Laboratório de Programação Concorrente

Lab11 - Using BlockingQueue - 25.1

Objetivo

Neste lab vocês irão experimentar o uso da interface BlockingQueue do pacote java.util.concurrent Package de Java. Nesse contexto, desenvolva este laboratório seguindo as etapas abaixo:

Preparação. Neste laboratório, nós disponibilizamos o código serial de um sistema Produtor-Consumidor que considera um buffer ilimitado e nenhum controle de acesso concorrente. Neste primeiro momento você deve garantir que este código executa como esperado. Para isso, siga o passo a passo:

1. Clone o repositório do código base

   git clone [link do repositório]

2. Execução da versão serial do código
   a. Navegue até o diretório da implementação serial:

      cd fpc/2025.1/lab11/src/serial

   b. Compile o código

      bash build.sh

   c. Execute a versão serial especificando os parâmetros:

      bash /run.sh 1 150 1 150

Dessa forma, você inicia a execução de 1 produtor e 1 consumidor que levam o mesmo tempo para produzir e consumir um item no buffer (150ms). O esperado é que a execução rode indefinidamente adicionando itens no buffer sem que o consumidor consiga consumir.

Etapa 1. No diretório **src/concurrent/etapa1**, desenvolva uma solução concorrente protegida cuja a fila compartilhada (buffer) entre o produtor e consumidor faça uso de uma BlockingQueue implementada no tipo ArrayBlockingQueue. Neste caso, tanto o produtor como o consumidor executam sem parar. Experimente diferentes tempos para produção e consumo dos números, considerando cenários com valor de produção maior que consumo e com valor de produção menor que o valor de consumo. No mesmo diretório, crie o arquivo comentarios-e1.txt e

comente o que acontece quando o tempo de produção é maior que o de consumo e vice versa.

Etapa 2. No diretório **src/concurrent/etapa2**, a partir da solução anterior, modifique o código de tal forma que o produtor deve gerar 10.000 números aleatórios e o consumidor deve ter um timeout de 600 milissegundos na espera por números na fila. Dessa forma, espera-se que o programa não execute sem parar. No mesmo diretório, crie o arquivo comentarios-e2.txt e comente o que acontece quando você executa com as configurações de tempo que você testou.

Apesar de não ser obrigatório, fiquem à vontade para usar ExecutorService para o gerenciamento das threads produtora e consumidora se assim desejarem.

Visão geral do código base

https://github.com/giovannifs/fpc/tree/master/2025.1/lab11

O código está organizado de acordo com a hierarquia abaixo:

- ***src/serial/:*** Diretório que disponibiliza a implementação serial do sistema produtor-consumidor.
- ***src/concurrent/:*** Diretório onde você implementará a versão concorrente do sistema produtor-consumidor. **Note que, inicialmente, este diretório contém uma cópia da implementação serial, então você deve alterá-la para implementar a concorrência!**
- **scripts auxiliares nos diretórios serial e concurrent:**
  - *build.sh:* script para compilar o código do diretório
  - *run.sh:* script para executar as implementações do diretório. Este exige os seguintes argumentos:
    - *<numero_produtores>*
    - *<tempo_producao> (em milisegundos)*
    - *<numero_consumidores>*
    - *<tempo_consumo> (em milisegundos)*
- ***submit-answer.sh:*** Script que será utilizado para a submissão da resposta do laboratório.

- ***support-doc:*** Disponibiliza documentação e tutorial sobre o uso de threads, semáforos e sincronizações em Java.

Entrega

Você deve criar e manter um repositório privado no GitHub com a sua solução. No entanto, a entrega do laboratório deverá ser realizada por meio de submissão online utilizando o script submit-answer.sh, disponibilizado na estrutura de arquivos do próprio laboratório. Uma vez que você tenha concluído sua resposta, seguem as instruções:

1) Crie um arquivo lab11_matr1_matr2.tar.gz com o seu código fonte. Para isso, supondo que o diretório raiz é Lab11/src, você deve executar:
   tar -cvzf lab11_matr1_matr2.tar.gz Lab11/src
2) Submeta o arquivo lab11_matr1_matr2.tar.gz usando o script submit-answer.sh, disponibilizado no mesmo repositório do laboratório:
   bash submit-answer.sh lab11 lab11_matr1_matr2.tar.gz

Prazo

23/set/25 16:00

BlockingQueue

A interface BlockingQueue suporta controle de fluxo (além da fila) introduzindo bloqueio se a estrutura estiver cheia ou vazia. Uma thread tentando enfileirar um elemento em uma fila cheia é bloqueada até que alguma outra thread libere espaço na fila. Da mesma forma, ela bloqueia uma thread tentando excluir de uma fila vazia até que alguma outra thread insira um item. BlockingQueue não aceita um valor nulo.

Unbounded queue – pode crescer quase indefinidamente
capacidade máxima é Integer.MAX_VALUE

```
BlockingQueue<String> blockingQueue = new LinkedBlockingDeque<>();
```

Bounded queue – com capacidade máxima definida

```
BlockingQueue<String> blockingQueue = new LinkedBlockingDeque<>(10);
```

Adicionando elementos

- add(e) – retorna verdadeiro se a inserção foi bem-sucedida, caso contrário, lança uma IllegalStateException
- put(e) – insere o elemento especificado em uma fila, aguardando por um slot livre, se necessário
- offer(e) – retorna verdadeiro se a inserção foi bem-sucedida, caso contrário, falso
- offer(e, timeout, TimeUnit) – tenta inserir o elemento em uma fila e aguarda por um slot disponível dentro de um tempo limite especificado

Removendo elementos

- take() – espera por um elemento head de uma fila e o remove. Se a fila estiver vazia, ele bloqueia e espera por um elemento para ficar disponível
- poll(timeout, TimeUnit) – recupera e remove o head da fila, esperando até o tempo de espera especificado, se necessário, para que um elemento fique disponível. Retorna null após um timeout