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

Sistemas Operativos 22/23

Trabalho Prático - Programação em C para UNIX

O trabalho prático consiste na implementação de uma plataforma para gerir leilões. O trabalho prático aborda acima de tudo os conhecimentos do sistema Unix, e deve ser concretizado em linguagem C, para plataforma Unix (Linux), usando os mecanismos deste sistema operativo abordados nas aulas teóricas e práticas. 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()*). 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 caráter periférico à disciplina (por exemplo: a pergunta "devo usar uma lista ligada ou um *array* dinâmico?" terá como resposta um encolher de ombros).
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, 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

Neste trabalho deve implementar uma plataforma para gerir um sistema de leilões ("SOBay"), o qual faz a gestão da comunicação entre os clientes envolvidos, gere os itens à venda, verifica os preços e determina quem arremata os itens. A funcionalidade é intuitiva e semelhante (mas muito simplificada) a muitos sistemas reais de leilões online. Para evitar confusão, "plataforma" refere-se sempre ao SOBay e "sistema" refere-se sempre ao sistema operativo.

Conceitos principais

- **Item** - objeto posto à venda. Cada item tem um ID (atribuído automaticamente pela plataforma), um nome (descrição: uma palavra apenas) pertence a uma determinada categoria (uma palavra apenas), tem associado um valor base de licitação e poderá ter um valor para "comprar já".
- **Leilão** - um item é colocado à venda por um período de tempo máximo. No final, o utilizador que licitou o preço mais elevado fica com ele. O item é colocado à venda por um preço base (inteiro). De cada vez que é feita uma licitação, a plataforma assume que passa a ser esse o novo preço (a plataforma ignora ofertas por valores inferiores).
- **Valor "Compre Já"** - um item tem associado um valor (inteiro) pelo qual poderá ser adquirido de imediato. O primeiro utilizador que oferecer o valor indicado fica imediatamente com ele. O valor "compre já" pode ser zero o que significa que esse item não tem a opção de compra imediata.

[1] Documentadas na secção 2 das *manpages*
[2] Documentadas na secção 3 das *manpages*

- **ID de item**. Os itens passam a existir na plataforma quando são postos à venda. Nesse instante é-lhe atribuido plataforma um identificador "ID" numérico que é usado posteriomente pelos utilizadores. Cada item terá um ID seguinte ao item anterior. A plataforma deve manter o valor atual do ID mais elevado de forma a saber qual o ID a atribuir ao item seguinte. **Esse valor começa em 1 na primeira execução da plataforma e deve ser mantido e continuado nas execuções seguintes**, se a plataforma fore transferida para outro computador.
- **Promoção** - durante o decurso dos leilões poderão existir promoções sobre determinadas categorias de itens e com uma duração limitada no tempo. As promoções são "anunciadas" por um programa específico.
- **Tempo** - a plataforma gere a passagem de tempo com a precisão de 1 segundo. Todos os intervalos de tempo usados pela plataforma são indicados em segundos, nunca se agrupando em minutos ou horas. A passagem do tempo na plataforma é independente da hora real mantida pelo sistema operativo. O tempo é um número crescente, que **começa em 1 na primeira execução da plataforma e que é mantido e continuado nas execuções seguintes**. A forma como esta salvaguarda é feita é deixada ao critério do aluno desde que funcione e seja compatível com a hipótese do software da plataforma seja transferido para outro computador (por exemplo, o computador do professor).
- **Saldo** - cada utilizador tem um saldo. O valor do saldo é atualizado sempre que um item é adquirido ou vendido por ele. Um utilizador com saldo negativo fica impedido de licitar novos iten, mas mantêm-se todas as licitações já efetuadas por esse utilizador, podendo este ficar com saldo negativo em resultado disso.

Programas envolvidos

O sistema pretendido abarca 3 tipos de programas principais: ***frontend***, ***backend*** e ***promotor***.

- O programa ***frontend*** é usado pelo utilizador para comprar e vender itens. O utilizador pode ser vendedor e comprador em simultâneo (itens diferentes), pelo que o programa ***frontend*** deve possibilitar tanto a realização de compras (e.g., licitar), como a colocação de itens para venda. A funcionalidade do ***frontend*** é, essencialmente, a interface entre o utilizador e o programa ***backend*** dado que é este último que gere os leilões.
  Haverá um processo ***frontend*** a correr por cada utilizador ligado à plataforma, sendo ele o responsável por o lançar.
- O programa ***backend*** faz a gestão da plataforma de leilões (funciona como um servidor). Recebe pedidos e informações por parte dos ***frontend*** correspondentes às funcionalidades disponibilizados pela plataforma aos utilizadores (exemplos: colocar um item à venda, licitar por um item, efetuar consultar, etc.). Pode também enviar informação ao ***frontend*** sem que este a tenha pedido. Este programa interage também com o(s) ***promotor***(es). O ***backend*** é usado diretamente (sem outro programa intermediário) por um utilizador ***administrador***.
  Existirá apenas um e um só processo ***backend*** a correr a um dado instante.
- O programa ***promotor*** desencadeia a promoção de itens, afetando a forma como determinados itens são vendidos pelo ***backend***. Este programa interage diretamente com o ***backend*** e tem características importantes que são explicadas mais adiante. Existirão zero, um ou mais processos **promotor** em simultâneo.

Mais adiante são dados detalhes adicionais acerca destes programas.

Utilizadores

Toda a interface com o utilizador é feita em ambiente de consola (terminal modo-texto). As ações dos utilizadores são especificadas por comandos escritos, e a informação apresentada é sempre texto. Não são necessários gráficos nem cores.

Todos os utilizadores da plataforma correspondem ao mesmo utilizador do sistema operativo onde os vários programas da plataforma são executados. Para simular um novo utilizador da plataforma será simplesmente aberto um novo terminal através do qual esse novo utilizador da plataforma interage com esta.

Existem dois tipos de utilizador neste sistema:

- **Cliente**. É aquele que compra ou vende itens, podendo ser simultaneamente vendedor e comprador (de itens diferentes). Utiliza o programa ***frontend***, e apenas este. Cada utilizador executa um ***frontend***. A interação do cliente com o ***frontend*** segue a lógica de comandos escritos que são processados pelo ***frontend***.
- **Administrador**. Controla a plataforma, sendo o responsável por lançar o ***backend***. Interage com a plataforma através do programa ***backend*** segundo a lógica de comandos escritos que são processados pelo ***backend*** (descritos mais adiante). O "administrador" não tem nada a ver com o administrador (*root*) do sistema operativo.

Ficheiros de dados

Existe três ficheiros **de texto** que permitem especificar os utilizadores da plataforma, os promotores e guardar/recuperar o estado dos leilões. Pode assumir que os dados nos ficheiros indicados estarão sempre formatados de forma correta.

- **`ficheiros de utilizadores`** - tem a informação relativa aos utilizadores cliente da plataforma, constituída pelo *username*, *password* e saldo. Existe um utilizador por cada linha e os campos são separados por um espaço. Username e password são sempre uma palavra cada, e o saldo é um valor inteiro. É garantido que os dados no ficheiro estão corretos. Pode assumir que este ficheiro não sofre alterações durante a execução do ***backend***.
- **`ficheiro de promotores`** - ficheiro que guarda o nome dos ficheiros executáveis dos programas ***promotor***. Cada linha do ficheiro corresponde a um ***promotor***. O ***backend*** é responsável por lançar e gerir a execução dos promotores aqui listados. A informação no ficheiro está corretamente formatada, no entanto, existe a hipótese dos programas lá mencionados não existirem e isso deve ser previsto pelo ***backend***.
- **`ficheiro de itens`** - tem informação relativa aos itens que ainda estavam a leilão aquando do encerramento da plataforma: os respectivos leilões devem ser repostos, sendo a duração indicada no ficheiro o tempo que faltava quando a plataforma encerrou (a plataforma atualiza o ficheiro ao encerrar). Cada linha do ficheiro descreve um item à venda, e os campos em cada linha são separados por um espaço. Cada linha deve conter, por esta ordem: *identificador único do item, nome do item (uma palavra), categoria (uma palavra), valor atual (valor inicial ou valor da licitação mais elevada), valor "compre já", duração do leilão (= tempo restante), username do utilizador que vende, username do utilizador que licitou o valor mais elevado (ou "-" se não tiver ainda sido licitado)*. Este ficheiro pode não existir, o que significa que não havia / já não havia nenhum item à venda quando a plataforma (re)iniciou. Se existir um ficheiro, pode assumir que este estará corretamente formatado.

O nome destes ficheiros pode variar, sendo indicado em variáveis de ambiente descritas mais adiante.

2. Utilização do sistema (funcionalidade vista pelos utilizadores)

Do ponto de vista do utilizador *cliente* (programa *frontend*)

- O utilizador cliente executa o programa ***frontend*** indicando as suas credenciais (username e a password - **da plataforma**), por linha de comandos. Exemplo:

```
$./frontend manuel password_do_manuel
```

- O ***frontend***, como interface com o utilizador que é, envia ao ***backend*** as credenciais para validação. Caso a validação seja bem sucedida, o ***backend*** passa a aceitar pedidos desse ***frontend*** associando-as ao respetivo utilizador. Caso a autenticação falhe, o servidor ignora ordens desse cliente (exceto novos pedidos de *login*). Em ambos os casos, o ***frontend*** é informado do resultado da validação das credenciais.

- O utilizador interage com o ***frontend*** através de **comandos escritos**, que causam o envio de mensagens para o ***backend*** (não se trata necessariamente de enviar para o ***backend*** aquilo que o utilizador escreveu *ipsis verbis*).
- **IMPORTANTE**. A maior parte da interação entre o utilizador e a plataforma (ou seja, entre ***frontend*** e ***backend***) será na lógica do diálogo: "vai pergunta, vem resposta". No entanto, o ***backend*** irá enviar informação aos vários ***frontends*** em execução por iniciativa própria, o que significa que o ***frontend*** tem que estar apto a interagir tanto com o utilizador como com o ***backend***, em paralelo. Significa também que o ***backend*** terá de registar de alguma forma os ***frontends*** que estão atualmente em execução com utilizadores autenticados.

O utilizador ***cliente*** pode (comandos)

- **Colocar um item a leilão:** comando **sell** *<nome-item> <categoria> <preço-base> <preço-compre-já> <duração>*
  Deve indicar o nome do item, a categoria, o preço base e o preço "compre já" (ambos inteiros) e a duração do leilão
  - Exemplo: *sell martelo construcao 10 20 200*
- **Listar** todos os **produtos** atualmente **à venda**. Pode listar por categoria, por preço (atual) máximo, por vendedor. Em princípio, a plataforma apenas regista os itens que ainda estão à venda, esquecendo a existência dos que já foram vendidos
  - Listar todos os itens: comando **list**
  - Listar todos os itens de uma **categoria** (categoria = 1 palavra apenas): comando **licat** <nome-categoria>
    - Exemplo: *licat desporto*
  - Listar todos os itens de um **vendedor:** comando **lisel** *<username do vendedor>*
    - Exemplo: *lisel manuel*
  - Listar todos os itens com preço **até um determinado valor:** comando **lival** *<preço-máximo>*
    - Exemplo: *lival 120*
  - Listar todos os itens **com prazo até** a uma determinada hora: comando **litime** *<hora-em-segundos>*
    (A hora é sempre indicada em segundos, nunca se agrupando em minutos ou horas).
    - Exemplo: *litime 250*
- **Obter a hora** atual (em segundos): comando **time**
- **Licitar** um item: comando **buy** *<id> <valor>*
  Todos os itens estão associados a um identificador (ID) numérico único, atribuído e gerido pela plataforma de forma automática (por exemplo, um número sempre crescente). Este ID aparece sempre nas listagens (ponto anterior). Em ambos os tipos de leilão, a licitação feita pelo utilizador inclui o ID do item e o valor oferecido (pode ser superior ao exigido pelo leilão; se for inferior a licitação é ignorada). A licitação só será aceite se o saldo atual do licitador for superior ao valor oferecido.
  - Exemplo: *buy 15 99*
- **Consultar o saldo:** comando **cash**
- **Carregar** o saldo: comando **add** *<valor>*
  - Exemplo: **add** 200
- **Sair**, encerrando o frontend: comando **exit**
  (O ***backend*** é avisado)

Todos os comandos devem ter feedback que indique o sucesso/insucesso da operação e os dados relevantes, conforme a opção pedida. Exemplo: a colocação de um item à venda deve incluir no *feedback* o ID que lhe foi atribuído pela plataforma.

Outras funcionalidades associadas ao utilizador de carácter automático

- O utilizador recebe uma notificação/mensagem cada vez que um artigo é vendido ou simplesmente o tempo tenha terminado (não vendido), mesmo que não tenha participado no negócio. A notificação inclui ID, o nome do item, a categoria, o valor de venda, o username de quem o comprou (ou "por vender").
- O utilizador é informado quando um item é colocado à venda: ID, nome, categoria, preço base, preço "compre já".
- O utilizador é informado sempre que uma promoção começa ou termina.

Do ponto de vista do utilizador ***administrador*** (programa ***backend***)

O administrador executa uma instância do programa ***backend***, sendo que apenas uma instância (processo) deste tipo pode estar a correr. A validação deste aspeto fica a cargo do programa e não do utilizador. Exemplo de execução:

```
$./backend
```

O único utilizador que interage com o ***backend*** é o **administrador**, através de comandos escritos (não são menus) para efetuar ações de controlo da plataforma. Não existe nenhum processo de login: o **administrador** é simplesmente quem lança a execução do ***backend***, ficando a interagir com ele. As ações (comandos) disponíveis ao **administrador** são:

- Listar utilizadores cliente atualmente a usar a plataforma: comando **users**

  Hide "Clientes Online" if 0
  Write "Nao existem clientes online" instead
- Listar itens à venda: comando **list**

  Um item por linha: id, nome item, categoria, preço atual, preço compre já, vendedor, licitador mais elevado (ou "-").
- Banir um cliente atualmente logado (porque sim): comando **kick** *<username>*

  Tem o mesmo efeito que se o cliente tivesse feito *logout* (pode voltar a entrar imediatamente). O utilizador em questão é informado, e o seu programa *frontend* deve reconhecer a situação e terminar automaticamente.
  - Exemplo: *kick artur65*
- Listar os promotores atualmente ativos: comando **prom**
- Atualizar promotores: comando **reprom**
- Lê o ficheiro dos promotores novamente. Manda encerrar os promotores que deixaram de estar noficheiro (essas promoções terminam), carrega os novos promotores, mantém inalterados os restantes.
- Cancelar um promotor: comando **cancel** *<nome-do-executável-do-promotor>*

  O promotor deixa de estar em execução. As suas promoção que estivessem ativa terminam. Da próxima vez que o ficheiro de promotores for analisado, se o promotor em questão ainda lá estiver volta a ser executado.
  - Exemplo: *cancel blackFriday*
- Encerrar a plataforma: comando **close**

  Os ***frontends*** são notificados, devendo também terminar. Os promotores devem também terminar. São atualizados os ficheiros dos itens em venda e o dos utilizadores (novos saldos). Os recursos do sistema em uso são libertados.

3. Funcionamento do sistema

Backend

- Só aceita executar se ainda não estiver a correr nenhum outro backend.;
- Interagir com ***administrador***, ***frontends*** e ***promotores***, sem que a interação com um atrase a interação com os outros;
- Ao encerrar avisa todos os intervenientes para encerrarem também. O encerramento deve ser ordeiro;
- Envia informações aos ***frontend***. Estas informações podem ser respostas a pedidos feitos por estes, mas também podem ser informações enviadas por iniciativa própria (sem qualquer pedido inicial), e já anteriormente descritas.

- Implementa um mecanismo que deteta que um cliente terminou sem avisar a plataforma. Sempre que isto acontece remove a informação acerca desse cliente e avisa o administrador. Este mecanismo baseia-se num "indicação de vida" que o cliente envia ao balcão a cada N segundos. O valor de N está indicado na variável de ambiente **`HEARTBEAT`**.
- Lança e gere a execução dos programas ***promotores***, interagindo com estes para obter as promoções que estão em efeito. Mais adiante são dados mais pormenores acerca de **promoções** e **promotores**.
- Em caso de problemas, o programa deve sair de forma ordeira, libertando os recursos ocupados.
- Na implementação **pode assumir que existem máximos** para as seguintes entidades
    - **Utilizadores**: máximo **20**
    - Programas **promotores**: máximo **10**
    - **Itens à venda** (leilões ativos): máximo **30**

Ficheiros de dados usados pelo *backend*

No início da execução, o ***backend*** carrega informação dos ficheiros de texto que descrevem os programas promotores, os utilizadores e os itens atualmente à venda (os ficheiros já foram descritos atrás). Quanto ao **ficheiro dos utilizadores, e apenas esse**, será fornecida uma biblioteca em ficheiro objecto (.o) que implementa a interação com os dados deste ficheiro e o trabalho relativo a este aspeto consiste em integrar essa biblioteca com o código produzido (pormenores mais adiante).

O nome dos 3 ficheiros envolvidos estão descritos nas variáveis de ambiente:

- **`FPROMOTERS`** - tem o nome do ficheiro de texto dos promotores
- **`FUSERS`** - tem o nome do ficheiro de texto que descreve os utilizadores
- **`FITEMS`** - tem o nome do ficheiro de texto com os itens atualmente à venda

Nota: comandos e nomes de ficheiros estão em língua inglesa. Ite**M**s e não ite**N**s

Manipulação do ficheiro de utilizadores - Biblioteca de acesso aos dados de utilizador

Será fornecida uma biblioteca (ficheiro objecto .o) para integrar no projeto. Será disponibilizada em vários formatos (x68 e x64), devendo ser escolhida a versão adequada à a arquitetura usada no trabalho. A biblioteca trata da leitura e salvaguarda dos dados dos utilizadores em ficheiro e responde, através de funções já implementadas, a questões que o resto do projeto poderá necessitar. Os detalhes de implementação não serão publicados e o projeto deverá ser independente desses detalhes.

A biblioteca contém a seguinte funcionalidade

- Leitura do ficheiro de utilizadores (o armazenamento de dados é gerido internamente).
```
int loadUsersFile(char * pathname);
```
- Gravação do ficheiro de utilizadores (deve ser usada pelo menos no final da execução do ***backend***)
```
int saveUsersFile(char * filename);
```
- Verificação de conta de utilizador (username existe e password está correcta).
```
int isUserValid(char * username, char * password);
```
- Obtenção do saldo do utilizador
```
int getUserBalance(char * username);
```
- Atualização do saldo de um utilizador (adicionar ou remover valores).
```
updateUserBalance(char * username, int value);
```
- Obtenção de um texto descritivo acerca do último erro ocorrido nas funções da biblioteca
```
char * getLastErrorText();
```

O ficheiro header da biblioteca (a fornecer) conterá os protótipos das funções e uma explicação dos códigos de retorno.

Promotor

Um promotor é um programa que periodicamente envia para o seu ***stdout*** uma linha de texto com o formato

```
categoria desconto duração
```

Esta informação deve ser apanhada pelo ***backend*** o qual será responsável por garantir que os itens da categoria indicada têm o desconto indicado, ficando esta promoção válida por número de segundos indicado na duração. Exemplo:

```
construção 10 100
```

Esta linha terá como efeito todos os itens da categoria *construção* terem 10% de desconto ao comprador, se a venda for finalizada nos 100 segundos seguintes à emissão da linha de texto. O vbalor recebido pelo vendedor é o mesmo, sendo os 10% (ou outro valor que fosse) assumidos por um fundo especial dos patrocinadores da plataforma.

Importante:

- A gestão da promoção - início, concretização do efeito, final - é inteiramente feita pelo *backend*. O único papel do promotor é o de emitir estas linhas de texto periodicamente. Não se sabe à partida quantas vezes nem quando é que cada promotor emitirá uma nova linha, nem qual será o seu conteúdo. Apenas se sabe que o formato indicado acima será respeitado: a categoria é uma palavra, o desconto e a duração serão valores inteiros positivos, sendo estes elementos separados por um espaço. **A linha terá um \n no fina**l.
- Pode haver vários promotores em ação. Cada um deles poderá emitir linhas, podendo inclusivamente ocorrer essa emissão em simultâneo. O backend deverá gerir a execução dos promotores de forma a que não haja "atropelamento" dos seus outputs.
- Cada promotor mencionado no ficheiro de texto dos promotores é carregado, permanecendo em execução. O promotor permanecerá em execução e terminará se receber um sinal SIGUSR1.

Serão fornecidos vários promotores pelos professores. A plataforma deverá ser compatível com todos. Os alunos podem também implementar outros, se o desejarem.

Frontend

Faz a interação entre o utilizador **cliente** e o resto da plataforma. As suas varacterísticas gerais são:

- Só aceita executar se o ***backend*** estiver em funcionamento.
- Recebe o username e password por parâmetros da linha de comando.
- Está apto a interagir com o ***backend*** e com o utilizador em paralelo sem que uma interação impeça ou atrase a outra.
- Deve indicar ao ***backend*** que está vivo com uma periodicidade de N segundos, sendo N indicado na variável de ambiente **HEARTBEAT**.
- Deve informar o utilizador dos dados relevantes (indicados pelo ***backend***).
- Não deve permanecer em execução se o ***backend*** terminar, ou de se for informado que o seu utilizador foi excluído (comando kick do administrador).

4. Requisitos e restrições

Implementação

- A comunicação entre ***backend*** e ***promotor*** não é feita com ***named pipes***.
- Ficheiros regulares são repositórios de informação - não são mecanismos de comunicação.
- Não existe comunicação entre dois ***frontends***.

- O promotor faz apenas aquilo que foi referido atrás.
- O mecanismo de comunicação entre ***frontend*** e ***backend*** 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.
  - Só podem ser usados os mecanismos de comunicação que foram abordados nas aulas.
  - As bibliotecas fornecidas pelos professores são de uso obrigatório.
  - 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 abordada (exemplo: dup2 em vez de dup é aceite - mas uso de memória partilhada não).
  - O uso de bibliotecas de terceiros (exceto as fornecidas pelo 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 não podem ser resolvidas com soluções que atrasem ou impeçam essa simultaneidade. Essas situações, se ocorrerem, têm formas adequadas de solução que foram estudadas nas aulas e devem ser observadas.
  - Excerto de código "do stackoverflow / github" não poderá ser extenso nem abordar as questões centrais da matéria.
  - Todo o código terá que 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 contabilizada.

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 utilizador e programas com que interajam.

6. Regras gerais do trabalho, METAS e DATAS IMPORTANTES

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 (grupos de três não são admitidos e qualquer pedido nesse sentido será sempre negado ou ignorado).
- Existe defesa obrigatória. A defesa será efetuada em moldes a definir e anunciados através dos canais habituais na altura em que tal for relevante. A defesa será presencial.
- Existem a meta intermédia e a meta final, tal como descrito na FUC. As datas e requisitos das metas são indicados mais abaixo. Em todas as metas a entrega é feita via nónio (inforestudante) através da submissão de um único **arquivo *zip***[3] cujo **nome** respeita o seguinte padrão[4]:

  so_2223_tp_metaN_*nome1_numero1_nome2_numero2*.zip

  (metaN, nome e número serão adaptados à meta, nomes e números dos elementos do grupo)
- A não adesão ao formato de compressão indicado ou ao padrão do nome do ficheiro será penalizada, *podendo levar a que o trabalho nem sequer seja visto*.
- 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**.

Metas: requisitos e datas de entrega

Meta 1: 20 de Novembro

Requisitos:

- Planear e definir as **estruturas de dados** responsáveis por gerir as definições de funcionamento no ***frontend*** e no ***backend***. Definir os *header files* com constantes simbólicas e declarações associadas às estruturas de dados.
- Frontend: implementar a parte da leitura de comandos e respetiva validação. Todos os comandos devem ter a sua sintaxe validada. Os comandos não farão ainda nada, mas será reconhecido como válido ou inválido, incluindo parâmetros. Implementar a recepção das credenciais do utilizador.
- Backend:
  - Implementar a leitura de comandos do administrador, validando a sintaxe de todos. Os comandos não farão ainda nada, mas será reconhecido como válido ou inválido, incluindo parâmetros.
  - Implementar a parte de lançamento dos promotores e recepção das suas frases. Nesta meta, apenas será lançado o primeiro promotor.
  - Leitura e atualização dos utilizadores através da biblioteca fornecida para esse efeito. De cada vez que a funcionalidade é testada, os utilizadores perdem 1 no saldo ficando essa alteração gravada.
  - Leitura do ficheiro dos itens à venda. Os dados serão lidos e interpretados. Não se trata de apenas ler e imprimir um ficheiro de texto - os valores inteiros serão armazenados e mostrados como inteiros.
  - A verificação da funcionalidade dos comandos / execução do promotor / utilizadores / itens será feira por opção do utilizador: a função main irá perguntar algo como "deseja testar que funcionalidade?".
- frontend / backend: outros aspetos necessários às funcionalidades referidas atrás.
- *makefile* que possua os *targets* de compilação "all" (compilação de todos os programas), "frontend" (compilação do programa ***frontend***), "backend" (compilação do programa ***backend***) e "clean" (eliminação de todos os ficheiros temporários de apoio à compilação e dos executáveis).

Data de entrega: Domingo, 20 de Novembro, 2022 . Sem possibilidade de entrega atrasada.

Na **meta 1** deverá ser entregue um breve documento (pdf, incluído no ficheiro zip) com duas páginas descrevendo os pormenores da implementação e principais opções tomadas. A capa de título não conta como página.

Meta 2: 8 de Janeiro

Requisitos:

- Todos os requisitos expostos no enunciado.

Data de entrega: Domingo, 8 de Janeiro, 2023 . Sujeito a ajustes caso haja alterações no calendário escolar.

Na **meta final** deverá ser entregue um **relatório** (pdf, também no zip). O relatório compreenderá o conteúdo que for relevante para justificar o trabalho feito, deverá ser da exclusiva autoria dos membros do grupo. Caso venha a ser divulgado, entretanto, um guia de elaboração do relatório, então este deverá seguir as indicações dadas.

7. 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 do sistema devem ser usados de acordo com a sua natureza.
- **Relatório** – O relatório deve descrever convenientemente o trabalho. Descrições meramente superficiais ou genéricas de pouco servirão. De forma geral, o relatório descreve a estratégia e os modelos seguidos, a estrutura da implementação e as opções tomadas. Podem vir a ser dadas indicações adicionais sobre a sua elaboração. O relatório deve ser entregue juntamente com o código no arquivo submetido na meta em questão.
- **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.

  Apesar da defesa ser individual, ambos os elementos do grupo devem comparecer ao mesmo tempo. 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, deverá informar o professor que lhe faz a defesa. Se não o fizer e for detectada essa desigualdade, ambos os alunos ficarão prejudicados em vez de apenas aquele que trabalhou menos.