Desafio de Sistemas Operacionais

José Douglas Gondim Soares, 485347

29 de Setembro de 2022

1 Introdução ao problema

O desafio consiste em resolver um problema clássico de produtor/consumidor. Nós precisamos garantir que o produtor não vai continuar produzindo enquanto o buffer estiver cheio e que o consumidor não vai continuar consumindo quando o buffer estiver vazio. Como no desafio os métodos put() e take() da classe Dropbox operam sobre uma mesma seção crítica (a variável number), nós precisamos achar uma forma de sincronizar esses dois métodos para que o put() execute primeiro que o take() e que eles executem de forma obrigatoriamente alternada deste ponto em diante.

Além desse problema de sincronização, nós também precisamos garantir que o consumidor de números pares consuma apenas números pares e que o consumidor de números ímpares consuma apenas números ímpares.

2 Solução

2.1 Explicação

Para resolver o problema de sincronização, vamos utiliza a classe Semaphore da linguagem Java que é importada a partir de java.util.concurrent.Semaphore.

Nós vamos utilizar dois semáforos para garantir essa sincronização. Um semáforo para o produtor (producerSemaphore) e um semáforo para o consumidor (consumerSemaphore). Nós podemos fechar um semáforo utilizando o método acquire() e abri-lo utilizando o método release().

A ídeia é que antes do produtor produzir algo, o semáforo do produtor deve ser fechado e após a produção, o semáforo do consumidor deve ser aberto. E antes do consumidor consumir algo, o semáforo do consumidor deve ser fechado e após o produto ser consumido, o semáforo do produtor deve ser aberto.

É muito importante também que o semáforo do consumidor seja inicializado fechado, pois isso vai garantir que o produtor vai executar primeiro e já vai existir um produto a ser consumido antes do primeiro consumidor executar.

Isso já vai garantir a sincronização dos processos, mas ainda precisamos resolver o problema de consumidores que consomem apenas números pares e aqueles que consomem apenas números ímpares. Para isso, basta colocar uma condição antes de consumir qualquer produto que checa se aquele produto é do tipo que o consumidor consome ou não. Caso a condição seja falsa, precisamos reabrir o semáforo do consumidor e não consumir o produto.

2.2 Código

2.2.1 Inicialização dos semáforos do produtor e do consumidor

```
4     static Semaphore consumerSemaphore = new Semaphore(permits: 0);
5     static Semaphore producerSemaphore = new Semaphore(permits: 1);
```

Figura 1: O semáforo do consumidor é inicializado fechado e o do produtor aberto.

2.2.2 Método produtor put()

```
27    public void put(int number) {
28        try {
29            producerSemaphore.acquire();
30        }
31        catch (InterruptedException e) {
32            System.out.println(x: "InterruptedException");
33        }
34
35        this.number = number;
36        this.evenNumber = number % 2 == 0;
37        System.out.format(format: "PRODUTOR gera %d.%n", number);
38        consumerSemaphore.release();
39    }
40 }
```

Figura 2: O método put() fecha o semáforo do produtor (linha 29) antes de produzir algo e abre o do consumidor (linha 38) após o produto ter sido produzido. Após a produção ele atualiza as variáveis number e evenNumber com o produto e o tipo do produto, respectivamente.

2.2.3 Método consumidor take()

```
10    public int take(final boolean even) {
11        try {
12            consumerSemaphore.acquire();
13        }
14        catch (InterruptedException e) {
15            System.out.println(x: "InterruptedException");
16        }
17
18        if(even != evenNumber) {
19            consumerSemaphore.release();
20            return -1;
21        }
22        System.out.format(format: "%s CONSUMIDOR obtem %d.%n", even ? "PAR" : "IMPAR", number);
23        producerSemaphore.release();
24        return number;
25    }
```

Figura 3: O método take() fecha o semáforo do consumidor (linha 12) antes de consumir o produto e abre o do produtor (linha 19) após consumir. Ele também checa se o produto é do tipo que o consumidor consome (linha 18) e não consome o produto caso a condição não seja satisfeita.

3 Resultados

3.1 Antes:

Figura 4: Execução sem a utilização de semáforos para sincronização.

3.2 Depois:

Figura 5: Execução com a utilização de semáforos para sincronização.