



Ncozinheiros.c

C 2.92KB

oi deep! esse código resolve o problema dos dining savages para N cozinheiros. Contudo, está dando deadlock quando ainda tem porções na panela mas todos os alunos já comeram. Um dos cozinheiros não finaliza. Como resolver?



Análise do Deadlock no Problema dos Dining Savages

O problema que você está enfrentando ocorre quando todos os alunos já terminaram de comer (`acabaram = 1`), mas ainda há porções na panela, deixando um cozinheiro bloqueado esperando pelo sinal `panela_vazia` que nunca virá.

Problemas identificados:

1. `porcoes == 0` ficam bloqueados.
2. A variável `acabaram` é verificada apenas depois que o cozinheiro já pegou o semáforo `panela_vazia`.

Solução proposta:

1. Modificar a função do cozinheiro para verificar `acabaram` antes de esperar pelo `panela_vazia`.
2. Enviar um sinal `panela_vazia` para cada cozinheiro quando todos os alunos terminarem.

Aqui está a correção:

```

void* f_cozinheiro(void *v) {
    int id = *(int*) v;
    while (1) {
        if (acabaram) {
            printf("Cozinheiro %d: finalizando, pois os alunos já comeram.\n", id);
            break;
        }
        sem_wait(&panela_vazia);
        if (acabaram) {
            printf("Cozinheiro %d: finalizando, pois os alunos já comeram.\n", id);
            break;
        }
        printf("Cozinheiro %d: a panela está vazia, preparando recarga...\n", id);
        putServingsInPot(id);
    }
    return NULL;
}

// Na função main, modificar a finalização para:
acabaram = 1;
// Envia sinal para todos os cozinheiros
for(int i = 0; i < N_COZINHEIROS; i++) {
    sem_post(&panela_vazia);
}

```

Explicação da solução:

1. Agora o cozinheiro verifica `acabaram` antes e depois de esperar pelo `panela_vazia`.
2. Quando todos os alunos terminam, o main envia um sinal `panela_vazia` para cada cozinheiro, garantindo que todos serão desbloqueados e poderão verificar a flag `acabaram`.
3. Isso evita que os cozinheiros fiquem permanentemente bloqueados esperando por um sinal que nunca viria.

Esta solução garante que todos os cozinheiros serão devidamente finalizados quando todos os alunos tiverem terminado de comer, independentemente do estado atual da panela.

mas aí o mesmo cozinheiro não vai imprimir 2 vezes?



Você está certo! Se verificarmos `acabaram` tanto antes quanto depois do `sem_wait(&panela_vazia)`, o cozinheiro pode imprimir a mensagem de finalização duas vezes se `acabaram` for definido enquanto ele está esperando no `sem_wait`.

Solução refinada:

Vamos ajustar para que cada cozinheiro imprima a mensagem de finalização apenas uma vez, usando uma abordagem mais limpa:

```
c                                                                    Copy

void* f_cozinheiro(void *v) {
    int id = *(int*) v;
    while (1) {
        // Tenta pegar panela_vazia com timeout (não bloqueante)
        if (sem_trywait(&panela_vazia) == 0) {
            if (acabaram) {
                printf("Cozinheiro %d: finalizando, pois os alunos já comeram.\n", id);
                break;
            }
            printf("Cozinheiro %d: a panela está vazia, preparando recarga...\n", id);
            putServingsInPot(id);
        } else {
            // Se não conseguiu pegar panela_vazia, verifica se acabou
            if (acabaram) {
                printf("Cozinheiro %d: finalizando, pois os alunos já comeram.\n", id);
                break;
            }
            // Espera um pouco antes de tentar novamente
            usleep(100000); // 100ms
        }
    }
    return NULL;
}

// Na função main, manter:
acabaram = 1;
// Não precisa mais postar panela_vazia, pois os cozinheiros verificam 'acabaram' periodicamente
```

Por que isso resolve:

1. **Sem dupla impressão:** Agora cada cozinheiro verifica `acabaram` em um loop, sem ficar bloqueado indefinidamente no `sem_wait`.
2. **Mais robusto:** Usando `sem_trywait()` + `usleep()`, evitamos bloqueios

permanentes e ainda mantemos a responsividade.

3. **Mais simples de finalizar:** Não precisamos mais enviar sinais artificiais de `panela_vazia` para desbloquear os cozinheiros.

Alternativa (se não quiser usar `sem_trywait`):

Se preferir manter o `sem_wait`, podemos usar uma variável adicional para controle:

```
c                                                                    Copy

void* f_cozinheiro(void *v) {
    int id = *(int*) v;
    while (1) {
        if (acabaram) break;

        sem_wait(&panela_vazia);

        if (acabaram) {
            sem_post(&panela_vazia); // Libera para outro cozinheiro
            break;
        }

        printf("Cozinheiro %d: a panela está vazia, preparando recarga...\n", id);
        putServingsInPot(id);
    }
    printf("Cozinheiro %d: finalizando, pois os alunos já comeram.\n", id);
    return NULL;
}

// Main:
acabaram = 1;
sem_post(&panela_vazia); // Apenas um sinal é suficiente
```

Esta versão garante:

1. Mensagem única de finalização
2. Liberação correta dos recursos
3. Sem deadlocks

Escolha a abordagem que melhor se adapta ao seu contexto! A primeira é mais "ativa" (polling), a segunda mais "passiva" (bloqueante).

só dar sem post p/ todos os cozinheiros não resolve? eu só estava dando uma vez no código original



Você está absolutamente certo! Dar `sem_post` para todos os cozinheiros **resolve sim** o problema original de forma mais simples e elegante, sem precisar das mudanças complexas que sugeri anteriormente. O erro no código original era justamente enviar apenas **um** sinal (`sem_post(&panela_vazia)`), quando na verdade você tem **N_COZINHEIROS** que podem estar bloqueados.

Solução definitiva (e mais simples):

Mude **apenas** a parte do `main()` onde você finaliza os cozinheiros:

```
c                                                                    Copy
// ... (código anterior igual)

acabaram = 1; // Avisa que os alunos terminaram

// Envia um sinal para CADA cozinheiro (N_COZINHEIROS vezes)
for (int i = 0; i < N_COZINHEIROS; i++) {
    sem_post(&panela_vazia); // Cada post libera um cozinheiro bloqueado
}

// ... (restante do código igual)
```

Por que isso funciona:

1. **Cada** `sem_post(&panela_vazia)` **libera um cozinheiro** que estava esperando em `sem_wait(&panela_vazia)`.
2. **Se houver mais cozinheiros do que sinais**, alguns ficarão bloqueados (por isso você precisa de **N_COZINHEIROS** `sem_post`).
3. **Quando liberados, os cozinheiros verificam** `acabaram` e saem do loop.

Vantagens:

- Mantém a lógica original do código quase intacta.
- Não introduz verificações extras ou `sem_trywait`.
- Garante que **todos** os cozinheiros serão liberados, mesmo se a panela não estiver vazia.
- Não imprime mensagens duplicadas.

Exemplo de saída correta:

Copy

```
Aluno 6: terminei de comer 10 vezes.  
Cozinheiro 1: finalizando, pois os alunos já comeram.  
Cozinheiro 0: finalizando, pois os alunos já comeram.  
Muçamos! Todos os alunos comeram 10 vezes.
```

Observação:

Se você tivesse **apenas 1 cozinheiro**, o código original já funcionaria (pois bastaria 1 `sem_post`). O problema só aparece com **N cozinheiros**, porque cada um pode estar bloqueado no `sem_wait(&panela_vazia)`, e um único `sem_post` liberaria apenas **um** deles.