Relatório do Projeto

Leitura do ficheiro config.txt e criação da memória partilhada
Lemos o ficheiro config.txt e separamos os seus dados usando o strtol(). Depois de verificar se todos os dados são válidos criamos a memória partilhada e guardamos o seu ponteiro numa variável global, para permitir o acesso à mesma a partir de qualquer parte do código.

Criação do processo Gestor de Corrida e Gestor de Avarias e criação do named pipe
Utilizamos a função fork() para criar os processos e criamos o named pipe no simulador de corrida.

Escrever a informação estatística no ecrã como resposta ao sinal SIGTSTP
Para escrever as informações utilizamos a função sigtstp() que reúne todos os dados necessários e faz o print.

Captura o sinal SIGINT, termina a corrida e liberta os recursos
Redirecionamos para a função sigint() e nela libertamos os mutexes, a memória partilhada, esperamos pelos processos filhos e ainda imprimimos os dados da corrida.

Criação dos processos Gestores de Equipa e Criação dos unnamed pipes
Criamos os unnamed pipes e os processos das equipas no início do processo gestor de corrida e guardamos os pipes num array global para facilitar o acesso aos mesmos.

Envio sincronizado do output para ficheiro de log e ecrã
Utilizamos a função log_out() em conjunto com um mutex para apenas escrever um output de cada vez.

Ler e validar comandos lidos do named pipe
Lemos os comandos do named pipe dentro de um while e, dependendo do comando, escrevemos o output no ficheiro log e no ecrã através da função log_out().

Começar e terminar uma corrida
Para começar uma corrida fazemos uso de uma variável de condição e de um mutex. As equipas antes de criarem as threads carro esperam pela variável de condição assim como o malfunction manager antes de começar a gerar avarias. Quando o comando START RACE! É introduzido estes processos são notificados. Para terminar utilizamos um contador em shared memory que decrementa consoante os carros acabam a corrida ou ficam sem combustível. Quando este atinge o valor 0 uma flag em shared memory passa a true e o malfunction manager envia uma mensagem às equipas e acaba. Estas recebem a mensagem e esperam que todas as threads carro terminem e acabam também. O race manager espera que todas as equipas acabem e acaba também depois, e de forma semelhante o race simulador espera pelo race manager e pelo malfunction manager.

Tratar SIGUSR1 para interromper a corrida
Redirecionamos o sinal para a função sigusr() onde acabamos todos os processos equipa e as suas threads carro, que ficam com os valores das suas variáveis originais, e o malfunction manager. Criamos outro malfunction manager, colocamos as variáveis de start_race e end a 0 e chamamos a função race manager de novo.

Atualizar SHM com informações dos carros, gerir os vários estados de cada carro e notificar o Gestor de Corrida através dos unnamed pipes
Tudo isto é feito nas threads carro sem necessidade de algum tipo de sincronização.

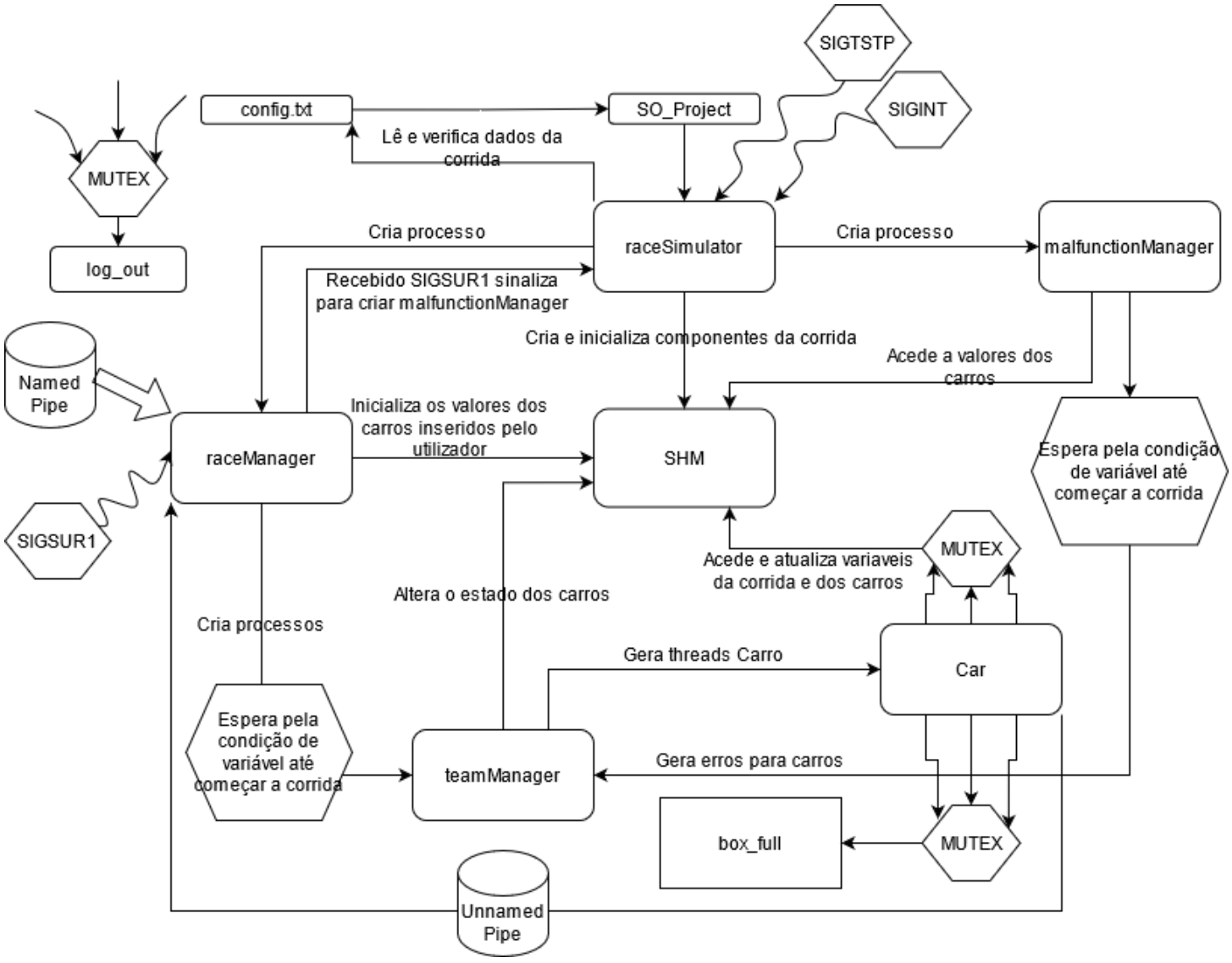
Criar threads carro e ler as avarias da MSQ e responder adequadamente
Feito pelo team manager que notifica o carro específico sobre a avaria, mudando o seu estado.

Marco da Cruz Pais Nº2019218981
Simão Carvalho Monteiro Nº2019215412

Gerir box e gerir abastecimento **dos carros**
Feito nas threads carro com a ajuda de uma variável com o estado da box e com um mutex.

Gerar avarias para os vários carros, baseado na fiabilidade de cada um
Com a ajuda de um rand() percorremos a lista de carros e baseado na sua fidelidade o malfunction manager envia ou não a mensagem à equipa correspondente.

Arquitetura do Projeto



Conclusão

Este projeto ajudou-nos a consolidar a matéria dada nas aulas teóricas, teórico-práticas e práticas e apesar de termos encontrado problemas ao longo da resolução, tentamos solucionar todos eles da melhor forma que conseguimos.
Foram despendidas cerca de 24hrs, por cada elemento, distribuídas ao longo de vários dias.

Marco da Cruz Pais Nº2019218981
Simão Carvalho Monteiro Nº2019215412