Programação Concorrente - Trabalho 1

Lucas Mafra Chagas
12/0126443

13 de Novembro de 2018

1 Introdução

Programação Concorrente consiste de uma coleção de processos e objetos compartilhados. Cada processo é definido por um programa sequencial e os objetos compartilhados permitem que esses programas consigam trabalhar entre si para completar algum objetivo [1].

Durante o semestre, foram estudados práticas para otimizar e garantir a concorrência. Essas práticas tem como objetivo garantir com que não aconteça condição de corrida, por meio de exclusão mútua da região crítica, além de evitar problemas de starvation e deadlock, durante a construção de projetos e sistemas.

Condição de corrida ocorre quando dois ou mais processos estão acessando dados compartilhados e o resultado final depende de quem executa e quando executa. Neste caso, as instruções dessas duas ou mais seções críticas podem ser executadas de forma intercalada, e não em regime de exclusão mútua [4]. É indicado a utilização de exclusão mútua para garantir com que não ocorra a condição de corrida.

Exclusão mútua impede com que dois processos atinjam a região crítica ao mesmo tempo [3]. A região crítica é definida como a parte do código onde é acessado a memória compartilhada [5]. Não pode ser acessada por mais de um processo ao mesmo tempo, fazendo com que o acesso de dois ou mais ao mesmo tempo leva a ocorrência da condição de corrida.

Além dos problemas de condição de corrida, existem outros problemas enfrentados na construção de programas concorrentes. Os mais conhecidos e enfrentados durante o semestre são starvation e deadlock.

Starvation acontece quando um processo tem o seu acesso a memória compartilhada perpetuamente negado pelos outros processos [6]. Normalmente ocorre quando todas os seus pedidos de utilização da memória compartilhada são negadas, fazendo com que o processo não atue no programa. Já o

deadlock acontece quando dois ou mais processos estão esperando um outro processo liberar o acesso a memória compartilhada [2]. Deadlocks fazem com que nenhum processo tenha acesso a memória, fazendo com que o programa não finalize.

Para o projeto levantado, foi idealizado uma solução de um problema de comunicação entre os processos através de memória compartilhada que envolva condições de corrida utilizando mecanismos de sincronização entre processos, utilizando a biblioteca POSIX Pthreads.

2 Formalização do Problema Proposto

O projeto proposto tem como objetivo trabalhar as interações que acontecem dentro do estádio de futebol. Tanto os fãs do esporte bretão, como os jogadores dois times (Time A e Time B), chegam ao mesmo tempo no estádio, porém o jogo só começa quando todos os 11 jogadores de cada time estiverem no estádio prontos para a partida. Enquanto isso, os torcedores podem comprar um delicioso cachorro-quente a qualquer momento, antes e durante a partida. Depois da partida, os dois times vão para as suas casas, além dos torcedores.

3 Descrição do Algoritmo Desenvolvido

O algoritmo desenvolvido utilizou, para a solução do problema proposto, variáveis de condição, semáforo e lock. O código foi separado em 5 funções principais: Partida; Time A; Time B; Torcedores; e Cachorro-quente. Além dessas funções, o algoritmo desenvolvido também apresenta a função Main e funções responsáveis pela escrita na tela, mostrando a interação das threads.

A partida só começa quando ambos os times chegam no estádio. Quando a partida termina, ambos os times saem do estádio. A solução proposta foi a utilização de variáveis de condição. Após pegar o lock, a partida vai esperar o sinal de inicio de jogo enviado pelos dois times. Caso não receba o sinal, ele libera o lock. Quando ele recebe esse sinal, ele começa a partida, após isso, ele libera o lock para qualquer outra interação. Depois disso, ele pega o lock, informa que acabou a partida e avisa a todos que acabou a partida. Foi levantado, durante a execução do projeto, a utilização de barreiras para o controle dos jogadores chegando ao estádio, porém a solução não conseguiu resolver os problemas enfrentados. Portanto, foi utilizado variáveis de condição, já que alcançou o objetivo proposto da função.

O time A e o time B apresentam lógicas parecidas. Ambos os times

chegam em momentos aleatórios. Após os 11 jogadores de ambos os times chegarem no estádio, a partida começa. Após a partida terminar, ambos os times vão embora. Após pegarem o lock, os times verificarão se o time está completo. Se estiver, eles esperarão o ônibus para irem ao estádio. Se não, os jogadores chegarão 1 a 1, avisando que chegaram. Eles liberam o lock depois dessa interação. Depois de pegarem o lock novamente, eles avisarão, caso estejam com todos os jogadores, que já pode começar a partida. Por fim, eles esperarão a partida acabar, indo embora logo em seguida.

Os torcedores apresentam duas interações. Com o estádio e com o vendedor de cachorro-quente. Como o estádio tem limite, os torcedores que não conseguiram chegar a tempo, infelizmente não vão poder entrar. Portanto, eles avisarão que não puderam entrar pois o estádio já está lotado. Para isso, a solução proposta foi a utilização de semáforos, já que é uma representação compacta de contadores. Para os que conseguiram entrar, eles vão pegar um cachorro quente com o vendedor de cachorro-quente. Caso não tenha cachorros-quentes, eles irão avisar ao vendedor de cachorro-quente que acabou. A solução proposta foi a utilização de variáveis de condição. Ao verificarem que não tem mais cachorro-quente, eles esperam o cachorro-quente ficar pronto. Caso tenha cachorro-quente, eles pegam na ordem que chegam no estádio. O último a pegar também avisa ao vendedor de cachorro-quente que acabou.

O vendedor de cachorro-quente verifica se tem cachorro-quente para os torcedores. Se tiver, ele avisa que ainda tem e não mexe na quantidade, mas se não tiver, ele irá cozinhar, depois avisa aos torcedores que tem cachorro-quente. A solução proposta também foi a utilização de variáveis de condição.

4 Conclusão

O projeto proposto buscou utilizar da maioria das técnicas vistas, como semáforos, variáveis de condição e utilização da exclusão mútua, colocando em prática a união entre diversos mecanismos de sincronização de processos vistos durante o semestre. Com a execução do projeto, foi possível compreender melhor a utilização de programação concorrente dentro de diversos espectros da computação, além de aprender como utilizar melhor a memória compartilhada.

References

[1] Gregory R Andrews and Fred B Schneider. Concepts and notations for concurrent programming. *ACM Computing Surveys (CSUR)*, 15(1):3–43, 1983.

[2] George F Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed systems: concepts and design*. pearson education, 2005.

[3] Edsger W Dijkstra. Solution of a problem in concurrent programming control. In *Pioneers and Their Contributions to Software Engineering*, pages 289–294. Springer, 2001.

[4] Mauricio Lima Pilla, Rafael Ramos dos Santos, and Gerson Geraldo H Cavalheiro. Introdução à programação para arquiteturas multicore. *Escola Regional de Alto Desempenho, Caixias do Sul*, 2009.

[5] Michel Raynal. *Concurrent programming: algorithms, principles, and foundations*. Springer Science & Business Media, 2012.

[6] Andrew S Tanenbaum. Modern operating systems, 2001. *Prentice-Hall, Inc., second edition, pages*, 200:184–185, 2009.