Problema do Banheiro Unissex usando *threads* e concorrência

Alaide Lisandra Melo Carvalho

Instituto Metrópole Digital - IMD / lisandra.melo.095@ufrn.edu.br

José Victor Ferreira da Fonseca

Instituto Metrópole Digital - IMD/ jose.victor.ferreira.125@ufrn.edu.br

---

1. Introdução: Projeto da Solução

O projeto teve a proposta inicial de criar uma solução usando concorrência para o problema do Banheiro Unissex. O problema relata a situação de um escritório que "contém um banheiro que pode ser utilizado tanto por homens quanto por mulheres, mas não por ambos ao mesmo tempo". Nesse sentido, deveríamos criar uma lógica que permitisse o uso de uma área por somente pessoas de um sexo ao mesmo tempo. Além disso, deveríamos considerar a quantidade de boxes do banheiro quando esse deveria ser ocupado e quanto tempo cada pessoa passaria usando o banheiro

Nesse sentido, considerando o banheiro como buffer, e as pessoas como produtores e consumidores, foi decidido tomar como base a ideia do algoritmo de concorrente de produtor consumidor. Nesse sentido, o banheiro representaria o buffer compartilhado e seus boxes cada elemento do buffer, as pessoas - separadas por um atributo de gênero - então poderia consumir (sair do buffer) e produzir (entrar no buffer), contudo isso deveria levar em consideração as regras de uso propostas inicialmente.

Então, baseado nessa ideia de funcionamento, foi projetada uma solução que contaria com uma classe de buffer (Banheiro), uma classe para produção e consumo (Pessoa) e uma classe principal para a execução do programa (Main).

2. Lógica, Técnicas e Mecanismos

Para o presente trabalho utilizamos a lógica de sincronização dos monitores, visto que eles tem uma implementação mais simples e alguns benefícios que auxiliam a concorrência. Uma das principais vantagens é que apenas um processo pode estar ativo no monitor por vez, sendo assim, já temos a exclusão mutua garantida. Além de possuir as variáveis de condição, necessárias para verificar se existem boxes disponíveis e também verificar o gênero que está usando o banheiro naquele momento, que, caso seja oposto ao primeiro indivíduo da fila, não aciona o método de inserir pessoa.

No código, existe um buffer (banheiro) que tem um espaço limitado, uma região crítica, que pode ser acessado por threads concorrentes. Porém, os métodos que adicionam e removem pessoas

do banheiros estão em exclusão mutua e sincronizados, impedindo que ocorra inserção e remoção ao mesmo tempo.

3. Corretude da solução

Para a garantia da corretude da solução primeiro determinaremos quais especificações serão consideradas na análise de corretude. Primeiro analisaremos se o programa é capaz de executar de modo que as regras determinadas sejam respeitadas, primeiro tem-se que o programa é executado até que a fila de pessoas seja inteiramente "consumida", isto é, o algoritmo é pensado de modo a executar até a alocação de todas das pessoas na fila dentro dos banheiros, isso pode ser visto no trecho de código entre as linhas 66 e 80 do arquivo `Banheiro.java` apresentadas a seguir.

```
while(true){
    if(filaGeral.size() > 0){
        //Codigo
    }else{
        break;
    }
}
```

Além disso, devemos ver se além de consumir a fila o programa segue as regras definidas. Para isso, estudaremos o trecho de código localizado entre as linhas 68 e 76. Nesse trecho, temos a lógica da adição pessoas caso o banheiro esteja vazio inicialmente, caso contrário, verifica-se a existência de boxes vazios e o gênero atual das pessoas no banheiro, somente caso as regras sejam respeitadas então uma pessoa é adicionada. Portanto, as pessoas são completamente adicionadas e seguindo as regras de forma correta.

Agora, tomando a execução relativamente a seu tratamento de concorrência, sob posse dos mecanismos de sincronia mencionados na seção anterior temos a garantia de controle de acesso e de exclusão mútua necessária para a execução correta do programa. Além disso, pelo código de execução da thread pessoa temos que a thread deve executar no máximo por 5 segundos e depois ser finalizada, portanto, nesse sentido o programa deve ser finalizado em algum momento, já que as pessoas estão sendo sendo removidas da fila sempre ao seu final de execução.

4. Relatos da Experiência

Durante o experimento as dificuldades encontradas foram relativas a estrutura do código, pois a linguagem a ser utilizada e o método de sincronização a serem utilizados foram decididos de maneira rápida.

Para a estruturação, como usamos a ideia do produtor/consumidor – exemplo corriqueiro nos estudos de concorrência – houve uma certa dúvida se seria preciso criar a classe homem e a classe

mulher. Entretanto, como ambas são classes muito semelhantes, poderia ser criada uma classe única de pessoa, diferindo apenas no gênero, que pode ser uma variável.

Outra dificuldade foi na criação do monitor, visto que ficamos em dúvida quanto a escolha das estruturas de dados, mas que logo foram decididas para a que melhor se adequavam ao nosso propósito.

5. Instruções de Compilação e execução

Para compilar o programa utilize o comando abaixo.

```
javac -d bin src\Banheiro.java src\Main.java src\Pessoa.java
```

Para executá-lo, use a sequência de comandos abaixo.

```
cd bin
java Main
```

6. Conclusão

O presente projeto tinha como objetivo a prática dos ensinamentos passados em sala de aula. Como intenção principal, entendemos que era importante a criação de uma lógica para o problema proposto e também quais os mecanismos de sincronização usar para melhor satisfazer nossa implementação.

Por fim, foi desenvolvida uma solução que contemplava todos os pontos e condições prescritas no documento inicial, usando apenas os conteúdos dados em sala, com base em exemplos utilizados na disciplina.