Instituto Superior de Engenharia de Coimbra
Departamento de Engenharia Informática e de Sistemas

Sistemas Operativos 25/26

Trabalho Prático - Programação em C para UNIX

Regras gerais

O trabalho prático aborda os conhecimentos do sistema Unix, e deve ser concretizado com a linguagem e API e mecanismos abordados nas aulas. O trabalho foca-se no uso correto dos mecanismos e recursos do sistema e não é dada particular preferência a escolhas na implementação de aspetos de carácter periférico à disciplina.

No que respeita à manipulação de recursos do sistema, deve ser dada prioridade ao uso de chamadas ao sistema operativo[1] face ao uso de funções biblioteca[2] (por exemplo, devem ser usadas *read()* e *write()* em vez de *fread()* e *fwrite()*). Excetua-se aqui a leitura de ficheiros de natureza secundária, tais como ficheiros de configuração ou salvaguarda de dados que podem ser manipulados com funções biblioteca (mas no uso de *named pipes* devem mesmo ser usadas as funções sistema).

Não é permitido o recurso a bibliotecas de terceiros que façam parte do trabalho, ou que ocultem o uso dos recursos e mecanismos estudados na disciplina. O uso de API do sistema Unix que não tenha sido abordado nas aulas é permitido desde que dentro da mesma categoria de recursos estudados - ou seja, variações menores das funções estudadas, **e terá** que ser devidamente explicado, em particular durante a defesa.

Não é permitida uma abordagem baseada na mera colagem de excertos de outros programas ou de exemplos. Todo o código apresentado terá que ser entendido e explicado por quem o apresenta, caso contrário não será contabilizado.

1. Descrição geral, conceitos principais e elementos envolvidos

O trabalho consiste na simulação de uma plataforma de gestão de um serviço de táxi composto por uma frota de automóveis autónomos. A plataforma é composta por três aplicações: **cliente** – interação entre o utilizador e a plataforma de agendamento de serviços de transporte; **controlador** – responsável pela gestão da frota automóvel, receção de pedidos de transporte, alocação de veículos e contabilização do total de km percorridos; e **veículo** – simula o funcionamento de um veículo autónomo, interagindo com o **cliente**, sempre que necessário. Um utilizador deste serviço pode agendar um serviço de transporte, através da aplicação **cliente**. A plataforma (aplicação **controlador**) é responsável por alocar, à hora pretendida pelo cliente, um dos veículos autónomos que compõem a frota de táxis. O veículo autónomo (aplicação **veículo**) desloca-se para o local indicado pelo cliente e, antes de iniciar o serviço, solicita diretamente ao cliente o local para onde este se pretende deslocar.

A interface com o utilizador é em ambiente de consola (texto). Haverá vários utilizadores, devendo cada um usar um terminal diferente, mas sempre na mesma máquina. *Utilizador da plataforma* e *utilizador do sistema* operativo são conceitos diferentes.

Os utilizadores começam por se identificar (ou seja, "começam por se identificar, usando para isso a aplicação **cliente**") junto da plataforma através de um *username*. Os utilizadores não necessitam de qualquer tipo de registo prévio, bastando indicar um *username* que ainda não tenha sido utilizado por qualquer um dos outros utilizadores que já se encontram ligados nesse momento. Não é necessária nenhuma *password*.

[1] Documentadas na secção 2 das *manpages*

[2] Documentadas na secção 3 das *manpages*

Os utilizadores podem agendar serviços de transporte, consultar os serviços agendados e cancelar um determinado serviço previamente agendado, mas ainda não realizado (aplicação **cliente**). O utilizador pode solicitar um ou mais serviços de transporte, indicando a *hora*, *local de início* e *distância do percurso*. O local específico para onde se pretende deslocar é indicado pelo utilizador ao **veículo** autónomo, quando este chegar junto dele. Pode assumir que existe um número máximo (limitado) de serviços e de utilizadores. Não é necessário efetuar verificações de *regra de negócio* tais como impedir o mesmo utilizador de solicitar duas viagens diferentes à mesma hora (não é esse o foco do trabalho).

À hora indicada, o **controlador** envia um veículo autónomo para efetuar o serviço de transporte solicitado. Para esse efeito deve lançar um processo **veículo**, passando-lhe a informação do serviço a efetuar através de argumentos da linha de comandos. Tenha em atenção que o **veículo** terá de contactar o **cliente** e, por esse motivo, precisa de conhecer o "contacto" dele (ou seja, o controlador envia também essa informação, igualmente através de argumentos de linha de comandos). Desde o momento que lança o **veículo**, o **controlador** fica a receber informação do estado o **veículo** (corresponde ao conceito muito em voga de telemetria). A informação é enviada pelo **veículo** pelo seu *stdout*, devendo o **controlador** capturar essa informação. O controlador fará isto para todos os veículos que estejam em ação, e podem ser vários ao mesmo tempo. A informação relevante é: o cliente entrou na viatura, a percentagem da distância já percorrida (o veículo reporta a cada 10%) e o cliente saiu da viatura (indicando se completou a viagem ou se o cliente revolveu sair antes).

O **veículo** é uma aplicação que simula o funcionamento de um veículo autónomo. Para cada serviço agendado, será executada uma aplicação **veículo, lançada pelo programa controlador (como referido no parágrafo anterior)** que termina quando o cliente sai da viatura (por vontade própria a meio do percurso, ou quando chega ao seu destino). As informações acerca do serviço e do cliente são especificadas através de argumentos da linha de comandos. Desde que é executado até terminar, o processo **veículo** deve enviar para o monitor (*stdout*) a informação de estado do veículo (telemetria), tal como indicado antes. O formato da informação está em aberto e deve ser decidido pelo implementador: pode ser uma única *string* num qualquer formato que encapsula toda a informação a indicar no momento em questão, ou pode ser um envio por cada elemento de informação, sendo indicado qual o elemento de informação e o respetivo valor. O requisito é que a informação deve ser passada de **veículo** para o **controlador** pelo *stdout*, e deve respeitar a periodicidade (por exemplo, os 10% da distância). Esta informação é recolhida pelo **controlador** para saber o estado de toda a frota automóvel – que/quantos **veículos** estão a efetuar serviços e em que estado se encontram.

2. Programas envolvidos

A plataforma é totalmente concretizada através dos três programas já referidos: **cliente**, **veículo** e **controlador.**

Cliente

O programa **cliente** é usado pelo utilizador para agendar, consultar e cancelar serviços, para além das ações de entrar e sair (a meio do percurso) de um veículo. O envio de comandos (ação que pretende realizar) e a receção de informação (chegada do veículo de transporte, etc.) podem ocorrer em simultâneo (significa que enquanto o utilizador está a escrever um comando, pode surgir no terminal o conteúdo de uma mensagem que tenha sido enviada pelo **controlador** e/ou **veículo**). A funcionalidade do programa **cliente** é, essencialmente, a interface de utilizador, sendo toda a gestão da plataforma feita no programa **controlador**. O utilizador apenas conseguirá utilizar a funcionalidade da plataforma depois de se identificar e tendo essa identificação sido aceite. A identificação consiste num *username* (uma palavra apenas), sem qualquer *password*, sendo sempre aceite desde que não haja já outro utilizador com esse nome. Cada utilizador lançará uma instância do programa **cliente**, podendo haver mais do que uma em simultâneo a um dado instante.

Qualquer troca de informação entre o **cliente** e o **veículo**, ou entre o **cliente** e o **controlador**, será feita por ***named pipes***.

Veículo

O programa **veículo** simula um veículo autónomo. É executado pelo **controlador** à hora indicada para o serviço solicitado e recebe, através de argumentos da linha de comandos, os dados do serviço a efetuar (incluindo o contacto do cliente a quem se destina o serviço). Reporta todos os eventos do serviço para o monitor (*stdout*). A comunicação com o **cliente** é feita através de *named pipes*. Não pode utilizar *named pipes* para a comunicação com o **controlador** – deve utilizar os argumentos da linha de comandos (para receber informação do **controlador**) e o envio de informação de estado para o monitor (*stdout*) (para enviar informação ao **controlador**). Podem existir vários processos **veículo** em execução, estando este valor limitado pelo tamanho da frota automóvel (definido através de uma variável de ambiente que é considerada pelo **controlador**).

Controlador

O programa **controlador** gere a frota de veículos autónomos. Pode interagir diretamente com um utilizador (administrador **da plataforma**, não do sistema), o qual especifica comandos na lógica de comandos escritos na consola (e recebidos pelo **controlador** no seu *stdin*) que permitem, por exemplo, desligar a plataforma, mostrar os utilizadores ativos, a lista de serviços agendados, a lista e o estado dos veículos que estão a executar serviços, etc. O **controlador** comunica: i) com o(s) **cliente**(s) por *named pipes, ii)* com o(s) **veículo**(s) através de argumentos da linha de comandos e redireccionamento da saída (*stdout*) do(s) processo(s) **veículo**(s). Em cada momento existirá no máximo um único processo **controlador** em execução.

Funciona com servidor e recebe pedidos dos vários clientes (em última análise, dos utilizadores) por *named pipes*. Os pedidos podem ser de **agendamento**, **cancelamento** e **consulta** de serviços de transporte (apenas os seus). É responsável por "enviar" o veículo à hora indicada para o local de partida (ou seja, lançar o processo veículo que corresponde ao serviço, à hora solicitada pelo utilizador, especificando os detalhes do mesmo através da linha de comandos), por receber o estado de um serviço de cada um dos veículos. Esta aplicação deve manter atualizado um contador com o número total de quilómetros percorridos por todos os veículos.

A dimensão da frota de veículos está definida na variável de ambiente de NVEICULOS. Isto significa que não haverá mais do que NVEICULOS **veículos** em execução ao mesmo tempo. Por "enviar" um veículo entende-se lançar um processo **veículo**.

O **controlador** é responsável por manter atualizada a informação de estado da frota. Para esse efeito os **veículos** reportam o estado do serviço que se encontram a realizar, informando o seguinte:

- o início do serviço (quando o cliente entrou na viatura, o que ocorre *quando* e *por* interação com o **cliente** desse utilizador);
- a distância já percorrida (percentagem da distância total, indicado a cada 10%); e
- o fim do serviço (o cliente saiu da viatura a meio ou chegou ao seu destino).

Para cancelar um serviço que já está a ser realizado, o **controlador** deve enviar o sinal SIGUSR1 ao processo **veículo**.

Limites predefinidos

Na implementação pode assumir que existem valores máximos para as seguintes entidades

- Utilizadores: máximo 30
- Veículos: máximo 10 (o valor real é especificado através da variável de ambiente NVEÍCULOS)

2. Utilização da plataforma

Utilizadores - Existem dois tipos de utilizador neste sistema:

- **Utilizador**. Este é aquele que usa o **cliente** e, após se identificar, utiliza a plataforma para solicitar serviços de transporte (agendar, consultar e cancelar) e executar ações num veículo (entrar e sair). Não tem forma de contactar diretamente um **veículo**, apenas o poderá fazer após o contacto inicial do **veículo** (aquando do início do serviço). Não

existe qualquer interação entre cliente-cliente. Para lidar com um novo utilizador será simplesmente aberto um novo terminal através do qual interage com o **cliente**.

- **Administrador**. Controla o **controlador** e é responsável por lançar a sua execução. Pode interagir com a plataforma através do programa **controlador**, seguindo a lógica de comandos escritos no *stdin* deste (estes comandos serão descritos em detalhe mais adiante). É importante salientar que o "administrador" não tem qualquer relação com o administrador (*root*) do sistema operativo.

Todos os utilizadores da plataforma correspondem ao mesmo utilizador do sistema operativo onde os vários programas **cliente** são executados. Simular a existência de mais um utilizador corresponde a executar mais um **cliente** num outro terminal.

Controlador

Do ponto de vista do utilizador administrador (programa **controlador**):

- O único utilizador que interage com o **controlador** é o administrador da plataforma, através de comandos escritos para efetuar ações de controlo. Não existe nenhum procedimento de *login*: o administrador é simplesmente quem executa o programa **controlador**, ficando a interagir com ele. O administrador executa uma instância do programa **controlador**, sendo que apenas uma instância (processo) deste programa pode estar a correr num dado instante. A validação deste aspeto fica a cargo do programa e não do utilizador. Exemplo de execução:

  ```
  $ ./controlador
  ```

- As ações (comandos) disponíveis ao administrador são:

O **controlador** dispõe de uma interface, seguindo paradigma da linha de comandos, que permite ao administrador realizar os seguintes comandos:

- **listar** – Mostra a informação de todos os serviços agendados.
- **utiliz** – Mostra a lista dos utilizadores atualmente ligados (com indicação à espera de carro / em viagem).
- **frota** – Mostra a percentagem da viagem (% do percurso já feito) de cada um dos veículos.
- **cancelar** <id> – Cancela um serviço através do seu *id*, quer esteja agendado ou a ser realizado. Se o *id* for 0 (zero), cancela todos os serviços agendados, incluindo os que já estão a ser realizados (neste caso deve cancelar esses serviços a meio da sua execução).
- **km** – Mostra o número total de quilómetros percorridos por todos os veículos.
- **hora** – Mostra o valor atual do tempo simulado (não é a hora do sistema)
- **terminar** – Termina a execução de todo o sistema (deve cancelar todos os serviços e notificar todos os **clientes**).

Cliente

Funciona como uma interface entre os utilizadores e a plataforma**.** Envia pedidos, via *named pipes,* utilizando o paradigma de linha de comandos. Do ponto de vista de um utilizador da plataforma (programa **cliente**):

- O utilizador cliente executa o programa **cliente** indicando o seu *username*, através da linha de comandos. Exemplo:

  ```
  $ ./cliente pedro
  ```

  O **cliente** só aceita executar se o **controlador** estiver em funcionamento e não deve permanecer em execução se o **controlador** terminar.

- O **cliente** envia inicialmente ao **controlador** o nome do utilizador. Caso a validação seja bem-sucedida (não exista outro utilizador com o mesmo nome e não tenha sido atingido o limite máximo de utilizadores), o programa **cliente** fica a aguardar comandos do utilizador. Caso contrário, o **controlador** informa o utilizador do sucedido através do **cliente**.

- O utilizador interage com o **cliente** através de comandos de texto que permitem, entre outras coisas, gerir os seus serviços de transporte, interagir com o veículo autónomo e a correspondente receção de informação da plataforma.
- A comunicação entre o **cliente** e o **veículo** é feita diretamente entre os dois, não envolvendo o **controlador** (como intermediário). No entanto, pode haver situações em que o **cliente** precise de falar com ambos **veiculo** e **controlador**.

O programa **cliente** deve lidar em simultâneo com a informação que chega das aplicações **controlador** e **veículo** (via *named pipes*) e com os comandos introduzidos pelo utilizador. Dependendo do comando introduzido, este será enviada ao **controlador** ou ao **veículo**. Sem prejuízo da usabilidade da interface com utilizador, não é obrigatório o recurso à biblioteca **ncurses**, sendo suficiente o uso de uma interface baseada no paradigma de consola (*scroll-up* do texto) desde que seja clara e lógica.

O utilizador pode introduzir (no cliente) os seguintes comandos (que serão dirigidos ao **controlador**, ou ao **veículo** conforme o caso e o necessário para cumprir a ordem do utilizador:

- **agendar** <hora> <local> <distancia>

  Agenda um serviço para a *hora* indicada, do *local* de partida para um destino a uma *distância* especificada. A hora é especificada em número de segundos (contados desde o início do **controlador**), o local de partida é meramente informativo (pode assumir um valor qualquer – *string*) e a distância é indicada em quilómetros.

- **cancelar** <id>

  Cancela um serviço previamente agendado, que ainda não foi realizado. Se o *id* for 0 (zero), cancela todos os serviços agendados pelo utilizador (quem gere o agendamento é o **controlador**).

- **consultar**

  Mostra a informação dos serviços agendados por si (esta informação está no **controlador**).

- **entrar** <destino>

  Entrar no **veículo** após a sua "chegada" (há um contacto por parte do **veículo** a informar o **cliente** que chegou ao local de partida). Apenas pode ser utilizado no início de um serviço e deve indicar o local de destino (sendo a distância a percorrer previamente conhecida pelo **veículo**). O destino pode ser especificado como uma *string* (como por exemplo: Coimbra, Porto, Lisboa, etc.), cujo conteúdo não tem qualquer outra utilização para além de informativa. Este comando é direcionado ao **veículo**.

- **sair**

  Permite indicar ao **veículo** que quer sair antes de chegar ao destino. Neste caso o **veículo** dá por terminado o serviço, sendo o controlador avisado. A distância efetivamente percorrida pode ser assumida como sendo a da última atualização da distância percorrida pelo veículo – múltiplos de 10%, como o instante em que o cliente sai do veículo. Este comando é direcionado ao **veículo**.

- **terminar**

  Permite sair da aplicação **cliente**, desde que não tenha nenhum serviço em execução (apenas poderá sair quando o serviço for concluído e emitido o seu pagamento pelo **controlador**). Caso tenha serviços agendados, estes devem ser cancelados pelo **controlador** (o utilizador tem de informá-lo da sua intenção).

Todos os comandos devem ter *feedback* no terminal que indique o sucesso/insucesso da operação e os dados relevantes, conforme o comando escrito.

As mensagens recebidas no cliente devem ter efeitos visíveis na interface do utilizador com indicação de onde provêm (**controlador** ou **veículo**).

Veículo

Serve para simular um veículo autónomo. Para esse efeito, envia para o monitor (*stdou*t) toda a informação de estado do serviço que se encontra a realizar: entrada de um cliente, distância já percorrida, serviço cancelado a meio por intenção do cliente (se for o caso) e serviço concluído.

Recebe através da linha de comandos o *local*, a *distância* (número de quilómetros a percorrer) e a forma de chegar até ao cliente (por exemplo, o *named pipe* utilizado pelo processo **cliente**).

Esta aplicação é lançada pelo **controlador** à hora indicada para o serviço. Assim que é colocada em execução (assume-se que chega instantaneamente ao local de partida do serviço) deve contactar o **cliente** (contacto direto via *named pipes*) para dar início ao serviço, reportando esse facto. Recorde-se que "hora" no contexto do enunciado é um simples contador que começa em 1 quando o **controlador** inicia e incrementa de 1 em 1.

Durante a execução do serviço o **veículo** deve reportar o estado de execução do serviço a cada 10% da distância total percorrida. Para esse efeito, todos os veículos têm a mesma velocidade que é de 1 km por unidade de tempo (o tal tempo simulado). Caso receba o sinal SIGUSR1 deve cancelar o serviço que está a executar, informando o **cliente** desse facto, e terminando o processo de imediato.

No final deve reportar a conclusão o serviço (saída do **cliente**) e terminar a sua execução (o processo termina, dispondo o **controlador** de mais um veículo para os próximos serviços).

Esta aplicação não recebe qualquer comando diretamente do utilizador (ao contrário do que acontece com as outras aplicações – **cliente** e **controlador**), interagindo apenas com as aplicações **cliente** e **controlador** (utilizando os mecanismos de comunicação já indicados). No entanto, o **veículo** pode ser afetado indiretamente pelo utilizador: por exemplo, se o utilizador decidir sair a meio da viagem (esta intenção é comunicada em primeira mão ao **cliente**, que depois dá andamento à situação).

Gestão do tempo pelo programa controlador

Não é necessário utilizar a hora do sistema operativo para especificar a "hora" a que um **cliente** pretende agendar um serviço. Pode assumir que "hora" é especificada como um valor inteiro, que corresponde ao número de segundos que passaram desde o lançamento da aplicação **controlador**.

Ficheiros de dados envolvidos

Não existe nenhum ficheiro de configuração ou de dados.

3. Requisitos e restrições

Implementação

- **Não pode** utilizar o mecanismo *select* no programa **controlador**.
- Ficheiros regulares são repositórios de informação - **não são** mecanismos de comunicação.
- O mecanismo de comunicação entre o **cliente** e as outras aplicações (**controlador** e **veículo**) é o *named pipe*. O número de *named pipes* envolvidos, quem os cria, e a forma como são usados devem seguir os princípios exemplificado nas aulas. O uso de *named pipes* a mais do que aquilo que é necessário pode ser alvo de penalização.
- Os mecanismos de comunicação entre o **controlador** e o(s) **veículo**(s) são os argumentos da linha de comandos (**controlador** para **veículo**) e os *pipes anónimos* – redireccionamento saída do(s) **veículo**(s) (**veículo** para **controlador**).
- Só podem ser usados os mecanismos de comunicação que foram abordados nas aulas.
- A API do sistema deve ser aquela que foi estudada. Qualquer variação deve ser dentro da API estudada. Uma função pouco abordada, mas relacionada com as estudadas, é aceite, mantendo-se dentro do contexto do que foi abordado (exemplo: *dup2* em vez de *dup* é aceite – desde que explicada na defesa, mas uso de memória partilhada já não).
- O uso de bibliotecas de terceiros (exceto as fornecidas pelos professores) não é aceite.

- As questões de sincronização que possam existir devem ser acauteladas e tratadas da forma adequada.
- Situações que obriguem os programas a lidar com ações que possam ocorrer em simultâneo devem ser resolvidas com soluções que não atrasem ou impeçam essa simultaneidade. Essas situações, se ocorrerem, têm formas adequadas de solução que foram estudadas nas aulas.
- Usos de excertos de código de exemplos (livros, stackoverflow, github, bots, etc.) não poderão ser extensos nem abordar as questões centrais da matéria, devendo ser explicados na defesa.
- Todo o código terá de ser explicado na defesa, tenha ou não sido retirado de exemplos - se não for explicado, será entendido como "o conhecimento não está lá" e, por conseguinte, a parte não explicada não é valorizada. A frase seguinte e respetivas variações não é uma explicação: "foi como vi nas aulas e faço igual, mas não sei porquê nem o que significa". Esta também não: "estava a ter problemas e uns colegas disseram para fazer assim".

Terminação dos programas

- Quer seja feita a pedido do utilizador, ou por não estarem reunidos os pressupostos para o programa executar, ou por situação de erro em *runtime*, os programas devem terminar de forma ordeira, libertando os recursos usados, avisando tanto quanto possível o utilizador e os programas com que interajam.

4. Regras gerais do trabalho

Aplicam-se as seguintes regras, descritas na primeira aula e na ficha da unidade curricular (FUC):

- O trabalho pode e deve ser realizado em grupos de dois alunos.
- Existe defesa obrigatória, que é individual e presencial. Podem existir moldes adicionais a definir e anunciados através dos canais habituais na altura em que tal for relevante (por exemplo, ser ou não necessário trazer computador).
- A entrega do trabalho prático é feita via nónio (inforestudante) através da submissão de um único **arquivo *zip***[3] cujo **nome** respeita o seguinte padrão[4]:

  so_2526_tp_*nome1_numero1_nome2_numero2*.zip

  (*nome* e *número* são os nomes e números de aluno dos elementos do grupo)
- A não adesão ao formato de compressão indicado (.zip) ou ao padrão do nome do ficheiro será penalizada, *podendo levar a que o trabalho nem sequer seja visto*.
- Do arquivo zip deverão constar:
  - Todo o **código fonte** desenvolvido;
  - Ficheiro(s) ***makefile*** com *targets* de compilação "all" (compilação de todos os programas), "controlador" (compilação do programa ***controlador***), "cliente" (compilação do programa ***cliente***), "veiculo" (compilação do programa ***veículo***) e "clean" (eliminação de todos os ficheiros temporários e dos executáveis – *.o);
  - Um **relatório** (em pdf) com o conteúdo relevante e que deverá ser da exclusiva autoria dos membros do grupo.
- Cada grupo submete o trabalho uma vez, sendo indiferente qual dos dois alunos o faz.
- **É obrigatório** que o aluno que faz a submissão **associe no nónio a entrega também ao outro aluno do grupo**.

[3] Leia-se "***zip***" - não é *arj*, *rar*, *tar*, ou outros. O uso de outro formato poderá ser **penalizado**. Há muitos utilitários da linha de comando UNIX para lidar com estes ficheiros (zip, gzip, etc.). Use um.

[4] O não cumprimento do formato do nome causa atrasos na gestão dos trabalhos recebidos e será **penalizado**.

- **É conveniente que ambos estejam inscritos em turmas práticas (mesmo que seja em turmas diferentes). A não inscrição impede o registo do trabalho na plataforma, e, por conseguinte, pode levar à perda da nota.**

Data de entrega: Sábado, 13 de dezembro, 2025.

5. Avaliação do trabalho

Para a avaliação do trabalho serão tomados em conta os seguintes elementos:

- **Arquitetura do sistema** – Há aspetos relativos à interação dos vários processos que devem ser planeados de forma a apresentar-se uma solução elegante, leve e simples. A arquitetura deve ser bem explicada no relatório.
- **Implementação** – Deve ser racional e não desperdiçar recursos do sistema. As soluções encontradas devem ser claras e bem documentadas no relatório. O estilo de programação deve seguir as boas práticas. O código deve ter comentários relevantes. Os recursos e API do sistema devem ser usados de acordo com a sua natureza.
- **Relatório** – O relatório deve descrever bem o trabalho. Descrições superficiais ou genéricas não servem. O relatório deve descrever e justificar a estratégia e os modelos seguidos, a estrutura da implementação e as opções tomadas. Deve incluir uma tabela com as funcionalidades requeridas, e, para cada uma, a indicação de cumprido, parcialmente cumprido, não cumprido (e porquê). O relatório é entregue em formato pdf dentro do arquivo submetido.
- **Defesa** – Os trabalhos são sujeitos a defesa individual, durante a qual será verificada a autoria e conhecimentos, podendo haver mais do que uma defesa caso subsistam dúvidas. A nota final do trabalho é diretamente proporcional à qualidade da defesa. Elementos do mesmo grupo podem ter notas diferentes consoante o desempenho e grau de participação individuais que demonstraram na defesa.
  A falta à defesa implica automaticamente a perda da totalidade da nota do trabalho.
- Plágios e trabalhos feitos por terceiros: o regulamento da escola descreve o que acontece nas situações de fraude.
- Os trabalhos que não funcionem serão fortemente penalizados independentemente da qualidade do código-fonte ou arquitetura apresentados. Trabalhos que nem sequer compilam terão uma nota extremamente baixa.
- A identificação dos elementos de grupo deve ser clara e inequívoca (tanto no arquivo zip como no relatório). Trabalhos anónimos não são corrigidos.
- Qualquer desvio quanto ao formato e forma nas submissões (exemplo, tipo de ficheiro) dará lugar a penalizações.

Importante: O trabalho deve ser realizado por ambos os elementos do grupo. Não são aceites divisões herméticas em que um elemento faz uma parte e apenas essa, nada sabendo do restante. Se num grupo existir uma participação desigual, o professor que faz a defesa deve ser informado do facto no início da defesa. Não tendo sendo informado e sendo detetada essa desigualdade, ambos os alunos ficarão prejudicados em vez de apenas aquele que trabalhou menos.