Projeto de Sistemas Operativos

Tomás Mendes – 2019232272  
Joel Oliveira – ?

**Tempo Despendido**

Tomás: 15h ?

Joel: 15h ?

Total: 30h ?

**Introdução**

Este projeto tinha como objetivo implementar um simulador de corridas, usando todas as estruturas lecionadas e trabalhadas nas aulas PL. Assim, foram usados alguns métodos de sincronizção entre processos e *threads,* tais como semáfros, variáveis de condição e sinais. Foram também usados *named* e *unnamed pipes* bem como *message queues.*

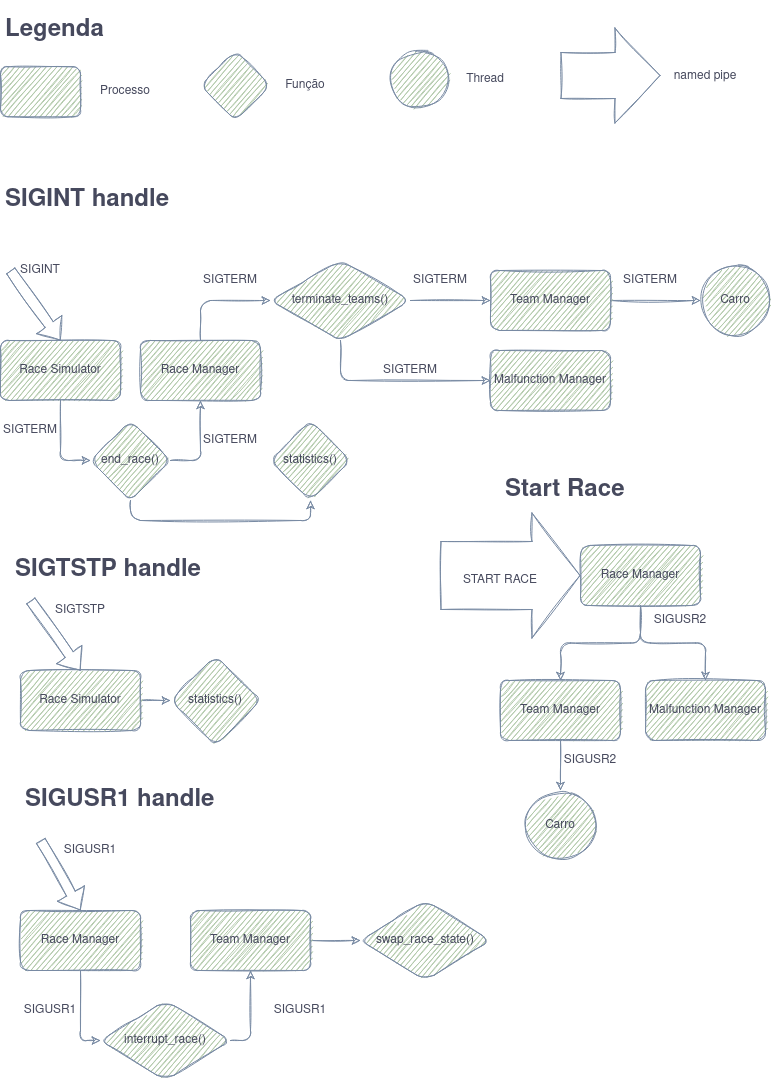
**Explicação das escolhas feitas**

Começando pela memória partilhada, onde está tudo o que é necessário para que os vários processos do programa consigam funcionar. A estrutura da memória encontra-se no diagrama abaixo. De notar que o *array* dos carros que está em memória partilhada, tem *slots* para o número máximo de carros. Cada equipa tem um indíce associado, atribuido por ordem de criação da equipa, que pode ser usado para aceder aos seus carros, por exemplo, a 3ª equipa a ser criada poderá aceder aos seus carros na posição NR\_CARS\*3, sendo NR\_CARS o número máximo de carros por equipa. Os acessos à memória partilhada estão também, quando necessário, devidamente sincronizados. Cada *thread* carro atualiza apenas a sua estrutura, sendo a sincronização, do array de carros, apenas necessária para as estatisticas durante a corrida.

O ínicio da corrida, bem como a interrupção da corrida, após a receção do SIGUSR1, e a terminação do programa, após a receção do SIGINT, é feita pela propagação de sinas pelos processos e thread, dependendo do caso. O diagrama abaixo exemplifica a propação dos sinais em cada caso.

Cada equipa tem *mutex* para controlar o acesso à variável com o estado da box. A escrita no ficheiro de *log* e no *stdout* é também controlado por um *mutex* do tipo *named* criado pelo processo principal, *Race Simulator*.

A interrupção da corrrida é feita após a receção do SIGUSR1 pelo *Race Manager* e após todos os carros terem chegado à linha da meta*.* É invertida uma flag e as thread carro ficam num wait de uma váriável de condição. Após uma interupção da corrida, apenas o comando START RACE, recebido pelo named pipe, recomeça a corrida do ponto em que estava, voltando a inverter a variável e fazendo o *broadcast* da variável de condição.

 As estatísticas, quando chamadas após a receção do sinal SIGTSTP, copia o array dos carros que está em memória partilhada e ordena-o por ordem na pista. Se um carro já tiver acabado a corrida, terá a posição em que acabou na variável end\_position, sendo esse o fator principal para ordenar os carros. Os restantes carros são ordenados pela volta e pela posição na pista. Para garantir a integridade dos dados, as estatisticas alteram uma flag, fazendo com que todos os carros parem para fazer uma cópia do array dos carros. Para tal, cada carro...(acabar isto)