Projeto de Sistemas Operativos

Tomás Mendes – 2019232272

Joel Oliveira – 2019227468

Tempo Despendido (estimativa)

Tomás: 15h

Joel: 15h

Total: 30h

Introdução

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

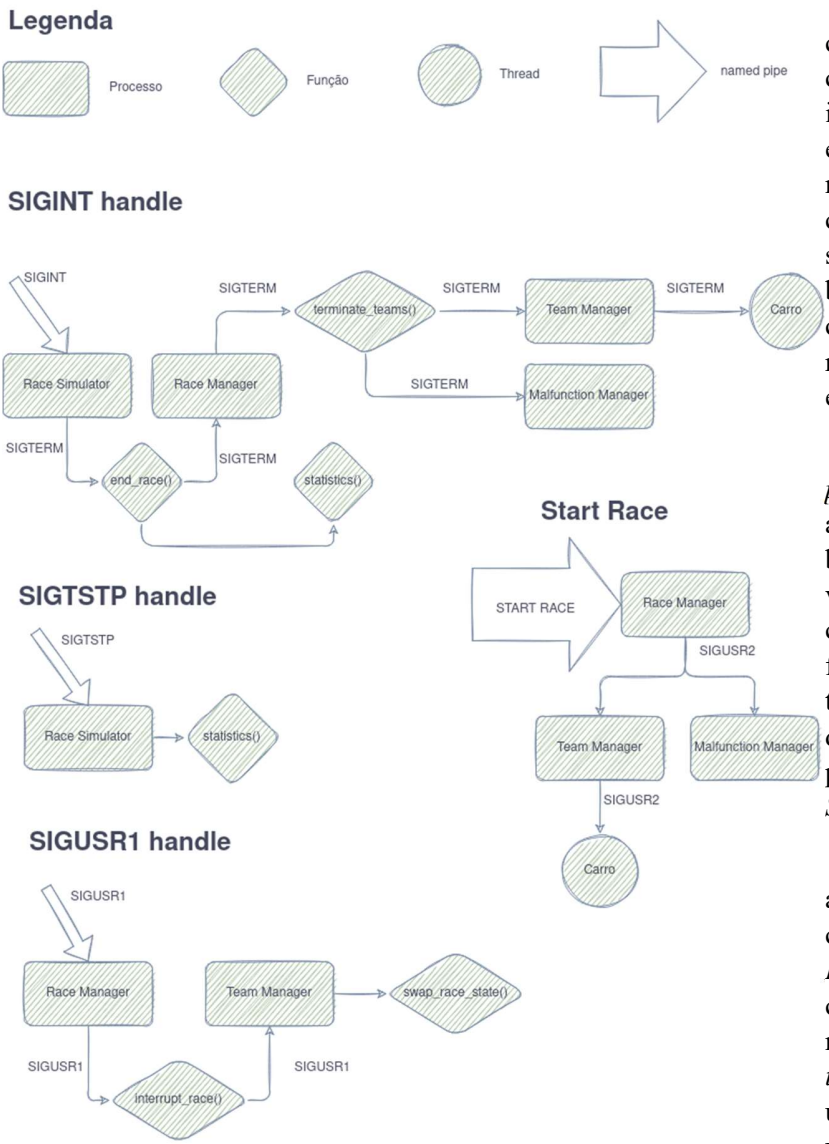
Explicação das escolhas feitas

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 índice associado, atribuído 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 estatísticas, durante a corrida.

O início da corrida é controlado por uma variável de condição pela qual os carros esperam. Após todos os processos *team_manager* criarem as *threads* associadas ao número de carros da equipa, este comunica ao *race_manager* que os seus carros estão prontos e fica em espera do sinal de partida, num *sigwait*. Após receção desta informação de todas as equipas, é enviado um sinal de partida a cada *team_manager* (SIGUSR2), estes seguem o seu procedimento, alterando o valor da condição pela qual as threads esperam, e sinalizando todas que avancem, com um *conditional_broadcast()*.

A interrupção da corrida, após a receção do SIGUSR1, bem como o término do programa, após a receção do SIGINT, é feita pela propagação de sinas pelos processos e *threads*, dependendo do caso. O diagrama, presentes na próxima página, exemplifica a propação dos sinais em cada caso.

No caso das *threads* do processo do *team_manager*, a receção dos sinais é controlada por máscaras, para certificar que é o processo *team_manager* a receber o SIGUSR1 para a interrupção da corrida, e que os carros ignoram sempre esse sinal. Para o caso da finalização da corrida, o processo *team_manager* sinaliza as suas *threads* com o sinal SIGUSR2;



O controlo de espera pelo cruzamento da meta por parte dos carros para a interrupção/finalização da corrida é realizado com estas mesmas máscaras. Sempre que um carro cruza a meta os sinais presentes o sinal SIGUSR2 é desbloqueado, e bloqueado novamente até ser cruzada a meta. Isto também é realizado quando o carro se encontra na box.

Cada equipa tem um *pthread_mutex* para controlar o acesso à variável com o estado da box, sendo que a memória desta variável está alocada à memória de cada processo. 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 corrida é feita após a receção do SIGUSR1, como referido anteriormente, 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 variável de condição. Após uma interrupção 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 estatísticas alteram uma *flag*, fazendo com que todos os carros parem para fazer uma cópia do *array* dos carros.

Esta paragem é controlada por uma série de semáforos POSIX que simulam uma variável de condição. Um semáforo com o valor igual ao número de carros em pista, que gera a variável de condição de espera das estatísticas. Dois semáforos iniciados a 0, um para interromper as estatísticas até o valor da condição alterar e outro para interromper os carros da mesma maneira. A condição que os carros verificam é um *boolean* que está na memória partilhada.

O procedimento descrito está exemplificado abaixo, sendo que o *mutex* está iniciado a 1, os *condx* estão iniciados a 0, e o nr_carros contem o valor do número de carros em pista.