Sistemas Operativos 2020/2021

Trabalho Prático
Simulador de corridas

1. Introdução

O sistema a desenvolver deve simular, de forma simplificada, uma corrida de carros, semelhante a corridas Fórmula 1. Várias equipas irão competir pelo primeiro lugar na corrida. Cada equipa poderá ter um ou mais carros na corrida, mas cada uma apenas terá um lugar na *box*. O carro vencedor é o primeiro que conseguir terminar o número de voltas da corrida. Os carros em competição necessitam de ir às boxes para abastecer com combustível ou quando têm uma avaria. As avarias dos carros em corrida são aleatórias e caso ocorra uma avaria a um carro, ele entra em modo de segurança (velocidade reduzida, e prioridade na *box*) e tem que ir à *box* da equipa para reparação. Os carros terão de gerir o seu nível de combustível de modo a efetuar o abastecimento na *box* assim que possível. Caso o nível de combustível atinja um nível mínimo, o carro entra em modo de segurança.

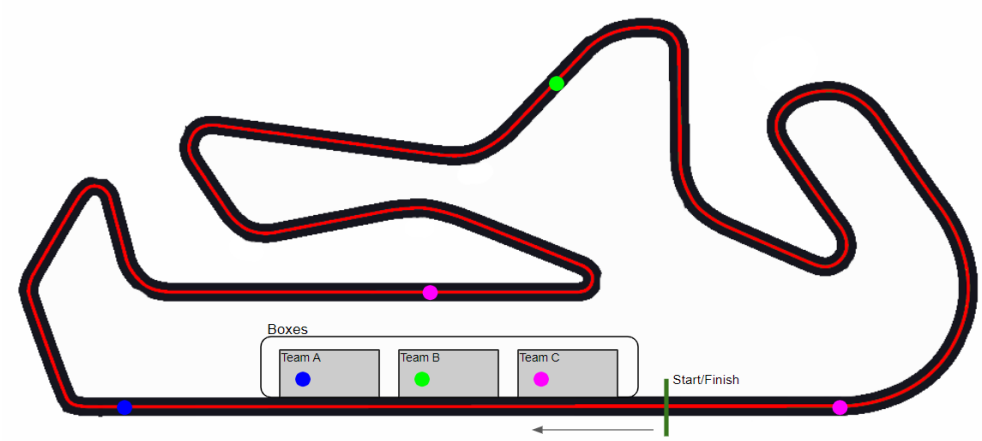

Figura 1: Imagem ilustrativa do simulador de corridas

2. Descrição do sistema de simulação

Nesta secção serão descritos os componentes do sistema de simulação, as suas funcionalidades e a forma como comunicam entre si.

2.1. Estrutura do sistema de simulação

A estrutura do sistema de simulação é apresentada na imagem seguinte.

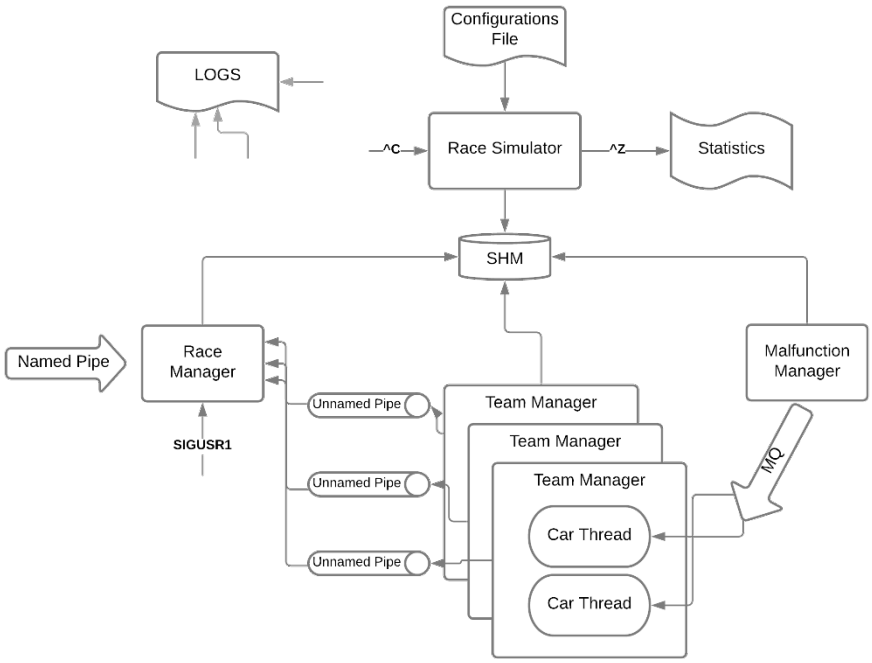

Figura 2: Visão geral do simulador a desenvolver

Tal como é representado na Figura 2, o sistema é baseado em vários processos e *threads*:

- **Simulador de Corrida (*Race Simulator*)** - Processo responsável por iniciar o sistema e os restantes processos do simulador.
- **Gestor de Corrida (*Race Manager*)** - Processo responsável pela gestão da corrida (início, fim, classificação final) e das equipas em jogo.
- **Gestor de Equipa (*Team Manager*)** - Processo responsável pela gestão da *box* e dos carros da equipa. É ainda responsável pela reparação dos carros da equipa e por atestar o depósito de combustível. Existe um por equipa.
- **Carros (*Car Thread*)** - Responsável pela gestão das voltas à pista, pela gestão do combustível e pela gestão do modo de circulação (normal ou segurança). Existe uma *thread* por carro.
- **Gestor de Avarias (*Malfunction Manager*)** - Processo responsável por gerar aleatoriamente as avarias dos carros, a partir da informação da sua fiabilidade.

E também por vários IPCs:

- **SHM** - Zona de memória partilhada acedida pelos processos **Simulador de Corrida**, **Gestor de Corrida, Gestor de Equipa** e **Gestor de Avarias**. Pode conter todos os dados necessários à boa gestão da corrida.
- ***Named Pipe*** - Permite enviar comandos ao processo **Gestor de Equipa**.
- ***Unnamed pipe*** - Permitem a comunicação entre o processo **Gestor de Corrida** e cada **Gestor de Equipa**.
- **MQ** - *Message Queue* que permite o envio de informação do **Gestor de Avarias** e cada uma das *threads* **Carros**.

Existirá também um ficheiro de ***log*** onde serão escritas todas as informações para análise posterior.

2.2. Descrição dos componentes e das funcionalidades

De seguida são apresentadas com detalhe as características e funcionalidades dos diversos componentes.

Simulador de Corrida

- Lê configurações do **Ficheiro de Configuração** (ver exemplo fornecido)
- Cria ***Named Pipe*** de comunicação com o **Gestor de Corrida**
- Inicia o processo **Gestor de Corrida**
- Inicia o processo **Gestor de Avarias**
- Captura o sinal SIGTSTP e imprime estatísticas
- Captura o sinal SIGINT para terminar a corrida e o programa. Ao receber este sinal, deve aguardar que todos os carros cruzem a meta (mesmo que não seja a sua última volta) - os carros que se encontram na box no momento da instrução devem terminar. Após todos os carros concluírem a corrida, deverá imprimir as estatísticas do jogo e terminar/libertar/remover todos os recursos utilizados.

Gestor de Corrida

- Cria processos **Gestor de Equipa** (1 por equipa)
- Aceita comandos pelo ***Named Pipe***:
    - Adição de novos carros (ver exemplo); tem de validar a informação e ver se não excede o número de equipas permitidas;
        - Todas as linhas com informação errada ou carros de equipas que não podem ser inscritas devem ser recusadas e essa informação escrita no ecrã e em ***log***.
    - O comando "START RACE!" começa a corrida caso exista o número de equipas inscritas definido no ficheiro de configurações.
        - Caso contrário deve ser gerada uma mensagem de erro no ecrã e para o ***log***.
    - Após começar a corrida, se forem recebidos novos carros, deverá imprimir os dados recebidos seguido da informação "Rejected, race already started!"; esta informação também deve constar no ***log***.
- Escreve as informações de cada carro, recebidas do ***Named Pipe***, na memória partilhada.
- Notifica as diversas equipas do início, fim e interrupção da corrida.
- Captura o sinal SIGUSR1 para interromper uma corrida que esteja a decorrer. A corrida deverá terminar mal todos os carros cheguem à meta (tal como acontecia após a receção de um SIGINT) e a sua estatística final apresentada. A informação da interrupção deverá ser escrita no ***log***. Se receber novo comando "START RACE!" a corrida poderá ser novamente iniciada.

Carros

- São *threads* criadas pelo Gestor de Equipa
- Podem estar num de 5 estados:
    - **Corrida** - consomem `CONSUMPTION` litros de combustível e avançam `SPEED` metros, por unidade de tempo (valores obtidos através do ***Named pipe***);
    - **Segurança** - consomem apenas `0.4xCONSUMPTION` litros de combustível e avançam `0.3xSPEED` metros por unidade de tempo; estão neste estado por terem uma avaria, ou por terem pouco combustível;
    - **Box** - estão na box, na posição da partida (distância 0);
    - **Desistência** - quando ficam sem combustível a meio da pista;
    - **Terminado** – carro que já terminou a corrida.
- Quando recebem uma nova mensagem do **Gestor de Avarias**, a informar de uma avaria no carro, passam para modo **Segurança**.
- Quando atingem combustível apenas suficiente para realizar 4 voltas (em modo **Corrida**) começam a tentar entrar na box. Ao chegarem ao ponto de partida da pista, se a box da equipa estiver em estado **livre**, entram na box e abastecem. Caso a box esteja **reservada** ou **ocupada**, devem continuar.
- Quando atingem combustível suficiente para realizar apenas 2 voltas (em modo **Corrida**) passam para modo **Segurança**.
- Se por algum motivo o carro ficar sem combustível, este deverá passar para o estado **Desistência** e ficará parado até ao fim da corrida.
- Atualizam em memória partilhada o estado do carro.
- Todas as alterações de estado (incluindo a terminação da corrida) têm de ser notificadas ao **Gestor de Corrida** através dos ***unnamed pipes***.

Gestor de Equipa

- Cria as *threads* **Carro**;
- Mantém atualizada na memória partilhada o estado da box: **livre**, **ocupada** ou **reservada**.
    - Cada equipa tem apenas uma box para todos os carros da equipa (só um carro pode estar em reparação ou a abastecer durante a corrida).
    - A box está em estado **Reservado** quando um ou mais carros da equipa estão em modo **Segurança**.
    - A box localiza-se no marco 0 metros. Não são consideradas questões de aceleração/desaceleração na entrada e saída da box.
- Repara carro, quando este está na box da equipa, podendo demorar um tempo aleatório *X* entre **T_Box_min** e **T_Box_max** a reparar o carro (valores obtidos do **Ficheiro de Configuração**).
- Atesta o depósito do carro, gastando para isso 2 unidades de tempo.

Gestor de Avarias

- Após o início da corrida executa a cada **T_Avaria** unidades de tempo (valor obtido do **Ficheiro de Configuração**), e para cada carro, determina com base na fiabilidade, se ocorre uma avaria.
- Comunica a avaria ao carro por ***Message Queue*** (na mesma execução pode gerar múltiplas ou nenhuma avaria, mas no máximo uma por carro).

Estatísticas

- Top 5 da grelha da corrida, por ordem decrescente do número de voltas completas realizadas, apresentando o número do carro, o número da equipa, o total de voltas realizadas e o número de paragens na box.
- Carro em último lugar, apresentando os mesmos campos identificados na estatística anterior.
- Total de avarias ocorridas durante a corrida (contemplando todos os carros/equipas).
- Total de abastecimentos realizados durante a corrida (contemplando todos os carros/equipas).
- Número de carros em pista (após receber instrução para finalizar o programa, todos os carros que cruzam a meta devem deixar de ser considerados em pista).

Ficheiro de Configurações

O ficheiro de configurações deverá seguir a seguinte estrutura:

```
Número de unidades de tempo por segundo
Distância de cada volta*, Número de voltas da corrida
Número de equipas**
Número máximo de carros por equipa
Número de unidades de tempo entre novo cálculo de avaria (T_Avaria)
Tempo mínimo de reparação (T_Box_min), tempo máximo de reparação (T_Box_Max)***
Capacidade do depósito de combustível (em litros)
```

* Em metros
** Mínimo de 3 equipas, caso contrário rejeita o ficheiro de configurações
*** Unidades de tempo

Exemplo do ficheiro de configurações:

```
1
12000, 10
5
2
30
10, 30
36
```

Comandos a enviar através do *named pipe*

Adicionar carros (têm de ser especificados todos os campos):

```
ADDCAR TEAM: {nome}, CAR: {número carro}, SPEED: {metros por unidade de tempo},
CONSUMPTION: {litros por unidade de tempo}, RELIABILITY: {% de fiabilidade}
```

Exemplo:

```
ADDCAR TEAM: A, CAR: 20, SPEED: 30, CONSUMPTION: 0.04, RELIABILITY: 95
ADDCAR TEAM: B, CAR: 01, SPEED: 32, CONSUMPTION: 0.04, RELIABILITY: 90
```

Iniciar corrida:

```
START RACE!
```

Log da aplicação

Todo o *output* da aplicação deve ser escrito de forma legível num ficheiro de texto "log.txt". Cada escrita neste ficheiro deve ser precedida pela escrita da mesma informação na consola, de modo a poder ser facilmente visualizada enquanto decorre a simulação.
Deverá pôr no *log* os seguintes eventos acompanhados da sua data e hora:

- Inicio e fim do programa;
- Adição de novo carro
- Erros ocorridos
- Mudança de estado de cada carro
- Envio de avaria
- Sinais recebidos

Exemplo do ficheiro de *log*:

```
18:00:05 SIMULATOR STARTING
18:00:10 WRONG COMMAND => ADDCAR 10 20 30
18:00:23 NEW CAR LOADED => TEAM: A, CAR: 01, SPEED: 30, CONSUMPTION: 0.04, RELIABILITY: 95
18:00:25 NEW COMMAND RECEIVED: START RACE
18:00:25 CANNOT START, NOT ENOUGH TEAMS
18:00:26 NEW CAR LOADED => TEAM: B, CAR: 02, SPEED: 40, CONSUMPTION: 0.06, RELIABILITY: 90
18:00:27 NEW CAR LOADED => TEAM: C, CAR: 03, SPEED: 35, CONSUMPTION: 0.05, RELIABILITY: 91
18:00:27 NEW COMMAND RECEIVED: START RACE
18:00:31 NEW PROBLEM IN CAR 02
(…)
18:02:00 SIGNAL SIGTSTP RECEIVED
(…)
18:05:00 CAR 02 WINS THE RACE
18:06:00 SIMULATOR CLOSING
```

3. *Checklist*

Esta lista serve apenas como indicadora das tarefas a realizar e assinala as componentes que serão objeto de avaliação na defesa intermédia.

Item	Tarefa	Avaliado na defesa intermédia?
Simulador de corrida	Arranque do servidor, leitura do ficheiro de configurações, validação dos dados e aplicação das configurações lidas	S
	Criação do processo Gestor de Corrida e Gestor de Avarias	S
	Criação da memória partilhada	S
	Criação do *named pipe*	
	Escrever a informação estatística no ecrã como resposta ao sinal SIGTSTP	
	Captura o sinal SIGINT, termina a corrida e liberta os recursos	
Gestor de Corrida	Criação dos processos Gestores de Equipa	S
	Criação dos *unnamed* pipes	
	Ler e validar comandos lidos do *named pipe*	
	Começar e terminar uma corrida	
	Tratar SIGUSR1 para interromper a corrida	
Threads Carro	Atualizar SHM com as suas informações	
	Gerir os vários estados de cada carro (corrida, segurança, box, desistência e terminado)	
	Lerem as avarias da MSQ e responderem adequadamente	
	Notificar o Gestor de Corrida através dos *unnamed pipes*	
Gestor de Equipa	Criar *threads* carro	S (preliminar)
	Gerir box	
	Gerir abastecimento dos carros	
Gestor de Avarias	Gerar avarias para os vários carros, baseado na fiabilidade de cada um	
Ficheiro de *log*	Envio sincronizado do *output* para ficheiro de *log* e ecrã.	S
Geral	Criar um makefile	
	Diagrama com a arquitetura e mecanismos de sincronização	S (preliminar)
	Suporte de concorrência no tratamento de pedidos	
	Deteção e tratamento de erros.	
	Sincronização com mecanismos adequados (semáforos, *mutexes* ou variáveis de condição)	S (preliminar)
	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.	

4. 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**. Dependendo da gravidade poderão ainda levar a processos disciplinares.
- Todos os trabalhos serão escrutinados para deteção de cópias de código.
- Para evitar cópias, os alunos não podem colocar código em repositórios de acesso público.

- Leia atentamente este enunciado e esclareça dúvidas com os docentes.
- Em vez de começar a programar de imediato pense com tempo no problema e estruture adequadamente a sua solução. Soluções mais eficientes e que usem menos recursos serão valorizadas.
- Inclua na sua solução o código necessário à deteção e correção de erros.
- Evite esperas ativas no código, sincronize o acesso aos dados sempre que necessário e assegure a terminação limpa do servidor, ou seja, com todos os recursos utilizados a serem removidos.
  - Esperas ativas serão fortemente penalizadas!
  - Acessos concorrentes que, por não serem sincronizados, puderem levar à corrupção de dados, serão fortemente penalizados!
- 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 student2.dei.uc.pt.
  - Compilação: o programa deverá compilar com recurso a uma *makefile*; não deve ter erros em qualquer uma das metas; evite também os *warnings*, exceto quando tiver uma boa justificação para a sua ocorrência (é raro acontecer!).
  - A não compilação do código enviado implica uma classificação de **ZERO valores** na meta correspondente.
- 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 (grupos com apenas 1 aluno devem ser evitados e grupos com mais de 2 alunos não são permitidos).
- A nota da defesa é individual pelo que cada um dos elementos do grupo poderá ter uma nota diferente;
- Os alunos do grupo devem pertencer a turmas PL do mesmo docente. Grupos com alunos de turmas de docentes diferentes são exceções que carecem de aprovação prévia.
- Ambas as defesas, intermédia e final, devem ser realizadas na mesma turma e com o mesmo docente.

5. Metas, entregas e datas

Data	**Meta**	
Data de entrega no Inforestudante 05/04/2021-22h00	Entrega intermédia	Crie um arquivo no formato **ZIP** (**NÃO SERÃO ACEITES OUTROS FORMATOS**) com todos os ficheiros do trabalho e submeta-o no Inforestudante: • Os **nomes** e **números** dos alunos do grupo devem ser colocados no início dos ficheiros com o código fonte. • Inclua todos os ficheiros fonte e de configuração necessários e também um Makefile para compilação do programa. • Não inclua quaisquer ficheiros não necessários para a compilação ou execução do programa (ex. diretórios ou ficheiros de sistemas de controlo de versões) • Com o código deve ser entregue **1 página A4 com a arquitetura e todos os mecanismos de sincronização a implementar descritos**. • **Não serão admitidas entregas por e-mail**.
Aulas PL da semana de 05/04/2021	Demonstração /defesa intermédia	• A demonstração deverá contemplar todos os pontos referidos na *checklist* que consta deste enunciado. • A demonstração/defesa será realizada nas aulas PL. • A defesa intermédia vale **20%** da cotação do projeto.
Data de entrega final no Inforestudante 15/05/2021-22h00	Entrega final	Crie um arquivo no formato **ZIP** (**NÃO SERÃO ACEITES OUTROS FORMATOS**) com todos os ficheiros do trabalho e submeta-o no Inforestudante: ● Os **nomes** e **números** dos alunos do grupo devem ser colocados no início dos ficheiros com o código fonte. ● Inclua todos os ficheiros fonte e de configuração necessários e também um Makefile para compilação do programa. ● Não inclua quaisquer ficheiros não necessários para a compilação ou execução do programa (ex. diretórios ou ficheiros de sistemas de controlo de versões) ● Com o código deve ser entregue um **relatório** sucinto (no máximo 2 páginas A4), no formato **pdf (NÃO SERÃO ACEITES OUTROS FORMATOS)**, 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. ● **Não serão admitidas entregas por e-mail**.
17/05/2021 a 02/06/2021	Defesa final	• A defesa final vale **80%** da cotação do projeto e consistirá numa análise detalhada do trabalho apresentado. • Defesas em grupo nas aulas PL. • É necessária inscrição para a defesa.