UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

CAMPUS CHAPECÓ

CIÊNCIA DA COMPUTAÇÃO

SISTEMAS OPERACIONAIS

PROFESSOR: NEIMAR MARCOS ASSMANN

SOLUÇÕES DO PROBLEMA DO JANTAR DOS FILÓSOFOS

Yuri Luis Malinski Lanzini - 2221100006

1. Explicação das Implementações:

Solução de inversão Aleatória:

A Solução de Inversão Aleatória é uma abordagem engenhosa para evitar impasses em um problema clássico conhecido como "O Problema dos Filósofos Jantando". Nesta solução, cada filósofo é designado para pegar seus hashis de maneira aparentemente aleatória, mas controlada, evitando assim a situação em que todos os filósofos tentam pegar o hashi à esquerda simultaneamente, resultando em um impasse.

Aqui está o trecho de código que representa essa parte da implementação:

```c
void *threadFilosofo(void *arg)
{
    int i = (long int)arg;
    while (1)
    {
        medita(i);

        int h1 = i;
        int h2 = (i + 1) % NUMFILO;

        if (rand() % 2 == 0)
        {
            int temp = h1;
            h1 = h2;
            h2 = temp;
        }

        pega_hashi(i, h1);
        pega_hashi(i, h2);
        larga_hashi(i, h1);
        larga_hashi(i, h2);

    }
    pthread_exit(NULL);
}
```

- Os hashis são representados pelos índices h1 e h2, onde h1 é o hashi à esquerda do filósofo e h2 é o hashi à direita.
- Para evitar impasses (deadlocks) onde todos os filósofos pegam o hashi esquerdo primeiro e, em seguida, esperam indefinidamente pelo hashi direito, é realizada uma inversão aleatória.
- int h1 = i; e int h2 = (i + 1) % NUMFILO; declaram e inicializam as variáveis h1 e h2, que representam os hashis que o filósofo i vai tentar pegar. h1 é o hashi à sua esquerda, e h2 é o hashi à sua direita. NUMFILO é uma constante que define o número total de filósofos na mesa.
- A inversão é feita através de uma condição onde é verificado se um número aleatório gerado pelo rand() % 2 é igual a 0 ou 1.
- Se o número aleatório for igual a 0, os hashis são mantidos na ordem original (o hashi à esquerda é h1 e o hashi à direita é h2).
- Se o número aleatório for igual a 1, os hashis são trocados, ou seja, o hashi à direita se torna h1 e o hashi à esquerda se torna h2.
- Essa inversão aleatória é fundamental para garantir que todos os filósofos tenham a mesma chance de pegar ambos os hashis, reduzindo assim a probabilidade de impasses e melhorando a eficiência do algoritmo.

Solução do saleiro:

A solução proposta para o problema do jantar dos filósofos é baseada em um conjunto de passos cuidadosamente planejados para evitar impasses e garantir que cada filósofo possa acessar os hashis de forma segura. O uso de um semáforo denominado "saleiro" desempenha um papel fundamental nesse processo, controlando o acesso aos hashis e permitindo que apenas um filósofo por vez os utilize. Quando um filósofo está usando os hashis, os outros devem esperar até que ele os libere, evitando assim impasses.

Aqui está o trecho de código que representa essa parte da implementação:

```c
void *threadFilosofo(void *arg)
{
    int i = (long int)arg;
    while (1)
    {
        medita(i);

        sem_wait(&saleiro);
        pega_hashi(i, i);
        pega_hashi (i, (i+1) % NUMFILO);
        sem_post(&saleiro);

        come(i);
        larga_hashi(i, i);
        larga_hashi (i, (i+1) % NUMFILO);


    }
    pthread_exit(NULL);
}
```

- sem_wait(&saleiro): Este semáforo (saleiro) é uma parte crucial da solução. Ele controla o acesso aos hashis, permitindo que apenas um filósofo por vez os pegue. Se um filósofo está usando os hashis, os outros filósofos devem esperar até que ele termine e os libere. Isso previne o impasse.
- pega_hashi(i, i): O filósofo pega o hashi à sua esquerda.
- pega_hashi(i, (i+1) % NUMFILO): O filósofo pega o hashi à sua direita. O % NUMFILO é usado para garantir que, quando i for igual a NUMFILO - 1, o próximo filósofo seja o filósofo 0, fechando assim o círculo.
- sem_post(&saleiro): Após pegar os dois hashis, o filósofo libera o semáforo saleiro, indicando que os hashis estão ocupados.

Tentativa:

A solução proposta para o problema do jantar dos filósofos utiliza semáforos para controlar o acesso aos hashis. Cada filósofo tenta pegar um hashi por vez e, se não conseguir pegar o segundo hashi imediatamente, ele devolve o primeiro para evitar impasses. Isso garante que cada filósofo possa eventualmente comer de forma segura, evitando bloqueios mútuos.

Aqui está o trecho de código que representa essa parte da implementação:

```c
void pega_hashi(int f, int h)
{
    printf("%sF%d quer  h%d\n", space[f], f, h);
    sem_wait(&hashi[h]);
    printf("%sF%d pegou h%d\n", space[f], f, h);

    // Tenta pegar o segundo hashi
    int segundo_hashi = (h + 1) % NUMFILO;
    if (sem_trywait(&hashi[segundo_hashi]) == 0)
    {
        printf("%sF%d pegou h%d\n", space[f], f, segundo_hashi);
        come(f);
        comeu[f]++;
        larga_hashi(f, h);

    }
    else
    {
        // Não conseguiu pegar o segundo hashi, então larga o primeiro
        printf("%sF%d não conseguiu pegar h%d, devolvendo h%d\n", space[f], f, segundo_hashi, h);
        sem_post(&hashi[h]);
    }
}
```

- int segundo_hashi = (h + 1) % NUMFILO: Aqui, é calculado o índice do segundo hashi que o filósofo tentará pegar. Isso é feito adicionando 1 ao índice do hashi atual (h) e aplicando a operação de módulo NUMFILO para garantir que o índice fique dentro do intervalo válido de hashis (de 0 a NUMFILO - 1).
- if (sem_trywait(&hashi[segundo_hashi]) == 0): Esta linha tenta pegar o segundo hashi (segundo_hashi). sem_trywait() retorna imediatamente, indicando sucesso (retorna 0) se o semáforo estiver disponível e falha (retorna -1) se o semáforo estiver bloqueado.
- Else: Se o filósofo não conseguir pegar o segundo hashi imediatamente, este bloco else será executado.
- sem_post(&hashi[h]): Aqui, o primeiro hashi (h) é devolvido, ou seja, o semáforo associado a esse hashi é incrementado para indicar que ele está disponível novamente para uso.

2. Análise Comparativa:

Solução de inversão aleatória:

Vantagens:

- **Mitiga impasses:** A introdução de aleatoriedade na ordem em que os filósofos tentam pegar os hashis ajuda a mitigar a condição de espera circular, reduzindo significativamente a probabilidade de impasses ocorrerem.
- **Simplicidade de implementação:** Apesar da introdução de aleatoriedade, a solução ainda é relativamente simples de entender e implementar.

Desvantagens:

- **Não elimina completamente impasses:** Embora reduza a probabilidade de impasses, a solução de inversão aleatória não os elimina completamente. Ainda existe uma pequena chance de impasses ocorrerem, especialmente em cenários específicos.
- **Potencial para Starvation:** Dependendo da aleatoriedade introduzida na ordem de pegar os hashis, alguns filósofos podem ter prioridade sobre outros, o que pode resultar em starvation (inanição) para alguns filósofos se eles raramente conseguirem pegar os hashis.

Solução do saleiro:

Vantagens:

- **Robustez contra Deadlocks:** A principal vantagem desta solução é que ela evita efetivamente impasses (deadlocks). Ao garantir que um filósofo só possa pegar os garfos se ambos estiverem disponíveis, impede-se a possibilidade de um impasse onde todos os filósofos pegam um garfo e esperam pelo outro.
- **Justiça na Alocação de Recursos:** Com a solução do Saleiro, todos os filósofos têm uma chance justa de comer, pois não podem pegar os garfos a menos que ambos estejam disponíveis.

Desvantagens:

- **Complexidade de Implementação:** Comparado com a solução original, a implementação do Saleiro é mais complexa devido à necessidade de coordenação entre os filósofos para acessar o saleiro antes de tentar pegar os garfos.

- **Potencial para Starvation:** Se um filósofo estiver constantemente sendo interrompido por outros filósofos que conseguem acessar o saleiro primeiro, ele pode acabar ficando faminto (starvation). Isso pode acontecer em situações de alta concorrência por recursos.

Tentativa:

Vantagens:

- **Prevenção de Deadlocks:** A solução proposta evita impasses (deadlocks) ao permitir que os filósofos liberem os hashis se não conseguirem pegar o segundo imediatamente. Isso garante que nenhum filósofo fique permanentemente bloqueado e que todos possam eventualmente comer.
- **Eficiência:** A solução é eficiente em termos de utilização de recursos, pois cada filósofo só bloqueia um hashi por vez e, se necessário, devolve-o imediatamente. Isso minimiza o tempo de espera e o uso desnecessário de recursos do sistema.

Desvantagens:

- **Potencial para Starvation:** Embora a solução evite impasses, pode haver casos em que um filósofo nunca consiga pegar ambos os hashis devido à constante competição com outros filósofos. Isso pode levar à starvation, onde um filósofo fica indefinidamente sem conseguir comer.
- **Uso Ineficiente de Recursos em Casos de Starvation:** Em situações onde um ou mais filósofos enfrentam starvation, pode haver um uso ineficiente de recursos, já que os filósofos continuam tentando pegar os hashis sem sucesso, consumindo recursos do sistema sem progresso significativo.

3. Avaliação de Desempenho:

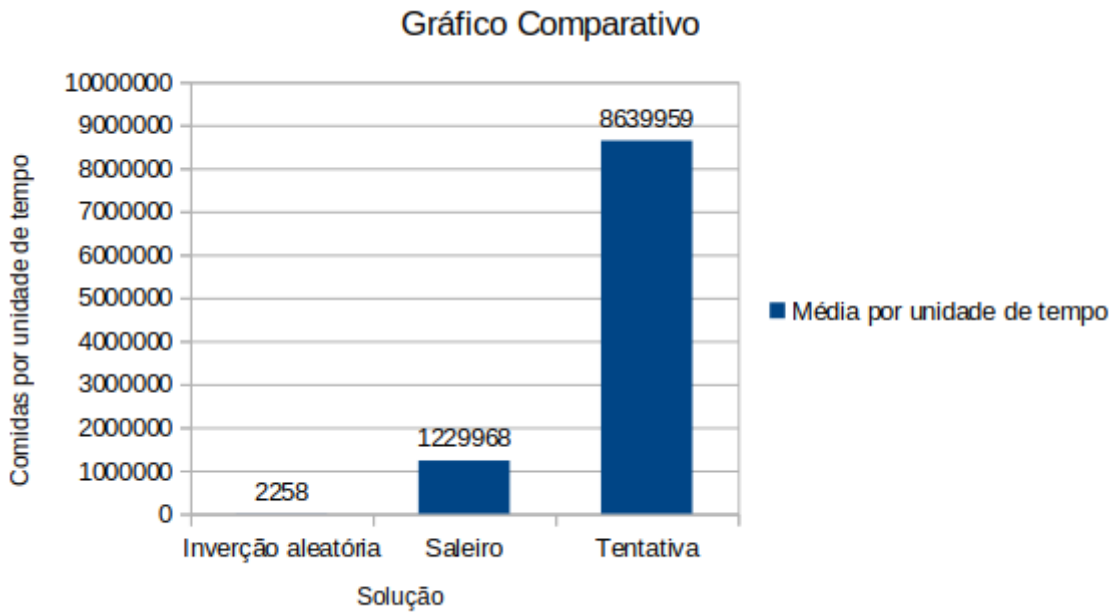
Em todas as três soluções, foram implementadas as seguintes metodologias:

- **Throughput:** é calculado ao final do período de medição (TEMPO_DE_MEDICAO), onde o programa imprime quantas vezes cada filósofo conseguiu comer durante esse período e calcula o throughput total. O throughput total é calculado somando o número de vezes que cada filósofo comeu e dividindo pelo tempo de medição.
  A função imprime_throughput() é responsável por essa tarefa. Ela itera sobre a matriz comeu[], que registra quantas vezes cada filósofo comeu, e imprime esses valores. Em seguida, calcula e imprime o throughput total.

- **Desabilitação das Pausas:** Para desabilitar as pausas durante as funções medita() e come(), foram comentadas as chamadas da função espera(). Originalmente, essa função gera um tempo de espera aleatório entre 0 e n segundos, onde n é o parâmetro passado para a função. Comentando as chamadas da função, o tempo de espera é efetivamente zero.

Não foi aplicado o conceito de equidade, uma vez que nas soluções do problema do saleiro e da tentativa, os filósofos estavam se alimentando de forma relativamente uniforme. Apenas na solução que envolvia inversão aleatória houve um pequeno problema nesse sentido. No entanto, as técnicas de equidade que experimentei acabaram por alterar significativamente os resultados em comparação com quando não eram aplicadas. Por isso, optei por não utilizar nenhum método de equidade.

4. Criação de Gráfico Comparativo:



5. Análise Crítica:

Solução de Inversão Aleatória:

- Média de comidas por unidade de tempo: 2258
- Essa solução aborda o impasse ao introduzir uma aleatoriedade controlada na seleção de hashis pelos filósofos. Isso impede que todos os filósofos tentem pegar o hashi à esquerda simultaneamente, evitando bloqueios mútuos.
- No entanto, a baixa média de comidas por unidade de tempo indica que essa solução pode não ser eficiente em permitir que os filósofos comam rapidamente. Embora resolva o problema de impasses, pode não ser a solução mais eficiente em termos de desempenho.

Solução do Saleiro:

- Média de comidas por unidade de tempo: 1229968
- Essa solução utiliza semáforos, como o "saleiro", para controlar o acesso aos hashis. Apenas um filósofo por vez pode usar os hashis, evitando impasses.
- A alta média de comidas por unidade de tempo indica que essa solução é altamente eficiente em permitir que os filósofos comam rapidamente, enquanto também resolve o problema de impasses.

Solução da Tentativa:

- Média de comidas por unidade de tempo: 8639959
- Nesta solução, os filósofos tentam pegar um hashi por vez e, se não conseguirem pegar o segundo imediatamente, devolvem o primeiro para evitar impasses.
- A média de comidas por unidade de tempo mais alta entre as três soluções indica que essa abordagem é a mais eficiente em permitir que os filósofos comam rapidamente, enquanto também resolve o problema de impasses.

Em termos de eficiência em permitir que os filósofos comam rapidamente e evitar impasses, a Solução da Tentativa parece ser a mais eficaz, seguida de perto pela Solução do Saleiro. A Solução de Inversão Aleatória, embora resolva o problema de impasses, parece ser menos eficiente em permitir que os filósofos comam rapidamente, devido à sua baixa média de comidas por unidade de tempo.

Em resumo, todas as soluções implementadas resolveram o problema de impasses ao removerem as condições necessárias para que eles ocorram. Cada solução adota uma abordagem diferente, seja através da introdução de aleatoriedade controlada que evita a situação em que todos tentam pegar o hashi à esquerda simultaneamente, do uso de semáforos para controlar o acesso aos recursos que remove a condição de competição por recursos simultânea, ou da estratégia de tentativa e devolução para evitar a espera circular.