# Relatório do projeto final
# "Simulador de corrida"

## Introdução

O projeto consiste em desenvolver um simulador de corrida de carros semelhante a corridas Fórmula 1. Durante a corrida várias equipas competem pelo primeiro lugar, sendo que o primeiro carro que conseguir terminar o número de voltas é o vencedor. Relativamente ao desenvolvimento do projeto, cada membro teve várias tarefas distribuídas, tendo cada um elaborado o seu próprio código ao mesmo tempo e nos mesmos ficheiros, em tempo real, através da extensão LiveShare do VSCode. Cada membro despendeu cerca de 5 a 7h semanais, somando-se no final cerca de 60 a 70 horas de trabalho por membro.

## Estrutura

1) diagrama de **arquitetura do programa**, ilustrando o seu funcionamento;
2) ficheiro **makefile**, permite compilar o programa para executá-lo;
3) ficheiro **config.txt**, contém as configurações da corrida;
4) ficheiro **log.txt**, onde são mantidas todas as alterações relevantes que ocorrem durante a corrida;
5) código fonte, contido nos ficheiros **main.c, functions.c, processes.c e declarations.h.**

A composição do programa é dividida da seguinte forma: o **Simulador de Corrida** assume a forma da main, o **Gestor de Corrida** é um processo que gera vários processos **Gestor de Equipa**, que por sua vez geram várias **threads Carro**. Existe também um processo **Gestor de Avarias** que funciona à parte e comunica avarias às threads Carro por **Message Queue**. Existem também comunicações feitas de um Gestor de Equipa para o Gestor de Corrida através de **unnamed pipes**, sendo que este recebe também comandos do **named pipe**. Todos os processos e threads têm acesso à **Shared Memory**, havendo mecanismos de sincronização que controlam todos os acessos que são feitos, sendo eles **semáforos mutex, posix e variáveis de condição**. Por fim, existem também vários sinais que permitem imprimir **Estatísticas** ou terminar a corrida.

O programa é compilado com o comando "make" e é executado com o comando "./races". Noutro terminal são introduzidos comandos para o named pipe "np_race_manager", como por exemplo, "echo "ADDCAR TEAM: A, CAR: 1, SPEED: 30, CONSUMPTION: 0.12, RELIABILITY: 100" > np_race_manager".

## Implementação

**Pré-corrida**

- Criação dos sinais, criação de um named semaphore para sincronizar escritas no ficheiro log, leitura do ficheiro de configurações, criação da shared memory e dos arrays de estruturas de carros e equipas, criação de semáforos mutex, criação de variáveis de condição, criação de named pipe, criação da message queue e criação dos processos raceManager() e malfunctionManger().
- O processo malfunctionManager() fica à espera do arranque da corrida para poder funcionar. Por outro lado, o raceManager() começa por criar os unnamed pipes e faz a leitura dos comandos enviados por named pipe. Se a equipa ainda não existir, cria-a e o contador de carros da corrida é incrementado; se já existir, apenas o contador é incrementado.
- Quando um processo teamManager() é criado, a box da equipa é criada dentro do processo e, uma vez recebido do named pipe o comando de início de corrida, cria todas as threads carro pertencentes à equipa em questão.
- Após a leitura e verificação da existência de um número certo de equipas, de acordo com o ficheiro de configurações, se receber o comando "START RACE!" a corrida arranca e todos os processos

de equipa e threads de carro criados no momento começam a funcionar, assim como o malfunctionManager().

**Corrida**

- **malfunctionManager()**: percorre todos os carros e gera um nº aleatório entre 0 e 100; se esse número for maior que a "reliability" do carro então gera uma avaria e manda-a ao carro por message queue.
- **raceManager()**: continua a receber comandos mas não os aceita; recebe as mudanças de estado dos carros por unnamed pipe e, se o estado for TERMINADO ou DESISTENCIA, incrementa o número de carros que acabaram a corrida. Se esse número for igual ao número de carros total, a corrida termina, podendo ser recomeçada se receber um comando "START RACE", exceto no caso de um SIGINT.
- **teamManager()**: a box fica à espera de receber um carro. Inicialmente possui um atributo car_id = -1, significando que não existe nenhum carro na box (uma vez que todos os carros têm um id positivo). Se o id do carro na box mudar, então é porque recebeu um carro e vai abastecer e/ou repará-lo, consoante tenha uma avaria ou não. Após isso, o processo liberta o carro da box. Quando os seus carros terminam de correr, espera pelas threads e volta ao estado inicial para poder assim dar início outra corrida, ou limpar os recursos e sair caso seja sinalizado.
- **carThread()**: dá sleep e avança *speed* unidades, retirando sempre *-consumption* ao combustível, com valores diferentes consoante o estado do carro seja CORRIDA ou SEGURANCA. Cada vez que muda de estado notifica o raceManager() por unnamed pipe. Se o carro estiver em condições de entrar na box, passa para o estado BOX e avisa a própria box da sua entrada. Após isso, a box trata do carro. Uma vez que o carro entrou em estado TERMINADO ou DESISTENCIA, ele fica à espera que a corrida retome novamente, exceto no caso de receber um sinal SIGINT.
- Ao receber o sinal SIGSTP, o processo principal imprime no ecrã as estatísticas da corrida. Ao receber o sinal SIGINT, o *Race Simulator* irá sinalizar o fim da simulação e esperar pela saída do *Race Manager* e *Malfunction manager*. Este termina imediatamente, enquanto o primeiro vai, antes de limpar os seus recursos, aguardar pelo fim de todos os processos *Team Manager*, que por sua vez aguardam o fim de todas as suas threads.