



## 1. Introdução

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

O objetivo geral do projeto será o desenvolvimento de uma aplicação em C para simular um sistema de pedidos e respostas, com três tipos de participantes diferentes: clientes, restaurantes e motoristas. Esta aplicação, chamada **MagnaEats**, simula uma aplicação de encomenda e entrega de comida ao domicílio.

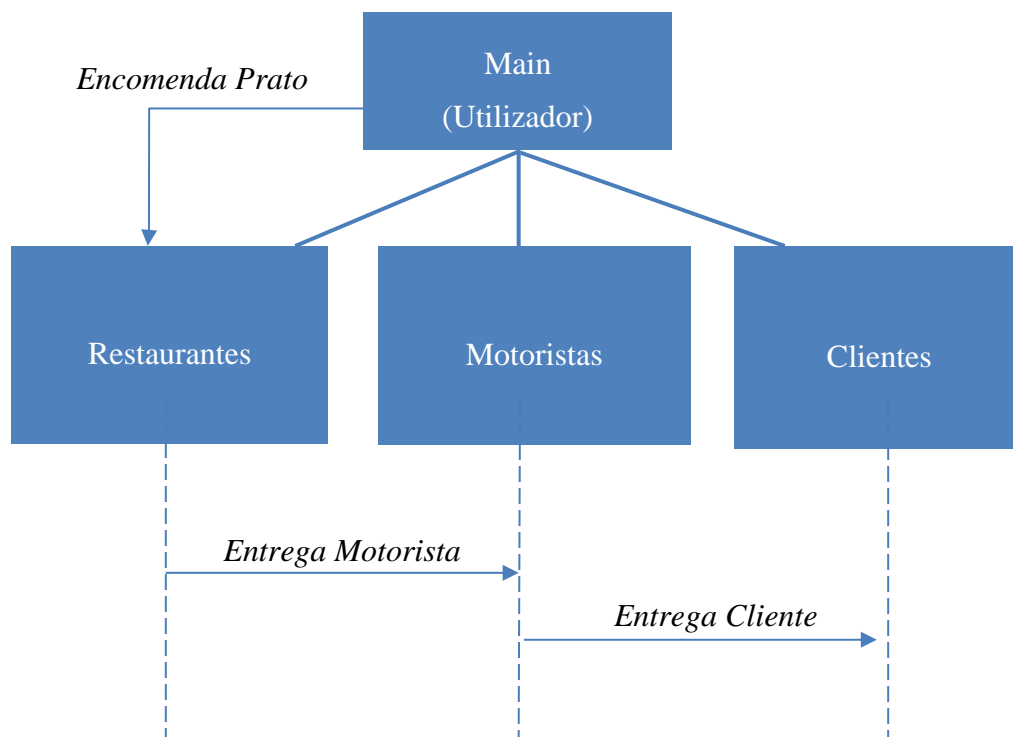
O **MagnaEats** será realizado em 2 fases. A primeira fase tem como objetivo fundamental a familiarização com a linguagem C, a criação do ficheiro `makefile` (para compilação e manutenção do projeto), a gestão de processos e a alocação de memória. Neste primeiro enunciado é feita uma apresentação geral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

## 2. Funcionamento Geral

Como referido acima, no modelo do sistema do **MagnaEats**, existem três tipos de participantes: clientes, restaurantes e motoristas. Em cada execução do **MagnaEats** pode haver um ou mais participantes de cada tipo, conforme configurado pelo utilizador do sistema. Para além desses três participantes, existe também o processo principal (processo pai), i.e., o *Main*, que gere todos os outros processos (filhos) e a interação com o utilizador. O *Main* oferece um menu com opções para que o utilizador possa interagir com **MagnaEats**.

O menu contém quatro opções:

- (1) `request <cliente> <restaurante> <dish>` – criar um pedido de encomenda ao restaurante *restaurant*, feito pelo cliente *client*, pedindo o prato *dish*;
- (2) `status <id>` – consultar o estado do pedido especificado por *id*;
- (3) `stop` – terminar a execução do sistema **MagnaEats**;
- (4) `help` – mostrar as informações de ajuda sobre as opções anteriores.



**Figura 1:** Visão geral do sistema *MagnaEats* e interações entre os participantes.

Quando se pretende efetuar um pedido (opção request), o utilizador pede à *Main* (através de uma interface linha de comandos) para criar uma operação, especificando o seu *id* de cliente, o *id* do restaurante pretendido, e o prato que pretende encomendar. A operação é então enviada para o restaurante pretendido, que após preparar a refeição, adiciona o seu *id* à operação (confirmando que foi mesmo ele que processou a operação) e envia a mesma para um motorista disponível. No caso dos motoristas, qualquer motorista pode entregar qualquer encomenda. O motorista selecionado adiciona o seu *id* à operação e entrega a encomenda no cliente. Por fim, o cliente confirma a receção da encomenda e finaliza-a, adicionando, para tal, o seu *id* à operação no campo respetivo.

O modelo de interação entre os três participantes é o modelo produtor/consumidor. Um pedido em curso transita entre quatro estados que indicam em que estado o seu processamento se encontra. Os estados são: 'I' - pedido iniciado pela *Main*; 'R' - pedido processado pelo restaurante; 'D' - pedido processado por um motorista (*driver*), e 'C' pedido finalizado e recebido no cliente.

A ação de consulta de estado de um pedido (opção status), por sua vez, verifica se a resposta a um pedido já chegou e consulta o seu estado e o identificador do cliente que fez o pedido, restaurante requisitado e prato pretendido, *id* do restaurante que efetivamente tratou o pedido, *id* do motorista que fez a entrega, e *id* do cliente que recebeu a encomenda.

Por fim, a ação para terminar a execução do sistema *MagnaEats* (opção stop) imprime as estatísticas finais do mesmo, como o número de pedidos processados por cada cliente, restaurante e motorista, e termina o programa.

### 3. Descrição específica

A Fase 1 do projeto consiste na concretização (i.e., programação) dos seguintes módulos:

- *Main* (main.c e main.h – main.c/h), módulo principal que gere os outros módulos e faz a interação com o utilizador.
- Gestão de processos (process.c/h), com funções para criar e destruir processos (ex., Restaurante, Motorista e Cliente).
- Gestão de memória (memory.c/h), com funções para criação de memória dinâmica e de memória partilhada, assim como funções para escrita em diferentes tipos de estruturas de dados (ex., buffers circulares e de acesso aleatório).
- Restaurantes (restaurant.c/h), com funções para receber pedidos da *Main* e encaminhar para os Motoristas.
- Motoristas (driver.c/h), com funções para receber pedidos de Restaurantes e encaminhar os mesmos para os Clientes.
- Clientes (client.c/h), com funções para receber os pedidos dos motoristas, processá-los e enviar para a *Main* estatísticas dos mesmos.

Para cada um destes módulos, é fornecido um ficheiro com extensão *.h* com os cabeçalhos das funções, e que **não pode ser alterado**. As concretizações das funções definidas nos ficheiros *X.h* (em que *X* é o nome do ficheiro) terão de ser desenvolvidas pelo grupo de trabalho nos ficheiros *X.c*, utilizando os algoritmos e métodos que o grupo achar convenientes. Se o grupo entender necessário, ou se for pedido, também pode criar um ficheiro de cabeçalho *X-private.h* para acrescentar outras definições, cujas implementações serão também incluídas no ficheiro *X.c*.

Juntamente com o enunciado da Fase 1 do projeto é disponibilizado, aos alunos, um ficheiro zip contendo todos os cabeçalhos definidos para os módulos acima apresentados.

Para auxiliar no início do desenvolvimento do projeto, é também fornecida uma função *main* que os alunos podem usar:

```
int main(int argc, char *argv[]) {
    //init data structures
    struct main_data* data = create_dynamic_memory(sizeof(struct
main_data));
    struct communication_buffers* buffers =
create_dynamic_memory(sizeof(struct communication_buffers));
    buffers->main_rest = create_dynamic_memory(sizeof(struct
rnd_access_buffer));
    buffers->rest_driv = create_dynamic_memory(sizeof(struct
circular_buffer));
    buffers->driv_cli = create_dynamic_memory(sizeof(struct
rnd_access_buffer));

    //execute main code
    main_args(argc, argv, data);
    create_dynamic_memory_buffers(data);
    create_shared_memory_buffers(data, buffers);
    launch_processes(buffers, data);
    user_interaction(buffers, data);

    //release memory before terminating
    destroy_dynamic_memory(data);
    destroy_dynamic_memory(buffers->main_rest);
    destroy_dynamic_memory(buffers->rest_driv);
    destroy_dynamic_memory(buffers->driv_cli);
    destroy_dynamic_memory(buffers);
}
```

#### 4. Estrutura do projeto e makefile

O projeto deve ser organizado segundo a seguinte estrutura:

```
\MAGNAEATS
    \bin
    \include
    \obj
    \src
```

O diretório `bin` deverá conter o executável `magnaeats`. O diretório `include` deverá conter os ficheiros `.h` com a definição das estruturas de dados e declarações de funções. **Estes ficheiros não podem ser alterados** (à exceção dos ficheiros `-private.h`). O diretório `obj` deverá conter os ficheiros objeto gerados (ficheiros `.o`) pela execução do `makefile`. Por fim, o diretório `src` deverá conter os ficheiros fonte (ficheiros `.c`) com o código desenvolvidos pelos alunos.

O executável `magnaeats` será gerado a partir da execução do comando `make`. O `makefile` necessário para a execução do comando `make` será desenvolvido pelos alunos e deve ser colocado na raiz do diretório `MAGNAEATS`.

#### 5. Desenvolvimento e Testes

Esta fase do projeto inclui vários módulos e funções a desenvolver. Como tal, recomenda-se que os alunos as desenvolvam ao longo do semestre, após ter sido lecionada a matéria respetiva nas aulas TP, não devendo deixar a realização do mesmo para a última semana da data de entrega.

Para efeitos de teste, é fornecido aos alunos o executável `magnaeats_profs`, desenvolvido pelos professores da disciplina e compilado numa máquina virtual com a distribuição Linux instalada nos laboratórios de aula da FCUL. É esperado que, tendo sido introduzidos os mesmos argumentos, tanto o executável desenvolvido pelos alunos como o desenvolvido pelos professores devolvam resultados semelhantes.

Exemplo de utilização do executável `magnaeats` (e do `magnaeats_profs`):

```
$/magnaeats max_ops buffers_size n_restaurants n_drivers n_clients
```

onde `max_ops` é o número máximo de pedidos (encomendas) que podem ser criados, `buffers_size` é o tamanho máximo dos buffers, `n_restaurants` é o número máximo de Restaurantes, `n_drivers` é o número máximo de Motoristas e `n_clients` é o número máximo de clientes.

Recomenda-se aos alunos que comecem os testes com um número reduzido de operações e processos (ex., 10 pedidos, 1 restaurante, 1 motorista e 1 cliente). Caso o código funcione com este exemplo, então aumentem e testem com mais processos e operações.

De notar que nesta 1ª fase do projeto, e no caso de haver vários motoristas, pode acontecer que mais que um motorista receba e tente entregar a mesma encomenda. Este problema demonstra a necessidade de efetuar sincronização entre processos, o qual será resolvido na 2ª fase do projeto.

Depois do arranque da aplicação, o utilizar deve interagir com o sistema por meio de um menu com as opções `request`, `status` e `stop` que foram explicadas na Seção 2. Deve também ser possível por meio da ação `help` imprimir informações sobre as ações disponíveis.

## 6. Entrega

A entrega da primeira fase do projeto tem de ser feita de acordo com as seguintes regras:

1. Colocar todos os ficheiros do projeto, de acordo com a estrutura apresentada na Seção 4, **bem como um ficheiro README** onde os alunos podem incluir informações que julguem necessárias (ex., nome e número dos alunos que o desenvolveram, limitações na implementação do projeto, etc.), num ficheiro comprimido no formato ZIP. O nome do ficheiro será **grupoXX-projeto1.zip** (XX é o número do grupo).
2. Submeter o ficheiro **grupoXX-projeto1.zip** na página da disciplina no moodle da FCUL, utilizando a atividade disponibilizada para tal. Apenas um dos elementos do grupo deve submeter, considerando que será escolhida aleatoriamente uma submissão no caso de existirem várias.

Na entrega do trabalho, é ainda necessário ter em conta que:

- **(1) se não for incluído um Makefile e (2) se o mesmo não compilar os ficheiros fonte, ou (3) se houver erros de compilação (isto é, se não forem criados os ficheiros objeto e executáveis), o trabalho é considerado nulo.**
- Todos os ficheiros entregues devem começar com um cabeçalho com três ou quatro linhas de comentários indicando o número do grupo, o nome e número dos seus elementos.
- Os programas são testados no ambiente Linux instalado nos laboratórios de aulas, pelo que se recomenda que os alunos desenvolvam e testem os seus programas nesse ambiente. A imagem Linux instalada nos laboratórios pode ser descarregada de <https://admin.di.fc.ul.pt/importacao-da-imagem-para-virtualbox-tutorial/>

Se não se verificar algum destes requisitos o trabalho é considerado não entregue.

**Não serão aceites trabalhos entregues por mail nem por qualquer outro meio não definido nesta secção.**

## 7. Prazo de entrega

O trabalho deve ser entregue até dia **10 de abril de 2022 (domingo) às 23:59h**. Após esta data, a submissão do trabalho através do Moodle deixará de ser permitida.

## 8. Avaliação dos trabalhos

A avaliação do trabalho será realizada:

- (1) pelos alunos, pelo preenchimento do formulário de contribuição de cada aluno no desenvolvimento do projeto. O formulário será disponibilizado no Moodle e preenchido após a entrega do projeto.
- (2) pelo corpo docente sobre um conjunto de testes.

Para além dos testes a efetuar, os seguintes parâmetros serão avaliados: funcionalidade, estrutura, desempenho, algoritmia, comentários, clareza do código, validação dos parâmetros de entrada e tratamento de erros.

## 9. Divulgação dos resultados

A data prevista da divulgação dos resultados da avaliação dos trabalhos é 06 de maio de 2022.