Uma comparação entre uma aplicação single thread e multithread com e sem proteção da seção crítica na simulação de uma oficina mecânica.

Bianca Carvalho de Oliveira
Universidade Estadual do Oeste do Paraná - UNIOESTE
Cascavel, Brasil
bianca.c.o@unioeste.com

I. INTRODUÇÃO

Uma aplicação multithread pode ter vários fluxos de execução simultâneo diferentes, pertencentes a diferentes *threads*. Esses *threads* compartilham o mesmo espaço de endereço privado do processo e compartilham todos os recursos adquiridos pelo processo. Eles são executados no mesmo contexto de execução do processo e, portanto, um thread pode influenciar outras threads no processo [1].

Os processos modernos podem ser categorizados em dois tipos: *single-thread* (processos tradicionais) e *multithread*. A execução de um programa em um processo de *thread* único é sequencial, enquanto a execução de um programa em um processo *multithread* é concorrente.

Um processo concorrente pode afetar ou ser afetado pelos outros processos que estão executando no sistema. Eles podem compartilhar diretamente um espaço de endereçamento lógico (ou seja, código e dados), o que pode acarretar uma inconsistência de dados [2].

Para controlar o acesso a um recurso compartilhado, é declarado uma seção de código chamada seção crítica; em seguida, controla-se o acesso a essa seção de modo que quando um processo está executando sua seção crítica, nenhum outro processo deve ter autorização para fazer o mesmo. Assim, a execução das seções críticas pelos threads é mutuamente exclusiva no tempo [2][3].

Visando superar essa dificuldade, pode-se usar uma ferramenta de sincronização, denominada semáforo. Um semáforo S é uma variável inteira que, além da inicialização, só é acessada através de duas operações-padrão: *wait* () e *signal* (). Todas as modificações do valor inteiro do semáforo nas operações *wait* () e *signal* () devem ser executadas indivisivelmente. Isto é, quando um processo modifica o valor do semáforo, nenhum outro processo pode modificar o valor desse mesmo semáforo simultaneamente [2][3].

O presente trabalho tem como objetivo simular o uso de elevadores automotivos em uma oficina mecânica utilizando uma aplicação single thread e uma multithread, e alterando a execução delas com e sem a proteção da seção crítica, a fim de, posteriormente comparar o desempenho de ambas as abordagens, assim como identificar a importância da proteção da seção crítica em aplicação *multithread*.

O artigo está organizado da seguinte forma. A Seção 2 apresenta a descrição do problema. A seção 3 apresenta a descrição da implementação. Seção 4 descreve a análise dos resultados. Por fim, a seção 5 conclui o trabalho.

II. DESCRIÇÃO DO PROBLEMA

O problema escolhido foi uma oficina mecânica na qual há n de elevadores automotivos para atender uma demanda de m carros. Cada carro possui uma identificação e precisa utilizar um elevador por um determinado tempo. Já cada elevador pode atender somente um carro por vez.

O desafio da implementação desse problema consiste em fazer um algoritmo capaz de implementar cada um dos elevadores de modo que estes sempre consigam atender a demanda de carros. Por se tratar de uma implementação com acesso concorrente aos dados – no caso é a fila de demandas – deve-se considerar as situações na qual dois ou mais elevadores podem adquirir o mesmo carro e um ou mais carro pode ficar eternamente na fila.

III. DESCRIÇÃO DA IMPLEMENTAÇÃO

Visando analisar o desempenho de ambas as abordagens, bem como identificar os problemas descritos anteriormente, foi implementada uma solução utilizando a linguagem JAVA, dividida em três soluções principais: *single-thread*, *multithread* com a seção crítica protegida e *multithread* com a seção crítica desprotegida.

O intuito das soluções desenvolvidas foi simular o funcionamento de uma oficina mecânica que deve atender a uma fila de demandas obedecendo à regra de que somente um carro pode utilizar um elevador por vez e todos os carros devem ser atendidos. Em ambas as abordagens essa fila é gerada de maneira automática, nela armazena-se os carros, cada qual possui um identificador e um tempo de utilização do elevador – gerado de forma aleatória através de um *Random* e varia de 1 a 5 milissegundos.

A. *Implementação single thread*

Nesta abordagem utiliza-se a classe *FilaCarros* que contém um método *sequencial* – responsável por executar a aplicação de maneira sequencial. Este método apresenta o seguinte funcionamento: enquanto houver carro na fila, retirasse o carro para o elevador consumindo um tempo de serviço x. Isso é feito sucessivamente até que não haja mais carros na fila.

Como a execução é sequencial apenas um carro será atendido por vez, pegando um elevador, consumindo o tempo necessário para processar a demanda e devolvendo o elevador para que o próximo consiga utilizar.

B. *Implementação multithread com proteção da seção crítica*

Nesta abordagem utiliza-se as classes *Threads* e *FilaCarros* – responsáveis por simular o problema de forma concorrente. Da classe *Threads* utiliza-se os métodos:

criarThreads, responsável por inicializa um vetor de threads com n posições, onde cada *thread* representa um elevador; e o método *consumidor*, responsável por processar a fila de demanda. Já da classe *FilaCarros* utiliza-se o método *comProtecao* – responsável por consumir um carro com proteção da seção crítica através de um semáforo *mutex*.

Visto que cada carro deve ser processado apenas uma vez, a ação do elevador de adquirir um carro na fila de demanda deve ser realizada de maneira atômica, caracterizando a seção crítica do problema.

A fim de evitar os problemas citados na seção 2, quando um elevador está adquirindo um carro, ele tenta adquirir o semáforo. Caso consiga, ele pega o carro e ao terminar libera o semáforo. Caso contrário, ele aguarda até poder adquirir o semáforo.

C. *Implementação multithread sem proteção da seção crítica*

Nesta abordagem utiliza-se as classes *Threads* e *FilaCarros* – responsáveis por simular o problema de forma concorrente. Da classe *Threads* utilizou-se os mesmos métodos citados no ponto acima. Já da classe *FilaCarros* utiliza-se o método *semProtecao* – responsável por consumir um carro sem proteção da seção crítica.

Como o intuito dessa abordagem é expor os erros que podem ocorrer quando não se protege a seção crítica, não se utilizou semáforo para controlar o acesso a fila de demandas.

IV. Análise dos Resultados

Tabela I mostra a comparação do tempo médio de execução em milissegundos para as implementações single thread, multithread sem a seção protegida. E a Tabela II mostra a comparação para as implementações single thread, multithread com a seção protegida. Ambas têm como base a média do resultado de três iterações.

TABLE I. TEMPO EM EXECUÇÃO (EM MILISSEGUNDOS)

Tempo em execução da implantação sequencial e multithread sem proteção da seção crítica (em milissegundos)					
Tipo/ Fila	64	128	256	512	1024
Sequencial	193484,6	383977,0	735479,0	1503371,3	3079007,6
4 threads	55784,0	98575,0	188835,3	387639,0	777717,3
8 threads	31075,3	53810,0	96576,6	197178,3	395800,3
16 threads	17053,3	30068,6	54467,0	103626,6	210878,3
32 threads	9021,6	15390,3	30068,0	55143,6	110968,0
64 threads	5012,0	9353,0	15038,3	28416,6	56837,6
128 threads		5012,0	9021,3	15698,3	28422,0
256 threads			5013,3	9357,0	15752,6
512 threads				5032,6	9368,6

TABLE II. TEMPO EM EXECUÇÃO DA (EM MILISSEGUNDOS)

Tempo em execução da implantação sequencial e multithread com proteção da seção crítica (em milissegundos)					
Tipo/ Fila	64	128	256	512	1024
Sequencial	193484,7	383977,0	735479,0	1503371,3	3079007,7
4 threads	50109,0	97915,7	185487,0	378341,3	771698,3
8 threads	27060,3	50125,7	93916,7	189858,0	387429,7
16 threads	14698,7	26390,7	48134,0	96909,7	194830,7
32 threads	9032,3	14704,0	26055,3	50120,3	98905,0
64 threads	5011,7	9352,7	14696,3	26399,3	51122,7
128 threads		5009,3	9012,3	15042,0	27075,7
256 threads			5010,3	9022,0	15036,3
512 threads				5032,7	9039,7

Observando os resultados, nota-se que ao utilizar uma abordagem sequencial, o tempo de espera aumenta exponencialmente, em proporção ao tamanho da fila. Já as que utilizam multithread tem o tempo de execução menor, ao decorrer que o número de threads aumenta.

Nota-se também que a aplicação multithread com a seção crítica desprotegida tem um resultado pior do que a aplicação com a seção crítica protegida.

Isso pode ser observado de forma evidente por meio da comparação das três de abordagens, como mostrado na Figura 1

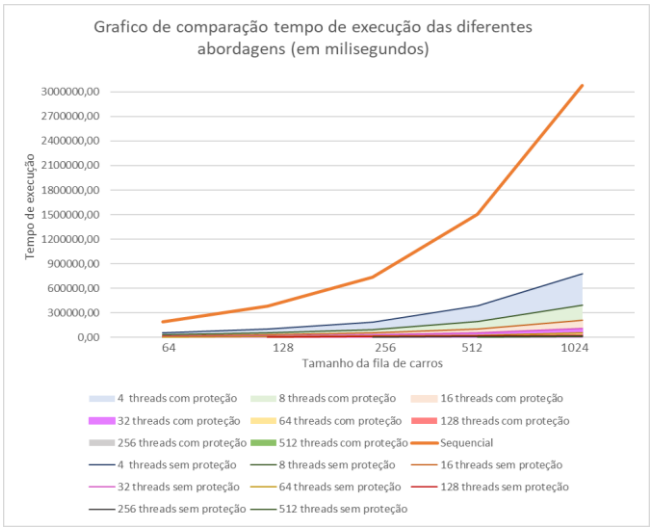

Figura 1: Gráfico comparativo.

Além disso com a análise dos dados obtidos a partir da execução da aplicação foi observado que a abordagem usando multithread com a seção crítica desprotegida apresenta dois problemas: como a seção crítica pode ser acessada por mais de um thread pode ocorrer de dois ou mais semáforos adquirirem o mesmo carro e um ou mais carros permanecer eternamente na fila de execução.

V. Conclusão

Considerando esses resultados, e possível concluir que o problema proposto é mais bem solucionado utilizando a abordagem multithread com seção crítica protegida, já que ela apresenta um melhor tempo de execução e evita a ocorrências de erros apresentados pela abordagem multithread com seção crítica desprotegida.

Conclui-se também que a proteção da seção crítica do código é extremamente necessária para aplicações multitread, a fim de evitar problemas.

VI. Referencias

[1] S. Haldar, and A. A. Aravind, Operating systems (self edition 1.1 Abridged). Redwood: Oracle Corporation, 2016

[2] A. Silberschatz, P. B, Galvin and G. Gagne, Sistemas operacionais: conceitos e aplicações. 8ª ed. Rio de Janeiro: Campus, 2001.

[3] A. Silberschatz, P. B, Galvin and G. Gagne, Fundamentos de sistemas operacionais. 9ª ed. Rio de Janeiro: LTC, 2010.