

Projeto *Scut-IUL* (Parte 3)

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 3 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** estão no ficheiro **common.h**, sendo que as estruturas **Passagem** e **Contadores** são iguais às do trabalho anterior, e acrescentam-se as seguintes:

```
typedef struct {
    Contadores contadores;
    Passagem lista_passagens[NUM_PASSAGENS];
} DadosServidor;

typedef struct {
    long tipoMensagem;
    struct {
        int action; // Ação associada à mensagem
        union { // Nos dados, OU vai um pedido OU vão contadores
            Passagem pedido_cliente;
            Contadores contadores_servidor;
        } dados;
    } conteudo;
} Mensagem;
```

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 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 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-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 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-3 (já que não têm permissão para editar nesta diretoria).

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

```
$ zip a123456-parte-3.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 3”;
- Dentro do formulário “Visualizar Exercício de carregamento: Trabalho Prático 2021/2022 Parte 3”, 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.

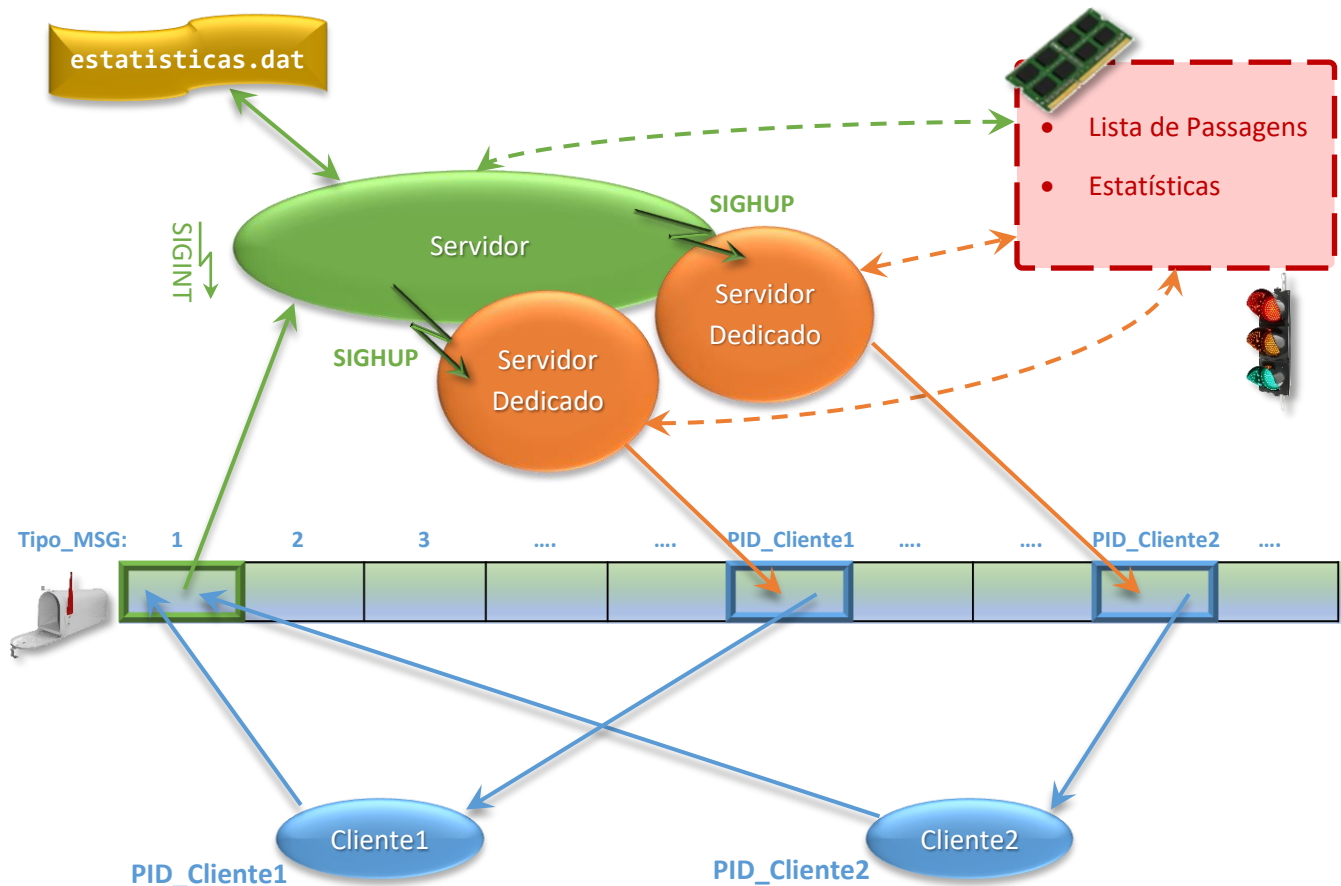
Recorda-se ainda que o Anexo I do Código de Conduta Académica, publicado a 25 de janeiro de 2016 em Diário da República, 2ª Série, nº 16, indica no seu ponto 2 que quando um trabalho ou outro elemento de avaliação apresentar um nível de coincidência elevado com outros trabalhos (percentagem de coincidência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

Data de entrega: **22 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 IPC e 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-3**.

Atenção: Apesar de vários ficheiros necessários para a realização do trabalho serem fornecidos na diretoria do Tigre **"/home/so/trabalho-2021-2022/parte-3"**, 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** Tenta abrir uma fila de mensagens (*Message Queue*) IPC que tem a KEY **IPC_KEY** definida em **common.h** (alterar esta KEY para ter o valor do nº do aluno, como indicado nas aulas). Deve assumir que a fila de mensagens já foi criada. Em caso de erro, dá **error C1 "<Problema>"** e termina o **Cliente** com *exit code* -1. Caso contrário dá **success C1 "<msg_id>"**;
- 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** Envia as informações do elemento **Passagem** numa mensagem com o tipo de mensagem **1** e action **1 – Pedido** para a *Message Queue*. Em caso de erro na escrita, dá **error C3 "<Problema>"**, e termina o processo **Cliente** com *exit code* -1. Caso contrário, dá **success C3 "Enviei mensagem"**;
- C4** Lê da *Message Queue* uma mensagem cujo tipo de mensagem é igual ao PID deste processo **Cliente**, mensagem essa que poderá vir do **Servidor Dedicado**. Se houver algum erro dá **error C4 "<Problema>"** e termina o **Cliente** com *exit code* -1. Caso contrário dá **success C4 "Li mensagem do Servidor"**.
- C5** Se a mensagem que chegou em **C4** veio com action **2 – Pedido ACK**, serve para o **Servidor Dedicado** indicar que o processamento da passagem foi iniciado. Se o **Cliente** receber essa mensagem, dá **success C5 "Passagem Iniciada"**, assinala que o processamento iniciou, e retorna para aguardar a conclusão do processamento do lado do **Servidor Dedicado**;
- C6** Se a mensagem que chegou em **C4** veio com action **3 – Pedido Concluído**, serve para o **Servidor Dedicado** indicar que o processamento da passagem foi concluído. Se o **Cliente** receber essa mensagem, que inclui os contadores de estatística atualizados, dá **success C6 "Passagem Concluída com estatísticas: <contador normal> <contador via verde> <contador anomalias>"**, e termina o processo **Cliente**. **ATENÇÃO:** Deverá previamente validar que anteriormente este **Cliente** já tinha recebido a mensagem com action **2 – Pedido ACK** (ver **C5**), indicando que o processamento do lado do **Servidor Dedicado** teve início, caso contrário, em vez de sucesso, dá **error C6** e termina o processo **Cliente**;
- C7** Se a mensagem que chegou em **C4** veio com action **4 – Pedido Cancelado**, serve para o **Servidor Dedicado** indicar que o processamento a passagem não foi concluído. Se o **Cliente** receber esse sinal, dá **success C7 "Processo Não Concluído e Incompleto"**, e termina o processo **Cliente**.

Valores possíveis do campo **action** das mensagens da *Message Queue*:

- **1 – Pedido:** Envio do **Cliente** para o **Servidor**, preenchendo também a estrutura **pedido_cliente**;
- **2 – Pedido ACK:** Envio do **Servidor Dedicado** para o **Cliente**, não precisa preencher mais nenhuma informação;
- **3 – Pedido Concluído:** Envio do **Servidor Dedicado** para o **Cliente**, para conclusão do processo de passagem, preenchendo também a estrutura **contadores_servidor**;
- **4 – Pedido Cancelado:** Envio do **Servidor Dedicado** para o **Cliente**, não precisa preencher mais nenhuma informação.

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** Tenta abrir uma *Shared Memory* (SHM) IPC que tem a KEY **IPC_KEY** definida em **common.h** (alterar esta KEY para ter o valor do nº do aluno, como indicado nas aulas). Se essa *Shared Memory* ainda não existir passa para o passo **S2** sem erro. Caso contrário, liga-se a ela. Em caso de erro, dá **error S1 "<Problema>"**, e termina o **Servidor** (-1). Senão dá **success S1 "Abri Shared Memory já existente com ID <shmId>"**.
- S2** Se no ponto **S1** a *Shared Memory* ainda não existia, então realiza as seguintes operações:
 - S2.1** Cria uma *Shared Memory* com a KEY **IPC_KEY** definida em **common.h** e com o tamanho para conter os Dados do **Servidor**, e liga-se a ela. Em caso de erro, dá **error S2.1 "<Problema>"**, e termina o processo **Servidor** (-1). Caso contrário, dá **success S2.1 "Criei Shared Memory com ID <shmId>"**;
 - S2.2** Inicia a lista de passagens, preenchendo em todos os elementos o campo **tipo_passagem=-1** ("Limpa" a lista de passagens). Em caso de qualquer erro, dá **error S2.2 "<Problema>"**, e termina o processo **Servidor** (-1). Caso contrário, dá **success S2.2 "Iniciei Shared Memory Passagens"**;
 - S2.3** Tem 3 contadores, para as passagens Normal, Via Verde e 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 **S6.2**) e carrega o valor nos contadores. Se houver erro na leitura do ficheiro, dá **error S2.3 "<Problema>"**, e termina o **Servidor** (-1). Caso contrário, dá **success S2.3 "Estatísticas Carregadas"**. Se o ficheiro não existir, inicia os três contadores com o valor 0 e dá **success S2.3 "Estatísticas Iniciadas"**;
- S3** Cria uma *Message Queue* com a KEY **IPC_KEY** definida em **common.h**, e arma os sinais **SIGINT** (ver **S6**) e **SIGCHLD** (programa para ignorar este sinal). Se houver erros em qualquer destas operações, dá **error S3 "<Problema>"** e termina o processo **Servidor** com *exit code* -1. Caso contrário, dá **success S3 "Criei mecanismos IPC"**;
- S4** Lê a informação da *Message Queue* numa mensagem com o tipo de mensagem **1**. Essa mensagem deverá ter a action **1 – Pedido** e deverá conter um elemento do tipo **Passagem**. Se houver erro na operação, dá **error S4 "<Problema>"**, e termina o **Servidor** (-1). Caso contrário, dá **success S4 "Li Pedido do Cliente"**;
- S5** Cria um processo filho (fork) **Servidor Dedicado**. Se houver erro, dá **error S5 "Fork"**. Senão, o processo **Servidor Dedicado** (filho) continua no passo **SD7**, e o processo **Servidor** (pai) dá **success S5 "Criado Servidor Dedicado com PID <pid Filho>"**. Em qualquer dos casos, recomeça o processo no passo **S4**;
- S6** 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 S6 "Shutdown Servidor"**, e depois:
 - S6.1** Envia o sinal **SIGHUP** a todos os **Servidores Dedicados** da Lista de Passagens, para que concluem o seu processamento imediatamente. Depois, dá **success S6.1 "Shutdown Servidores Dedicados"**;
 - S6.2** 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 S6.2**, caso contrário, dá **success S6.2 "Estatísticas Guardadas"**;
 - S6.3** Dá **success S6.3 "Shutdown Servidor completo"**, apaga a *Message Queue* e o grupo de Semáforos criados (mas não apaga a *Shared Memory*), e termina o processo **Servidor** com *exit code* 0.

SD7 O novo processo **Servidor Dedicado** (filho) arma os sinais **SIGHUP** (ver **SD13**) e **SIGINT** (programa para ignorar este sinal). Depois de armar os sinais, dá **success SD7 "Servidor Dedicado Armei sinais"**;

SD8 O **Servidor Dedicado** deve validar se o pedido que “herdou” do **Servidor** está corretamente formatado. Esse pedido inclui uma estrutura **Passagem** cuja formatação correta tem de validar se:

- 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.

Em caso de erro na formatação:

- Dá **error SD8 "<Problema>"**, e incrementa o contador de anomalias;
- Se **pid_cliente** é um valor > 0, envia uma mensagem com action 4 – **Pedido Cancelado**, para a *Message Queue* com tipo de mensagem igual ao **pid_cliente**. Se o envio tiver erro, dá **error SD8 "<Problema>"**;
- Ignora o pedido, e termina o processo **Servidor Dedicado** com *exit code -1*.

Caso contrário, se não houver erro na formatação, dá **success SD8 "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>"**;

SD9 Verifica se existe disponibilidade na Lista de Passagens. Se todas as entradas da Lista de Passagens estiverem ocupadas, dá **error SD9 "Lista de Passagens cheia"**, incrementa o contador de passagens com anomalia, manda uma mensagem com action 4 – **Pedido Cancelado**, para a *Message Queue* com tipo de mensagem igual ao **pid_cliente**, ignora o pedido, e termina o processo **Servidor Dedicado** com *exit code -1*. Caso contrário, preenche uma entrada da lista com os dados deste pedido, acrescentando a informação do seu PID, incrementa o contador de passagens do tipo de passagem correspondente e dá **success SD9 "Entrada <índice lista> preenchida"**;

SD10 O **Servidor Dedicado** envia uma mensagem com action 2 – **Pedido ACK**, para a *Message Queue* com tipo de mensagem igual ao **pid_cliente**, indicando o início do processamento da passagem, e dá **success SD10 "Início Passagem <PID Cliente> <PID Servidor Dedicado>"**;

SD11 O **Servidor Dedicado** calcula um valor aleatório (usando `my_rand()`) entre os valores **MIN_PROCESSAMENTO** e **MAX_PROCESSAMENTO**, dá **success SD11 "<Tempo>"**, e aguarda esse valor em segundos (*sleep*);

SD12 O **Servidor Dedicado** envia uma mensagem com action 3 – **Pedido Concluído**, para a *Message Queue* com tipo de mensagem igual ao **pid_cliente**, onde também deverá incluir os valores atuais das estatísticas na estrutura `contadores_servidor`, indicando o fim do processamento da passagem ao processo **<pid_cliente>**, apaga a entrada do **Cliente** na lista de passagens, dá **success SD12 "Fim Passagem <PID Cliente> <PID Servidor Dedicado>"**, e termina o processo **Servidor Dedicado**;

SD13 O sinal armado **SIGHUP** serve para o **Servidor** indicar que deseja terminar imediatamente o pedido de processamento da passagem. Se o **Servidor Dedicado** receber esse sinal, não incrementa o contador de passagens com anomalia, mas manda uma mensagem com action 4 – **Pedido Cancelado**, para a *Message Queue* com tipo de mensagem igual ao **pid_cliente**, apaga a entrada do **Cliente** na lista de passagens, dá **success SD13 "Processamento Cancelado"**, e termina o **Servidor Dedicado**.

SD14 Repare que os vários **Servidores Dedicados** têm todos acesso concorrente à *Shared Memory*. Acrescente em **S3** a criação de um grupo com dois semáforos do tipo **MUTEX**, um dedicado à lista de passagens e outro dedicado às estatísticas. Altere o código do Servidor e do Servidor Dedicado por forma a garantir a exclusão mútua no acesso a cada um destes dois elementos dos Dados do Servidor, garantindo que o tempo passado em exclusão é sempre o menor possível.

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(<Mensagem>)`

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

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

Exemplos:

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

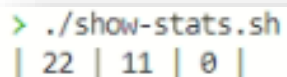
Tem a vantagem de que mostra sempre as mensagens de debug (não precisa sequer ser nunca apagado). Quando os alunos quiserem apagar as mensagens de debug, basta descomentarem uma linha no ficheiro `servidor.c` ou `cliente.c`, aquela que torna O valor `DEBUG_MODE TRUE` passa para `DEBUG_MODE FALSE`, e assim, mantendo o vosso programa intocado, não mostra nenhuma mensagem de debug.

Foi também fornecida uma função utilitária `shmView()` que mostra o conteúdo da *Shared Memory* em debug mode.

Análise dos ficheiros binários:

Neste trabalho são armazenadas informações num ficheiro em formato binário, **estatisticas.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 o conteúdo deste ficheiro:

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



```
> ./show-stats.sh  
| 22 | 11 | 0 |
```

A terminal window with a light gray background. The first line shows a green prompt character followed by the command `./show-stats.sh`. The second line shows the output of the script, which is a table with three columns and one row of data: `| 22 | 11 | 0 |`.

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-3** (mas poderá ser outra qualquer);
- Posicionem-se nessa diretoria **trab-so-parte-3** da vossa área:
`$ cd trab-so-parte-3`
- 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-trab3-validator`
- Agora, posicionem-se na subdiretoria do validador:
`$ cd so-2021-trab3-validator/`
- Criem soft-links do validador para a vossa diretoria de validação:
`$ ln -s /home/so/trabalho-2021-2022/parte-3/so-2021-trab3-validator/* .`
- E, finalmente, executem o script de validação do vosso trabalho, que está na diretoria “pai” (..)
`$./so-2021-trab3-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-trab3-validator** e correrem o script de validação como demonstrado acima.

Boa sorte!!!
Os docentes