Mestrado em Ciência e Tecnologia da Computação

Disciplina: PCO003 – Sistemas Operacionais
Wladmir Anibal Ribeiro de Bragança

Problema do Jantar dos Filósofos: Cinco filósofos desejam comer espaguete; No entanto, para poder comer, cada filósofo precisa utilizar dois garfos e não apenas um. Portanto, os filósofos precisam compartilhar o uso do garfo de forma sincronizada. Os filósofos comem e pensam;

Problemas que devem ser evitados:

Deadlock – todos os filósofos pegam um garfo ao mesmo tempo;

Starvation – os filósofos fiquem indefinidamente pegando garfos simultaneamente;

Explicação

HTML

- **Estrutura básica para exibir os filósofos e um botão "Iniciar Jantar".**
- **Cada filósofo é representado por um círculo colorido com texto dentro.**

CSS

- **Estiliza os filósofos como círculos. O estado dos filósofos muda a cor:**
  - **Pensando: Verde (`thinking`).**
  - **Comendo: Vermelho (`eating`).**
  - **Esperando: Cinza (`waiting`).**

JavaScript

Lógica:

- **O garçom controla quantos filósofos podem tentar pegar os garfos ao mesmo tempo.**
- **Apenas um número limitado de filósofos pode comer simultaneamente, evitando a situação em que todos estão esperando pelos garfos.**
- **Variável `MAX_FILOSOFOS_COMENDO`:**
  - **O garçom permite até n-1 filósofos comerem ao mesmo tempo, onde `n` é o número total de filósofos. Isso garante que nunca haverá uma situação em que todos os filósofos pegam um garfo ao mesmo tempo, evitando deadlock.**
- **Variável `filosofosComendo`:**
  - **Mantém o controle de quantos filósofos estão comendo simultaneamente. Se o número de filósofos comendo atingir o limite (`MAX_FILOSOFOS_COMENDO`), os outros filósofos precisam esperar.**

- **Garçom (`garcomDisponivel`):**
  - **Ela controla quando o garçom está disponível para permitir que os filósofos comam.**
- **Espera Assíncrona:**
  - **Caso um filósofo não consiga comer devido ao limite de filósofos comendo, ele espera de forma assíncrona (`esperarAleatorio`) antes de tentar novamente.**
- **Estado `esperando`:**
  - **Enquanto o filósofo aguarda a permissão do garçom para pegar os garfos, ele entra no estado `esperando`, o que previne que ele bloqueie outros filósofos.**

Como Funciona

- **Evita Deadlock: O deadlock é evitado porque o garçom garante que não mais que n-1 filósofos possam pegar os garfos ao mesmo tempo.**
- **Minimiza Starvation: A starvation é minimizada porque, eventualmente, cada filósofo terá a chance de pegar os garfos assim que outros terminarem de comer.**

Desenvolvido em HTML + CSS + JavaScript

filosofos.html

```
<!DOCTYPE html>

<html lang="pt-BR">

<head>

   <meta charset="UTF-8">

   <meta name="viewport" content="width=device-width,
initial-scale=1.0">

   <title>Problema dos Filósofos Jantando com Garçom</title>

   <link rel="stylesheet" href="style.css">

</head>
```

```
<body>

   <h1>Problema dos Filósofos Jantando com Garçom</h1>



   <!-- Mesa com os filósofos -->

   <div class="mesa">

     <div>

       <div id="filosofo-1" class="filosofo thinking">Filósofo
1</div>

       <div id="contador-1" class="contador">Comeu: 0
vezes</div>

     </div>

     <div>

       <div id="filosofo-2" class="filosofo thinking">Filósofo
2</div>

       <div id="contador-2" class="contador">Comeu: 0
vezes</div>

     </div>

     <div>

       <div id="filosofo-3" class="filosofo thinking">Filósofo
3</div>

       <div id="contador-3" class="contador">Comeu: 0
vezes</div>

     </div>

     <div>
```

```
        <div id="filosofo-5" class="filosofo thinking">Filósofo
5</div>

        <div id="contador-5" class="contador">Comeu: 0
vezes</div>

      </div>

      <div class="table-center"></div>

      <!-- Centro da mesa -->

      <div>

        <div id="filosofo-4" class="filosofo thinking">Filósofo
4</div>

        <div id="contador-4" class="contador">Comeu: 0
vezes</div>

      </div>

    </div>



    <button id="start-btn">Iniciar Jantar</button>



    <script>

        const NUM_FILOSOFOS = 5;

        const ITERACOES = 5;

        const MAX_FILOSOFOS_COMENDO = NUM_FILOSOFOS - 1; // O
garçom permite até n-1 filósofos comerem ao mesmo tempo

        let comerContador = Array(NUM_FILOSOFOS).fill(0);
```

```
        let filosofosComendo = 0;  // Controla quantos
filósofos estão comendo ao mesmo tempo

        let garcomDisponivel = true;



        // Função para alterar o estado do filósofo e atualizar
o contador de refeições

        function mudarEstado(id, estado) {

            const philosopher =
document.getElementById('filosofo-' + id);

            const contador =
document.getElementById('contador-' + id);



            if (estado === 'comendo') {

                philosopher.classList.add('eating');

                philosopher.innerHTML = `Filósofo ${id}
Comendo...`;

                philosopher.classList.remove('thinking',
'waiting');

                comerContador[id - 1]++;

                contador.innerHTML = `Comeu: ${comerContador[id
- 1]} vezes`;

            } else if (estado === 'pensando') {

                philosopher.classList.add('thinking');

                philosopher.innerHTML = `Filósofo ${id}
Pensando...`;
```

```
                philosopher.classList.remove('eating',
'waiting');

            } else if (estado === 'esperando') {

                philosopher.classList.add('waiting');

                philosopher.innerHTML = `Filósofo ${id}
Esperando...`;

                philosopher.classList.remove('eating',
'thinking');

            }



            // Se o filósofo comeu 5 vezes, ele termina

            if (comerContador[id - 1] === ITERACOES) {

                philosopher.innerHTML = `Filósofo ${id}
(Satisfeito)`;

                philosopher.classList.add('finish');  // Fica
vermelho quando satisfeito

                status.innerHTML = 'Terminado!';

            }

        }



        // Função que simula o comportamento do filósofo

        async function filosofo(id) {

            for (let i = 0; i < ITERACOES; i++) {
```

```
                mudarEstado(id, 'pensando');
                await esperarAleatorio(1000, 3000); // Simula o
tempo de pensamento

                mudarEstado(id, 'esperando');
                await esperarAleatorio(1000, 2000); // Simula o
tempo de espera

                // Pedir permissão ao garçom para comer
                while (filosofosComendo >=
MAX_FILOSOFOS_COMENDO || !garcomDisponivel) {
                    await esperarAleatorio(500, 1000); //
Espera um pouco e tenta de novo
                }

                // Permissão concedida
                filosofosComendo++;
                mudarEstado(id, 'comendo');
                await esperarAleatorio(2000, 5000); // Simula o
tempo de comer

                // Filósofo termina de comer
                filosofosComendo--;
```

```
            }


            mudarEstado(id, 'terminado');
        }


        // Função auxiliar para simular espera assíncrona
        function esperarAleatorio(min, max) {
            return new Promise(resolve => setTimeout(resolve,
Math.random() * (max - min) + min));
        }


        // Função para iniciar o jantar
        function iniciarJantar() {
            comerContador = Array(NUM_FILOSOFOS).fill(0); //
Reinicia os contadores
            filosofosComendo = 0;  // Reinicia o contador de
filósofos comendo
            for (let i = 1; i <= NUM_FILOSOFOS; i++) {
                filosofo(i); // Inicia o ciclo para cada
filósofo
            }
        }
```

```
       // Evento do botão de início

document.getElementById('start-btn').addEventListener('click',
iniciarJantar);

   </script>

</body>

</html>
```

style.css

```
body {

 font-family: Arial, sans-serif;

 text-align: center;

 background-color: #f4f4f4;

 margin: 0;

 padding: 20px;

}

.mesa {

 display: grid;

 grid-template-columns: 1fr 1fr 1fr;

 grid-template-rows: 1fr 1fr 1fr;
```

```
 gap: 100px;

 justify-items: center;

 align-items: center;

 margin: 0 auto;

 max-width: 600px;

 max-height: 400px;

}

.filosofo {

 display: inline-block;

 padding: 20px;

 border-radius: 50%;

 width: 100px;

 height: 100px;

 line-height: 100px;

 text-align: center;

 font-size: 18px;

 background-color: #add8e6;

 transition: background-color 0.5s ease;

}

.eating {

 background-color: #075f5f;

}
```

```css
.thinking {
 background-color: #66ff66;
}
.waiting {
 background-color: #ac8e08;
}
.finish {
 background-color: #ff6666;
}
.status {
 font-size: 16px;
 margin-top: 10px;
}
.contador {
 font-size: 16px;
}
#start-btn {
 margin-top: 20px;
 padding: 10px 20px;
 font-size: 18px;
 cursor: pointer;
}
```