Projeto (Fase I)

1 Introdução ao Projeto

A parte teórico-prática da disciplina de Sistemas Operativos pretende familiarizar os alunos com alguns dos problemas envolvidos na utilização dos recursos de um Sistema Operativo. O projeto de avaliação será realizado utilizando principalmente a linguagem de programação C e as APIs de Linux e POSIX (*Portable Operating System Interface*), mas inclui também alguma programação em `shell` (`bash`) e `makefile`.

O propósito geral do projeto é o desenvolvimento de uma aplicação em C para simular o fluxo central de um serviço de administração portuária com várias operações, tais como efetuar cargas, descargas, armazenagem de mercadorias, etc. Esta aplicação, chamada *SO_AdmPor*, envolverá múltiplos processos cooperativos para efetuar as suas operações. O fluxo de chamadas entre processos envolve: (i) o envio de descrições de operações pretendidas, (ii) a emissão de pedidos destas operações e (iii) o agendamento/execução das mesmas. De forma a se poder aferir a qualidade do serviço prestado, são também registadas informações de progresso sob a forma de um *log*, que podem posteriormente ser analisadas.

O *SO_AdmPor* será realizado em 2 fases. A primeira fase tem como objetivo fundamental a familiarização com código base fornecido, a criação do ficheiro `makefile` para compilação e manutenção do projeto, a gestão de processos e a alocação de memória. Neste primeiro enunciado é feita uma apresentação geral da *SO_AdmPor* juntamente com informação específica de suporte à realização da primeira fase. Na fase seguinte será disponibilizado um novo enunciado que complementará a informação contida neste enunciado.

2 Funcionamento Geral

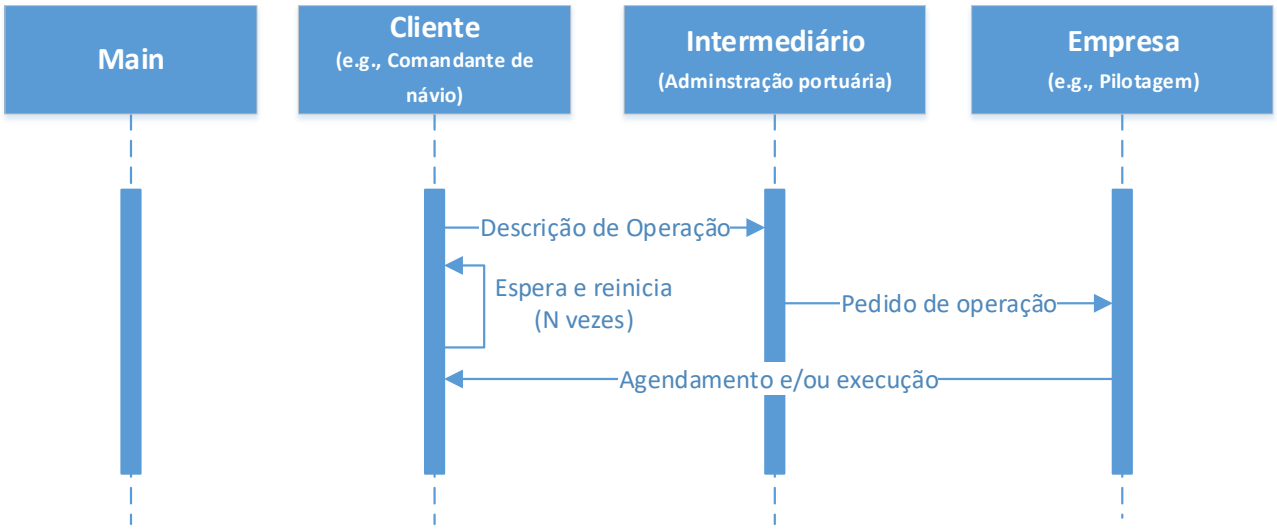

Figura 1 - Diagrama de sequência para a agendamento e/ou execução de operações portuários na SO_AdmPor

A Administração Portuária de um porto é um agente público que tem um papel muito particular e difícil de desempenhar, pois tem de coordenar e facilitar todas as atividades que ocorrem no seu porto. No contexto de SO, o *SO_AdmPor* vai agregar informação sobre um catálogo de operações disponibilizadas por vários

empresas ou profissionais prestadores de serviços (e.g., pilotagem, reboque, amarração, armazenagem, carga/descarga, etc.), facilitando o acesso aos mesmos a clientes (e.g., comandantes dos navios) através da utilização de um intermediário (i.e., administração portuária). O fluxo central do sistema a simular (Figura 1) compreende as seguintes etapas:

1. Um cliente (e.g., um comandante de um navio), preenche um formulário com a descrição da operação que pretende (e.g., descarga de mercadoria), coloca-o na lista de operações portuárias e aguarda indicação da administração portuária para realizar a operação.
2. Um intermediário (i.e., operador de administração portuária) obtém a descrição da operação, verifica a disponibilidade de empresas e/ou profissionais que possam satisfazer a operação e, caso haja, preenche um pedido de despacho de operação que coloca na lista de pedidos de operação portuárias a despachar.
3. Uma empresa e/ou profissional recebe um pedido de execução de operação, e se este puder ser satisfeito, atualiza a sua base de dados (i.e., inventário interno), envia o agendamento para o cliente, colocando-os na área de operações portuárias agendadas.
4. Um cliente cuja descrição de operação tenha sido satisfeita, descarrega o agendamento e aguarda pela sua execução. Após a conclusão da mesma, sai do porto, navega por um período aleatório, e retorna ao porto para solicitar uma nova operação (fluxo "Espera e reinicia" na figura). Um cliente pode efetuar no sistema um definido número N de operações.

3 Interação com o Sistema Operativo

Criação de Processos (Fase 1):
No *SO_AdmPor*, as entidades (clientes, intermediários e empresas) são representadas por processos que cooperam através de buffers para troca de descrição/pedido/agendamento de operações. São apresentadas de seguida as interações com o sistema operativo.

Comunicação entre Processos (Fase 1):
Os processos comunicam através da escrita e leitura em buffers que não são mais do que zonas de memória com uma determinada capacidade para guardar um determinado número de estruturas de dados de um dado tipo (vetor de estruturas). Cada estrutura de dados representa uma descrição, um pedido ou agendamento, consoante o buffer. Uma vez que um processo não pode aceder ao espaço de endereçamento de outros processos, estes buffers têm de ser concretizados através de **zonas de memória partilhada**.

Sincronização de Processos (Fase 2):
De modo a que o acesso às zonas de memória partilhada seja coerente, os processos terão de se **sincronizar e garantir o acesso em exclusão mútua a zonas críticas**. Adicionalmente, os vários processos também necessitam de se coordenar na realização das suas atividades.

Configuração e Resultados (Fase 1):
A execução do programa depende de uma configuração, contida num **ficheiro**, que define o número de processos, o número de empresas e/ou profissionais (i.e., capacidade portuária) e a capacidade dos buffers. Por omissão, os resultados das operações portuárias das diferentes entidades são escritos para a consola, mas podem também ser enviados para ficheiro. Desta forma, a configuração pode ser feita sem necessidade de recompilação da aplicação e os resultados permanecem guardados após a execução do *SO_AdmPor*.

Sinais e Alarmes (Fase 2):
A execução é acompanhada através da atualização de **relógios** que permitem registar o seu estado. Os estados são armazenados em ficheiro para posterior análise. Adicionalmente através de **alarmes** é possível mostrar regularmente no ecrã informação sobre a evolução do sistema, ou seja, que ações estão a decorrer na administração portuária.

Verificação de Resultados (Fase 1):
Finalmente, a verificação dos resultados é efetuada através de um **shell script**, que verifica se os ficheiros de output gerados pelo código fornecido pelo corpo docente e pelo projeto elaborado pelos alunos é o mesmo.

Esta primeira fase do projeto foca-se na organização de processos e alocação de memória e programação Shell, correspondendo aos temas de Criação de Processos, Comunicação entre Processos e Verificação de Resultados acima apresentados. Os temas Sincronização entre Processos e Sinais e Alarmes serão explicados em mais detalhe na segunda fase do projeto (no próximo enunciado), uma vez que serão objeto de estudo daquela fase.

4 Configuração

A execução do *SO_AdmPor* é parametrizada através de um ficheiro de configuração que descreve um determinado cenário. Em cada cenário são descritos precisamente quais as operações disponibilizadas pela administração portuária (e.g., cargas e descargas, depósito, entrega e receção de mercadorias, etc.), o número de clientes e as operações que pretendem realizar, o número de intermediários, o número de empresas e os serviços por estas prestados, e finalmente a capacidade das diversas listas de descrição/pedido/agendamento (buffers). Esta abordagem permite testar facilmente o sistema em diferentes cenários. Um cenário possível é o seguinte:

```
; tamanho máximo de cada linha = 500 chars

[OPERACOES]
; Cada número na linha "CAPACIDADE_PORTUARIA" representa uma operação
; diferente e quantas vezes ela pode ser pedida (independentemente
; de que empresa a serve). No exemplo seguinte existem 6 operações,
; em que a primeira operação (operação 0) pode ser pedida 1 vez, a
; operação 1 pode ser pedida 2 vezes, etc.
; Assim sendo, considera-se que as operações portuárias são identificadas
; sequencialmente por (0 - 9) e o número de operações portuárias é indicado pela
; quantidade (0 - ...). O número máximo de operações é 10.
CAPACIDADE_PORTUARIA = 1 2 3 4 5 6

[CLIENTES]
; Cada cliente escolhe uma (e apenas uma) operação para ser executada.
; O número total de clientes é portanto dado pelo número de elementos na
; lista "OPERACAO". Os clientes não têm identificadores, e a operação
; requisitada por cada um é indicada pelo número (0 - 9).
; No exemplo seguinte, o primeiro cliente pede a operação 0, o segundo
; pede a operação 1, etc.
OPERACAO = 0 1 2 3 4 5

; Adicionalmente, os clientes podem pedir que as suas operações sejam
; executadas N vezes. Para tal, existe o parâmetro "N". No exemplo seguinte,
; cada operação de cada cliente vai ser executada 5 vezes.
N = 5

[INTERMEDIARIOS]
; O número de intermediários é dado pelo número de elementos da lista
; "LIST". Cada intermediário é identificado por (I1, I2, ...). No exemplo
; seguinte existe um intermediário chamado I1.
LIST = I1
```

```
[EMPRESAS]
; O número total de empresas é dado pelo número de elementos na lista
; "OPERACOES". As operações prestadas por cada empresa são indicadas
; pelos números (0 - 9) e são separadas por virgulas. No exemplo
; seguinte, existem 2 empresas que prestam todas as operações
; de 0 a 5.
OPERACOES = 0 1 2 3 4 5, 0 1 2 3 4 5

[BUFFERS]
; Existem 3 buffers no sistema, um para guardar descrições de operações,
; um para guardar pedidos de operações, e um para guardar agendamentos.
; Cada número da linha "CAPACIDADE_BUFFER" representa a capacidade máxima
; de cada um destes buffers, respetivamente. Os valores da capacidade podem
; ser alterados, mas o número de buffers não. No exemplo seguinte, cada
; um dos três buffers tem capacidade 10.
CAPACIDADE_BUFFER = 10 10 10
```

5 Estrutura do projeto

Os ficheiros do projeto podem ser descarregados da página da disciplina. Depois de descomprimido o arquivo, serão encontrados os seguintes diretórios:

```
\SO_AdmPor
      \bin
      \include
      \logPlayer
      \obj
      \src
      \tests
            \in
            \log
            \out
```

O diretório bin contém o executável `soadmpor`. O diretório `include` contém os ficheiros `.h` com a definição das estruturas de dados e declarações de funções. Estes ficheiros não podem ser alterados. O diretório `logPlayer` contém o ficheiro fonte do depurador, `logPlayer`, que será apresentado no enunciado da segunda fase do projecto. O diretório `obj` contém os ficheiros objeto. O diretório `src` contém os ficheiros fonte que deverão ser alterados para realização do projeto. Finalmente, o diretório `testes` inclui os vários ficheiros de configuração (em `in`) da SO_AdmPor, e é também onde devem ser guardados os ficheiros gerados (`out` e `log`). O executável `soadmpor` será gerado a partir da execução do comando `make`. O `makefile` necessário para a execução do comando `make` deve ser colocado no diretório SO_AdmPor.

6 Ficheiros de output

Quando o SO_AdmPor é executado com um determinado ficheiro de configuração, é gerado um ficheiro de output. Os ficheiros de output do SO_AdmPor deverão ser colocados no diretório SO_AdmPor/tests/out. Estes ficheiros vão ter um aspeto que vai depender do cenário/configuração inicial, mas serão semelhantes ao seguinte:

```
CLIENTE 000 solicitou 0 e obteve 0
CLIENTE 001 solicitou 1 e obteve 1
CLIENTE 002 solicitou 2 e obteve 2
CLIENTE 003 solicitou 3 e obteve 3
```

```
CLIENTE 004 solicitou 4 e obteve 4
CLIENTE 005 solicitou 5 e obteve 5
Clientes: 06
Intermediarios: 01
Empresas: 01
Clientes servidos por intermediarios:     06
Clientes servidos por empresas:           06
Total agendamentos:                       01 01 01 01 02 00 = 06
Capacidade inicial portuária:       01 02 03 04 05 06 = 21
Capacidade final portuária:         00 01 02 03 04 05 = 15
```

Estes ficheiros sumarizam a execução do cenário/configuração dado como input e mostram estatísticas importantes e acontecimentos que decorreram ao longo da execução do programa.

7 Processos

Cada processo criado (para além do processo inicial) representará uma entidade que poderá ter um dos papéis referidos: cliente, intermediário ou empresa. Podem existir vários processos a desempenhar o mesmo papel.

O SO_AdmPor encerra quando todos os clientes forem atendidos. Nessa situação todos os processos devem terminar e, de seguida, o processo *main* pode indicar a conclusão da operação no SO_AdmPor.

De seguida, são especificadas as ações de cada tipo de processo:

7.1 Main

O processo main (Figura 1) prepara o ambiente de comunicação e sincronização, abre o SO_AdmPor, lança os processos que representam as várias entidades e fica a aguardar pela sua conclusão. No final fecha o SO_AdmPor, apresenta indicadores do funcionamento do sistema (statistics), liberta os recursos reservados para comunicação e sincronização e termina. Este processo realiza as instruções contidas no ficheiro main.c.

7.2 Cliente

Um processo cliente começa por aguardar a abertura do SO_AdmPor e solicita a operação que está indicada no ficheiro de configuração. Se a operação portuária estiver disponível então aguarda pelo agendamento e posterior execução. Após a conclusão, o cliente não abandona o SO_AdmPor, esperando para solicitar outra operação. O cliente poderá solicitar até N operações, onde N está especificado no ficheiro de configuração. Caso contrário, abandona imediatamente o SO_AdmPor.

7.3 Intermediário

Um processo intermediário começa por aguardar a abertura do SO_AdmPor e, enquanto existirem clientes, aguarda por descrições de operações de um cliente. Ao chegar uma descrição de operação, se existir capacidade portuária, então coloca um pedido de operação a uma empresa. Caso contrário, informa ao cliente de que as operações não estão disponíveis (i.e., não existem prestadores de serviço). O cliente deverá enviar uma nova descrição ao intermediário.

7.4 Empresas

Um processo empresa começa por aguardar a abertura do SO_AdmPor e, enquanto existirem clientes, aguarda um pedido de um cliente. Ao chegar um pedido de operação que esta possa tratar, prepara o agendamento e, após isso envia-os ao cliente respetivo.

8 Estruturas de dados

De modo a permitir a comunicação entre os vários processos estão estabelecidas várias estruturas de dados que agregam toda a informação. Estas estruturas de dados serão apresentadas em cada fase do projeto.

Nesta fase são introduzidas as seguintes estruturas de dados:

- `operation`
- `configuration`
- `statistics`
- `pointers`
- `request_d`
- `request_r`
- `response_s`

A estrutura `operation` representa: uma solicitação a um intermediário, um pedido à empresa e também um agendamento. Esta estrutura é criada pelo processo cliente como uma descrição de operação e passada ao intermediário. O intermediário recebe a estrutura e no caso de existência de capacidade portuária atualiza-a e passa-a à empresa. A empresa recebe esta estrutura, elabora o agendamento, atualiza a estrutura e entrega-a ao cliente respetivo.

A estrutura `configuration` contém os dados lidos do ficheiro de configuração.

A estrutura `statistics` contém os dados que permitem no final da execução aferir a concretização dos objetivos. A capacidade portuária é aqui guardada de modo a se poder comparar no final com a capacidade remanescente que está na estrutura anteriormente referida. Esta estrutura é composta pelos seguintes campos:

Campo	Contém	Tamanho do vetor
capacidade_inicial_portuaria	Capacidade inicial dos prestadores de operações portuários	Nº de prestadores de serviços
pid_clientes	PID por cliente	Nº de clientes
pid_intermedarios	PID por intermediário	Nº de intermediários
pid_empresas	PID por empresas	Nº de empresas
clientes_servidos_por_intermediarios	Nº de clientes atendidos por intermediário	Nº de intermediários
clientes_servidos_por_empresas	Nº de clientes atendidos por empresas	Nº de empresas
agendamentos_entregues_por_empresas	Matriz [Operação,Empresa] com agendamentos recebidos	Nº de operações × Nº de empresas
serviços_recebidos_por_clientes	Nº de agendamentos recebidas pelos clientes	Nº de operações

A estrutura `pointers` dá suporte à utilização de um buffer circular. Para isso oferece dois índices: in e out. o índice in indica a próxima posição do buffer a ser escrita por uma entidade do sistema (e.g., um cliente). O índice out indica a próxima posição do buffer a ser lida por uma entidade do sistema (e.g., um intermediário). Por exemplo, um cliente envia uma descrição de operação (escreve no buffer no índice in), enquanto um intermediário recebe uma descrição de operação (lê do buffer no índice out).

A estrutura `request_d` contém um ponteiro para um buffer de descrições, onde cada slot pode conter uma estrutura `operation`, e um ponteiro para um vetor de inteiros que indicam quais as posições do buffer que

estão livres (valor é 0) ou ocupadas (valor é 1). Esta estrutura serve para que os clientes encaminhem as suas descrições de operações para os intermediários.

A estrutura `request_r` contém um ponteiro para um buffer de pedidos, onde cada slot pode conter uma estrutura `operation`, e um ponteiro para uma estrutura `pointers`, com os índices das próximas posições a serem lidas e escritas. Esta estrutura serve para que os intermediários encaminhem os pedidos de operações para as empresas.

A estrutura `response_s` contém um ponteiro para um buffer de agendamentos, onde cada slot pode conter uma estrutura `operation`, e um ponteiro para uma estrutura `pointers`, com os índices das próximas posições a serem lidas e escritas. Esta estrutura serve para que as empresas façam chegar os agendamentos aos clientes.

De realçar que cada slot das estruturas `request_d`, `request_r` e `response_s`, contém sempre uma estrutura `operation`. Inicialmente, o cliente cria uma estrutura `operation` e preenche alguns campos. Os restantes campos da estrutura devem ir sendo preenchidos nas etapas seguintes por um dos intermediários ou empresas. Mais informação sobre estas estruturas de dados pode ser consultada nos ficheiros .h do projeto.

NOTA: O conteúdo dos ficheiros .h não pode ser alterado.

9 Objetivos

Esta primeira fase do projeto tem como objetivo a familiarização dos alunos com o projeto, criação do ficheiro `makefile` que permitirá a compilação automática do mesmo e a escrita de código nos ficheiros main.c e memory.c. Para tal, devem explorar o código fornecido e testar diferentes ficheiros de configuração como input. Devem identificar para cada alocação de memória se esta deve ser do tipo dinâmica (só acessível ao processo corrente) ou partilhada (acessível a outros processos).

Todas as zonas de desenvolvimento estão rodeadas com comentários do tipo "`// ==================`". Por exemplo, a instrução `so_main_wait_clientes()` deve ser comentada para que os alunos possam implementar o código que a permite à main esperar pela terminação dos clientes e atualizar estatísticas.

9.1 Ficheiro `makefile`

O ficheiro `makefile` deve conter pelo menos:

1. Regras de compilação para os ficheiros objeto;
2. Regra de ligação para o executável `soadmpor`, incluir bibliotecas `-lrt` e `-lpthread`
3. Regras de limpeza de projeto (`clean`) que devem limpar a diretoria de ficheiros objeto e o executável `soadmpor` (devem ter cuidado para não eliminar o ficheiro `so.o` fornecido pelos docentes);
4. Regras de teste.

9.2 Ficheiro `script.sh`

Para o ponto 4 do `makefile` (Regras de teste), deve ser escrito um *shell script* (script.sh) que ordena e compara os ficheiros de output gerados pelo código fornecido (com o projecto) com os ficheiros de output gerados pelo código elaborado pelos alunos, escrevendo no terminal "EQUAL" caso os ficheiros sejam iguais ou "DIFFERENT" caso contrário e indicando para este último o número de diferenças encontradas. *O shell script deve receber os nomes dos ficheiros a testar como parâmetros*. Como nesta fase apenas existe o `soadmpor` padrão, então é preciso "criar" outros ficheiros de output para teste do *shell script*. Estes podem ser criados partindo de um ficheiro de output gerado pelo `soadmpor` padrão, efetuando as seguintes alterações:

1. Troca da ordem das linhas do ficheiro (o resultado do teste deverá ser "EQUAL")

2. Alteração de valores no ficheiro (o resultado do teste deverá ser “DIFFERENT”)

9.3 Ficheiro main.c

Esta fase envolve também a escrita de código no ficheiro main.c. Os objetivos específicos para este ficheiro são os seguintes:

1. Adicionar o código que permita criar todos os processos filho (clientes, intermediários e empresas).
2. Adicionar o código que permita ao processo *main*:
   - aguardar a terminação dos filhos do tipo cliente, intermediário e emresa. Deverão ter em atenção e em conta as N operações que os filhos clientes podem realizar (ou seja, que o filho do tipo cliente após concluir uma operação, espera e poderá voltar a solicitar uma nova operação);
   - registar o estado que cada filho devolveu, através da chamada de sistema exit, na estrutura de dados statistics.

9.4 Ficheiro memory.c

Esta fase envolve a escrita de código no ficheiro `memory.c`. Os objetivos específicos para este ficheiro são os seguintes:

1. Criação e terminação de um buffer de acesso aleatório (`BDescricao`), como zona de memória partilhada entre processos independentes, para os clientes colocarem os pedidos para os intermediários. O tamanho do buffer deverá estar de acordo com o valor contido em `Config.BUFFER_DESCRICAO` (estrutura `configuration`).
2. Criação e terminação de um buffer circular (`BPedido`), como zona de memória partilhada entre processos independentes, para os intermediários colocarem os pedidos para as empresas. O tamanho do buffer deverá estar de acordo com o valor contido em `Config.BUFFER_PEDIDO` (estrutura `configuration`).
3. Criação e terminação de um buffer circular (`BAgendamento`), como zona de memória partilhada entre processos independentes, para as empresas colocarem os propostas e agendamentos para os clientes. O tamanho do buffer deverá estar de acordo com o valor contido em `Config.BUFFER_AGENDAMENTO` (estrutura `configuration`).
4. Adicionar o código que permite aos clientes, intermediários e empresas escreverem e lerem a informação dos buffers partilhados. Funções a alterar: `memory_request_d_write`, `memory_request_d_read`, `memory_request_r_write`, `memory_request_r_read`, `memory_response_s_write` e `memory_response_s_read`.
5. Adicionar o código que permita atualizar os campos da estrutura `operation` nas etapas adequadas, p. ex.: na função `memory_request_d_write` o intermediário deve colocar o seu `id` no campo `intermediario`.
6. Na função `memory_create_statistics`, reservar e libertar memória dinamicamente para cada um dos campos da estrutura indicadores com os tamanhos necessários (consultar estrutura `configuration`). Iniciar o vetor `capacidade_inicial_portuaria` com os valores contidos em `Config.capacidade_portuaria` e apagar o conteúdo dos vetores `clientes_servidos_por_intermediarios`, `clientes_servidos_por_empresas`, `agendamentos_entregues_por_empresas` e `agendamentos_recebidos_por_clientes`.

O nome a atribuir a cada zona de memória partilhada não está estabelecido, devendo ser escolhido pelo grupo utilizando uma nomenclatura consistente.

Um objetivo específico é a escolha correta do tipo de alocação de memória (dinâmica ou partilhada) para cada variável. Um objetivo geral é a correta reserva e libertação de memória. Este objetivo implica também a reserva do espaço de memória estritamente necessário.

Outro objetivo fundamental é que não existam fugas de memória. Para identificar essa situação deve ser utilizado a ferramenta Valgrind.

9.5 Tratamento de erros

Um objetivo geral que é comum a ambas as fases do projeto é que o resultado de cada chamada ao sistema deve ser tratado. Nomeadamente, devem ser verificadas e tratadas todas as situações de erro.

10 Teste

Os métodos de teste passam pela análise do ficheiro de configuração, dos ficheiros gerados (resultados e log) e nalguns casos também as saídas para a consola. A concretização de cada um dos objetivos desta fase pode ser verificado com o método seguinte:

1. Verificar que o SO_AdmPor reage à introdução de diferentes combinações de parâmetros.
2. Para um número significativo de clientes, verificar que todos os clientes foram atendidos e que a ordem de intermediários não será sequencial.
3. Verificar que todos os clientes foram atendidos antes do processo *main* do SO_AdmPor encerrar e que os indicadores (statistics) mostram informação válida.

O ficheiro de log permitirá aferir especificamente o funcionamento de cada buffer. Finalmente, os indicadores estatísticos deverão produzir informação válida.

11 Entrega

Os ficheiros `makefile`, `script.sh`, `main.c`, `memory.c` devem ser entregues até às 23h59 do dia 10/04/2020, utilizando o link de entrega disponível na página de SO no moodle. Deve-se criar um ficheiro zip com o nome **SO-0XX-proj1.zip**, onde **XX** é o número do grupo, contendo os ficheiros listados acima e um ficheiro `Readme.txt` com as limitações do trabalho. Apenas um dos elementos do grupo deve submeter o trabalho.

Não serão aceites trabalhos por e-mail, nem por qualquer outro meio não definido nesta secção. Se não se verificar algum destes requisitos o trabalho é considerado não entregue.