***Race Simulator***

Sistemas Operativos

Licenciatura em Engenharia Informática, FCTUC

2020/ 2021



Desenvolvido por:

João Filipe Guiomar Artur, 2019217853

Sancho Amaral Simões, 2019217590

**Descrição e objetivos**

Este projeto, designado de *Race Simulator*, foi desenvolvido no âmbito da unidade curricular de Sistemas Operativos, edição 2020/ 2021, da Licenciatura em Engenharia Informática, FCTUC.

O tema do trabalho é desenvolver uma simulação de corridas de carros e aspetos relacionados com a gestão de uma corrida de carros em pista. Alguns destes aspetos incluem gestão de equipas e carros participantes, gestão de avarias e reabastecimentos de combustível e gestão das dinâmicas das *boxes* das equipas. Assim, o tema e os aspetos referidos permitiram explorar de uma forma mais aprofunda conceitos abordados no contexto de aulas teórico-práticas e práticas, tais como:

* Criação de processos e *threads*;
* Sincronização de processos e *threads* através de semáforos, variáveis de condição e *pthread mutexes*;
* Comunicação entre diferentes processos através de *named pipes*, *unnamed pipes* e *message queues*;

**Mecanismos de sincronização**

Um dos aspetos fundamentais para o correto funcionamento é a sincronização quer do uso dos recursos partilhados entre os vários processos e *threads* quer da ordem temporal dos acontecimentos.

A nível de sincronização de recursos, apesar de na primeira meta ter sido considerada a utilização de semáforos POSIX, optou-se por uma utilização extensiva de variáveis de condição e *pthreads mutexes*. Esta decisão é suportada por testes de *benchmarking* efetuados que revelaram uma melhor eficiência e performance dos *pthread* *mutexes* comparativamente aos semáforos POSIX. Assim, os mecanismos de sincronização referidos são utilizados no controlo do início da corrida, de acessos a dados na região de memória partilhada e eventos de *output*.

Para garantir uma correta ordem temporal dos eventos gerados, decidiu-se implementar um *clock* interno, suportado por variáveis de condição e *mutexes*. A cada ciclo de relógio, os vários processos e *threads* executam a sua tarefa e ficam à espera que o clock assinale o início do próximo ciclo de relógio. Tendo em conta o ponto de vista do clock, este fica à espera de receber um aviso de todos os processos gestores e threads que estão à sua espera. Nessa altura, inícia um novo ciclo de relógio e notifica todas as entidades dependentes.

**Comunicação entre processos**

Tendo em conta os objetivos do trabalho, é também fundamental a comunição entre processos, sendo esta conseguida através de *message queues*, *named* e *unnamed pipes.*

Numa primeira etapa, antes do início da corrida, é apenas considerado um named pipe para a comunicação entre o utilizador e o processo *Race Manager* para a receção de comandos.

Na segunda etapa, que começa com o início da corrida e dura até ao fim desta, utilizam-se todos os mecanismos supracitados. Para uma correta gestão dos vários *pipes*, recorre-se ao conceito de multiplexagem, sendo o *named pipe* usado para o efeito descrito e os *unnamed pipes* utilizados para a comunição entre as *threads* que simulam os carros e o *Race Manager*. Além disso, a *message queue* é utilizada para a comunição entre o *Malfunction Manager* e as *threads* já referidas

**Estados da corrida**

<escrever explicação sobre os estados da corrida>

**Resposta aos sinais**

Associação aos estados da corrida

Um dos pontos considerado no desenvolvimento foi a resposta a diversos tipos de sinais. < fazer ligação com estados da corrida >

A nível do sinais de interrupção da corrida, SIGINT e SIGUSR1, as respostas são bastante semelhantes, sendo as únicas diferenças o facto de a resposta ao primeiro terminar a simulação e a resposta ao segundo terminar apenas a corrida com possibilidade de recomeçar mais tarde. Ressalte-se ainda a possibilidade de sobreposição dos dois sinais, isto é, após a receção de um SIGINT se se receber um SIGUSR1, a resposta a este último é priveligiada e vice-versa.

Para a resposta ao sinal SIGTSTP, optou-se por fazer uma cópia da região de memória partilhada em regime de exclusão mútua, com recurso à instrução *memcpy* que revelou um comportamento bastante eficiente. O cálculo das estatísticas é depois efetuado sobre esta cópia.

**Eventos de output**

Com o objetivo de assegurar uma correta ordem do *output* gerado, decidiu-se recorrer a um *phtread mutex* para sincronizar os eventos de *output* quer para o utilizador quer para o ficheiro de *log*.

Outro aspeto a considerar é o mapeamento do ficheiro de *log* em memória, sendo todas as operações de escrita efetuadas no ficheiro em memória. O ficheiro físico é atualizado quando a simulação é encerrada.

**Funcionalidades extra**

Atendendo à complexidade do desenvolvimento deste projeto, decidiu-se implementar um conjunto de exceções e um sistema de *debug* para melhor detetar a ocorrência de erros e permitir uma melhor compreensão da sua natureza. Refira-se ainda que no sistema de *debug* existem vários níveis que permitem acompanhar o comportamento de determinadas funcionalidades como o *clock* ou a sequência temporal dos eventos gerados durante a corrida, entre outros aspetos.

Além disso, para permitir uma maior modularização e reaproveitamento do código desenvolvido, criaram-se bibliotecas *wrapper* com conjuntos de funções genéricas que depois foram utilizadas em diversos contextos distintos.

**Considerações finais**

Para concluir, inclui-se uma pequena reflexão de aspetos/ aprendizagens retiradas no decurso do desenvolvimento deste projeto e uma estimativa do tempo gasto.

<escrever reflexão>

Por último, estima-se que cada um dos elementos tenha investido entre 65 a 75 horas, ambos com o objetivo final de desenvolver um projeto sólido e de elevada qualidade.