Race Simulator

Sistemas Operativos

Licenciatura em Engenharia Informática, FCTUC

2020/ 2021

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

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, tendo em conta os aspetos relacionados com a sua gestão. Alguns destes aspetos são a gestão de equipas e carros participantes, gestão de avarias, reabastecimentos de combustível e gestão das dinâmicas das *boxes* das equipas. Assim, o tema e os aspetos referidos permitiram explorar de uma forma mais profunda conceitos abordados no contexto das aulas teórico-práticas e práticas de SO, 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 *mutexes pthread*. Esta decisão é suportada por testes de *benchmarking* efetuados que revelaram uma melhor eficiência e *performance* dos *mutexes pthread* comparativamente aos semáforos *POSIX*. Assim, os mecanismos de sincronização referidos são utilizados no controlo do início, interrupção e fim da corrida, de acessos a dados na região de memória partilhada e de 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 (simulado pelo *clock*), os vários processos e *threads* executam a sua tarefa, numa unidade de tempo, e ficam à espera que o clock assinale o início do próximo ciclo de relógio. Por outro lado, o *clock* fica à espera de receber uma notificação de todas as threads que estão à sua espera. Após isso, dorme durante uma unidade de tempo e inícia um novo ciclo de relógio, notificando todas as *threads* dependentes.

Comunicação entre processos

Tendo em conta os objetivos do trabalho, é também fundamental a comunição entre processos, sendo esta alcançada 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 e interpretação 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*. Convém salientar que estes *pipes (named* e *unnamed)* são multiplexados através da instrução *select*. Além disso, a *message queue* é utilizada para a comunição entre o *Malfunction Manager* e as *threads* carro, sendo que o tipo da mensagem é especificado pelo ID interno do carro (gerado a partir da incrementação do número de carros na *shm* até ao momento).

Estados da corrida

Para facilitar a gestão da corrida, assumiu-se que esta poderia estar em um de cinco estados:

- *NOT_STARTED* – o simulador encontra-se em estado de espera;
- *RUNNING* – a corrida encontra-se a decorrer normalmente;
- *INTERRUPTED* – a corrida encontra-se a decorrer, mas acabará mal todos os

carros cruzem a meta (estado induzido pelos sinais *SIGINT* e *SIGUSR1)*;

- *FINISHED* – a corrida acabou dado que todos os carros cruzaram a meta ou foram desqualificados.
- *CLOSED* – o simulador foi fechado voluntariamente pelo utilizador através do envio do sinal *SIGINT* ou da inserção do comando "*EXIT*" na *named pipe*.

O simulador comporta-se, então, de modo semelhante a um autómato finito na medid a em que transita de estado para estado consoante os estímulos internos/ externos.

Resposta aos sinais

Um dos pontos considerado no desenvolvimento do projeto foi a resposta aos diversos tipos de sinais.

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 a 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 privilegiada 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 ter um elevado grau de eficiência. 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 dois *mutexes phtread* para sincronizar os eventos de *output* quer para o utilizador quer para o ficheiro de *log*.

Funcionalidades extra

Atendendo à complexidade do desenvolvimento deste projeto, decidiu-se implementar um mecanismo de tratamento de exceções e um sistema de *debug* para facilitar a deteção 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

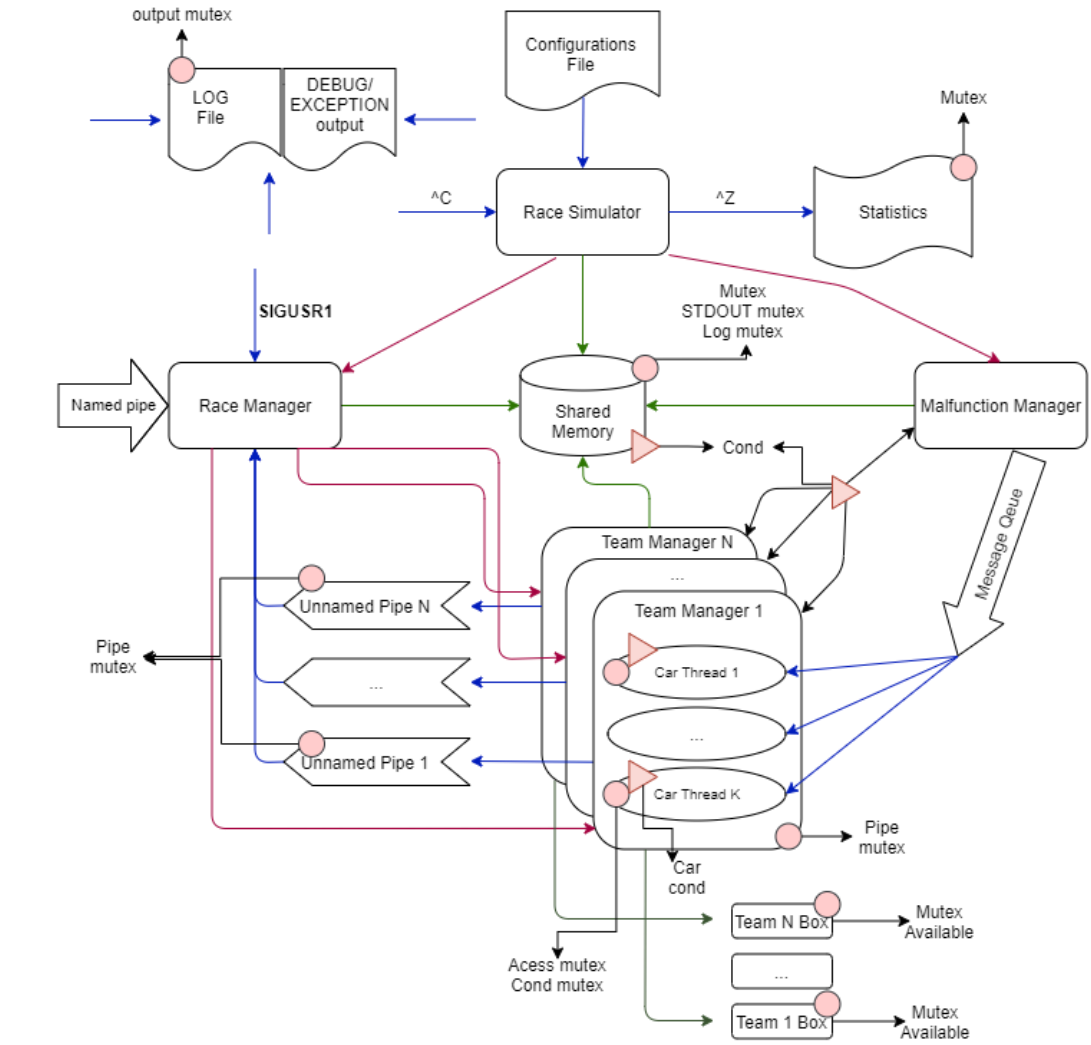
Para concluir, inclui-se uma pequena reflexão de aspetos/ aprendizagens retiradas no decurso do desenvolvimento deste projeto e uma estimativa do tempo gasto.

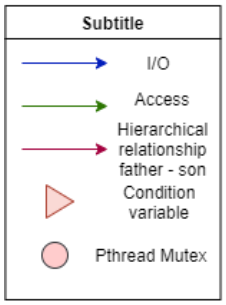
A implementação deste projeto permitiu-nos perceber que o desenvolvimento de aplicações que recorrem a multiprogramação deve ser feito de forma muito cuidadosa dado que é muito suscetível ao aparecimento de erros particularmente difíceis de detetar. Para resolver parcialmente este problema tivemos que recorrer ao *debugger GDB* com alguns comandos que permitem a navegação entre processos.

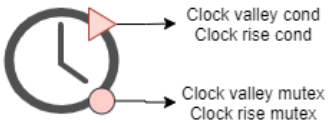
De um ponto de vista mais teórico, este projeto permitiu-nos perceber ainda melhor parte das dinâmicas consideradas na gestão de um sistema operativo. Além disso, alertou-nos para as consequências de más decisões durante implementação de novas funcionalidades em aplicações ou até no próprio SO.

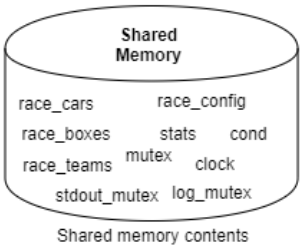
Por último, estima-se que cada um dos elementos tenha investido entre 110 a 120 horas, ambos com o objetivo final de desenvolver um projeto sólido.

RaceSimulator - Architecture & Syncronization Mechanisms









Shared memory contents

Authors:

- João Filipe Guiomar Artur, 2019217853
- Sancho Amaral Simões, 2019217590

LEI, University of Coimbra, 2nd semester, 2nd year, Operating Systems