Laboratório de Programação Concorrente

Lab6 Computing File Similarity - 25.1

Objetivo

O objetivo deste laboratório é melhorar o desempenho de um programa através de concorrência, mantendo o programa correto, e, para isso, protegendo algumas regiões críticas e coordenando a execução das threads. Neste laboratório, usaremos como cenário base, um problema já visto neste semestre, o **cálculo de sum de um arquivo**. No entanto, agora você deve desenvolver uma versão concorrente que explore o uso de múltiplas threads e atenda os requisitos abaixo:

1) implementar um controle de admissão de forma a limitar o consumo de memória processando no máximo N/2 arquivos concorrentemente (onde n é a quantidade de arquivos passados como parâmetro);

2)computar a similaridade entre os arquivos; e,

3) imprimir o valor da soma do sum de todos os arquivos.

Um arquivo pode apresentar algum grau de similaridade (um valor entre 0 e 1) com outro arquivo. Quanto maior a quantidade de pedaços (ou chunks) de um arquivo iguais a pedaços do outro arquivo (iguais conforme a função sum), mais similar os arquivos são.

Seu programa deve receber uma lista de arquivos e irá retornar para cada arquivo sua similaridade em relação aos demais. Entregaremos uma versão serial do programa funcional. Crie uma versão concorrente com base na versão serial entregue.

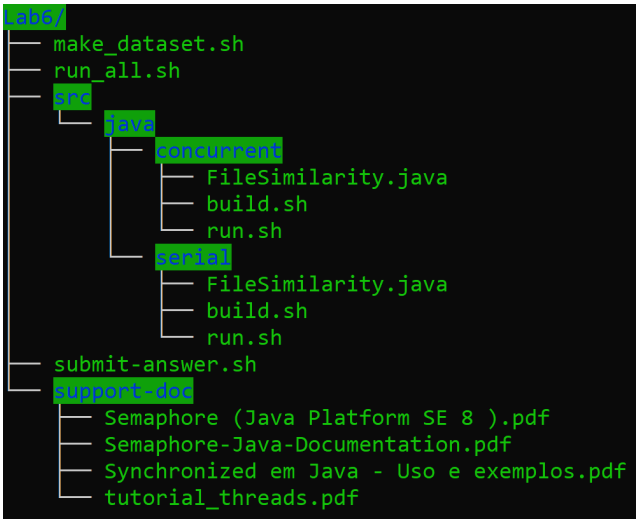
Visão geral do código base

No código base vocês encontrarão implementação serial da solução em Java. A implementação recebe como argumentos os “path” de arquivos que devem ter o sum calculado e as similaridades identificadas. A sua implementação deve adicionar concorrência a esse processo.

A entrega, detalhada nas seções seguintes, envolverá o código fonte. Iremos avaliar tanto as possibilidades de plágio entre os alunos quanto a geração automática de código.

https://github.com/giovannifs/fpc/tree/master/2025.1/Lab6

O código está organizado na seguinte hierarquia:



- 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 a versão concorrente. **Note que, inicialmente, esse diretório contém a mesma implementação que a serial, então você deve alterá-la para implementar a concorrência!**

- make_dataset.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 o número de arquivos a serem gerados como argumento, por exemplo:
  bash make_dataset.sh 5

- 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.

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

Preparação

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_dataset.sh 5
   ```

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

   ```
   bash run_all.sh
   ```

Ao final, na saída padrão deve-se indicar a similaridade entre os arquivos da seguinte forma:

```
Similarity between ../file.1 and ../file.2: 89.66221%
Similarity between ../file.1 and ../file.3: 89.7776%
Similarity between ../file.2 and ../file.3: 89.46289%
Total sum: 4010731557
```

Observação: É interessante que você siga o código fonte tentando entender quais partes do código fonte irão virar regiões críticas quando o programa for concorrente.

Solução em duas etapas

Etapa 1 - "Faça funcionar"

Você tem total liberdade para mudar o código base da maneira que achar melhor. Por exemplo, criando novas funções, alterando a assinatura das funções dadas bem como usando outras estruturas de dados. Apesar disso, recomendamos fortemente que você faça uma primeira versão que muda muito pouco o código base (e, que melhore o desempenho do programa). Você pode usar semáforos ou locks intrínsecos/monitores (synchronized de Java, discutido na última aula).

Etapa 2 - “Otimize sua solução”

Na etapa dois, você deve tentar melhorar ainda mais o desempenho do seu programa. Note que há potencial para usar concorrência tanto no cálculo do sum (em vários níveis) quanto na computação da similaridade. Ainda, há estruturas de dados que podem ser mais eficientes (a escolha das estruturas pode, inclusive, facilitar a escrita do código concorrente).

O comportamento da entrada e saída deve ser mantido para essa etapa.

Prazo

12/08/2025 às 16h00

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 lab6_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 lab6_pc, você deve executar:

   ```
   tar -cvzf lab6_matr1_matr2.tar.gz lab6_pc/src
   ```

2) Submeta o arquivo lab6_matr1_matr2.tar.gz usando o script submit-answer.sh, disponibilizado no mesmo repositório do laboratório:

   ```
   bash submit-answer.sh lab6 path/lab6_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.