UNIVERSIDADE FEDERAL DO ESTADO DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS ELETRÔNICOS
PROGRAMAÇÃO DE SISTEMAS ELETRÔNICOS
WUERIKE HENRIQUE DA SILVA CAVALHEIRO

RELATÓRIO DE RESOLUÇÃO DO PROBLEMAS "O BAR DOS FILÓSOFOS"

DESCRIÇÃO DO PROBLEMA

O Bar dos Filósofos é um problema de programação concorrente que visa abordar questões de gerenciamento de recursos compartilhados entre processos e/ou threads, foi proposto por Chandy e Misra em 1984 [1] como uma generalização de um problema anterior conhecido como Jantar dos Filósofos.

O problema diz que N filósofos estão numa bebedeira, eles podem estar dispostos em formas aleatórias, definidas num grafo em que cada vértice representa um filósofo enquanto as arestas expressão uma relação de vizinhança entre eles. Cada filosofo compartilha uma garrafa com cada um de seus vizinhos, durante as sessões de bebedeira os filósofos escolherão quantas garrafas querem beber e de quais vizinhos pegarão essas garrafas, um filósofo só poderá beber se adquirir todas as garrafas necessárias.

Nas sessões de bebedeiras os filósofos passam por 3 estados diferentes, na ordem, TRANQUILO, COM SEDE e BEBENDO. Quando TRANQUILO, um filósofo apenas aguarda por um tempo aleatório de 0 a 2 segundos até que fique COM SEDE. Neste momento, o filósofo em questão escolherá um número aleatório de garrafas que deve variar de 2 ao número máximo de vizinhos que ele possui. Este filósofo deve então solicitar as garrafas aos seus vizinhos com quem às compartilha, assim que possuir todas ele entra no estado de BEBENDO que dura por 1 segundo e então volta a TRANQUILO. Após beber o filósofo então pode disponibilizar as garrafas que está segurando para que os outros filósofos também possam beber.

A execução correta deste problema ocorre quando todos os filósofos são capazes de passarem por todos os estados um mesmo número de vezes de forma que o tempo médio no estado COM SEDE seja equilibrado.

DESCRIÇÃO DO ALGORITMO

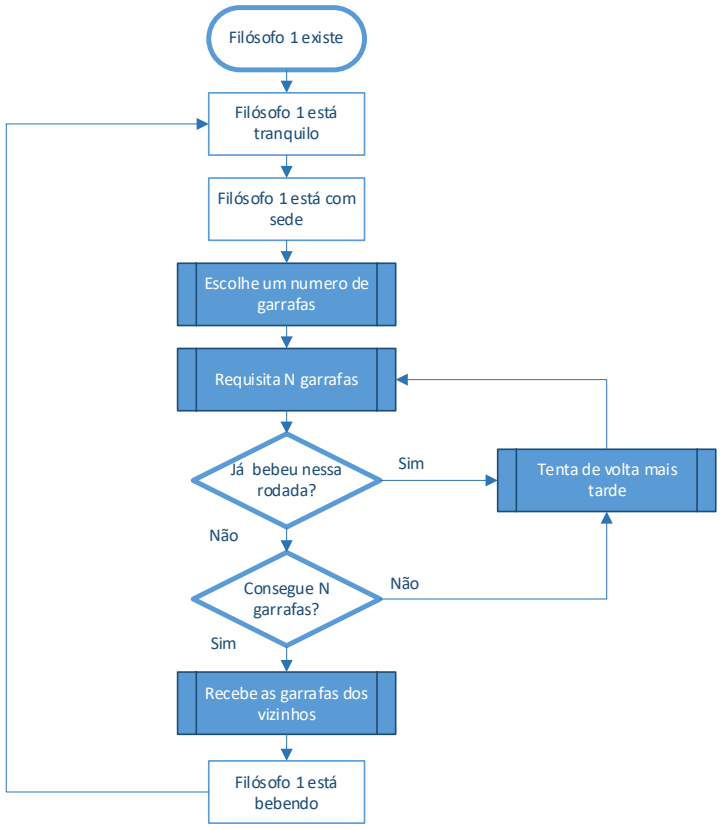
O algoritmo utilizado baseia-se essencialmente na solução proposta por Chandy e Misra, chamada de solução higiênica [1], onde o requisito para que a garrafa – ou garfo no caso do Jantar dos Filósofos – deve estar limpa para que um filósofo possa a usar. Enquanto um filósofo está segurando uma garrafa, estando ele no estado de BEBENDO ou em qualquer outro momento durante a transição de estados, a garrafa é considerada suja e o outro filósofo com quem a garrafa é compartilhada não a acessa.

Quando um filósofo termina de beber ele deixa de segurar a garrafa, a colocando sobre a mesa, nesse instante a garrafa ainda não está limpa, mas outros filósofos que precisam dela são capazes de perceber que não há ninguém bebendo. Nesse instante o filosofo que deseja beber requisita a garrafa ao seu vizinho, como a garrafa está suja sobre a mesa o vizinho é quem faz o trabalho de entregá-la – enquanto a limpa – ao filósofo que fez requisição.

Para garantir uma melhor ordenação, o filósofo não solicita garrafas aos seus vizinhos a menos que ele perceba ser capaz de pegar todas as garrafas que precisa, caso seja incapaz de fazer isso, ele então permite que outro filósofo faça sua requisição e tenta novamente mais tarde. Um filósofo só é impedido de tentar solicitar as garrafas desejadas se ele já houver bebido nessa rodada, isso garante que todos os filósofos vão beber uma vez antes que alguém beba duas, equilibrando assim os tempos de espera entre eles – e ninguém passará sede em excesso.

O fluxograma abaixo demonstra o algoritmo de comportamento de um dos possíveis n filósofos que podem ser simulados.

Figura 1 – Ciclo de comportamento para um filósofo.

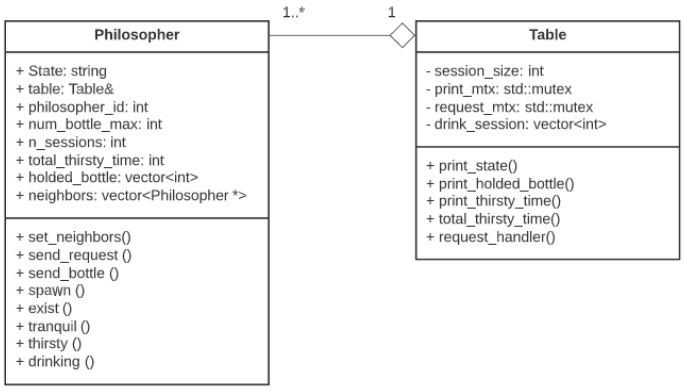


Fonte: O Autor.

DESCRIÇÃO DA IMPLEMENTAÇÃO

O problema foi resolvido através da de programação orientada a objetos, onde criou-se uma classe para filósofos e outra para a própria mesa onde ocorre a bebedeira. Todos os filósofos são instanciados da mesma forma, recebendo um ID único e uma instância da mesa, dessa forma, não se atribui prioridades pré definidas aos filósofos, deixando a cargo do algoritmo eleger qual filósofo tem prioridade sobre os recursos a cada requisição. O diagrama UML abaixo demonstra o relacionamento entre as classes utilizadas.

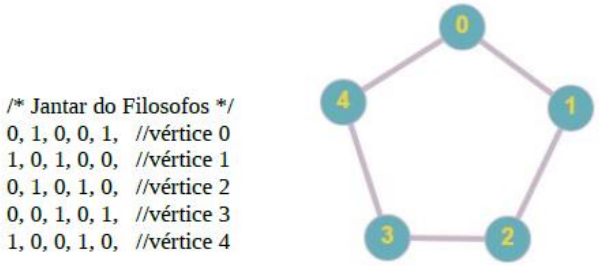
Figura 2 – Diagrama UML.



Fonte: O Autor.

Antes que os objetos envolvidos sejam devidamente instanciados, deve-se primeiro conhecer a quantidade de filósofos, a disposição deles e a quantidade de rodadas de bebedeira. Esses parâmetros são dados ao software ao executá-lo na linha de comando, sendo que a quantidade de filósofos e o relacionamento entre eles se dá através da interpretação de uma matriz, que por sua vez representa um grafo.

Figura 3 – Exemplo de matriz e grafo.



Fonte: [2].

Inicialmente o software interpreta a matriz e extrai a quantidade de vértices que grafo possui, que representa o número de filósofos que serão instanciados. Quando a quantidade correta de filósofos está instanciada, através de uma segunda análise da matriz, o software determina a relação de vizinhança entre os filósofos, de forma que cada filósofo recebe um vetor de ponteiro para a instância de seus filósofos vizinhos. O fim desta etapa de preparação ocorre ao criar uma thread para cada filósofo e então iniciar o ciclo de vida dos filósofos, conforme Figura 1.

Os filósofos então executarão, conforme número do parâmetro de entrada, as rotinas que definem seus estados de TRANQUILO, COM SEDE e BEBENDO. O primeiro e o ultimo estados são simples, onde o filósofo apenas aguarda por um determinado tempo, enquanto o segundo estado é o responsável por lidar com o compartilhamento das garrafas. A seguinte figura exibe a principal parte da rotina que descreve este estado.

Figura 4 – Núcleo da rotina do estado de sede.

```
// Choose bottles from 2 to number of neighbors
int num_bottle_needed = 2 + rand()%(this->num_bottle_max-1);

// Send a request for his neighbors
while(!send_request(neighbors, num_bottle_needed))
{
    // Timeout between requests
    sleep_for(milliseconds(50));
}

// Show the bottles he get
for (int i=0; i<holded_bottle.size(); i++){
    table.print_holded_bottle(philosopher_id, holded_bottle[i]);
}
```

Fonte: O Autor.

O filósofo escolhe um número de aleatório de garrafas que varia de 2 ao número de vizinhos que possui e então chama o método que requisita essas garrafas. Se esse método retornar falso, significa que as garrafas não estavam disponíveis e neste caso o filósofo deve fazer outra tentativa. Quando o filósofo enfim conseguir a quantidade de garrafas definidas inicialmente, exibe-se no terminal de quais foram as garrafas obtidas e então ele dá continuidade ao seu ciclo, passando para o estado de BEBENDO.

O método *send_request* chama o método *request_handler* da classe *Table*, que faz o gerenciamento dos recursos compartilhados. Ao entrar nesse método um *mutex* é bloqueado,

impedindo que outros processos o executem no mesmo instante, dessa forma a exclusão mútua evita uma disputa inválida pelos recursos. A principal parte do método *request_handler* pode ser observado abaixo.

Figura 5 – Núcleo do método *request handler*.

```
// run through the philosopher neighbors
for (int i = 0; i < neighbors.size(); i++)
{
    // for each neighbor check if he is holding the bottle
    vector<int>::iterator position = find(  neighbors[i]->holded_bottle.begin(),
                                            neighbors[i]->holded_bottle.end(),
                                            requester_id);
    // true if the neighbor is not holding the bottle
    if (position == neighbors[i]->holded_bottle.end())
    {
        // add the neihbor id to the available bottles list
        available_bottles.push_back(neighbors[i]->philosopher_id);
    }
}

// check if there is enough bottles
if (available_bottles.size() >= n_bottles)
{
    // look for the neighbors add to available bottles list
    // and then make then send the bottle to the requester
    for (int i = 0; i < neighbors.size(); i++)
    {
        for (int j = 0; j < available_bottles.size(); j++)
        {
            if(neighbors[i]->philosopher_id == available_bottles[j])
            {
                neighbors[i]->send_bottle(requester_id);
                sended_bottles++;
            }
        }
    }
}
// If there isn't enough bottles try again later
else
{
    return false;
}
```

Fonte: O Autor.

Sendo que todo esse trecho de código está sendo protegido pelo *mutex*, sabe-se que apenas um filósofo por vez executará essa rotina, bem como uma lógica na entrada garante que filósofos que ainda não beberam nesta rodada terão prioridade de execução.

Portanto, a requisição é processada em duas etapas, no primeiro laço verifica-se entre os vizinhos quantas garrafas estão disponíveis, adicionando o id de cada vizinho que não estiver segurando uma garrafa no vetor *available bottles*. Se houver uma quantidade suficiente de garrafas, então um segundo laço é executado, fazendo com que a quantidade de garrafas necessária seja enviada para o filósofo que as requisitou. Neste ponto do código pode-se observar a questão da solução higiênica, onde quem deseja as garrafas não as pega diretamente, mas quem compartilha essa garrafa com ele é quem as envia.

Na sequência do fluxo, ocorre o retorno para o método do estado COM SEDE, que se comporta como visto anteriormente. Os estados se repetem pela quantidade de vezes definida e então a simulação é encerrada.

RESULTADOS

Foram simuladas 3 situações diferentes, o primeiro caso é representado pelo grafo da figura 3, este grafo é a representação do problema Jantar do Filósofos, um grafo cíclico com 5 filósofos e 5 garrafas. O resumo dessa simulação pode ser visto na tabela abaixo.

Tabela 1 – Resumo da simulação 1.

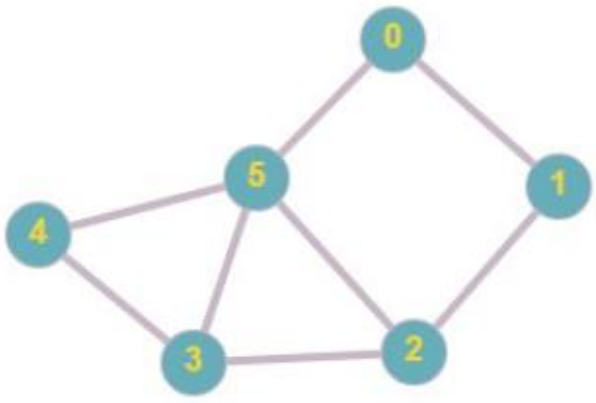
Filósofo	Tempo com sede total
0	6,802 s
1	7,298 s
2	7,145 s
3	7,040 s
4	5,358 s

Fonte: O Autor.

Foram simuladas 6 rodadas de bebedeira que demoraram 20,542 segundos para serem concluídas. Quanto aos tempos com sede, verifica-se que a variação entre o máximo e mínimo foi de aproximadamente 26,6%.

A segunda simulação foi feita em relação ao grafo apresentado na seguinte figura.

Figura 6 – Grafo usado na segunda simulação.



Fonte: [2].

Este grafo de 6 vértices também foi simulado por 6 rodadas, a Tabela 2 exibe o resumo obtido.

Tabela 2 – Resumo da simulação 2.

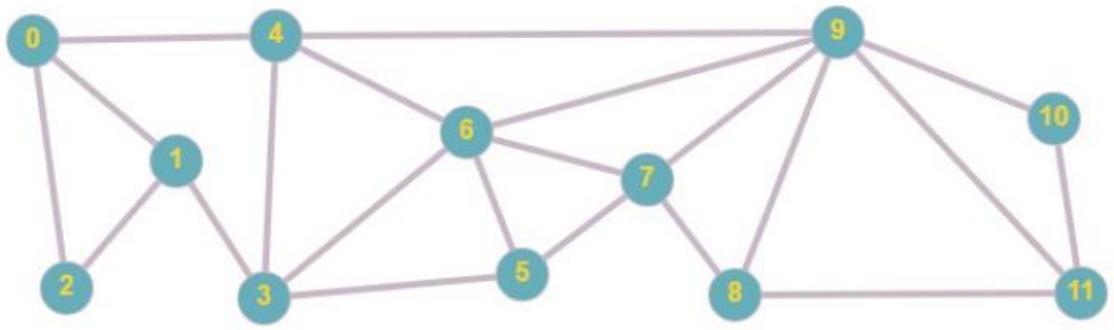
Filósofo	Tempo com sede total
0	11,271 s
1	07,254 s
2	10,064 s
3	08,731 s
4	09,691 s
5	07,907 s

Fonte: O Autor.

A duração total da simulação foi de 23,621 segundos, um filósofo a mais acarretou em pouco mais de 3 segundos de acréscimo em relação a simulação anterior. Neste caso, a variação entre o tempo com sede máximo e mínimo foi de 35,64%, um valor relativamente maior que o obtido anteriormente.

Por fim realizou-se a simulação 3, que executou o cenário proposto no grafo da Figura 7.

Figura 7 – Grafo usado na terceira simulação



Fonte: [2].

Este grafo representa um total de 12 filósofos a serem simulados, para este caso a simulação foi feita pra apenas 3 rodadas de bebedeira.

Tabela 3 – Resumo da simulação 3.

Filósofo	Tempo com sede total
0	4,362 s
1	3,996 s
2	4,696 s
3	4,762 s
4	3,434 s
5	3,528 s
6	6,670 s
7	4,336 s
8	5,725 s
9	4,690 s
10	6,579 s
11	3,260 s

Fonte: O Autor.

O tempo da simulação foi de 14,084 segundos, com variação entre tempos com sede máximo e mínimo de 51,1%, ou seja, o filósofo 6 passou sede por mais que o dobro do tempo que o filósofo 3.

Apesar de ter maior complexidade, o dobro de filósofos e metade das rodadas em relação a segunda simulação, o tempo de simulação foi significativamente menor, com quase 10 segundos a menos. Apesar lógica dizer que o dobro de filósofos e metade do numero de rodadas resultaria em tempos de simulações parecidos, deve-se se atentar que o tempo de processamento de disputa das garrafas tende a ser muito pequeno, e os maiores intervalos estão nos estados de TRANQUILO e BEBENDO, dessa forma o tempo de simulação depende mais do numero de repetições do que do número de filósofos.

Utilizando-se ainda do grafo de 12 vértices da Figura 7, realizou-se uma simulação com 100 rodadas que durou 423,759 segundos, aproximadamente 7 minutos.

Tabela 4 – Resumo da simulação 4.

Filósofo	Tempo com sede total
0	190,727 s
1	185,009 s
2	182,321 s
3	181,620 s
4	195,252 s
5	199,050 s
6	194,816 s
7	191,785 s
8	189,785 s
9	201,294 s
10	196,025 s
11	192,121 s

Fonte: O Autor.

Os resultados da Tabela 4 demonstram que quanto mais rodadas ocorrem a variação do tempo de sede vai diminuindo, neste caso por exemplo a variação foi de 9,77% entre o maior e menor tempo, filósofos 9 e 3 respectivamente. Esse fator pode ser novamente atribuído aos tempos nos outros estados, que são consideravelmente maiores, porém também pode-se imaginar que como a ordem de software é aleatória, um maior numero de rodadas pode justamente proporcionar um maior equilíbrio entre os filósofos.

CONCLUSÃO

Partindo do problema conhecido como o Bar dos Filósofos que trata do gerenciamento de recursos compartilhados, desenvolveu-se um algoritmo capaz de solucionar o desafio que rodou simulações com mais de uma hora, sem apresentar *data races* ou *deadlocks*.

Apesar de ter obtidos os resultados esperados, tem-se ciência de que algumas decisões de desenvolvimento não foram ideias, sendo a principal delas o fato de usar uma estrutura *while* para constantemente solicitar as garrafas que um filósofo precisa. Considero que essa foi uma solução que funcionou, porém que foi uma resolução baseada em força bruta. O desejado para esta parte era construir uma estrutura com filas e notificações, de forma que todas as requisições fossem adicionadas a uma fila. e se um filósofo não pudesse ser atendido ele voltasse para a fila para ter outra tentativa em outro momento.

Trabalhos futuros em cima desse desenvolvimento também necessitariam de uma melhor utilização da linguagem C++, como por exemplo uma melhor definição das classes envolvidas e o relacionamento entre elas, buscando evitar as inúmeras vezes que ponteiros de instâncias dos objetos foram passadas como parâmetros.

Referências

[1] K. M. CHANDY and J. MISRA. The Drinking Philosophers Problem. ACM Transactions on Programming Languages and Systems, Vol. 6, No. 4, October 1984, Pages 632-646. Disponível em: https://www.cs.utexas.edu/users/misra/scannedPdf.dir/DrinkingPhil.pdf.

[2] Prof. Marcial Fernández. Programação Concorrente e Paralela. Universidade Estadual do Ceará (UECE).