Relatório Projeto de Sistemas Operativos

Edgar Filipe Ferreira Duarte 2019216077
Pedro Guilherme da Cruz Ferreira 2018277677

1. Introdução

No âmbito da cadeira de Sistemas operativos, foi criado um simulador de corridas que, a partir de métodos de sincronização, gere diversos processos de forma a se sustentar uma corrida justa para todos os participantes. Para este efeito, foram criados 4 processos diferentes (*RaceSimulator*, *BreakdownManager*, *RaceManager e TeamManager*), sendo que o *TeamManager* cria *threads*, em que cada uma representa um carro diferente da equipa (no resto do projeto uma *thread* será denominadas por um "carro"). Nos capítulos seguintes será detalhado o funcionamento de cada um destes processos.

2. RaceSimulator

O *RaceSimulator* é o processo inicial da simulação. É este o processo que lê o ficheiro de configuração, cria a shared memory, cria os processos *RaceManager* e *BreakdownManager* e cria o named *pipe*, que serve para transmitir comandos do utilizador ao *RaceManager*. Para esta última funcionalidade, foi criado um programa independente (***commands.c***) que se dedica a ler input do utilizador e envia essa mesma informação pelo named pipe até ao *RaceManager*. A shared memory foi feita com recurso a aritmética de ponteiros de forma a garantir um qualquer número de carros/equipas. A *shared memory* é um array de equipas em que cada equipa tem um array de carros para, assim, ser mais fácil a realização de operações na mesma. Este processo também é responsável por receber os sinais: *SIGINT* (para terminar a corrida) e *SIGTSTP* (para imprimir as estatísticas da corrida). É de notar que o *SIGTSTP* está indisponível até a corrida começar (visto que não faz sentido haver estatísticas antes).

3. BreakdownManager

O *BreakdownManager* é o processo responsável pela criação de avarias nos diversos carros. Para isso, enquanto a corrida estiver a decorrer (não terminou / não está em pausa), este processo, em intervalos de tempo definidos pelo ficheiro de configurações, cria um inteiro aleatório de 0-100 e, se esse inteiro tiver um valor superior ao da fiabilidade do carro, gera uma avaria para o carro. Essa avaria é comunicada através das message quele aos próprios carros. Para se garantir que a avaria é comunicada para o carro correto, foi utilizada a fórmula seguinte, com o objetivo de cada carro possuir um message type único:

msgtype = index_equipa * total de carros máximo por equipa + index_carro + 1

Nota: index_equipa é o index da equipa na shared memory e index_carro é o index do carro no array da equipa

Quando recebe um SIGUSR2 (enviado pelo *RaceManager*) a sua execução termina.

4. RaceManager

O *RaceManager* tem duas funções. Primeiramente recebe comandos através do named pipe para criação de novos carros ou para o começo da corrida. Sempre que um carro de uma equipa desconhecida é pedido para ser adicionado, se ainda for permitido, essa nova equipa é criada, ou seja, um novo processo *TeamManager* é criado. Também cria um *unnamed pipe* para conseguir partilhar informações com o processo acabado de criar. Quando o comando "START RACE!" é detetado, se as condições o permitirem, o *RaceManager* envia um sinal a todos os outros processos (*RaceSimulator*, *BreakdownManager* e todos os *TeamManager* criados) para informar do início da corrida.

A outra função do RaceManager é: ou esperar pelo sinal SIGUSR1 para causar uma interrupção na corrida, ou receber informação acerca do término de um carro por qualquer um dos named pipes. Na primeira situação quando o sinal é recebido, o *RaceManager* envia um sinal para o *BreakdownManager* e o *TeamManager* a indicar a pausa da corrida. A partir do momento em que todos os carros fiquem parados, o *RaceManager* espera para receber um comando pelo named pipe com a informação "START RACE!" para reiniciar a corrida.

5. TeamManager

O *TeamManager* é o processo responsável pela criação dos Carros e pela gestão da box. Além da criação de threads carro, também cria uma thread cuja função é, ao longo da execução do programa, ir calculando o tempo, para em intervalos exatos e consistentes, avisar os carros de uma nova iteração. Foi criada esta thread reduzir delays entre threads para um valor reduzido e constante de forma a tornar a corrida justa.

Após receber o sinal de começo da corrida (do *RaceManager*), o *TeamManager* cria os Carros e a thread de gestão do tempo. O *TeamManager* sabe quantos carros deve criar pois esta informação foi guardada na shared memory enquanto que o *RaceManager* vai recebendo novos comandos para criação de carros.

A gestão da box é feita da seguinte forma: a *box* tem 2 semáforos, um para as reservas e outro para a entrada na box. Primeiramente, o *TeamManager* fica à espera que o seu semáforo da reserva seja libertado (um carro realizou uma reserva). Aí muda o seu estado para "RESERVADO" e agora espera por um carro entrar na box. Quando um carro entra na box (e liberta o semáforo da box), o *TeamManager* simula o tempo de reparação do carro, se necessário, e enche o combustível do carro. Quando estas ações terminam o *TeamManager* liberta um semáforo que está a fazer a *thread* do carro esperar e volta a repetir o processo descrito neste parágrafo. É de notar que enquanto um carro está na box o TeamManager pode receber sinais, sendo que, se receber, o carro que está na box é libertado.

Quando recebe um sinal para terminar fica à espera que todos os carros terminem a sua corrida. Quando todos terminarem, este processo termina a sua execução. Quando recebe um sinal para interromper a corrida fica à espera que todos parem para depois enviar um sinal ao *RaceManager* a indicar o sucedido. Depois fica à espera de um sinal do mesmo processo para recomeçar/terminar a corrida.

6. Carros

Estas threads são responsáveis pela simulação dos carros. Elas ficam num loop infinito, em que a cada iteração incrementa-se a distância percorrida na *lap* e decrementa-se o combustível. No fim de cada iteração ficam à espera numa variável de condição de ter permissão, da thread gestor de tempo (explicada no capítulo 5), para avançar para a iteração seguinte. No caso de receberem uma avaria (através da message queue explicada no capítulo 3) ou de o carro ficar com combustível para 2 laps ou menos, o carro reserva a box (caso ela ainda não esteja reservada) e entra em modo de "SEGURANCA".

Quando completa uma volta, primeiramente verifica-se se a corrida já terminou ou se já está interrompida. Se for a primeira opção, o carro fica à espera que todos os carros da equipa terminem a sua lap, para depois terminar a sua execução. Se for a segunda opção o carro fica à espera que todos os carros interrompam a sua execução e de seguida fica à espera de ser notificado pelo *TeamManager* para retomar a sua marcha. Se nenhuma dessas opções for escolhida, o carro verifica se terminou o número de laps suficiente para acabar a corrida. Se sim, comunica através do unnamed pipe o término ao *RaceManager* e fica à espera que todos os carros acabem a lap para depois morrer, se não verifica se precisa de entrar na box. Caso precise, verifica a disponibilidade da box e se ela estiver disponível então entra na box. Se não quiser entrar na box ou não puder entrar na box, o carro fica à espera de indicação do gestor de tempo para iniciar uma nova iteração, ou seja, iniciar uma nova lap.

Se o carro ficar sem combustível numa lap, ele desiste da corrida e espera que o resto dos carros da equipa terminem a sua execução. Se, porventura, todos os carros da mesma equipa desistirem, é comunicado ao *RaceManager* esta ocorrência (através de um sinal) e a equipa e as threads terminam a sua execução.

7. Conclusão

Está assim explicada a solução desenvolvida. É de referir que em vários locais críticos é feita a utilização de semáforos (demonstrado no diagrama da arquitetura e mecanismos de sincronização) para evitar erros graves no programa. Em termos de tempo consumido por cada membro do grupo, estima-se cerca de 80 horas de esforço.