ISCTE IUL Instituto Universitário de Lisboa Escola de Tecnologias e Arquitectura Dep. de Ciências e Tecnologias da Informação	Sistemas Operativos Ano letivo 2018/2019 1º Semestre versão 1.0

Projeto *LyftIUL*

O presente trabalho visa aplicar os conhecimentos adquiridos durante as aulas de Sistemas Operativos. O trabalho será composto de três partes, com o objectivo de desenvolver aspetos da plataforma *LyftIUL*. A Plataforma *LyftIUL* tem como objectivo ligar os estudantes do ISCTE-IUL que têm viatura própria (carro ou mota) com os estudantes que precisam de transporte para uma determinada viagem. Na Plataforma *LyftIUL* existem os seguintes conceitos:

Condutor: (*Driver*) é o estudante que faz o transporte. Um condutor tem um determinado perfil, que inclui a seguinte informação: número de estudante, nome, turma, número de telemóvel, email, tipo de viatura, marca, matrícula, número de viagens efetuadas, pontos e saldo atual.

Passageiro: (*Rider*) é o estudante que precisa de se deslocar de uma determinada *origem* para um determinado *destino*. Um Passageiro tem um determinado perfil, que inclui a seguinte informação: número de estudante, nome, turma, número de telemóvel, e-mail, numero cartão de crédito

Viatura: o meio de transporte utilizado pelo condutor para transportar o passageiro

Pedido: mensagem enviada pelo *Passageiro* a requisitar uma *Viatura* para o transportar naquele momento

Viagem: transporte de um *Passageiro* pelo *Condutor* num determinado período de tempo, com um determinado custo e com a qual foram obtidos pontos, utilizando uma Viatura.

Em termos gerais, quando um *Passageiro* precisa de transporte, utiliza o seu telemóvel para se ligar ao servidor *tigre.iul.lab* e requisitar um *Condutor*. Os condutores activos estão ligados ao servidor *Tigre* a aguardar pedidos.

Constituição dos grupos

O trabalho será realizado em grupos de 2 ou 3 estudantes (só em casos excepcionais podem ser aceites grupos de um estudante). Os grupos devem começar por se registar usando o seguinte formulário:

http://so.dcti.iscte.pt/www/myscripts/Grupos/inscrever_grupos.php

A constituição dos grupos deverá manter-se até ao final do semestre, mas caso se verifiquem alterações, deve ser criado um novo grupo que passará a ser usado em vez do anterior para submeter os trabalhos.

Observação. Não se esqueça da palavra passe, pois será necessária para mais tarde submeter os trabalhos.

Entrega, relatório e auto-avaliação

O trabalho será realizado em três partes, que devem ser entregues ser entregues via eletrónica, através dos seguintes formulários:

Parte 1: http://so.dcti.iscte.pt/www/myscripts/Trabs/submeter_trab01.php
Parte 2: http://so.dcti.iscte.pt/www/myscripts/Trabs/submeter_trab02.php
Parte 3: http://so.dcti.iscte.pt/www/myscripts/Trabs/submeter_trab03.php

Observação. A submissão de um ficheiro substitui a versão anteriormente submetida.

Em cada uma das submissões deverá enviar um ficheiro **zip** contendo os ficheiros criados ou modificados pelo estudante (ex: *.sh , *.txt, *.c, *.h, ...) de acordo com o formato pedido no enunciado, juntamente com o ficheiro *relatório* que é mencionado de seguida. No caso dos programas em C, não se devem incluir os ficheiros binários. Os programas entregues deverão respeitar o nome dos ficheiros mencionados no enunciado. Caso não o façam, o respectivo trabalho sofrerá uma penalização na nota final.

Para cada uma das 3 partes do projeto deverá ser criado um relatório em formato TXT ou PDF com informações sobre a elaboração do projeto. Esse ficheiro deve chamar-se ***relatorio.txt*** ou ***relatorio.pdf***, deverá ter apenas 1 página A4 e deverá ser submetido juntamente com o trabalho.

O projeto será avaliado em termos globais produzindo-se uma nota de projeto, que será distribuída por todos os elementos do grupo, caso estes tenham todos contribuído de igual forma para o projeto. Caso existam diferenças no nível de contribuição, então os estudantes que mais contribuíram poderão ter uma nota individual superior à nota de projeto e o inverso para os estudantes que menos contribuíram. No fundo, parte da nota individual dos estudantes que menos contribuíram pode ser transferida para os estudantes que mais contribuíram, recompensando assim o esforço adicional de quem mais contribuiu. Assim, o grupo deverá indicar no seu relatório uma estimativa de qual a *percentagem de contribuição de cada elemento do grupo para o conjunto das tarefas realizadas (entre 0% e 100%)*, por forma a aferir qual foi a contribuição individual de cada elemento do grupo. É ainda requerido uma justificação sucinta do porquê das diferenças nas percentagens atribuídas aos vários elementos do grupo (caso essas diferenças existam).

É importante que os estudantes que menos contribuíram para o projeto façam questão de garantir que essa menor contribuição fique expressa neste documento, pois, caso contrário, estão a ficar com os créditos dos colegas que mais se envolveram no projeto e, consequentemente, a impedir que as notas individuais desses mesmos colegas possam subir para além da nota de projeto. Portanto, é importante que toda a informação prestada seja realmente representativa do esforço relativo colocado pelos vários elementos, pois, caso contrário, a distribuição de notas poderá não ficar de acordo com as expectativas do grupo. Além disso, a nota final do projeto é atribuída através de uma oral, que acabará por apurar de uma outra forma a contribuição de cada elemento do grupo e que poderá ser comparada com as informações prestadas.

Assume-se que a informação prestada é consensual entre os vários elementos do grupo. Quando isso não for possível, tal facto deverá ficar expresso no documento.

Política em caso de fraude

Os alunos podem partilhar e/ou trocar ideias entre si sobre os trabalhos e/ou resolução dos mesmos. No entanto, o trabalho entregue deve corresponder ao esforço individual de cada grupo. São consideradas fraudes as seguintes situações:

- Trabalho parcialmente copiado
- Facilitar a copia através da partilha de ficheiros
- Utilizar material alheio sem referir a sua fonte.

Em caso de deteção de algum tipo de fraude, os trabalhos em questão não serão avaliados, sendo enviados à Comissão Pedagógica ou ao Conselho Pedagógico, consoante a gravidade da situação, que decidirão a sanção a aplicar aos alunos envolvidos. Serão utilizadas as ferramentas *Moss* e *SafeAssign* para detecção automática de cópias.

Recorda-se ainda que o Anexo I do Código de Conduta Académica, publicado a a 25 de Janeiro de 2016 em Diário da Republica, 2ª Série, nº 16, indica no seu ponto 2 que:

> Quando um trabalho ou outro elemento de avaliação apresentar um nível de coincidência elevado com outros trabalhos (percentagem de coincidência com outras fontes reportada no relatório que o referido software produz), cabe ao docente da UC, orientador ou a qualquer elemento do júri, após a análise qualitativa desse relatório, e em caso de se confirmar a suspeita de plágio, desencadear o respetivo procedimento disciplinar, de acordo com o Regulamento Disciplinar de Discentes do ISCTE - Instituto Universitário de Lisboa, aprovado pela deliberação n.o 2246/2010, de 6 de dezembro.

O ponto 2.1 desse mesmo anexo indica ainda que:

> No âmbito do Regulamento Disciplinar de Discentes do ISCTE- -IUL, são definidas as sanções disciplinares aplicáveis e os seus efeitos, podendo estas variar entre a advertência e a interdição da frequência de atividades escolares no ISCTE-IUL até cinco anos.

Ficheiros

O conjunto de *scripts* e programas irão operar com os seguintes ficheiros.

condutores.txt

Formato: número de estudante: nome: turma: telemóvel: e-mail: tipo de viatura: marca: matrícula: número de viagens: pontos: saldo

Exemplo:

```
88020:Maria Aguiar:EIB2:911555444:mguiar@gmail.com:carro:mercedes:88-TX-11:3:21:12.5
88110:Vasco Rapido:IB2:962222222:vrapid@gmail.com:mota:BWM:33-MT-99:1:5:4.0
...
```

passageiros.txt

Formato: número:nome:turma:telemóvel:e-mail:cartão de crédito

Exemplo:

```
88013:Bruno Silva:IB1:963333111:a88013@iscte-iul.pt:4222-2222-2222-2222
88030:Joana Bolota:ETB2:964444444:a88030@iscte-iul.pt:4444-4444-4444-4444
...
```

viagens.txt

Formato: data:número do estudante condutor:número do estudante passageiro:pontos (inteiro de 0 a 10):valor da viagem (euros)

Exemplo:

```
2018-10-01:88020:88013:6:4.2
2018-10-01:88020:88014:8:3.8
...
```

lyftadmin.pid

Formato: PID

Exemplo:

```
3111
```

Parte I - Shell Script (bash)

Data de entrega: **13** de outubro de 2018

Nesta primeira parte do trabalho, propõe-se que os estudantes desenvolvam a parte de Administração do servidor *LyftIUL*. Para tal deverão desenvolver os seguintes *scripts*.

1) cria_condutor.sh

Este script é executado na secretaria quando um estudante se inscreve como condutor e mostra os documentos necessários (carta de condução, documento único automóvel). O script é executado com os seguintes argumentos: *número de estudante*, *nome*, *turma*, *telemóvel*, *e-mail*, *tipo de viatura (carro ou mota)*, *marca* e *matrícula*. Exemplo:

```
./cria_condutor.sh 88020 "Maria Aguiar" EIB2 911555444 mguiar@gmail.com
    carro mercedes 88-TX-11
```

O script deve acrescentar o condutor em questão ao ficheiro *condutores.txt*, depois de garantir que o esse condutor ainda não consta do ficheiro. Nesta fase, o *número de viagens*, os *pontos* e o *saldo* deverão ficar com o valor 0 (zero).

2) cria_passageiros.sh

Este script deverá ler o ficheiro `/etc/passwd` do servidor *tigre.iul.lab*, transformar os utilizadores que nele constam em Passageiros do sistema *LyftIUL* e registá-los no ficheiro *passageiros.txt*. Como tal, para todos os estudantes, deverá extrair o *número de estudante* (sem a letra "a") e o *nome* e inserir essa informação no ficheiro. O *e-mail* deverá ser criado automaticamente tendo por base o número do estudante e acrescentando "@iscte-iul.pt". Os campos *turma*, *telemóvel*, *cartão de crédito* serão adicionados posteriormente, pelo que nesta fase devem ficar vazios. Exemplo de linhas adicionadas ao ficheiro após a importação:

```
88013:Bruno Silva:::a88013@iscte-iul.pt:
88030:Joana Bolota:::a88030@iscte-iul.pt:
```

3) lista_negra.sh

Com base no ficheiro *condutores.txt*, este script deve:

- identificar todos os condutores que têm um *rating* abaixo dos 5 valores, tendo em conta que o *rating* corresponde à soma dos seus pontos, dividida pelo número de viagens, arrendondada às unidades;
- se houver condutores nessa situação, deve perguntar ao utilizador se deve prosseguir com o envio de um e-mail a esses condutores;
- em caso afirmativo, deve enviar um email a todos esses condutores, informando-os do seu saldo e de que em breve poderão vir a ser excluídos da plataforma.

4) atualiza_condutores.sh

Este script deverá verificar se o ficheiro *viagens.txt* existe.

Caso exista, deverá percorrer todas as suas linhas e, para cada uma delas, deverá atualizar o número de *pontos*, *número de viagens* e *saldo* do correspondente condutor no ficheiro *condutores.txt*.

No final, deverá apagar o ficheiro *viagens.txt* ou eliminar todas as suas linhas.

5) stats.sh

Este script é executado pelo administrador do sistema para obter informações sobre o sistema. Nomeadamente, deve devolver a seguinte informação:

- Número de condutores e de passageiros na plataforma
- Valor total (soma) dos saldos dos condutores
- Top 5 condutores: lista dos 5 condutores com mais *pontos*
- Top 3 condutores com mais *rating*, em que o rating corresponde a dividir a soma dos pontos pelo número de viagens.
- Condutores por curso: estatística com o número de condutores por curso. Por exemplo: IGE: 5, ETI: 12, LEI: 2. Para tal, deve saber interpretar as turmas.
  Nota: pode simplificar o problema considerando apenas três cursos: IGE, ETI, LEI.

6) menu

Este script agrega os restantes, mostrando um menu com as opções:

1. Cria condutor
2. Cria passageiros
3. Lista negra
4. Atualiza condutores
5. Stats
6. Sair

Cada uma das opções anteriores, invoca o respectivo script descrito anteriormente. Na opção 1, as opções deverão ser pedidas ao administrador (user) e injectadas como argumentos do script.

Este script deverá manter-se em ciclo, permitindo realizar multi-operações, até ser escolhida a opção 7, que permite terminar o funcionamento do script.

7) dashboard.sh

Este *script* deverá produzir conteúdo em formato HTML, com base nos ficheiros existentes. O ficheiro resultante deverá estar estruturado de acordo com as seguintes entradas: 1) Condutores; 2) Passageiros; 3) Maus condutores (com *rating* abaixo dos 5 pontos). O código seguinte ilustra uma possível estrutura do ficheiro `HTML`:

```
<html>
<head>
  <meta charset="UTF-8">
  <title>Estatisticas LyftIUL</title>
</head>
<body>
<h1>Condutores</h1>
  88020, Maria Aguiar, viagens: 3, pontos: 21, rating: 8, saldo: 12.5<br>
<h1>Passageiros</h1>
  88013, Bruno Silva, IB1
<h1>Maus condutores</h1>
     88022, Manuel Caracol, viagens: 1, pontos: 3, rating: 3, saldo: 4.2<br>
</body>
</html>
```

Use o script para produzir o ficheiro `~/public_html/stats.html`. Verifique que o ficheiro foi bem produzido colocando o seguinte endereço no seu browser:

```
http://tigre.iul.lab/~axxxxx/stats.html
```

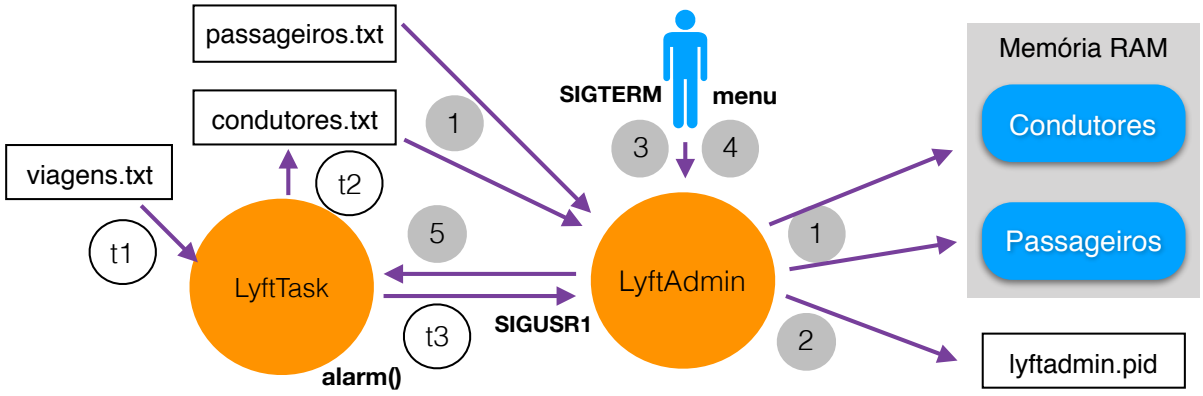
em que axxxxx é o seu username no tigre. Note que a página está disponível apenas dentro do ISCTE ou via VPN.

Deverá configurar o seu "cron", por forma a que este script seja executado automaticamente a cada 10 minutos.

Parte II - Processos e sinais

Data de entrega: **17** de novembro de 2018

Depois de ter desenvolvido as ferramentas de Shell Script que servem de apoio ao sistema *LyftIUL*, nesta parte do trabalho será desenvolvida a estrutura dos módulos **LyftAdmin** e **LyftTask** do sistema, utilizando a linguagem de programação C. Posteriormente, esta versão será ainda completada, no âmbito da parte 3 do trabalho, com conhecimentos sobre IPCs adquiridos em Sistemas Operativos. Considere o seguinte diagrama, que apresenta uma visão geral da arquitectura pretendida:



Nesta fase do trabalho pretende-se tratar dos módulos **LyftAdmin** e **LyftTask**.

LyftTask

O LyftTask é um processo responsável por processar as *viagens* anteriormente ocorridas. Este processo deve contemplar a seguinte sequência de tarefas e após o tratamento das mesmas deve terminar.

(t1) Ler o ficheiro *viagens.txt*.

(t2) Para cada linha do ficheiro *viagens.txt*, deve actualizar os valores da mesma nos campos *número de viagens*, *pontos* e *saldo* do ficheiro *condutores.txt*. Depois de ler toda a infomação do ficheiro, deverá removê-lo.

(t3) Caso tenha sido feita alguma alteração ao ficheiro *condutores.txt* e caso exista o ficheiro *lyftadmin.pid*, deve enviar o sinal "SIGUSR1" ao processo *LyftAdmin*, usando para isso o PID anteriormente escrito no ficheiro *lyftadmin.pid*.

LyftAdmin

O *LyftAdmin* é o processo iniciado pelo utilizador e que é responsável pelas seguintes acções (identificadas no diagrama):

1. Ler os ficheiros *condutores.txt* e *passageiros.txt* e carregar a informação para estruturas de dados em memória. O formato aconselhado para as estruturas de dados é apresentado abaixo:

Memória partilhada *Condutor*	Memória partilhada *Passageiro*

`typedef struct {`
`  int numero;`
`  char nome[50];`
`  char turma[10];`
`  char telemovel[15];`
`  char email[40];`
`  char tipo[20];`
`  char marca[20];`
`  char matricula[15];`
`  int viagens;`
`  int pontos;`
`  float saldo;`
`  int activo;`
`  long disponivel_desde;`
`  int PID;`
`} Tcondutor;`
	`typedef struct {`
	`  int numero;`
	`  char nome[50];`
	`  char turma[10];`
	`  char telemovel[15];`
	`  char email[40];`
	`  char c_credito[20];`
	`} Tpassageiro;`

2. Criar o ficheiro *lyftadmin.pid* e colocar lá o seu PID.

3. O programa *LyftAdmin* deve armar e tratar os seguintes sinais:

   SIGUSR1: sempre que o programa receber este sinal, deve reler o ficheiro *condutores.txt* e atualizar os *pontos, número de viagens e saldo* nas estruturas em memória.

   SIGTERM: sempre que o programa receber este sinal, deve re-escrever os ficheiros *condutores.txt* e *passageiros.txt* com base na informação presente nas estruturas de memória quer estes estejam alteradas ou não. Após a escrita dos ficheiros, deve terminar.

4. Apresentar um menu ao utilizador com as seguintes opções:

   Imprimir memória: deve mostrar informação sobre os condutores e passageiros que se encontram na memória. Deve listá-los e indicar quantos são.

   Alterar passageiro: permite ao utilizador alterar um passageiro. Para tal, o número de estudante é pedido ao utilizador e se o mesmo existir em memória, os campos podem ser novamente re-introduzidos.

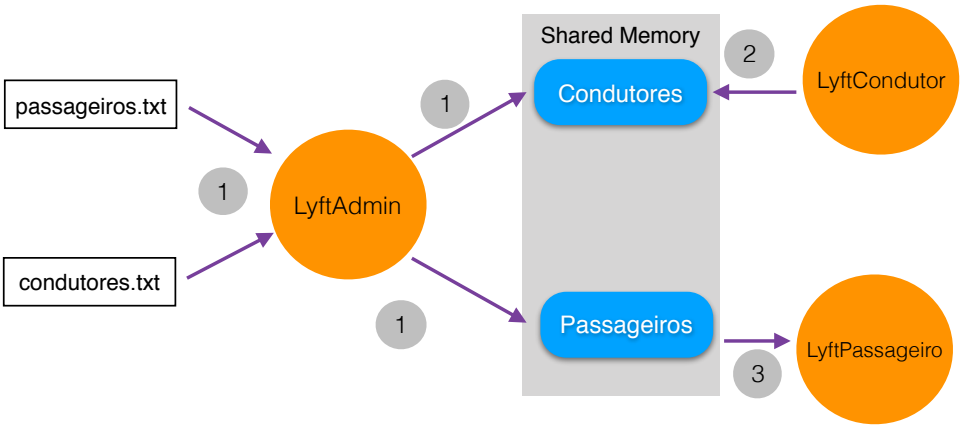
   Alterar condutor: permite ao utilizador alterar dados de condutor. Para tal, o número de estudante é pedido ao utilizador e se o mesmo existir em memória, os campos podem ser novamente re-introduzidos (excepto os campos *viagens*, *pontos* e *saldo*)

5. Através da chamada de sistema *alarm()*, de minuto a minuto é criado um novo processo filho do *LyftAdmin* e dentro do mesmo é executado o programa "*LyftTask*".

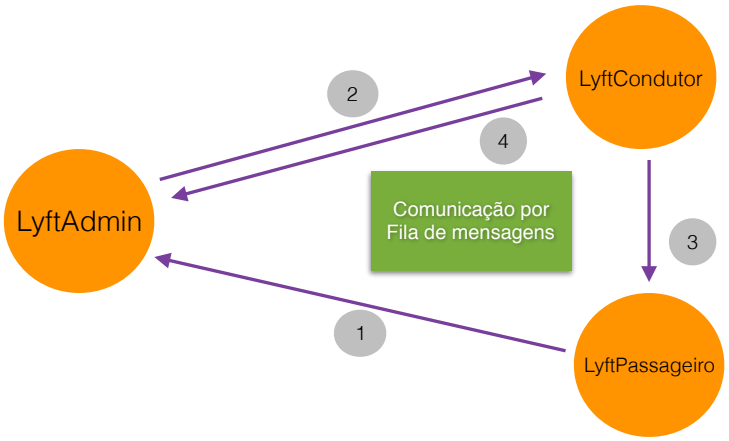
Parte III - Ficheiros e IPCs

Data de entrega: **15** de dezembro de 2018

Nesta fase do trabalho pretende-se que os estudantes exercitem a comunicação entre processos (IPCs) e outros mecanismos de comunicação e sincronização disponibilizados pelo Sistema Operativo. Assim, o código do programa *LyftAdmin* já entregue na parte 2 do trabalho será adaptado a novas funcionalidades e serão criados dois novos programas, *LyftCondutor e LyftPassageiro*.



O diagrama acima apresenta uma visão geral da arquitectura pretendida, envolvendo esses três módulos. O *LyftAdmin* é um processo iniciado pelo administrador e permite fazer a gestão de serviços. Este processo começa por (1) ler os ficheiros *condutores.txt* e *passageiros.txt* e carregar o seu conteúdo para as *memórias partilhadas* respectivas, que por sua vez podem ser utilizadas pelo *LyftCondutor* e pelo *LyftPassageiro*. O programa *LyftCondutor* é utilizado pelos condutores para receberem pedidos de transporte e (2) pode ler e escrever na memória partilhada *Condutores*. O *LyftPassageiro* é iniciado por um estudante que deseja fazer uma viagem e (3) pode ler o conteúdo da memória partilhada *Passageiros*.



Os três programas utilizam uma única fila de mensagens para comunicar entre eles de forma assíncrona. O diagrama acima resume a troca de mensagens, que funciona da seguinte forma: sempre que um passageiro pretende efetuar uma viagem envia uma mensagem ao *LyftAdmin* (1). O *LyftAdmin* contacta o respetivo condutor para efetuar o serviço (2). Esse condutor responde diretamente ao passageiro (3). No final da viagem o condutor

envia um recibo, informando o *LyftAdmin* sobre o custo da viagem e sobre os *pontos* que o passageiro lhe atribuiu através de um painel existente na viatura (4).

As estruturas de dados do sistema *LyftIUL* devem seguir o formato anteriormente definido. O campo *disponivel_desde* irá conter uma marca temporal em formato *UNIX time*, que correponde ao número de segundos decorridos desde 1 de Janeiro de 1970 e que pode ser obtido com a função `time()`. As mensagens trocadas são todas do tipo *MsgViagem* podendo ter ou não os campos preenchidos.

Estrutura das mensagens trocadas

```
typedef struct {
   long tipo;
   struct {
      int pid_passageiro;
      int pid_condutor;
      char local_encontro[100];
      long data;
      float pontos;
      float valor;
   } dados;
} MsgViagem;
```

Nota: os programas devem utilizar semáforos sempre que necessário, especialmente quando são utilizadas memórias partilhadas, por forma garantir exclusão mútua e evitar colisões.

LyftAdmin

Este programa deverá ser alterado para utilizar *memórias partilhadas*, de forma a partilhar os seus dados com os restantes processos tal como indicado no diagrama 1. Deve verificar se as *memórias partilhadas* já se encontram criadas. Caso não estejam, deve ler os ficheiros *condutores.txt* e *passageiros.txt* e carregar o seu conteúdo nas *memórias partilhadas* respectivas. De seguida deve apresentar o seguinte menu:

Imprimir memória: deve mostrar informação sobre os condutores e passageiros que se encontram nas *memórias partilhada*s. Deve listá-los, indicar quantos são e quantos condutores se encontram ativos.

Alterar passageiro: permite ao utilizador alterar informação sobre um passageiro. Para tal, o número de estudante é pedido ao utilizador e se o mesmo existir em memória, os campos *turma*, *telemovel, email, c_credito* podem ser novamente introduzidos

Alterar condutor: permite ao utilizador alterar dados de condutor. Para tal, o número de estudante é pedido ao utilizador e se o mesmo existir em memória, os campos podem ser novamente re-introduzidos (excepto os campos *pontos, viagens* e *saldo*)

O programa *LyftAdmin* vai ainda ter de monitorizar *fila de mensagens* "Viagens". Para tal, deverá criar um processo filho que será responsável por estar bloqueado à espera de mensagens de tipo "1". Tal como se mostra no diagrama 2, as mensagens podem ser enviadas por passageiros, contendo pedidos de transporte (mensagem 1), ou por condutores, contendo recibos (mensagem 4). O *LyftAdmin* deve perceber que tipo de mensagem é e agir em conformidade.

- Um pedido de viagem (mensagem 1) reconhece-se pelo facto do PID do passageiro estar preenchido e o PID do condutor não.
  - Deve ir à *memória partilhada*, identificar o condutor activo (enable igual a "1") com a marca temporal *disponivel_desde* (em formato UNIX time) mais baixo e atribuir-lhe a viagem.
  - Para lhe atribuir a viagem: 1) coloca o condutor inativo; 2) envia uma mensagem (mensagem 2) contendo o *PID* do passageiro e o *local_encontro*, com o tipo igual ao PID do condutor.
- Os recibos dos condutores (mensagem 4) reconhecem-se por terem PID de passageiro, PID de condutor e um valor.
  - Deve ir à *memória partilhada* e aumentar o saldo do respectivo condutor pelo *valor* que consta na mensagem. Deve também somar os pontos recebidos aos pontos já constantes da memória partilhada.
  - Deverá ainda colocar a marca temporal *disponivel_desde* com a data actual (em formato UNIX time) na estrutura do condutor respectivo e colocar o condutor como ativo.

LyftPassageiro

- No arranque deverá pedir o número de estudante do passageiro, validá-lo usando a memória partilhada *Passageiros* e mostrar o respetivo nome do estudante.
- De seguida deverá perguntar o local de encontro e enviar uma mensagem (mensagem 1) para o *LyftAdmin*, preenchendo os seguintes campos da estrutura *MsgViagem*
  - **tipo**: preencher a "1" que é o tipo das mensagens enviadas ao servidor
  - **pid_passageiro**: o seu PID
  - **local_encontro**: a morada do sítio onde está
  - **data**: a data actual em UNIX time
- Após o envio, fica à espera de uma mensagem (mensagem 3) cujo tipo é o seu PID. Essa mensagem é enviada pelo próprio condutor e contém informação sobre o mesmo. Quando esta mensagem chegar, imprime no ecrã o PID do condutor (para garantir que é a boleia certa) e sai do programa.

LyftCondutor

Quando é iniciado, deve pedir o número de condutor, validá-lo usando a *memória partilhada* "condutores" e alterar os seguintes campos nessa memória partilhada:

- **activo**: colocar a "1", para dizer que está activo
- **disponivel_desde**: colocar a data actual em UNIX time
- **PID**: o seu PID

De seguida deverá ficar à escuta de mensagens para si (tipo igual ao seu PID) na fila "viagens". Ao receber uma mensagem (mensagem 2) vinda do *LyftAdmin*, deverá:

- Imprimir o PID do passageiro no ecrã
- Enviar uma mensagem (mensagem 3) para o passageiro com os campos seguintes preenchidos:
  - **tipo**: com o PID do passageiro recebido na mensagem.
  - **pid_condutor**: o seu próprio PID
  - **local_encontro**: a morada do sitio onde está (para o passageiro ter uma ideia de quanto tempo demorará)
- De seguida, para simular o seu serviço (viagem), deverá agardar um número aleatório de segundos, entre 1 a 20 segundos.
- No final da viagem, deverá ainda pedir ao utilizador que introduza o valor da viagem e os pontos (0 a 10) e enviar uma nova mensagem (mensagem 4) com o recibo, que será recebida pelo *LyftAdmin:*
  - **tipo**: 1, que é o tipo das mensagens enviadas ao servidor
  - **pid_condutor**: o seu próprio PID
  - **pid_passageiro**: o PID que anteriormente recebeu
  - **pontos**: o número de pontos que recebeu
  - **valor**: valor da viagem

A combinação de teclas *Ctrl+C,* que corresponde ao sinal SIGINT, deverá ser tratada. Ao ser recebida, deve indicar que já não se encontra de serviço, colocando campo *activo* a 0 na *memória partilhada,* e sair.