Sistemas Operativos 2019/2020

Trabalho Prático
Aeroporto

1. Objetivos do trabalho

- Desenvolver um simulador de uma *Torre de Controlo* de um aeroporto. Servirá para controlar a descolagem e aterragem de voos num aeroporto com 4 pistas.
- Explorar mecanismos de gestão de processos, *threads*, comunicação e sincronização em Linux.

2. Contextualização

O sistema a ser desenvolvido deve simular um ambiente onde existem uma *Torre de Controlo*, duas pistas de aterragem, duas pistas para descolagem e voos a aterrar e a descolar. A torre tem uma posição fixa e controla os voos que se dirigem para aterrar no aeroporto e aqueles que aguardam uma oportunidade para descolar. A Figura 1 representa uma visualização do aeroporto, cujo problema subjacente é descrito abaixo.

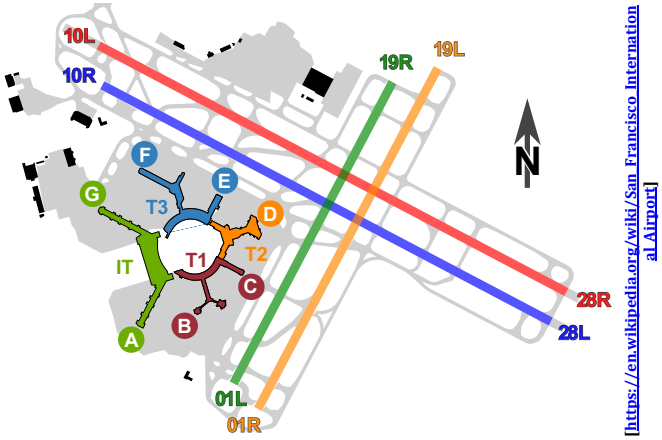

Figure 1 – Localização das pistas do aeroporto e *Torre de Controlo* (T2).

As pistas 01L e 01R são utilizadas para decolagens enquanto que as pistas 28L e 28R são usadas para aterragens. Devido às particularidades do aeroporto em questão, aterragens e descolagens não podem acontecer em simultâneo, enquanto que podem acontecer até 2 aterragens ou 2 descolagens em simultâneo. Na mesma pista, aterragens e descolagens têm de respeitar um tempo mínimo entre voos. Desta forma, é necessária uma *Torre de Controlo* particularmente eficiente capaz de garantir um alto *throughput* ao mesmo tempo que evita *starvation* dos voos.
Os voos chegam ao sistema com o objetivo de descolar ou aterrar, existindo uma fila de espera para cada um dos casos. Cada voo que deseja descolar é caracterizado pela sua hora desejada de partida. Cada voo com o objetivo de aterrar é caracterizado pelo tempo que demora até à pista (*Estimated Time of*

Arrival - ETA) e pelo seu combustível disponível. Os aviões podem aterrar até ao seu combustível chegar a `0`. A partir desse momento apenas têm combustível de reserva e serão automaticamente desviados para outro aeroporto. O tempo é dado em *time units* e o combustível em *fuel units*, sendo que os aviões gastam uma unidade de combustível por cada unidade de tempo em voo.
A *Torre de Controlo* coordena a operação dos voos de acordo com as regras apresentadas na secção 4, com o objetivo de maximizar o número de operações e minimizar os tempos de espera. Quando um voo se aproxima do aeroporto, a torre deve decidir se há condições para ele aterrar, caso contrário deve ordenar ao voo que circule no ar enquanto espera (*holding*). No entanto, há que respeitar o combustível disponível, sendo que apenas na extrema impossibilidade de o voo aterrar este deve ser desviado para um outro aeroporto (ver 4.2).
Durante o processo de descolagem ou aterragem, a pista em uso fica indisponível.

3. Visão geral do funcionamento da aplicação

A Figura 2 apresenta uma visão geral do funcionamento do sistema a implementar no contexto do Trabalho Prático.

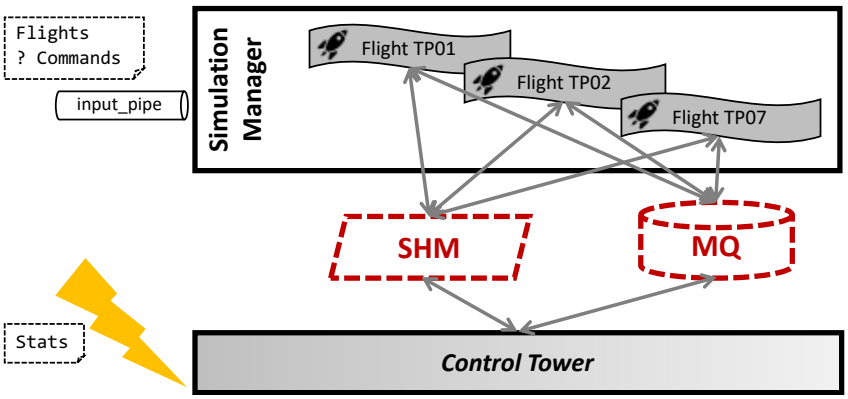

Figure 2 – Visão geral do sistema a implementar.

Tal como é representado na figura, o sistema é baseado em vários processos e *threads*, entre os quais:

- **Gestor da Simulação (*Simulation Manager*)** – é responsável por iniciar todo o sistema. Deve criar a fila de mensagens (MQ), as zonas de memória partilhada (*SHM*) e o processo *Torre de Controlo*. Deve também criar o *named pipe* ("*input_pipe*" da Fig. 2) onde vai receber os voos que entram no sistema, criando uma *thread* Voo para representar cada um.
- **Torre de Controlo** (***Control Tower***) – deve coordenar as operações de descolagem e aterragem, mantendo filas para as *threads* voo e guardando estatísticas. Ao receber um sinal SIGUSR1 deve ler as estatísticas a partir da memória partilhada e apresentá-las ao utilizador.
- **Voos (*Flights*)** – cada voo é representado por uma *thread* individual, criada pelo processo *Gestor da simulação*. Cada uma aterra, descola ou é desviada para um outro aeroporto de acordo com as instruções da *Torre de Controlo*.

De seguida explicam-se em maior detalhe as funcionalidades a implementar.

4. Descrição das funcionalidades a implementar

4.1. Introdução dos voos no sistema

É responsabilidade do processo *Gestor da simulação* iniciar as *threads* que simulam cada voo. Assim, deve monitorizar o *named pipe* ("*input_pipe*" Fig. 2) para receber a informação dos voos, que são expressas na forma de comandos que podem significar um novo voo a chegar ou a partir. **A estrutura dos comandos a receber é apresentada abaixo e deve ser seguida rigorosamente.**

```
# estrutura do comando para adicionar uma partida
# DEPARTURE {flight_code} init: {initial time} takeoff: {takeoff time}

DEPARTURE TP440 init:  0 takeoff: 100
DEPARTURE TP89  init: 10 takeoff: 100
DEPARTURE TP73  init: 10 takeoff: 100

# estrutura do comando para adicionar uma chegada
# ARRIVAL {flight_code} init: {initial time}  eta: {time to runway}
#      fuel: {initial fuel}

ARRIVAL TP437   init:  0  eta: 100 fuel: 1000
ARRIVAL TP88    init: 10  eta: 200 fuel: 1500
ARRIVAL TP70    init: 10  eta: 100 fuel: 1500

# Todos os tempos são dados em unidades de tempo (ut); os tempos de início
# e de descolagem são contados em ut desde o início da simulação; o initial
# time é o instante em que deve ser criada a thread que simula o voo.
```

Como podemos observar, um voo de partida tem um nome, o instante inicial em que fica pronto para entrar na fila e o instante desejado de descolagem. Por outro lado, um voo de chegada tem um nome, o instante inicial em que deve entrar no radar do aeroporto, o tempo até à pista (ETA) e o combustível no instante inicial. O tempo é medido desde o arranque da simulação e deve ser em unidades de tempo, de acordo com as configurações definidas na subseção 4.6.

Quando o processo *Gestor da simulação* recebe um comando, deve validar a sua sintaxe e correção (e.g. verificar se os instantes incluídos são posteriores ao atual e se fazem sentido - por exemplo, o combustível deve dar para voar até ao ETA). Os comandos inválidos devem ser registados no ficheiro de *log* (ver subseção 4.5) como erróneos e esses voos descartados. Caso os dados sejam válidos deverá escalonar o início da respetiva *thread* para o instante *init*. O escalonamento poderá ser feito usando uma estratégia à sua escolha, mas é importante que a *thread* não seja criada imediatamente para evitar a sobrecarga do sistema nos casos em que muitos voos são enviados em *batch* pelo *pipe*.

4.2. Funcionamento dos voos de partida e chegada

Cada voo é representado por uma *thread* a correr dentro do processo *Gestor Simulação* e deve arrancar apenas no instante *init* correspondente. A *thread* deve

a partir daí assumir as responsabilidades pelo voo, até conseguir completar a sua operação de descolagem/aterragem, ou ser reencaminhada para outro aeroporto.

Os voos de partida, ao serem criados, devem enviar uma mensagem à *Torre de Controlo*, através da MSQ a dizer qual é o seu instante de partida desejado. Os voos de chegada devem igualmente notificar a *Torre de Controlo* através da fila de mensagens, mas neste caso do seu tempo até à pista (ETA) e do seu combustível.

Em ambos os casos, a *Torre de Controlo* responderá pela mesma MSQ com o número do *slot* em memória partilhada atribuído a esse voo específico. Será através desse *slot* que as *threads* receberão a indicação de descolagem/aterragem, bem como das informações de necessidade de uma manobra de *holding* ou de desvio para outro aeroporto. Caso qualquer voo receba como resposta da *Torre de Controlo* que o pedido de descolagem/aterragem foi rejeitado, a *thread* deverá terminar.

Caso um voo de chegada apenas tenha combustível para voar até ao instante '`4 + ETA + duração da aterragem`', a mensagem a enviar à *Torre de Controlo* através da fila de mensagens deve ser prioritária e lida antes das outras. A *Torre de Controlo* deverá marcar este voo como urgente e evitar que ele tenha de efetuar manobras de *holding*. A *thread* deverá escrever no *log* o instante em que enviou a mensagem de emergência (ver exemplo em 4.5).

As *threads* correspondentes às partidas recebem a indicação de descolagem pelo seu *slot* em memória partilhada. Assim que receber a ordem para descolar, a *thread* deve tirar partido dos mecanismos de sincronização para aguardar, sem espera ativa, que uma das pistas esteja vazia, que o aeroporto possa receber descolagens e que nenhum voo esteja a aterrar. Poderá então começar o processo de descolagem, que demora ***T*** unidades de tempo (ver 4.6).

As *threads* correspondentes às chegadas recebem as instruções da *Torre de Controlo* através da memória partilhada - aterragem, *holding*, desvio para outro aeroporto. Caso a *Torre de Controlo* verifique que é necessário adiar a aterragem, o voo será ordenado a realizar uma manobra de *holding* que irá alterar o ETA. Os voos de chegada demoram ***L*** unidades de tempo a aterrar (ver 4.6).

Cada *thread* deverá escrever no *log* (ver 4.5) o tempo em que foi iniciada e em que terminou. Qualquer evento que leve a que uma *thread* termine também deverá ser registado no *log* e nas estatísticas (exs: rejeição pela *Torre de Controlo*, chegada do combustível a `0`).

4.3. Funcionamento da *Torre de Controlo* e notificação dos voos

A *Torre de Controlo* é um processo criado pelo *Gestor Simulação* e que deve incluir o menor número de *threads* que achar adequadas para o cumprimento das suas funções. As responsabilidades da *Torre de Controlo* são as seguintes:

Acolher a chegada de novos voos. Cada *thread* que simula um voo, ao ser criada, envia pela MSQ uma mensagem à *Torre de Controlo*. A *Torre de Controlo* monitoriza a fila de mensagens por onde recebe informações de novos voos de partida ou chegada a entrar no sistema. Caso o número de voos já esteja no máximo permitido (definido na subseção 4.6.), o voo deve ser imediatamente rejeitado, sendo que este evento deve ser reportado nas estatísticas e *log*.

Os voos de partida devem enviar uma mensagem a dizer qual é o seu instante de partida desejado. A torre deve guardar a informação recebida e responder pela mesma MSQ com o número do *slot* no espaço de memória partilhada que é atribuído a esse voo específico. Deve também colocar o voo na fila de espera para partidas, de acordo com o instante desejado de partida.

Os voos de chegada devem igualmente notificar a *Torre de Controlo* através da fila de mensagens, mas neste caso do seu tempo até à pista (ETA) e do seu combustível. A *Torre de Controlo* deve guardar a informação recebida e responder pela mesma MSQ com o número do *slot* no espaço de memória partilhada que é atribuído a esse voo específico. Deve também colocar o voo na fila de espera para aterragens, de acordo com o ETA.

A *Torre de Controlo* pode receber mensagens de emergência enviadas pelos voos a pedir para que esse mesmo voo seja considerado prioritário (explicado na subseção 4.2). Essas mensagens devem ser lidas primeiro que as restantes. Baseado nestas mensagens, a *Torre de Controlo* deve dar prioridade a esse voo, fazendo com que aterre o mais cedo possível evitando que ele tenha de efetuar manobras de *holding*.

Atualização de dados dos voos. A *Torre de Controlo* deve manter atualizados os valores de combustível de cada avião, sabendo que por cada unidade de tempo em voo os aviões gastam uma unidade de combustível. Quando o combustível do voo chega a `0` quer dizer que acabou o combustível principal, sendo que ainda lhe resta o combustível de reserva. Nesse caso o voo deve ser informado pela memória partilhada que deverá ser imediatamente redirecionado para outro aeroporto, saindo do sistema e libertando os recursos associados. Esta é uma situação que deve acontecer apenas excecionalmente, pelo que deve pensar os seus algoritmos de escalonamento de forma a minimizá-la (ver subseção 4.6).

Manter ordenadas as filas de aterragem e descolagens. A *Torre de Controlo* deve gerir a ordem das operações de descolagem e aterragem. Essa informação é passada aos voos através da memória partilhada. Como há duas pistas (01L e 01R) para descolagens e mais duas pistas (28L e 28R) para aterragens, a *Torre de Controlo* pode sempre designar até dois aviões no topo de cada uma das filas, de forma a que estes possam usar ambas as pistas em paralelo, assim que cada uma delas fique disponível. No caso das aterragens, sempre que o voo tiver mais que 5 outros voos à sua frente na fila, deve ser ordenado a realizar uma manobra de *holding*, sendo as filas devidamente atualizadas. A manobra de holding irá adicionar uma duração entre *min* e *max* (das configurações na subseção 4.6) ao

ETA do voo. O voo deve receber esta informação através da sua zona de memória partilhada. Sempre que a manobra de *holding* levar a que o combustível do voo chegue a `0`, o voo deve ser redirecionado para outro aeroporto, desaparecendo do sistema, sendo que este facto deve ser registado nas estatísticas e *log*. Sempre que uma operação termina, a *Torre de Controlo* tem de atualizar imediatamente as filas e selecionar os próximos voos a aterrar/levantar – os 2 primeiros de cada fila.

Alternar entre aterragens e descolagens. A *Torre de Controlo* deve decidir quando alternar entre permitir aterragens e permitir descolagens. Deve escolher o algoritmo de escalonamento que preferir de modo a otimizar o tráfego no aeroporto. Isto será observável nas estatísticas.

Apresentar as estatísticas. Após receber o SIGUSR1, este processo deve ler as estatísticas da memória partilhada e apresentá-las tanto na consola. Isto deve ser feito sem prejuízo das suas restantes possibilidades, e este deve ser o único processo a realizar esta tarefa.

4.4. Informação sobre estatísticas

Pretendem-se manter estatísticas relativas ao funcionamento do sistema. Estas estatísticas devem ser mantidas na memória partilhada (SHM), de forma a que possam ser atualizadas tanto pelo processo *Torre de Controlo*, como pelas *threads* Voo (Fig. 2). Após a receção de um sinal do tipo SIGUSR1, o processo *Torre de Controlo* deverá escrever para o ecrã a seguinte informação estatística:

- Número total de voos criados
- Número total de voos que aterraram
- Tempo médio de espera (para além do ETA) para aterrar
- Número total de voos que descolaram
- Tempo médio de espera para descolar
- Número médio de manobras de holding por voo de aterragem
- Número médio de manobras de holding por voo em estado de urgência
- Número de voos redirecionados para outro aeroporto
- Voos rejeitados pela *Torre de Controlo*

4.5. Log da aplicação

Todo o *output* da aplicação deve ser escrito de forma legível num ficheiro de texto "log.txt". Cada escrita neste ficheiro deve ser precedida pela escrita da mesma informação na consola, de modo a poder ser facilmente visualizada enquanto decorre a simulação.
Deverá pôr no *log* os seguintes eventos acompanhados da sua data e hora:

- Inicio e fim do programa;
- Inicio e fim de cada *thread* voo;
- Atribuição de uma manobra de *holding* a um voo (incluindo o *holding* time atribuído, o nome do avião, a distância e o combustível)
- Ordem de aterragem e descolagem (incluir o nome do avião e pista atribuída)
- Leitura de um comando do *pipe*

- Erros nos comandos recebidos pelo *pipe*
- Mensagem de emergência de um avião
- Desvios de aviões para outro aeroporto

Exemplo do ficheiro de *log*:

```
18:00:05 NEW COMMAND => ARRIVAL TP437 init: 100  eta: 100 fuel: 1000
18:00:10 WRONG COMMAND => ARRIVAL TP437 init: AAA  eta: 100 fuel: 1000
18:00:23 TP89 DEPARTURE 1R started
18:00:25 TP70 HOLDING 100
18:00:26 TP438 EMERGENCY LANDING REQUESTED
18:00:27 TP89 DEPARTURE 1R concluded
18:00:27 TP438 LANDING 28L started
18:00:31 TP438 LANDING 28L concluded
18:00:32 TP77 LEAVING TO OTHER AIRPORT => FUEL = 0
```

4.6. Arranque e terminação do sistema

No seu arranque, o processo *Gestor da Simulação* deverá ler a configuração inicial do ficheiro "config.txt" que deverá conter os seguintes dados:

```
unidade de tempo (ut, em milissegundos)
duração da descolagem (T, em ut), intervalo entre descolagem (dt, em ut)
duração da aterragem (L, em ut), intervalo entre aterragens (dl, em ut)
holding duração mínima (min, em ut), máxima (max, em ut)
quantidade máxima de partidas no sistema (D)
quantidade máxima de chegadas no sistema (A)
```

Nota: o intervalo entre descolagens/aterragens é contado a partir do momento em que a operação é completa.

Exemplo de um ficheiro de configuração:

```
500
30, 5
20, 10
75, 100
100
1000
```

De seguida, deverá criar todos os restantes recursos de comunicação e sincronização necessários, e o processo *Torre de Controlo*. O PID dos processos criados deverá ser escrito no ecrã e no ficheiro de log.

O sistema deverá estar preparado para terminar após a receção, por parte do processo *Gestor da Simulação*, de um sinal do tipo SIGINT. Nessa altura, o *Gestor da Simulação* deverá deixar de receber novos pedidos escrevendo para o *log* os comandos restantes no *pipe*, sem lhes dar seguimento. De seguida deve aguardar a terminação de todos os pedidos pendentes, e só depois de todos as aterragens e partidas serem efetuadas é que deve terminar todos os outros processos e fazer a limpeza de todos os recursos partilhados.

Os sinais que não são mencionados no enunciado devem ser tratados de forma a que não produzam efeitos nocivos para o funcionamento da aplicação.

5. Checklist

Esta lista serve apenas como indicadora das tarefas a realizar e assinala as componentes que serão objeto de avaliação na defesa intermédia.

Processo	Tarefa	Avaliado na defesa intermédia?
Gestor da simulação	Arranque do servidor e aplicação das configurações existentes no ficheiro "config.txt"	S
	Criação do processo *Torre de Controlo*	S
	Criação do *named pipe*	S
	Leitura e validação dos comandos pelo *named pipe*	S
	Criação das *threads* voo no instante correto	S
	Criação da fila de mensagens	S
	Criação da memória partilhada	S
Torre de Controlo	Criação das filas para gestão de partidas e chegadas	
	Gestão de partidas e chegadas	
	Leitura da MSQ e gestão de voos prioritários	
	Gestão da necessidade de manobras de *holding*	
	Envio de instruções para os aviões através da SHM	
	Descartar voos que excedem os limites	
	Escrever a informação estatística no ecrã como resposta ao sinal SIGUSR1	
Threads Voo	Troca de mensagens com a *Torre de Controlo*	
	Ler indicações da *Torre de Controlo* através da SHM	
	Descolagem de voos com uso exclusivo da pista e respeitando o intervalo de segurança	
	Aterragem de voos com uso exclusivo da pista e respeitando o intervalo de segurança	
Ficheiro de *log*	Envio sincronizado do output para ficheiro de *log* e ecrã.	S
Geral	Diagrama com a arquitetura e mecanismos de sincronização	S (preliminar)
	Suporte de concorrência no tratamento de pedidos	
	Deteção e tratamento de erros.	
	Sincronização com mecanismos adequados (semáforos, *mutexes* ou variáveis de condição)	S
	Prevenção de interrupções indesejadas por sinais e fornecer a resposta adequada aos vários sinais	
	Após receção de SIGINT, terminação controlada de todos os processos e *threads,* e libertação de todos os recursos.	

6. Notas importantes

- **Não será tolerado plágio, cópia de partes de código entre grupos ou qualquer outro tipo de fraude**. Tentativas neste sentido resultarão na **classificação de ZERO valores** e na consequente **reprovação na cadeira**. Dependendo da gravidade poderão ainda levar a processos disciplinares.
- Todos os trabalhos serão escrutinados para deteção de cópias de código.
- Para evitar cópias, os alunos não podem colocar código em repositórios de acesso público.

- Leia atentamente este enunciado e esclareça dúvidas com os docentes.
- Em vez de começar a programar de imediato pense com tempo no problema e estruture adequadamente a sua solução. Soluções mais eficientes e que usem menos recursos serão valorizadas.
- Inclua na sua solução o código necessário à deteção e correção de erros.
- Evite esperas ativas no código, sincronize o acesso aos dados sempre que necessário e assegure a terminação limpa do servidor, ou seja, com todos os recursos utilizados a serem removidos.
  - Esperas ativas serão fortemente penalizadas!
  - Acessos concorrentes que, por não serem sincronizados, puderem levar à corrupção de dados, serão fortemente penalizados!
- Inclua informação de *debug* que facilite o acompanhamento da execução do programa, utilizando por exemplo a seguinte abordagem:

```
#define DEBUG //remove this line to remove debug messages
(...)
#ifdef DEBUG
printf("Creating shared memory\n");
#endif
```

- Todos os trabalhos deverão funcionar na VM fornecida ou, em alternativa, na máquina student2.dei.uc.pt.
  - Compilação: o programa deverá compilar sem erros em qualquer uma das metas; evite também os *warnings*, exceto quando tiver uma boa justificação para a sua ocorrência.
  - A não compilação do código enviado implica uma classificação de **ZERO valores** na meta correspondente.
- A defesa final do trabalho é obrigatória para todos os elementos do grupo. A não comparência na defesa final implica a classificação de **ZERO valores** no trabalho.
- O trabalho pode ser realizado em grupos de até 2 alunos (grupos com apenas 1 aluno devem ser evitados e grupos com mais de 2 alunos não são permitidos).
- A nota da defesa é individual pelo que cada um dos elementos do grupo poderá ter uma nota diferente;
- Os alunos do grupo devem pertencer a turmas PL do mesmo docente. Grupos com alunos de turmas de docentes diferentes carecem de aprovação prévia.
- As defesas intermédia e final devem ser realizadas na mesma turma e com o mesmo docente.

7. Metas, entregas e datas

Data	**Meta**	
Data de entrega no Inforestudante 11/11/2019	Entrega intermédia	• Crie um arquivo no formato **ZIP** (**NÃO SERÃO ACEITES OUTROS FORMATOS**) com todos os ficheiros do trabalho e submeta-o no Inforestudante. o Não inclua quaisquer ficheiros não necessários para a compilação ou execução do programa (ex. diretórios ou ficheiros de sistemas de controlo de versões)
Aulas PL da semana de 11/11/2019	Demonstração /defesa intermédia	• A demonstração deverá contemplar todos os pontos referidos na *checklist* que consta deste enunciado. • Durante a defesa deverá ser apresentada **1 página A4 com a arquitetura e todos os mecanismos de sincronização a implementar descritos**. • A demonstração/defesa será realizada nas aulas PL • A defesa intermédia vale **20%** da cotação do projeto.
Data de entrega no Inforestudante Sem penalização: 08/12/2019-22h00 Com penalização de 10%: 09/12/2018-23h59 (Não serão aceites submissões após esta data.)	Entrega final	O projeto final deverá ser submetido no Inforestudante, tendo em conta o seguinte: ● Os **nomes** e **números** dos alunos do grupo devem ser colocados no início dos ficheiros com o código fonte. ● Com o código deve ser entregue um **relatório** sucinto (no máximo 2 páginas A4), no formato **pdf (NÃO SERÃO ACEITES OUTROS FORMATOS)**, que explique as opções tomadas na construção da solução. Inclua um esquema da arquitetura do seu programa. Inclua também informação sobre o tempo total despendido (por cada um dos dois elementos do grupo) no projeto. ● Crie um arquivo no formato **ZIP** (**NÃO SERÃO ACEITES OUTROS FORMATOS**) com todos os ficheiros do trabalho: o Todos os ficheiros fonte e de configuração necessários o Makefile para compilação do programa o Não inclua quaisquer ficheiros não necessários para a compilação ou execução do programa (ex. diretórios ou ficheiros de sistemas de controlo de versões) ● Não serão admitidas entregas por e-mail.
10/12/2019 a 20/12/2019	Defesa final	• A defesa final vale **80%** da cotação do projeto e consistirá numa análise detalhada do trabalho apresentado. • Defesas em grupo nas aulas PL. • É necessária inscrição para a defesa.