MIEEC – Sistemas Operativos 2018/2019

Trabalho Prático
Gestão de Encomendas

1. Objetivos do trabalho

- Desenvolver um simulador de um sistema de entrega de encomendas baseado em geolocalização. Servirá para coordenar os processos de encomendas de um sistema composto por drones e armazéns.
- Explorar mecanismos de gestão de processos, *threads*, comunicação e sincronização em Linux.

2. Contextualização

O sistema a ser desenvolvido deve simular um ambiente delimitado com coordenadas geoespaciais (x,y), onde existem um conjunto de armazéns, para além de uma equipa de distribuição constituída por uma central de comando, quatro bases e vários drones geolocalizados. Os armazéns têm posição fixa, enquanto que os drones se deslocam pelo espaço procurando responder a encomendas, nunca saindo dos limites da área definida. A Fig. 1 representa uma visualização do problema, descrito abaixo.

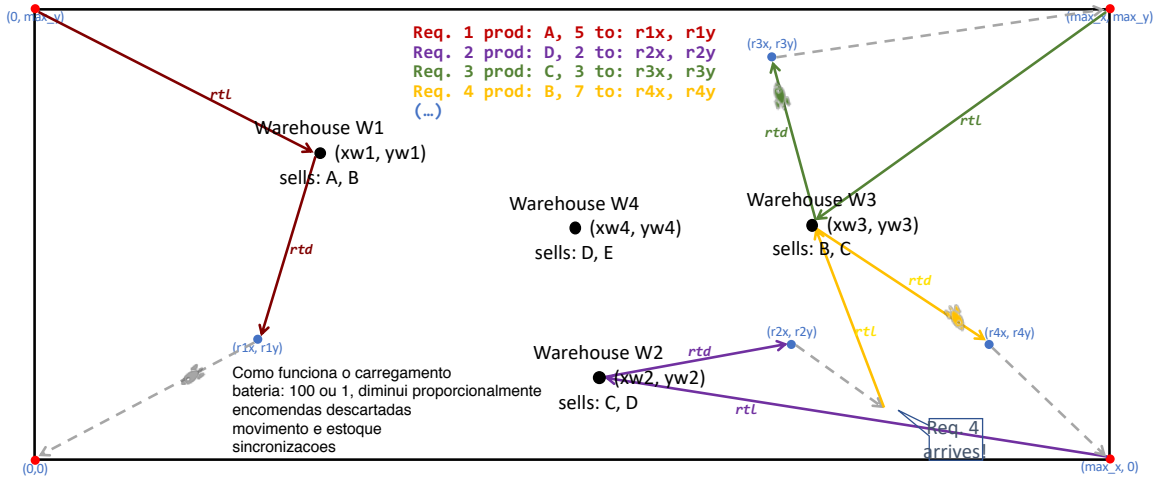

Figure 1 – Armazéns e drones nos seus percursos.

Cada armazém é caracterizado por um *nome*, pelas *coordenadas da sua localização*, pelos *três tipos de produtos que tem para vender* e pela *quantidade existente* de cada um. Os armazéns são abastecidos periodicamente, recebendo de cada vez apenas produtos de um dos tipos, como detalhado em 4.3.
Os drones são coordenados pelo processo *Central*, ao qual as encomendas vão chegando. Cada encomenda é caracterizada por um *número sequencial único*, *tipo de produto*, *quantidade* e *coordenadas do destino de entrega*. Existem quatro bases de drones. Para cada encomenda, a *Central* escolhe o drone desocupado com bateria suficiente, e com o caminho mais curto entre si e o destino da entrega, passando por um armazém que tenha o produto encomendado disponível na quantidade desejada.
Após ser escolhido pela *Central*, o drone desloca-se em linha reta até ao armazém (identificados com *rtl* na Fig. 1) e inicia o processo de carregamento dos materiais.

Seguidamente o drone movimenta-se até ao destino da entrega (*rtd* na Fig. 1), onde descarrega os produtos. Após terminar, dirige-se para a base mais próxima da sua posição (cinzento tracejado na Fig. 1). Deve, no entanto, interromper o regresso se lhe for atribuída uma nova encomenda, a qual deve começar imediatamente a atender (como no caso do Req. 4, a amarelo na Fig. 1). Cada drone tem a sua própria bateria que gasta quando em deslocação e recarrega na base.

3. Visão geral do funcionamento da aplicação

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

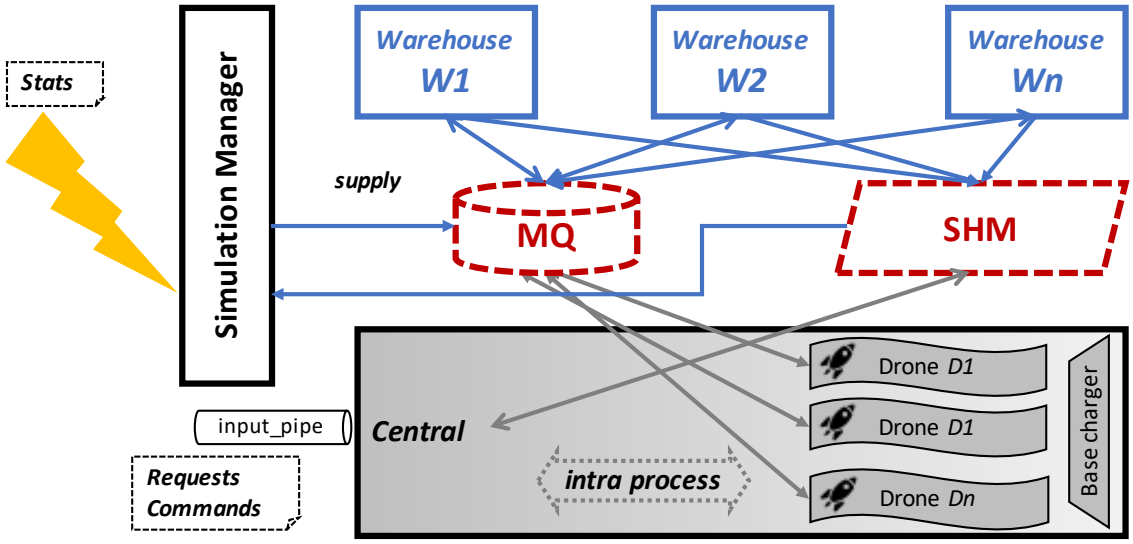

Figure 2 – Visão geral do sistema a implementar.

Como a figure mostra, o sistema é baseado em vários processos, entre os quais:

- **Gestor da simulação (*Simulation manager*)** – é responsável por iniciar todo o sistema. Deve criar a fila de mensagens (MQ), as zonas de memória partilhada (*SHM*) e os restantes processos. Ao receber um sinal SIGUSR1 deve ler as estatísticas a partir da SHM e apresentá-las ao utilizador.
- **Processo Central** (***Central***) – iniciar, coordenar e manter as *threads* relativas a cada um dos drones, e ao carregador de baterias (charger). Deve também criar o *named pipe* ("*input_pipe*" da Fig. 2) onde vai receber as encomendas e outros comandos relativos ao comportamento dos drones.
- **Processos Armazém (*Warehouse*)** – cada armazém é representado por um processo individual, criado pelo processo *Gestor da simulação*. Cada um mantém a memória partilhada (SHM) atualizada relativamente ao seu *stock* de produtos e monitoriza a fila de mensagens para perceber quando chega algum drone ou abastecimento.

4. Descrição das funcionalidades a implementar

4.1. Chegada das encomendas e funcionamento da Central

É responsabilidade do processo *Central* criar e manter as *threads* correspondentes aos *drones* (Dx na Fig. 2) e *charger*, bem como coordenar o trabalho deles. Para isso, deve

criar as estruturas intra processo necessárias para saber a cada momento se os drones se encontram ocupados/desocupados, saber a carga de bateria e sua localização e atribuir tarefas.
A *Central* monitoriza ao mesmo tempo o *named pipe* ("*input_pipe*" Fig. 2) para receber as tarefas a realizar. Estas tarefas são expressas por comandos que podem servir para criar uma encomenda nova e para alterar o número de drones. **A estrutura dos comandos a receber é apresentada abaixo e deve ser seguida rigorosamente.**

```
# estrutura do comando para criar encomenda
# ORDER order name prod: product name, quantity to: x, y
ORDER Req 1 prod: A, 5 to: 300, 100
ORDER Req 2 prod: D, 2 to: 600, 100
ORDER Req 3 prod: C, 3 to: 900, 700

# comando para alterar o número de drones
DRONE SET 20
```

Quando o processo *Central* recebe uma encomenda, deve:

1. Validar a sintaxe e a correção dos comandos (e.g. verificar se o tipo de produto existe e se as coordenadas são válidas). Os comandos inválidos devem ser registados no ficheiro de *log* (ver 4.5) como erróneos e descartados.
2. Atribuir-lhe um número único sequencial (ORDER_NO), que vai acompanhá-la durante o tratamento.
3. Verificar os armazéns que têm o produto e quantidade necessária para a encomenda.
4. Avaliar a distância entre cada um dos drones com bateria suficiente e desocupados ao destino da entrega, passando por um armazém que contenha o tipo e quantidade de produtos desejados (i.e., a distância rtd+rtl na Fig. 1), escolhendo o drone com a distância total a percorrer mais curta. Para isso deve consultar na memória partilhada as coordenadas e o stock de cada armazém.
5. Caso existam drones disponíveis e os produtos necessários em *stock*, deve reservar a quantidade de produtos no armazém através de memória partilhada (SHM na Fig. 2) e de seguida notificar a *thread* correspondente a esse drone através de meios de comunicação intra processo, para que este trate da encomenda (ver 4.2). A reserva não deve ser efetuada diminuindo a quantidade de stock disponível no armazém.

Caso não haja um drone disponível, ou não existam produtos disponíveis, a Central deve descartar o pedido e escrever para o log o pedido, identificando a razão pela qual foi descartado. No entanto, se houver drones disponíveis, mas com energia insuficiente, a tarefa deve ser atribuída ao drone que seja capaz de a terminar o mais cedo possível considerando o tempo de carregamento, e este deverá realizá-la quando tiver energia suficiente para o fazer e regressar à base.
Quando a *Central* recebe um comando de alteração do número de drones deve-o fazer sem interromper nenhuma encomenda em curso. Os novos drones devem ser distribuídos sequencialmente pelas bases, usando o algoritmo *round robin*.

4.2. Funcionamento dos Drones

Como dito anteriormente, o comportamento de um drone é implementado por uma *thread* (*Dx* na Fig. 2). Existem em cada momento um número D de drones no sistema, de acordo com as configurações. Os drones começam em repouso com a bateria inicial

definida nas configurações, distribuídos pelas bases existentes, uma base em cada canto do mapa (representadas a vermelho na Fig. 1).

Cada drone é caracterizado pela sua *posição atual*, que vai registando nas estruturas intra processo criadas pela *Central*, por um *número único* que o identifique, pelo seu *estado* (ocupado/desocupado), pela sua *bateria*, e pelo seu *destino* (caso exista), o que leva a que assuma um dos seguintes comportamentos:

1) **Repouso** – o drone encontra-se numa das bases sem se movimentar até que receba ordem para tal. O drone está a recarregar, a sua bateria aumenta **5** pontos por unidade de tempo, até ao máximo definido nas configurações, sendo a *thread charger* responsável por isto. Está desocupado, pode receber uma encomenda para tratar através de estruturas intra processo, assumindo o comportamento **2)**.
2) **Deslocação para carregamento** – o drone desloca-se para o armazém que lhe foi atribuído, movimentando-se a cada unidade de tempo usando a função `move_towards`. Gasta uma unidade de bateria por cada movimento. Está ocupado, não pode ser interrompido. Quando chega ao destino, assume o **3)**.
3) **Carregamento** – o drone começa por enviar uma mensagem ao armazém através da fila (MQ na Fig. 2) a indicar que chegou e que o carregamento pode começar. A mensagem enviada deve identificar a encomenda com um número único, para que o armazém responda com uma mensagem desse tipo. O drone fica à espera da resposta e está ocupado, não podendo ser interrompido. Não gasta bateria neste processo. O carregamento deverá demorar um número de unidades de tempo igual à quantidade de produtos. Quando receber a resposta sabe que o carregamento terminou e assume o comportamento **4)**.
4) **Deslocação para entrega** – semelhante ao comportamento **2)**, mas em direção ao destino da entrega. Gasta uma unidade de bateria por cada movimento. A entrega demora uma unidade de tempo a realizar. Após terminar a entrega assume o comportamento **5)**.
5) **Retorno à base** – o drone desloca-se até à base mais próxima, movimenta-se a cada unidade de tempo usando a função `move_towards`. Está desocupado e gasta uma unidade de bateria por cada movimento. Se a meio do retorno à base receber uma encomenda para tratar, assume o comportamento **2)**.

Será necessário registar, em cada um dos momentos acima descritos, os dados importantes para mais tarde obter a informação estatística (detalhado em 4.4).

Nota: A função `move_towards` é fornecida em conjunto com este enunciado.

4.3. Comunicação com o armazém e transferência dos produtos

Cada armazém é representado por um processo, criado pelo processo *Gestor da simulação*. Quando estes processos são criados é-lhes atribuído um identificador sequencial único (W_NO). O armazém é responsável por atualizar na memória partilhada todos os produtos que disponibiliza e a quantidade de cada um em *stock*, valores estes que são lidos pela *Central* no seu processo de decisão. Cada armazém fica também à escuta da fila de mensagem por mensagens destinadas ao seu identificador (W_NO).

Ao receber uma mensagem da parte de um drone (relativa a uma encomenda), o armazém deve:

1. Interpretar esta mensagem obtendo o tipo de produtos, a quantidade e o identificador para o qual responder.
2. Processar a mensagem durante o número de unidades de tempo correspondentes à quantidade de produtos pedidos na encomenda. Cada armazém só processa um pedido de cada vez.
3. No fim do carregamento da encomenda deve avisar o drone pela fila de mensagens, usando o identificador da encomenda incluído na mensagem recebida.
4. Concluída a encomenda, deve atualizar o seu *stock* na memória partilhada, assim como a quantidade de produtos reservados.

O armazém pode também receber abastecimento de produtos. Na prática, o processo *Gestor da simulação* envia a cada S unidades de tempo (definido na configuração) uma mensagem (através da MQ) para um armazém, com um tipo de produto e com uma quantidade Q (definido nas configurações). O armazém deve ser escolhido sequencialmente (*round robin*) e o produto deve ser escolhido aleatoriamente de entre os produtos que o armazém disponibiliza. Quando o armazém recebe estas mensagens deve atualizar o seu *stock*.

O armazém deve atualizar em todas estas operações as estatísticas mantidas em memória partilhada, de forma a permitir as funcionalidades enumeradas abaixo.

4.4. Informação sobre estatísticas

Pretendem-se manter estatísticas relativas ao funcionamento do sistema. Estas estatísticas devem ser mantidas na memória partilhada (SHM), de forma a que possam ser atualizadas tanto pela central, como pelos drones e armazéns.

Tal como a Fig. 2 ilustra, o processo *Gestor da simulação* é capaz de obter informação sobre encomendas aceites e tratadas pelos drones e armazéns, através da memória partilhada. Após a receção de um sinal do tipo SIGUSR1, deverá escrever para o écran a seguinte informação estatística:

- Número total de encomendas atribuídas a drones;
- Número total de produtos carregados de armazéns;
- Número total de encomendas entregues;
- Número total de produtos entregues;
- Número total de encomendas descartadas;
- Tempo médio para conclusão de uma encomenda (desde que é recebida a ordem pela *Central* até chegar ao destino da entrega);

Ao receber um sinal do tipo SIGUSR2 deverá escrever a mesma informação para o ficheiro de *logs*, limpando de seguida todas as estatísticas.

4.5. Output da aplicação e logs

Todo o *output* da aplicação deve ser apresentado de forma legível na consola e também em num ficheiro de texto (ficheiro de *log*, mencionado em outras partes deste enunciado).
Deverá pôr no *log* os seguintes eventos acompanhados da sua data e hora:

- Inicio e fim do programa;
- Criação e fim de cada um dos processos armazém (inclua o PID do processo);
- Alteração nas configurações;
- Abastecimento de um armazém;
- Receção de nova encomenda (escrevendo o nome e o número atribuído);
- Entrega de um novo pedido a um drone, identificando o número do pedido e o drone que irá fazer o transporte;
- Encomenda suspensa por falta de *stock*;
- Data de entrega de uma encomenda no destino, identificando o número do pedido e o drone que fez o transporte.

Exemplo do ficheiro de *log*:

```
18:00:23 Encomenda Req1-34 recebida pela Central
18:00:24 Encomenda Req1-34 suspensa por falta de stock
18:00:30 Armazém 2 recebeu novo stock
18:00:30 Encomenda Req1-34 enviada ao drone 2
18:00:35 Encomenda Req1-34 entregue no destino
```

4.6. Arranque e terminação

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

```
Comprimento maximo (max_x), Altura maxima (max_y)
Todos os tipos de produtos existentes no sistema
Número de drones no sistema (D), bateria inicial (Binit), bateria máxima (Bmax)
Frequência de Abastecimento (S, em unidades de tempo), quantidade (Q), unidade de tempo (em milissegundos)

Número de armazéns no sistema (W)
Nome do Armazém xy: coordenadas (x, y) prod: 3 produtos (tipo, quantidade inicial)
```

Exemplo de um ficheiro de configuração:

```
1280, 800
Prod_A, Prod_B, Prod_C, Prod_D, Prod_E
20, 100, 200
10, 20, 200

2
Warehouse1 xy: 100, 200 prod: Prod_A, 10, Prod_D, 10, Prod_E, 20
Warehouse2 xy: 800, 500 prod: Prod_B, 10, Prod_C, 20, Prod_D, 10
```

De seguida, deverá criar o processo *Central*, todos os restantes recursos de comunicação e sincronização necessários, e os processos armazém (*Warehouse*). O PID de cada um dos processos armazém criados e o seu W_NO deverão ser escritos no ecrã e no *log*.

O Servidor deverá estar igualmente preparado para terminar, após a receção, por parte do processo *Gestor da simulação*, de um sinal do tipo SIGINT. Nessa altura, o servidor deverá deixar de receber novos pedidos escrevendo para o *log* os comandos restantes no *pipe*, sem lhes dar seguimento. De seguida deve aguardar a terminação de todos os

pedidos pendentes, e só depois de todos os drones retornarem a uma base é que deve terminar os processos e fazer a limpeza no sistema de todos os recursos partilhados.

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

5. Notas importantes

- **Não será tolerado plágio, cópia de partes de código entre grupos ou qualquer outro tipo de fraude**. Tentativas neste sentido resultarão na **classificação de ZERO valores** e na consequente **reprovação na cadeira**.
- Em vez de começar a programar de imediato pense com tempo no problema e estruture adequadamente a sua solução.
- Inclua na sua solução o código necessário à deteção e correção de erros.
- Evite esperas ativas no código e assegure a terminação limpa do servidor, ou seja, com todos os recursos utilizados a serem removidos.
  - Esperas ativas serão fortemente penalizadas!
- Utilize um *makefile* para simplificar o processo de compilação. **O *makefile* tem uma cotação associada**: deve ser mostrado na defesa intermédia, e deve ser incluído na submissão final.
- Inclua informação de *debug* que facilite o acompanhamento da execução do programa, utilizando por exemplo a seguinte abordagem:

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

- Todos os trabalhos deverão funcionar na VM fornecida ou, em alternativa, na máquina `alunos.deec.uc.pt`
- A defesa final do trabalho é obrigatória para todos os elementos do grupo. A não comparência na defesa final implica a classificação de **ZERO valores** no trabalho.
- O trabalho pode ser realizado em grupos de até 2 alunos.
- O programa deverá compilar sem *erros* nem *warnings* em qualquer uma das metas. A não compilação do código enviado implica uma classificação de ZERO valores na meta correspondente.

6. Metas, entregas e datas

Data	**Meta**	
Semana de 09/04/2019	Demonstração intermédia	A submeter no InforEstudante: **1 página A4 com a arquitetura e mecanismos de sincronização que vão implementar descritos.** Os alunos deverão **apresentar** o seu trabalho nas aulas PL. A demonstração irá focar a versão actual do programa que deve incluir as funcionalidades relativas a todos os pontos assinalados na *checklist* que incluída neste enunciado. A defesa intermédia vale **20%** da cotação do projeto.
Data de entrega **Sem penalização:** 18/05/2019-22h **5% penalização** 20/05/2019-23h59 **Não serão aceites submissões após esta data.**	Entrega final	O projeto final deve ser submetido no InforEstudante, tendo em conta o seguinte: ● Os **nomes** e **números** dos alunos do grupo devem ser colocados no início dos ficheiros com o código fonte. ● Com o código deve ser entregue um **relatório** sucinto (no máximo 2 páginas A4), no formato **pdf**, que explique as opções tomadas na construção da solução. Inclua um esquema da arquitetura do seu programa. Inclua também informação sobre o tempo total despendido (por cada um dos dois elementos do grupo) no projeto. ● Crie um arquivo no formato **ZIP** com todos os ficheiros do trabalho. ● Não serão admitidas entregas por e-mail. A defesa final valerá **80%** da cotação do projeto e consistirá numa análise detalhada do trabalho apresentado.
21/05/2019 a 30/05/2019	Defesa	Defesas em grupo nas aulas PL.

7. Checklist

Esta lista serve apenas como indicadora das tarefas a realizar e assinala as componentes que serão objeto de avaliação na defesa intermédia, que correspondem às funcionalidades marcadas com "S".

Processo	Tarefa	Avaliado na defesa intermédia?
Gestor da simulação	Arranque do servidor e aplicação das configurações existentes no ficheiro "config.txt"	S
	Criação do processo *Central*	S
	Criação de todos os processos armazém	S
	Criação da memória partilhada	S
	Criação da fila de mensagens	
	Envio de mensagens de abastecimento	
	Escrever a informação estatística no ecrã como resposta ao sinal SIGUSR1	
Central e Drones	Criação das *threads* drone	S
	Criação do *named pipe*	
	Carregamento da bateria na base	S
	Leitura e validação dos comandos pelo *named pipe*	
	Escolha do drone e sua notificação	S
	Movimentação do drone até armazém	S
	Movimentação do drone até local de entrega da encomenda	
	Movimentação do drone até à base	
	Interrupção do regresso de um drone por receção de novo pedido	
	Envio de mensagem do drone ao armazém	
	Escrita na memória partilhada de dados estatísticos pela *Central* e drones	S
	Atualização de dados dos drones nas estruturas intra processo	S
Processos armazém	Leitura da chegada de um drone	
	Notificar drone que o carregamento foi terminado	
	Tempo de espera para transferência de produtos	
	Atualizar informação de stocks e estatísticas na memória partilhada	S
Geral e Ficheiro de *log*	Envio sincronizado do output para ficheiro de *log* e ecrã.	S
	Suporte de concorrência no tratamento de pedidos	
	Deteção e tratamento de erros.	S
	Sincronização com mecanismos adequados (semáforos, *mutexes* ou variáveis de condição)	
	Prevenção de interrupções indesejadas por sinais e fornecer a resposta adequada aos vários sinais	
	Após receção de SIGINT, terminação controlada de todos os processos e *threads,* e libertação de todos os recursos.	