

Projeto *Scut-IUL* (Parte 2)

O presente trabalho visa aplicar os conhecimentos adquiridos durante as aulas de Sistemas Operativos e será composto por três partes, com o objetivo de desenvolver os diferentes aspetos da plataforma **Scut-IUL**. Iremos procurar minimizar as interdependências entre partes do trabalho.



Este enunciado detalha apenas as funcionalidades que devem ser implementadas na parte 2 do trabalho.

A plataforma **Scut-IUL** destina-se à gestão da utilização de um sistema de pagamento automático de portagens.

Algumas definições básicas e tipos de dados para interoperabilidade entre **Cliente** e **Servidor**:

```
#define MIN_PROCESSAMENTO 1 // Tempo mínimo de processamento do Servidor
#define MAX_PROCESSAMENTO 7 // Tempo máximo de processamento do Servidor
#define MAX_ESPERA 7 // Tempo máximo de espera por parte do Cliente
#define NUM_PASSAGENS 20 // Tamanho máximo do buffer de Passagens

typedef struct {
    int tipo_passagem; // Tipo de Passagem: 1-Normal, 2-Via Verde
    char matricula[9]; // Matrícula da viatura
    char lanco[50]; // Lanço da autoestrada
    int pid_cliente; // PID do processo (Cliente) que pede passagem
    int pid_servidor_dedicado; // PID do processo (Servidor) dedicado ao pedido
} Passagem;

Passagem lista_passagens[NUM_PASSAGENS];
```

Os alunos deverão, em vez de **printf** (não será analisado para efeito de avaliação), utilizar sempre as macros **success** (para as mensagens de sucesso) e **error** (para as mensagens de erro) definidas em **utils.h** (a sintaxe destas macros está descrita na secção **Anexo**) para escrever TODAS as mensagens, respetivamente, de sucesso e erro resultantes dos vários passos da aplicação, devendo analisar a secção **Anexo** para ver exemplos de invocação das mesmas. Quando no enunciado estiverem indicados os pedidos de valores entre **< >**, o aluno deverá substituir esse texto pelos valores indicados.

Procedimentos 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 de código (*.c *.h). Cada um dos módulos será desenvolvido com base nos ficheiros fornecidos, e que estão na diretoria do Tigre “/home/so/trabalho-2021-2022/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 bem descritivos). Naturalmente, deverão copiar todos estes ficheiros para a vossa área, e não editar os ficheiros da diretoria /home/so/trabalho-2021-2022/parte-2 (já que não têm permissão para editar nesta diretoria).

Para criarem o ficheiro ZIP, usem, no Tigre, o comando **zip a<nº aluno>-parte-2.zip <ficheiros>**, por exemplo:

```
$ zip a123456-parte-2.zip *.c *.h
```

O ficheiro ZIP deverá depois ser transferido do Tigre para a vossa área local (Windows/Linux/Mac) via SFTP, para depois ser submetido via e-learning.

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

A entrega desta parte do trabalho deverá ser feita por via eletrónica, através do e-learning:

- e-learning da UC Sistemas Operativos, Seleccionam a opção sub-menu “Conteúdo/Content”;
- Seleccionem o link “Trabalho Prático 2021/2022 Parte 2”;
- Dentro do formulário “Visualizar Exercício de carregamento: Trabalho Prático 2021/2022 Parte 2”, seleccionem “Anexar Arquivo” e anexem o vosso ficheiro .zip. Podem submeter o vosso trabalho as vezes que desejarem. **Apenas a última submissão será contabilizada.** Certifiquem-se que a submissão foi concluída, e que esta última versão tem todas as alterações que desejam 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 sua fonte.

Em caso de deteção de algum tipo 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.

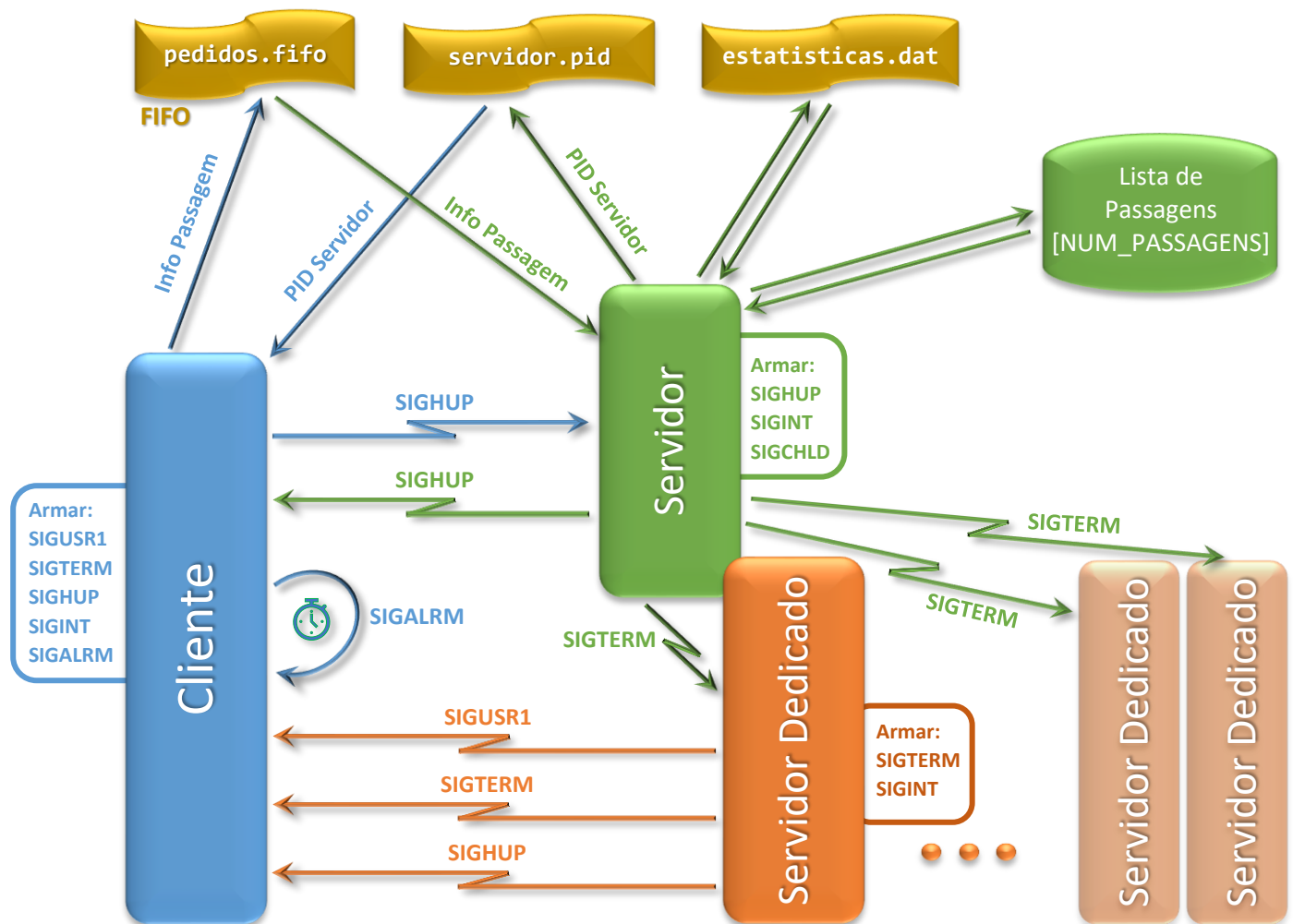
Recorda-se ainda que o Anexo I do Código de Conduta Académica, publicado a 25 de janeiro de 2016 em Diário da República, 2ª Série, nº 16, indica no seu ponto 2 que quando um trabalho ou outro elemento de avaliação apresentar um nível de coincidência elevado com outros trabalhos (percentagem de coincidência 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 II – Processos e Sinais

Data de entrega: 1 de maio de 2022

Nesta parte do trabalho, será implementado um modelo simplificado de gestão da utilização de um sistema de pagamento automático de portagens no sistema **Scut-IUL**, baseado em comunicação por sinais 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 a realização de passagens em portagens no sistema **Scut-IUL**. Assim, teremos dois módulos – **Cliente** e **Servidor**.

Copie para a sua diretoria local todos os ficheiros que se encontram no servidor Tigre, especificamente na diretoria `/home/so/trabalho-2021-2022/parte-2`.

Atenção: Apesar dos vários ficheiros necessários para a realização do trabalho serem fornecidos na diretoria do Tigre “/home/so/trabalho-2021-2022/parte-2”, assume-se que, para a sua execução, os programas executáveis e todos os ficheiros de input e de output estarão todos **sempre** presentes na mesma diretoria, que não deve estar *hardcoded*, ou seja, os programas entregues devem correr em qualquer diretoria.

1) cliente.c

O módulo **Cliente** é responsável pelo pedido das passagens. Este módulo será utilizado para solicitar a passagem das viaturas pelas portagens. Após identificação da viatura, é realizado o pedido da respetiva passagem, ficando este módulo a aguardar até que o processamento esteja concluído. Assim, definem-se as seguintes tarefas a desenvolver:

- C1** Lê a informação acerca do PID do **Servidor** de Passagens que deve estar registado no ficheiro **servidor.pid**. No caso de o ficheiro não existir ou de não ter um PID registado no ficheiro, dá **error C1 "<Problema>"** e termina o processo **Cliente**. Caso contrário, assume que a única informação no ficheiro (em formato de texto) é o PID do **Servidor** (pai, único PID do lado do **Servidor** que este **Cliente** conhece), e dá **success C1 "<PID Servidor>"**;
- C2** Pede ao Condutor (utilizador) que preencha os dados referentes à passagem da viatura (Matrícula e Lanço), criando um elemento do tipo **Passagem** com essas informações, e preenchendo o valor **pid_cliente** com o PID do seu próprio processo **Cliente**. Em caso de ocorrer qualquer erro, dá **error C2"<Problema>"**, e termina o processo **Cliente**; caso contrário dá **success C2"Passagem do tipo <Normal | Via Verde> solicitado pela viatura com matrícula <matricula> para o Lanço <lanco> e com PID <pid_cliente>"**;
- C3** Arma os sinais **SIGUSR1** (ver **C6**), **SIGTERM** (ver **C7**), **SIGHUP** (ver **C8**), **SIGINT** (ver **C9**), e **SIGALRM** (ver **C10**), dando, no fim de os armar, a mensagem **success C3 "Armei sinais"**;
- C4** Valida se o ficheiro com organização FIFO (*named pipe*) **pedidos.fifo** existe. Se esse ficheiro FIFO não existir, dá **error C4** e termina o processo **Cliente**. Caso contrário, escreve as informações do elemento **Passagem** (em formato binário) nesse FIFO **pedidos.fifo**. Em caso de erro na escrita, dá **error C4** e termina o processo **Cliente**, caso contrário, dá **success C4 "Escrevi FIFO"**;
- C5** Configura um alarme com o valor de **MAX_ESPERA** segundos (ver **C10**), dá **success C5 "Inicia Espera de <MAX_ESPERA> segundos"**, e fica a aguardar o resultado do processamento do pedido por parte do **Servidor**.
- C6** O sinal armado **SIGUSR1** serve para o **Servidor Dedicado** indicar que o processamento da passagem foi iniciado. Se o **Cliente** receber esse sinal, dá **success C6 "Passagem Iniciada"**, assinala que o processamento iniciou, e retorna para aguardar a conclusão do processamento do lado do **Servidor Dedicado**;
- C7** O sinal armado **SIGTERM** serve para o **Servidor Dedicado** indicar que o processamento da passagem foi concluído. Se o **Cliente** receber esse sinal, dá **success C7 "Passagem Concluída"**, e termina o processo **Cliente**. **ATENÇÃO:** Deverá previamente validar que anteriormente este **Cliente** já tinha recebido o sinal **SIGUSR1** (ver **C6**), indicando que o processamento do lado do **Servidor Dedicado** teve início, caso contrário, em vez de sucesso, dá **error C7** e termina o processo **Cliente**;
- C8** O sinal armado **SIGHUP** serve para o **Servidor Dedicado** indicar que o processamento a passagem não foi concluído. Se o **Cliente** receber esse sinal, dá **success C8 "Processo Não Concluído e Incompleto"**, e termina o processo **Cliente**;
- C9** O sinal armado **SIGINT** serve para que, no caso de o veículo ter uma avaria, ou por outro motivo qualquer, o condutor (utilizador) possa cancelar o pedido do lado do **Cliente**, usando o atalho **<CTRL+C>**. Se receber esse sinal (do utilizador via Shell), o **Cliente** envia o sinal **SIGHUP** ao **Servidor**, para que esta passagem seja sinalizada como anomalia, dá **success C9 "Processo Cancelado pelo Cliente"**, e retorna para aguardar que o **Servidor Dedicado** conclua o processo (o **Servidor Dedicado** deverá mais tarde enviar o sinal **SIGHUP** a este **Cliente**, ver **C8**);
- C10** O sinal armado **SIGALRM** serve para que, se o **Cliente** em **C5** esperou mais do que **MAX_ESPERA** segundos sem resposta, o **Cliente** envia o sinal **SIGHUP** ao **Servidor**, para que esta passagem seja sinalizada como anomalia, dá **success C10 "Timeout Cliente"**, e retorna para aguardar que o **Servidor Dedicado** conclua o processo (o **Servidor Dedicado** deverá mais tarde enviar o sinal **SIGHUP** a este **Cliente**, ver **C8**).

2) servidor.c

O módulo **Servidor** de Passagens é responsável pelo processamento de pedidos de passagem que chegam ao sistema **Scut-IUL**. Este módulo é, normalmente, o primeiro dos dois (**Cliente** e **Servidor**) a ser executado, e deverá estar sempre ativo, à espera de pedidos de passagem. O tempo de processamento destes pedidos varia entre os **MIN_PROCESSAMENTO** segundos e os **MAX_PROCESSAMENTO** segundos. Findo esse tempo, este módulo sinaliza ao condutor de que a sua passagem foi processada. Este módulo deverá possuir contadores de passagens por tipo, um contador de anomalias e uma lista com capacidade para processar **NUM_PASSAGENS** passagens. O módulo **Servidor** de Passagens é responsável por realizar as seguintes tarefas:

- S1** Inicia a lista de passagens, preenchendo em todos os elementos o campo **tipo_passagem=-1** ("Limpa" a lista de passagens). Em seguida, dá **success S1 "Init Servidor"**;
- S2** Deverá manter um contador por cada tipo de passagem (Normal ou Via Verde) e um contador para as passagens com anomalia. Se o ficheiro **estatisticas.dat** existir na diretoria local, abre-o e lê os seus dados (em formato binário, ver formato em **S10.4**) para carregar o valor guardado de todos os contadores. Se houver erro na leitura do ficheiro, dá **error S2 "<Problema>"**, caso contrário, dá **success S2 "Estatísticas Carregadas"**. Se o ficheiro não existir, inicia os três contadores com o valor 0 e dá **success S2 "Estatísticas Iniciadas"**;
- S3** Cria o ficheiro **servidor.pid**, e escreve nesse ficheiro o PID do **Servidor** (em formato de texto). Se houver erro em qualquer das operações, dá **error S3** e termina. Caso contrário, dá **success S3 "<PID Servidor>"**;
- S4** Cria o ficheiro com organização FIFO (*named pipe*) **pedidos.fifo**. Se houver erro na operação, dá **error S4 "<Problema>"**, caso contrário, dá **success S4 "Criei FIFO"**;
- S5** Arma e trata os sinais **SIGINT** (ver **S10**), **SIGHUP** (usando `sigaction()`, ver **S11**) e **SIGCHLD** (ver **S12**). Depois de armar os sinais, dá **success S5 "Armei sinais"**;
- S6** Lê a informação do ficheiro **pedidos.fifo**, (em formato binário) que deverá ser um elemento do tipo **Passagem**. Se houver erro na operação, dá **error S6 "<Problema>"**, caso contrário, dá **success S6 "Li FIFO"**;
- S7** O **Servidor** deve validar se o pedido está corretamente formatado. A formatação correta de um pedido será:
 - O Tipo de passagem é válido (1 para pedido Normal, ou 2 para Via Verde);
 - A Matrícula e o Lanço não são strings vazias (não é necessário fazer mais validações sobre o seu conteúdo);
 - O **pid_cliente** é um valor > 0. Não se fazem validações sobre **pid_servidor_dedicado**.

Em caso de erro na formatação, dá **error S7**, com o erro detetado, incrementa o contador de anomalias, manda o sinal **SIGHUP** ao processo com PID **<pid_cliente>**, ignora o pedido, e recomeça o processo no passo **S6**. Senão, dá **success S7 "Chegou novo pedido de passagem do tipo <Normal | Via Verde> solicitado pela viatura com matrícula <matricula> para o Lanço <lanco> e com PID <pid_cliente>"**;

- S8** Verifica se existe disponibilidade na Lista de Passagens. Se todas as entradas da Lista de Passagens estiverem ocupadas, dá **error S8 "Lista de Passagens cheia"**, incrementa o contador de passagens com anomalia, manda o sinal **SIGHUP** ao processo com PID **<pid_cliente>**, ignora o pedido, e recomeça o processo no passo **S6**. Caso contrário, preenche uma entrada da lista com os dados deste pedido, incrementa o contador de passagens do tipo de passagem correspondente e dá **success S8 "Entrada <índice lista> preenchida"**;
- S9** Cria um processo filho (fork) **Servidor Dedicado**. Se houver erro, dá **error S9 "Fork"**. Caso contrário: O processo **Servidor Dedicado** (filho) continua no passo **SD13**, e o processo **Servidor** (pai) completa o preenchimento da entrada atual da Lista de Passagens com o PID do **Servidor Dedicado**, e dá **success S9 "Criado Servidor Dedicado com PID <pid Filho>"**. Em qualquer dos casos, de erro ou de sucesso, recomeça o processo no passo **S6**;

S10 O sinal armado **SIGINT** serve para o Diretor da Portagem encerrar o **Servidor**, usando o atalho <CTRL+C>. Se receber esse sinal (do utilizador via Shell), o **Servidor** dá **success S10 "Shutdown Servidor"**, e depois:

S10.1 Envia o sinal **SIGTERM** a todos os **Servidores Dedicados** da Lista de Passagens, para que conclua o seu processamento imediatamente. Depois, dá **success S10.1 "Shutdown Servidores Dedicados"**;

S10.2 Remove o ficheiro **servidor.pid**. Em caso de erro, dá **error S10.2**, caso contrário, dá **success S10.2**;

S10.3 Remove o FIFO **pedidos.fifo**. Em caso de erro, dá **error S10.3**, caso contrário, dá **success S10.3**;

S10.4 Cria o ficheiro **estatisticas.dat**, escrevendo nele o valor de 3 inteiros (em formato binário), correspondentes

a

<contador de passagens Normal>	<contador de passagens Via Verde>	<contador Passagens com Anomalia>
--------------------------------	-----------------------------------	-----------------------------------

Em caso de erro, dá **error S10.4**, caso contrário, dá **success S10.4 "Estatísticas Guardadas"**;

S10.5 Dá **success S10.5** e termina o processo **Servidor**.

S11 O sinal armado **SIGHUP** serve para o **Cliente** indicar que deseja cancelar o pedido de processamento a passagem. Se o **Servidor** receber esse sinal, dá **success S11 "Cancel"**, e em seguida, terá de fazer as seguintes ações:

S11.1 Identifica o PID do processo **Cliente** que enviou o sinal (usando sigaction), dá **success S11.1 "Cancelamento enviado pelo Processo <PID Cliente>"**;

S11.2 Pesquisa na Lista de Passagens pela entrada correspondente ao PID do **Cliente** que cancelou. Se não encontrar, dá **error S11.2**. Caso contrário, descobre o PID do **Servidor Dedicado** correspondente, dá **success S11.2 "Cancelamento <PID Filho>"**, e incrementa o contador de passagens com anomalia;

S11.3 Envia o sinal **SIGTERM** ao **Servidor Dedicado** da Lista de Passagens correspondente ao cancelamento, para que conclua o seu processamento imediatamente. Depois, dá **success S11.3 "Sinal de Cancelamento enviado ao Servidor Dedicado"**, e recomeça o processo no passo **S6**.

S12 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, dá **success S12 "Servidor Dedicado Terminou"**, e em seguida:

S12.1 Identifica o PID do **Servidor Dedicado** que terminou (usando wait), dá **success S12.1 "Terminou Servidor Dedicado <PID Filho>"**;

S12.2 Pesquisa na Lista de Passagens pela entrada correspondente ao PID do Filho que terminou. Se não encontrar, dá **error S12.2**. Caso contrário, “apaga” a entrada da Lista de Passagens correspondente (colocando **tipo_passagem=-1**), dá **success S12.2**, e recomeça o processo no passo **S6**.

SD13 O novo processo **Servidor Dedicado** (filho) arma os sinais **SIGTERM** (ver **SD17**) e **SIGINT** (programa para ignorar este sinal). Depois de armar os sinais, dá **success SD13 "Servidor Dedicado Armei sinais"**;

SD14 O **Servidor Dedicado** envia o sinal **SIGUSR1**, indicando o início do processamento da passagem, ao processo <pid_cliente> que pode obter da estrutura **Passagem** do pedido que “herdou” do **Servidor** ou da entrada da Lista de Passagens, e dá **success SD14 "Início Passagem <PID Cliente> <PID Servidor Dedic>"**;

SD15 O **Servidor Dedicado** calcula um valor aleatório (usando **my_rand()**) entre os valores **MIN_PROCESSAMENTO** e **MAX_PROCESSAMENTO**, dá **success SD15 "<Tempo>"**, e aguarda esse valor em segundos (**sleep**);

SD16 O **Servidor Dedicado** envia o sinal **SIGTERM**, indicando o fim do processamento da passagem, ao processo <pid_cliente>, dá **success SD16 "Fim Passagem <PID Cliente> <PID Servidor Dedicado>"**, e termina o processo **Servidor Dedicado**;

SD17 O sinal armado **SIGTERM** serve para o **Servidor** indicar que deseja terminar imediatamente o pedido de processamento da passagem. Se o **Servidor Dedicado** receber esse sinal, envia o sinal **SIGHUP** ao <pid_cliente>, dá **success SD17 "Processamento Cancelado"**, e termina o **Servidor Dedicado**.

Anexo

Macros fornecidas, com Mensagens de **sucesso**, **erro**, **debug** e **validação de Scripts**:

Mensagens de output com Erro (com exemplos): Macro `error(<Passo>, <Mensagem>)`

A sintaxe é semelhante à do `printf()`; esta macro, tem como output no STDOUT:

```
"@@Error@@ {<Passo>} <Mensagem>"
```

Exemplos:

- O ficheiro `FILE_SERVIDOR` não existe:
 - `error("C1", "O ficheiro %s não existe", FILE_SERVIDOR);`
- Não é possível criar o FIFO `FILE_PEDIDOS`:
 - `error("S4", "Não foi possível criar o FIFO %s", FILE_PEDIDOS);`

Mensagens de output com Sucesso (com exemplos): Macro `success(<Passo>, <Mensagem>)`

A sintaxe é semelhante à do `printf()`; esta macro, tem como output no STDOUT:

```
"@@Success@@ {<Passo>} <Mensagem>"
```

Exemplos:

- O **Cliente** indica que iniciou o período de espera:
 - `success("C5", "Inicia Espera de %d segundos", MAX_ESPERA);`
- O **Servidor** indica que recebeu um novo pedido de passagem:
 - `success("S7", "Chegou novo pedido de passagem do tipo Via Verde solicitado pela viatura com matrícula %s para o Lanço %s e com PID %d", "AJ21RG", "Lisboa-Coimbra", 1234);`

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

A sintaxe é semelhante à do `printf()`; esta macro, tem como output no STDOUT:

```
"@@Debug@@:<Ficheiro Source>:<Nº Linha Source>:<Nome Função>: {<Passo>} <Mensagem>"
```

Exemplos:

- `debug("S8", "Entrada atual: %d", var_Entrada_Atual);`
- `debug("SD13", "Passei por aqui");`

Tem a vantagem de que mostra sempre as mensagens de debug (não precisa sequer ser nunca apagado). Quando os alunos quiserem apagar as mensagens de debug, basta alterar uma linha no header file `utils.h`: O valor `DEBUG_MODE TRUE` passa para `DEBUG_MODE FALSE`, e assim, mantendo o vosso programa intocado, não mostra nenhuma mensagem de debug.

Análise dos ficheiros binários:

Neste trabalho são armazenadas informações em dois ficheiros em formato binário, **pedidos.fifo** e **estatisticas.dat**. Não é fácil visualizar estes ficheiros 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, foram fornecidos dois scripts que ajudam a visualizar os conteúdos destes ficheiros.

pedidos.fifo:

Sendo um FIFO, não é fácil obter o conteúdo do mesmo, mas pode-se fazer usando o comando Shell:

```
$ cat pedidos.fifo > pedido.dat
```

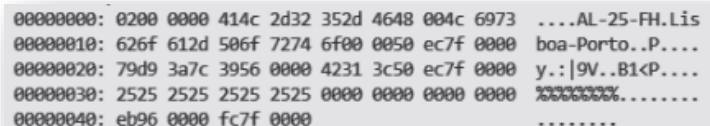
Assumindo uma estrutura Passagem armazenada no ficheiro, com o conteúdo:

```
passagem.tipo_passagem = 2;           // Tipo de Passagem: 1-Normal, 2-Via Verde
passagem.matricula = "AL-25-FH";      // Matrícula da viatura
passagem.lanco = "Lisboa-Porto";      // Lanço da autoestrada
passagem.pid_cliente = 38635;         // PID do processo (Cliente) que pede passagem
pid_servidor_dedicado = 32764;        // PID do processo (Servidor) dedicado ao pedido
```

consegue-se ver o conteúdo do ficheiro usando a aplicação **xxd**:

```
$ xxd pedido.dat      # Visualização de um ficheiro binário usando xxd
```

Um exemplo do output deste comando pode ser visualizado na imagem à direita:

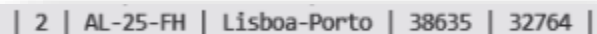


```
00000000: 0200 0000 414c 2d32 352d 4648 004c 6973  ....AL-25-FH.Lis
00000010: 626f 612d 506f 7274 6f00 0050 ec7f 0000  boa-Porto..P....
00000020: 79d9 3a7c 3956 0000 4231 3c50 ec7f 0000  y.:|9V..B1<P....
00000030: 2525 2525 2525 2525 0000 0000 0000 0000  %333333%.....
00000040: eb96 0000 fc7f 0000                                     .....
```

Em vez de usar a aplicação **xxd**, podemos usar o script fornecido **show-pedido.sh**. Usando este script, o mesmo ficheiro binário anterior pode ser visualizado da seguinte forma:

```
$ ./show-pedido.sh pedido.dat
```

Um exemplo do output deste comando pode ser visualizado na imagem em baixo:



```
| 2 | AL-25-FH | Lisboa-Porto | 38635 | 32764 |
```

Uma alternativa para visualizar esta informação é:

```
$ cat pedidos.fifo | ./show-pedido.sh
```

estatisticas.dat:

De forma análoga, é difícil analisar o ficheiro de estatísticas **estatisticas.dat** usando **cat** ou até **xxd**, mas pode-se fazer usando o comando Shell:

```
$ ./show-stats.sh
```

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 vosso trabalho:

- Garantam que o vosso trabalho (i.e., os ficheiros de código *.c *.h) está localizado numa diretoria local da vossa área. Para os efeitos de exemplo para esta demonstração, assumiremos que essa diretoria terá o nome **trab-so-parte-2** (mas poderá ser outra qualquer);
- Posicionem-se nessa diretoria **trab-so-parte-2** da vossa área:
`$ cd trab-so-parte-2`
- Deem o comando `$ pwd`, e validem que estão mesmo na diretoria correta;
- Deem o comando `$ ls -l`, e confirmem que todos os ficheiros *.c *.h do vosso trabalho estão mesmo nessa diretoria, e que esses ficheiros têm as permissões suficientes;
- Deem o comando para criar a diretoria do validador na vossa diretoria local (.):
`$ mkdir so-2021-trab2-validator`
- Agora, posicionem-se na subdiretoria do validador:
`$ cd so-2021-trab2-validator/`
- Criem soft-links do validador para a vossa diretoria de validação:
`$ ln -s /home/so/trabalho-2021-2022/parte-2/so-2021-trab2-validator/* .`
- E, finalmente, executem o script de validação do vosso trabalho, que está na diretoria “pai” (..)
`$./so-2021-trab2-validator.py ..`
- Resta agora verificarem quais dos vossos testes “passam” (✓) e quais “chumbam” (✗);
- Façam as alterações para correção dos vossos scripts;
- Sempre que quiserem voltar a fazer nova validação, basta novamente posicionarem-se na subdiretoria **so-2021-trab2-validator** e correrem o script de validação como demonstrado acima.

Boa sorte!!!
Os docentes