Laboratório de Programação Concorrente

Lab9 - Log Analyzer - 25.1

Objetivo

O objetivo deste laboratório é explorar a API ExecutorService para simplificar o gerenciamento de threads em Java. A atividade será desenvolvida de forma incremental, evoluindo de uma solução serial até o uso de Executors e Futures para controle e retorno de resultados.

Descrição

Você foi contratado para implementar um processador de logs de um sistema web. Cada arquivo de log contém várias linhas, onde cada linha representa uma requisição HTTP. Seu objetivo será contar quantas requisições bem-sucedidas (código 200) e quantas falhas (código 500) existem em cada arquivo, além de calcular os totais considerando todos os arquivos processados.

Etapas Incrementais

- **Etapa 1 – Versão Serial (Código Base):** A classe *LogAnalyzer.java* foi disponibilizada. Esta classe recebe como argumento uma lista de paths para arquivos de log. Cada arquivo é processado sequencialmente, contabilizando:
  - Número de respostas 200
  - Número de respostas 500

  Ao final, este código imprime o total de cada código de resposta.

  A Etapa 1 consiste em você executar o código base de acordo com as instruções presentes na próxima seção.

- **Etapa 2 – Versão Concorrente com Threads:** No diretório concurrent, crie a classe LogAnalyzer2.java, copiando da anterior. Evolua para que cada arquivo seja processado por uma thread independente (uma instância de Runnable).

  No final, agregue os resultados de todas as threads e imprima os totais. A saída deve ser no mesmo formato que a versão serial.

- **Etapa 3 – ExecutorService:** Ainda no diretório concurrent, você vai criar três classes LogAnalyzer3s.java, LogAnalyzer3c.java e LogAnalyzer3f.java. Nessas classes, você deve evoluir o código da etapa 2 para usar ExecutorService no gerenciamento das threads. As classes LogAnalyzer3s.java, LogAnalyzer3c.java e LogAnalyzer3f.java devem usar, respectivamente, os seugintes tipos de ExecutorService:
    - Executors.newSingleThreadExecutor()
    - Executors.newCachedThreadPool()
    - Executors.newFixedThreadPool(4)

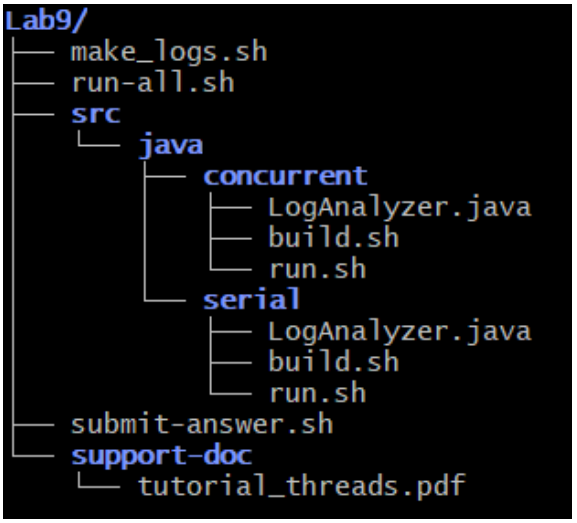
    Meça o tempo de execução em cada caso e compare.
    Questão para reflexão: O comportamento observado corresponde ao esperado para cada tipo de executor?

- **Etapa 4 – Callable e Future:** Crie a classe LogAnalyzer4.java. Reescreva a lógica para que cada tarefa de processamento de arquivo seja submetida como um Callable, retornando um objeto com os resultados (200 e 500). Use os objetos Future para recuperar os resultados das execuções e consolidá-los ao final.

Visão geral do código base

Aqui temos a descrição e o código fonte deste laboratório.

https://github.com/giovannifs/fpc/tree/master/2025.1/Lab9



- src/java/serial/:
  Diretório com a implementação serial, esse é o ponto de partida para entender o funcionamento do código.

- src/java/concurrent/:
  Diretório onde você implementará as versões concorrentes. **Note que, inicialmente, esse diretório contém a mesma implementação que a serial, então você deve alterá-la para implementar a concorrência!**
    - Seguem os nomes dos arquivos solicitados:
        - LogAnalyzer2.java
        - LogAnalyzer3s.java
        - LogAnalyzer3c.java
        - LogAnalyzer3f.java
        - LogAnalyzer4.java

- make_logs.sh:
  Script para gerar um conjunto de *n* arquivos que podem ser considerados para input das execuções. Os arquivos serão gerados em um diretório chamado dataset. Para executar esse script, informe, como argumento, o número de arquivos e a quantidade de linhas dos arquivos a serem gerado, por exemplo:
  bash make_logs.sh 10 10000

- run_all.sh:
  Script para compilar e executar as implementações serial e concorrente, em seus respectivos diretórios. Este script considera que os arquivos do diretório dataset serão passados como argumentos das execuções. **Você precisa editar este e o arquivo run.sh para executar os novos arquivos que você desenvolverá.**

- submit-answer.sh:
  Script que será utilizado para a submissão de sua resposta.

Execução da Etapa 1

1. Clone o repositório do código base

   ```
   git clone [link do repositório]
   ```

Faça uma comparação de desempenho entre as versões serial e concorrente (detalhes abaixo):

2. Execute o script make_dataset.sh para gerar arquivos para testar o código

   bash make_logs.sh 10 10000

3. Execute o script run_all.sh para executar todos os cenários considerando os arquivos dataset como argumentos das execuções

   bash run_all.sh

Prazo

09/set/25 16:00

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 lab9_matr1_matr2.tar.gz somente com o “src” do repositório que vocês trabalharam. Para isso, supondo que o diretório raiz de seu repositório privado chama-se lab9_pc, você deve executar:

   tar -cvzf lab9_matr1_matr2.tar.gz lab9_pc/src

2) Submeta o arquivo lab9_matr1_matr2.tar.gz usando o script submit-answer.sh, disponibilizado no mesmo repositório do laboratório:

   bash submit-answer.sh lab9 path/lab9_matr1_matr2.tar.gz

Lembre-se que você deve manter o seu repositório privado no GitHub para fins de comprovação em caso de problema no empacotamento ou transmissão online. Alterações no código realizadas após o prazo de entrega não serão analisadas.