Produtor-Consumidor com Semáforo

Bianca Gallicchio Tavares — 2122102BCC

Visão geral do código

O programa implementa o problema do Produtor-Consumidor utilizando múltiplas threads em **Java**. É criado um buffer circular de tamanho fixo, onde as threads produtoras inserem itens aleatórios e threads consumidoras retiram esses itens. Os semáforos são utilizados para controlar o acesso ao buffer e sincronizar a produção e o consumo para evitarmos condições de corrida. O programa vai finalizar a sua execução depois de um número total definido de operações e o estado do buffer e dos semáforos é exibido no terminal a cada operação.

Assim, temos um buffer com tamanho fixo de 5 elementos (variável *TAM_BUFFER*), o qual será acessado pelos Produtores, representados por threads gerando itens aleatórios de 0 a 99, e pelos Consumidores, representados por threads consumindo os itens que estão no buffer. Também há os semáforos, representados por três variáveis: *espacos*, que controla o número de posições disponíveis para produção; *itens*, que controla o número de posições preenchidas e que podem ser consumidas; e *mutex*, usada para garantir exclusão mútua na seção crítica.

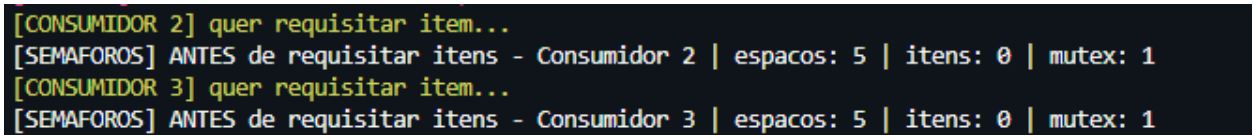
1 - O que acontece quando a thread termina a escrita?

```
[PRODUTOR 2] quer requisitar espaco para produzir: 46
[SEMAFOROS] ANTES de requisitar espacos - Produtor 2 | espacos: 5 | itens: 0 | mutex: 1
[SEMAFOROS] APOS adquirir espacos - Produtor 2 | espacos: 4 | itens: 0 | mutex: 1
[SEMAFOROS] DENTRO da regiao critica - Produtor 2 | espacos: 4 | itens: 0 | mutex: 0
[PRODUTOR 2] produziu: 46 na posicao 1 | Total produzidos: 2
[SEMAFOROS] APOS producao - Produtor 2 | espacos: 4 | itens: 1 | mutex: 1
[SEMAFOROS] APOS adquirir itens - Consumidor 2 | espacos: 4 | itens: 0 | mutex: 1
BUFFER: [__ 46 __ __ __]
```

Quando ela termina de escrever, a thread libera o mutex, saindo da seção crítica. Assim, o semáforo *itens* é incrementado, permitindo que consumidores que estavam bloqueados possam consumir. Podemos ver na captura de tela na linha “APÓS produção” que o Produtor 2 preencheu o buffer com o item 46 e o mutex foi liberado, retornando ao valor 1. Essa saída e a liberação do mutex é gerada na função *private static void produtor(int id)*.

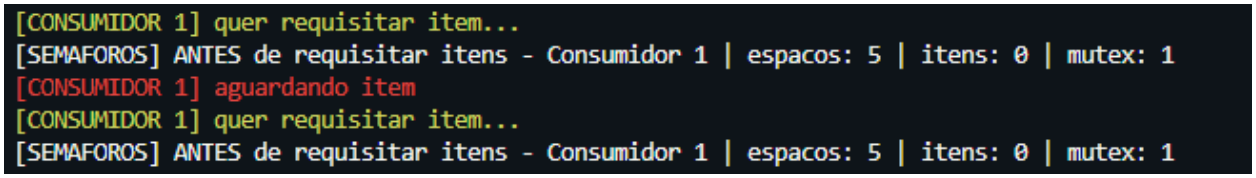
2 - O que acontece quando temos múltiplas threads lendo?

Aqui, somente uma thread pode entrar na seção crítica por vez. Então, múltiplos consumidores podem tentar ler, mas apenas um consome por vez, os outros ficam bloqueados no mutex.acquire().

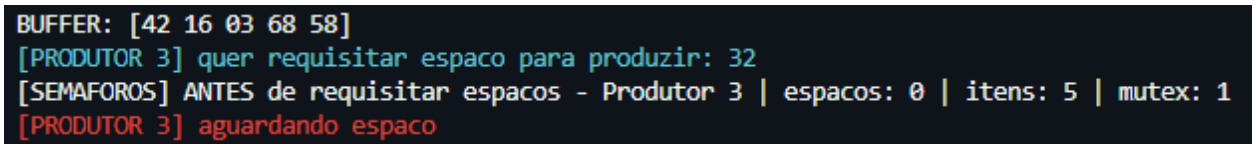


3 - Quantas threads de escrita e leitura podem estar bloqueadas no wait(w_or_r)?

As threads bloqueadas variam dinamicamente com o estado do buffer. Se *espacos* == 0, produtores que tentarem produzir ficam bloqueados. Consumidores, por sua vez, ficam bloqueados quando *itens* == 0. Então, qualquer número de threads produtoras ou consumidoras pode ficar bloqueado nesse wait.



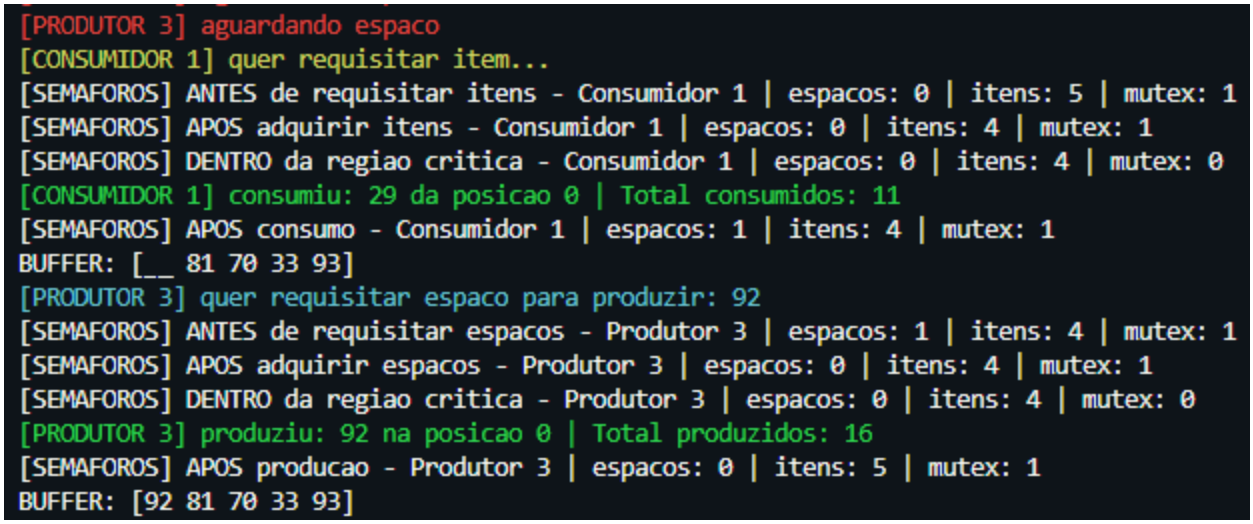
No print acima, há um consumidor lendo o semáforo *itens* para ver se algum item está disponível, mas, como a variável é 0, ele precisa aguardar um Produtor gerar um item.



No print acima, temos um produtor lendo o semáforo *espacos* para verificar se pode produzir algum item, mas, como a variável é 0, ele precisa aguardar um Consumidor retirar algum item do buffer.

4 - Quando que uma thread de escrita pode ser desbloqueada?

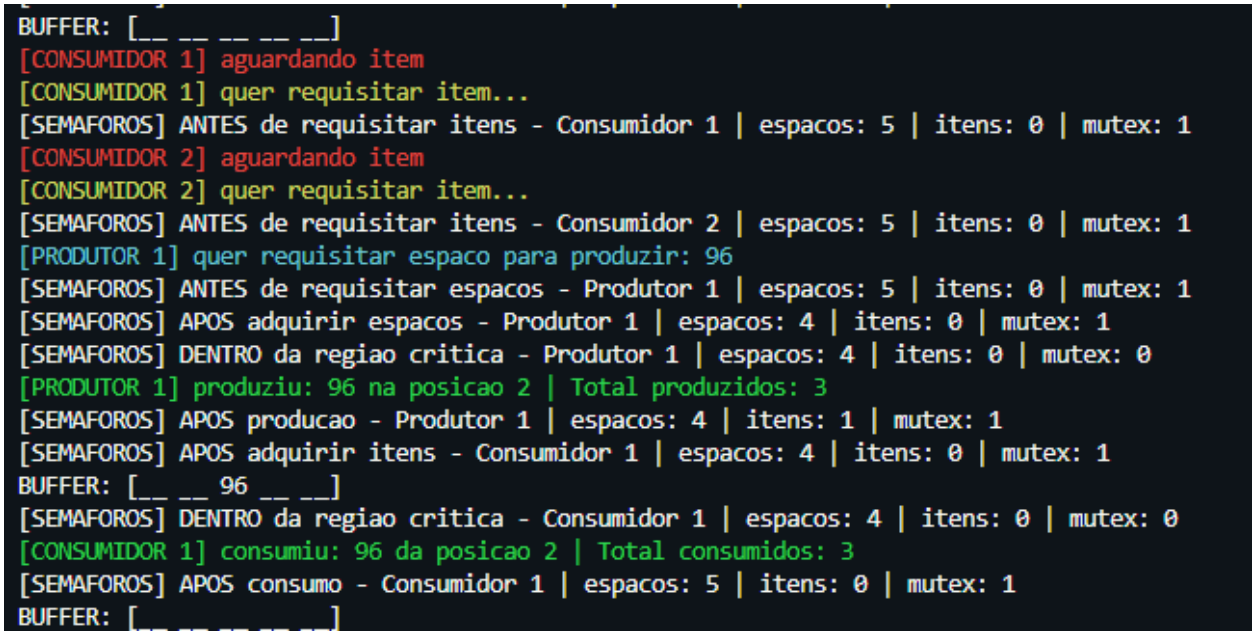
Ao consumir um item, o consumidor incrementa o semáforo vazio. Um produtor bloqueado aguardando espacos.tryAcquire() é então desbloqueado e pode escrever.



No print, o Produtor 3 estava bloqueado, mas surgiu um Consumidor que liberou espaço, então ele pode produzir um item.

5 - O signal(w_or_r) de leitura pode encontrar outra thread de leitura bloqueada no wait(w_or_r)?

No problema do produtor–consumidor, o signal de leitura (signal(itens)) nunca libera mais de um consumidor ao mesmo tempo. Cada thread consumidora só prossegue quando um item fica disponível e sempre entra sozinha na região crítica para removê-lo.



Acima, dois consumidores querem requisitar um item do buffer, mas ele está vazio. Logo surge um Produtor e, depois de agir, o Consumidor 1 é o selecionado para a região crítica onde pode retirar um item do buffer. O Consumidor 2 ainda precisou aguardar.