PROBLEMA DO JANTAR DOS FILÓSOFOS

Daniel Winter Santos ROCHA[1]; Filipe Soares FERNANDES[1]; Júlio Cesar Machado ÁLVARES[1]; Marco Aurélio Monteiro LIMA[1];Marcus Vinícius Rodrigues CAMPOS[1]; Samuel Pereira DIAS[2];

[1] Alunos do Curso de Engenharia de Computação do IFMG -*Campus* Bambuí

[2]Professor do IFMG campus Bambuí

Bambuí - MG - Brasil

{danielwinterifmg; filipe.soares94; juliocmalvares07; marco.monteirolima; marcusrodriguesc321}@gmail.com, samuel.dias@ifmg.edu.br

RESUMO

O jantar dos filósofos é caracterizado como um problema clássico de sincronismo.O artigo prevê uma solução para o problema, portanto, dois critérios devem ser inicialmente atendidos: os filósofos não devem tentar comer ao mesmo tempo (*deadlock*), nenhum filósofo deve morrer de fome (*starvation*). Deve-se ainda resolver o problema de seção crítica e, para isso, três condições devem ser satisfeitas. O resultado esperado é a implementação de um código em que nenhum dos problemas acima ocorra.

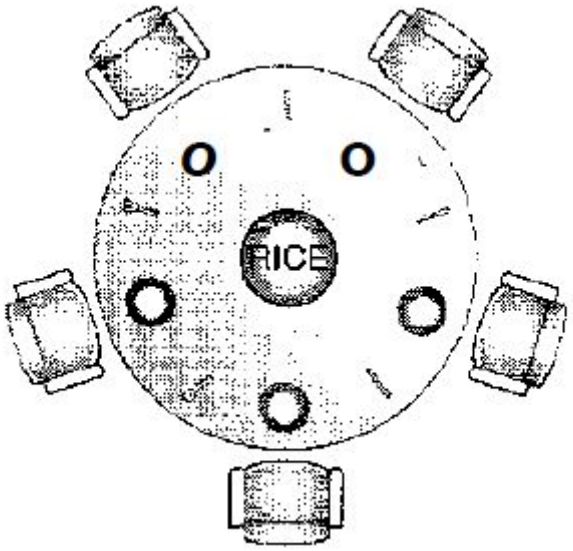
Palavras-chave: *starvation*, *deadlock*, jantar dos filósofos, problema de sincronismo.

1 INTRODUÇÃO

O problema do jantar dos filósofos é uma situação específica: Existem cinco filósofos que passam suas vidas comendo e pensando.Eles compartilham uma mesa com cinco cadeiras, cada uma pertencente a um filósofo e ao centro, existe uma tigela com arroz. São dispostos na mesa 5 hashis. Para comer, um determinado filósofo necessita possuir dois hashis, podendo pegar apenas um por vez, começando pelo da esquerda e depois o da direita.

O objetivo é desenvolver um algoritmo que simule o jantar dos filósofos corretamente, livre de deadlock, starvation e satisfazendo os requisitos do problema da seção crítica.

Figura 1- A situação dos filósofos jantando.



Fonte: Silberschatz (2008)

Dois problemas principais precisam ser evitados. O primeiro é o *deadlock*. Se todos os filósofos se apropriam de um hashi e tentam comer, nenhum deles irá ceder o hashi que adquiriu, sendo assim, nenhum deles irá comer, pois para isso, precisam de dois hashis.
O segundo problema é o *starvation*. Ocorre quando o filósofo não consegue pegar o hashi da esquerda porque seu vizinho à esquerda está usando ou quando ele consegue pegar o hashi da esquerda mas o hashi da direita está sendo usado pelo vizinho à direita. Isso faz com que o filósofo nunca consiga comer, logo, morre de fome.

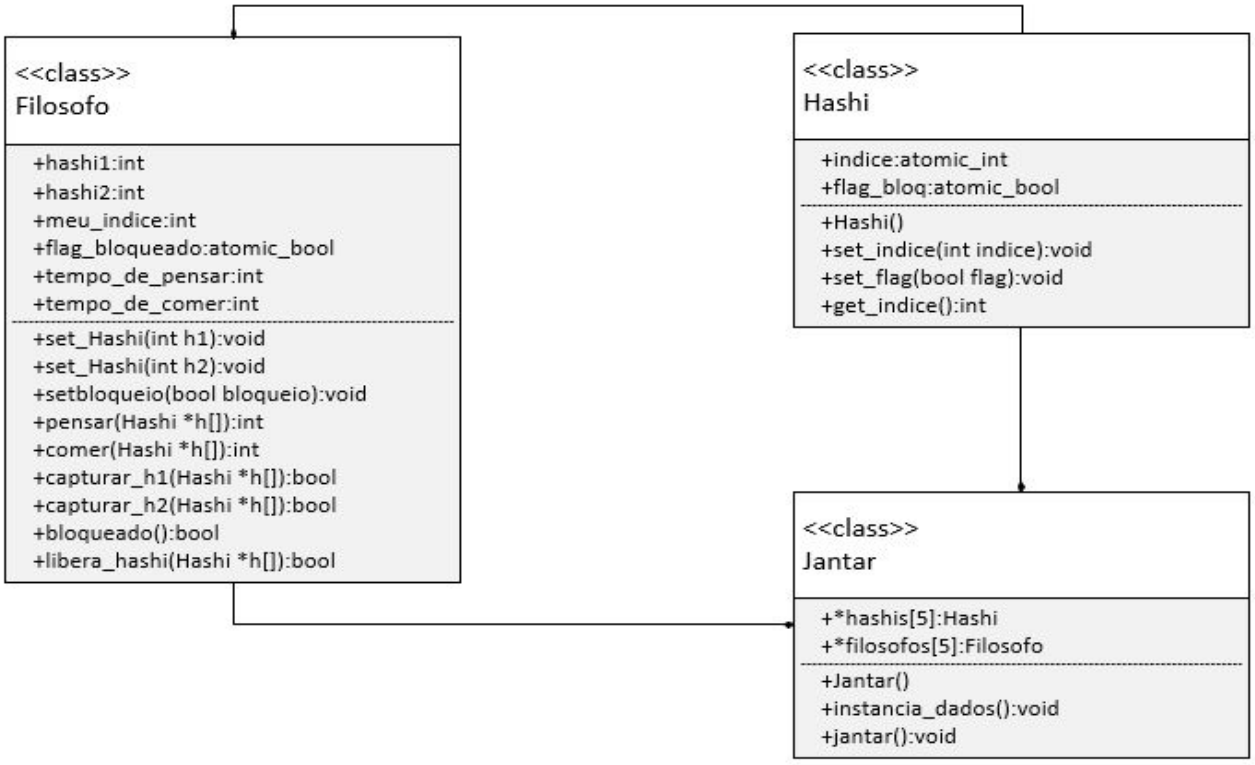
Um terceiro problema a ser levado em conta, é o problema da seção crítica. Segundo Silberschatz (2008, p.148) ”Cada processo possui um segmento de código, chamado seção crítica, onde o processo pode alterar variáveis comuns, atualizando uma tabela, gravando um arquivo, e assim por diante”. Para projetar uma solução para o problema da seção crítica três requisitos devem ser satisfeitos. Primeiramente, exclusão mútua. Somente um processo pode executar sua seção crítica por vez. Depois, progresso, prevê que todos os processos serão eventualmente serão executados. Por último, espera limitada, limita o número de vezes que os

processos acontecem, para garantir que quaisquer outros processos executem pelo menos uma vez, evitando o *starvation* dos processos.

2 DESENVOLVIMENTO

Para a solução do problema, foi desenvolvido um *software* na linguagem C++ versão 11, utilizando do paradigma orientado a objetos e com auxílio do ambiente de desenvolvimento Code Blocks versão 16.01. Contamos ainda com o uso das bibliotecas *atomic, thread, time, chrono*, além das bibliotecas nativas *iostream* e *cstdlib*. Primeiramente, foi desenvolvido o diagrama de classes UML e em seguida foi desenvolvido o código.

Figura 2- Diagrama UML da aplicação.



Fonte: (OS AUTORES, 2017)

2.1 Solução do problema de *deadlock.*

O deadlock foi resolvido fazendo com que, para um filósofo pegar os hashis ele precisa, primeiramente, verificar um atributo booleano atômico de cada recurso em questão.

Se um dos hashis estiver bloqueado, o filósofo irá liberar o hashi que já capturou, tornando-o cooperativo. Para que o filósofo possa comer, o mesmo deve ter posse de ambos os hashis. Foi implementado um bloqueio de dois níveis, sendo esse feito nos hashis e nos filósofos, evitando que filósofos concorrentes peguem um recurso que já está sendo usado. Quando o filósofo termina a operação de comer, ele libera ambos hashis usados e desbloqueia a si mesmo para realizar outra operação. Para impedir que filósofos concorrentes acessem o mesmo dado ao mesmo tempo, ocasionando condição de corrida, as variáveis atômicas são modificadas com operações utilizando o método "*memory_order_acquire"*, "onde as operações concorrentes são ordenadas para acontecerem uma vez que todos os acessos a posição de memória da variável acontecer". (CPLUSPLUS.COM, 2017)

2.2 Solução do problema de *starvation.*

O problema de starvation foi resolvido seguindo a premissa de que, a probabilidade de um filósofo não comer por diversas vezes seguidas é remota, tendo em vista que as 5 *threads* tem acesso a todos os filósofos. A forma como cada *thread* escolhe um filósofo é aleatória e o segundo nível de bloqueio implementado, força a *thread* a escolher um filósofo que não esteja comendo. Logo, para que um filósofo não coma por diversas rodadas, os seus vizinhos precisam comer por diversas rodadas seguidas e os mesmos precisam ser desbloqueados no mesmo instante da solicitação do filósofo em questão.

2.3 Solução do problema de seção crítica.

Para resolver a exclusão mútua, o filósofo deve necessariamente tentar pegar os dois hashis. Caso seja possível, o filósofo pode comer, caso não, ele libera os hashis para os outros. O problema de espera limitada foi resolvido partindo do princípio que as *threads* tem acesso aleatório aos filósofos e ainda há condições para priorizar filósofos que não estejam bloqueados, logo, a probabilidade de espera de todos os filósofos é igual. A progressão é garantida não só pelo uso das bibliotecas *atomic* e *thread*, mas como também pelo fato de que na implementação, os filósofos têm limites de tempo para suas funções de comer e pensar.

3 CONCLUSÃO

Quase todos os objetivos do trabalho foram atingidos. O algoritmo satisfaz os os requisitos do problema de seção crítica e possui tratamento eficaz a respeito de *deadlocks*. Existe, porém, uma deficiência na solução do *starvation*, a possibilidade de um filósofo morrer de fome é remota, mas existente.

DINING-PHILOSOPHERS PROBLEM

ABSTRACT

The Dining-Philosophers Problem is characterized as a classic synchronism problem. The article aims a solution to the problem, to that, two criteria must be met initially: all the philosophers should not try to eat at the same time(deadlock), no philosopher should starve(starvation). There must yet solve the critical-section problem and, for that, three conditions must be satisfied. The expected result is the implementation of a code in which none of the cited problems above happen.

***Keywords**: starvation, deadlock, Dining-Philosophers Problem. synchronism problem.*

REFERÊNCIAS

SILBERSCHATZ, Abraham; GALVIN, Peter Baer; GAGNE, Greg. **Operating System Concepts**.7ed. Disponível em: <https://users.dimi.uniud.it/~antonio.dangelo/OpSys/materials/Operating_System_Concepts.pdf>. Acessado em: 10 dez. 2017.

Atomic Library. Disponível em: <http://pt.cppreference.com/w/cpp/atomic>. Acessado em: 9 dez. 2017.

Memory order. Disponível em: <http://www.cplusplus.com/reference/atomic/memory_order/>. Acessado em: 9 dez. 2017.