da estrutura **CheckIn** correspondente a este **Cliente** no ficheiro de base de dados, e fecha o ficheiro **bd_passageiros.dat**. Em caso de erro, dá **so_error**, caso contrário, dá **so_success**. Em ambos casos, termina o **Servidor Dedicado**.

SD14 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", e termina 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 Valida se o ficheiro com organização FIFO (*named pipe*) do **Servidor**, de nome **server fifo**, existe na diretoria local. Se esse FIFO não existir, dá **so_error** e termina o **Cliente**. Caso contrário, dá **so_success**.
- C2 Arma e trata os sinais **SIGUSR1** (ver C8), **SIGHUP** (ver C9), **SIGINT** (ver C10), e **SIGNALRM** (ver C11). Em caso de qualquer erro a armar os sinais, dá **so_error** e termina o **Cliente**. Caso contrário, dá **so_success**.
- C3 Pede ao utilizador que preencha os dados referentes à sua autenticação (**NIF** e **Senha**). 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 termina o **Cliente**.



- C4 Cria um elemento **clientRequest** do tipo **CheckIn** com as informações introduzidas, preenchendo também o campo **pidCliente** com o PID do próprio processo **Cliente**. Os outros campos de **clientRequest** não precisam ser preenchidos. Dá **so_success <nif> <senha> <pidCliente>**.
- C5 Abre o FIFO do **Servidor** (**server fifo**). Escreve as informações do elemento **CheckIn NIF, Senha e PID Cliente** no FIFO do **Servidor**, escrevendo a informação na ordem indicada, em formato de texto (ASCII), com cada elemento a ocupar uma linha de texto, terminada por '**\n**':

server fifo

```
<NIF>
<Senha>
<PID Cliente>
```

Depois de escrever todos os dados indicados no FIFO, fecha o FIFO do **Servidor**. Em caso de erro na escrita, dá **so_error**, e termina o **Cliente**. Caso contrário, dá **so_success**.

- C6 Configura um alarme com o valor de **MAX_ESPERA** segundos (ver C11), e dá **so_success "Espera resposta em <MAX_ESPERA> segundos"**.
- C7 Fica à espera (**pause**) que haja um qualquer evento.
- C8 O sinal armado **SIGUSR1** serve para o **Servidor Dedicado** indicar que o check-in foi concluído com sucesso. Se o **Cliente** receber esse sinal, dá **so_success "Check-in concluído com sucesso"**, e termina o **Cliente**.
- C9 O sinal armado **SIGHUP** serve para o **Servidor Dedicado** indicar que o check-in foi concluído sem sucesso. Se o **Cliente** receber esse sinal, dá **so_success "Check-in concluído sem sucesso"**, e termina o **Cliente**.
- C10 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 termina o **Cliente**.
- C11 O sinal armado **SIGNALRM** serve para que, se o **Cliente** em esperou mais do que **MAX_ESPERA** segundos sem resposta, o **Cliente** dá **so_error "Cliente: Timeout"**, e termina o **Cliente**.

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 **S3**, o **Servidor** armou corretamente os sinais SIGINT e SIGCHLD:
 - `so_success("S3", "");` // Ainda que não haja mensagem de sucesso, tem de ter pelo menos 2 strings
- Em **C6**, o **Cliente** indica que iniciou o período de espera:
 - `so_success("C6", "Espera resposta em %d segundos", MAX_ESPERA);`
- Em **SD10.2**, o **Servidor Dedicado** indica que descobriu a entrada do cliente no ficheiro **bd_passageiros.dat**:
 - `so_success("SD10.2", "%d", indexClient);`
- Em **S4**, o **Servidor** leu um pedido do **Cliente** na forma de um elemento **CheckIn**:
 - `so_success("S4", "%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 **SD10.1**, para ver os valores da entrada atual lida da BD:
 - `so_debug("Entrada atual BD: NIF: %d; Senha: %s", itemDB.nif, itemDB.senha);`
- Em **S6.2**, 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[40];                          // Senha do passageiro
    char nome[60];                            // 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
} 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	
225608358	34erdfcv	Fabio Cardoso	FR2101	-1	-1	
275067109	12qwaszx	David Gabriel	FR5102	-1	-1	
213654377	09polkmn	Nuno Garrido	PR3201	-1	-1	
212345678	123qweasd	Carlos Coutinho	PT6202	-1	-1	
298765432	098poilkj	Jorge Rafael	FR2301	-1	-1	
245678912	1qa2ws3ed	Filipa Prudencio	PT5000	-1	-1	
231564897	0ok9ij8uh	Ricardo Fonseca	PT5030	-1	-1	
258147369	321ewqdsa	Paulo Jose Pereira	FR2101	-1	-1	
247158369	890uiojkl	Miguel Tavares	FR5102	-1	-1	
269358147	213wqesad	David Fernandes	PR3201	-1	-1	
289456123	980iuokjl	Paulo Jorge Pereira	PT6202	-1	-1	

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-2** (mas poderá ser outra qualquer).
- Posicione-se nessa diretoria **parte-2** da sua área:
\$ cd parte-2
- 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_trab2_validator**:
\$ ls -l
- Vá para a subdiretoria onde está o validador:
\$ cd so_2023_trab2_validator
- E, finalmente, dentro dessa diretoria, execute o script de validação do seu trabalho:
\$./so_2023_trab2_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_trab2_validator** e correr o script de validação como demonstrado acima.

A aplicação **so_2023_trab2_validator.py** tem algumas opções que podem ser úteis aos alunos:

- Se fizer **so_2023_trab2_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 2024-03-18

- Primeira versão (original) do trabalho.

Versão 2: publicada em 2024-04-01

- **SD11.2** e **SD13.1**: O ficheiro **bd_passageiros.dat** tem de ser aberto para leitura e para escrita ("r+"), senão a operação de abertura (só para escrita) apagaria todo o conteúdo do ficheiro.
- **SD10.2**: Foi separada a comparação do NIF e da Senha. Se encontrar o NIF, então para o **CICLO3** e compara as senhas, dando erro se forem diferentes. Foi também acrescentado o envio do sinal **SIGHUP** para o **Servidor Dedicado** indicar ao **Cliente** que o Check-in não foi corretamente efetuado.
- **SD10.1**: Foi acrescentado o envio do sinal **SIGHUP** para o **Servidor Dedicado** indicar ao **Cliente** que o Check-in não foi corretamente efetuado.
- **SD14**: Foi acrescentado o envio do sinal **SIGHUP** para o **Servidor Dedicado** indicar ao **Cliente** que o Check-in não foi corretamente efetuado.
- **SD11.3, SD11.4, SD13.2, e SD13.3**: Em vez de escrever apenas os campos **pidCliente** e **pidServidorDedicado** no ficheiro **bd_passageiros.dat**, para facilitar, passa a escrever-se o elemento **clientRequest** completo, por cima do elemento **itemDB**.
- **Estas alterações implicam uma alteração nos templates servidor.c e cliente.c, terão de ser copiados novamente.**

Versão 3: publicada em 2024-04-02

- Foi removido o passo de **SD10**, que fazia validação do pidCliente no **Servidor Dedicado**, já que essa validação já é feita no passo **S4.1** na leitura dos dados pelo **Servidor**.
- **Estas alterações implicam uma alteração nos templates servidor.c e cliente.c, terão de ser copiados novamente.**

Versão 4: publicada em 2024-04-02

- **SD11**: Foram acrescentados os envios do sinal **SIGHUP** (indicando erro) ao processo **Cliente**, aquando dos erros.
- **SD12**: Foi acrescentada uma chamada a **so_success** que faltava.