Trabalho 2 INE5611

Alexis Armin Huf

5 de maio de 2017

1 Introdução

Após a fama e a glória obtida com seu trabalho na Pedrólio, sua equipe foi contratada pelo DAFVENR Departamento de Acessibilidade para Formas de Vida Extraterrestres Não-Reptilianas. Uma espécie de alienígenas conhecida como Cercebianos tem tido dificuldades ao navegar na internet terráquea. Os olhos dos Cercebianos vêm as cores de forma muito diferente dos humanos e por seus isso eles não conseguem apreciar a rica cultaura internética do planeta Terra.

Felizmente, há um algoritmo para converter as imagens em imagens agradáveis aos cercebianos. No entanto, a conversão exige certo processamento, e o DAFVENR não gostaria que um Cercebiano lendo um artigo científico tivesse de esperar a conversão de centenas de memes do facebook. Você deve desonvoler um esquema capaz de receber multiplas requisições de conversão e processá-las levando em conta fatores como sua prioridade e tamanho.

O DAFVENR forneceu um programa em Java já com alguns testes escritos. Sua equipe deve implementar a interface `ScheduledConverter` na classe `PriorityScheduledConverter`.

2 Detalhes

A interface `ScheduledConverter` a ser implementada tem os seguintes métodos:

```
1  public interface ScheduledConverter extends AutoCloseable {
2      ConverterTask convert(InputStream inputStream, OutputStream outputStream, String mediaType,
3                            long inputBytes, Priority priority);
4      void processFor(long interval, TimeUnit timeUnit) throws InterruptedException;
5      Collection<ConverterTask> getAllTasks();
6      int getQuantum(Priority priority);
7      void setQuantum(Priority priority, int milliseconds);
8  }
```

Figura 1 – A interface `ScheduledConverter`

`convert()` recebe todos os parametros necessários para conversão e cria um objeto representando a tarefa. Esse objeto deve permitir consultar o estado da tarefa (pronta ou não) e cancelar a tarefa. A conversão é realizada por uma implementação da classe `Converter`, que é fornecida na construção do `ScheduledConverter`. **Os casos de teste já fornecem uma instância de** `Converter`. O método `processFor()` bloqueia, por no máximo o tempo especificado, processando tarefas de conversão de acordo com as regras de priorização (Subseção 2.1).

Note que `ScheduledConverter` implementa `AutoCloseable`. Recursos adquiridos pelo objeto devem ser liberados no método close() e tarefas não processadas devem ser canceladas.

O escalonamento de tarefas deve ser feito em pequenos pedaços: Na presença de n tarefas no `ScheduledConverter`, cada uma deve executar por um período de tempo definido pelas funções `setQuantum()`

e `getQuantum()` (Figura 1). No entanto, note que há regras específicas que devem ser respeitadas (Subseção 2.1), e portanto, pode ser que uma mesma tarefa seja repetidamente executada.

Se, enquanto uma tarefa estiver sendo executada pelo `Converter`, uma nova tarefa de maior prioridade (de acordo com as regras da Subseção 2.1) for submetda ao `PriorityScheduledConverter`, a execução da tarefa deve cessar imediatamente e a nova tarefa deve inciar a execução. Para fins de escalonamento, considera-se que a tarefa que parou de ser processada executou por um ciclo, e a nova tarefa executará pelo tempo relativo a sua priopridade, específicado via `setQuantum`.

2.1 Regras de Prioridade

Há três níveis de prioridade no sistema:

```
1  public enum Priority {
2      LOW,
3      NORMAL,
4      HIGH;
5
6      public static List<Priority> decreasing() { /*...*/ }
```

Figura 2 – Enumeração `Priority`.

Além disso, cada tarefa deve ser relacionada a órdem de submissão. Por exemplo 1ª tarefa submetida, 2ª tarefa, etc.

As seguintes regras devem ser respeitadas:

1. Se há uma tarefa de prioridade x nenhuma tarefa de prioridade y, tal que $x > y$, deve ser executada antes da tarefa com prioridade x terminar. Por exemplo, se há uma tarefa com prioridade `HIGH` submetida, não deve ser executada nenhuma tarefa `LOW` ou `NORMAL` até que a tarefa `HIGH` seja completada.

2. Entre tarefas de prioridade `LOW`, a política adotada deve minimizar o tempo médio de espera para que as tarefas iniciem processamento. Considere que o tempo de execução de uma tarefa é proporcional ao seu tamanho em bytes (fornecido ao método `convert`). Se tarefas com prioridade `LOW` possuem o mesmo tamanho, entre elas deve valer a mesma política usada pra prioridade `NORMAL`.

3. Entre tarefas com prioridade `NORMAL`, deve ser garantido que todas as tarefas progridam em conjunto. Isto é, deve ser uniformizado o tempo entre os períodos de execução (determinado via `setQuantum`) alocados às tarefas.

4. Entre tarefas com prioridade `HIGH`, deve ser garantido que a primeira tarefa termine primeiro, que a segunda em segundo, e assim por diante. Diferentemente do que ocorre nos demais níveis, não há alternação entre as tarefas de alta prioridade. Ainda assim, é necessário contabilizar os ciclos de processamento alocados para as tarefas de acordo com o configurado em `setQuantum`.

2.2 Testes Unitários

Além desse documento foi fornecido um conjunto de testes unitários. Em caso de dúvidas quando ao presente documento, consulte a classe `PriorityScheduledConverterTest`.

3 Relatório e Instruções de Entrega

Além da implementação, na forma de um arquivo `.tar.gz` (ou `.zip`), entregue um relatório em formato PDF com identificação do grupo e respondendo as seguintes perguntas.

1. Relacione os conceitos relacionados ao escalonamento e gerência de processos, de ao menos um dos livros no plano de ensino da disciplina, com a implementação realizada nesse trabalho. Escreva um texto coerente, com uma organização lógica.
2. Aponte as diferenças entre o escalonador que foi implementado e escalonadores de processos/threads de um sistema operacional.

4 Avaliação

Cada componente do trabalho tem o seguinte peso:

1. Implementação: 4.0;
   - Testes unitários fornecidos passando: 2.0
   - Testes extras (secretos!) passando: 2.0
   - Haverá desconto de até 1.0 por más práticas já abordadas em aula ou em outros trabalhos.
2. Relatório: 3.0 (2.0 para a questão 1 e 1.0 para a questão 2);
3. Apresentação (nota individual): 3.0.

A nota máxima do trabalho é 10.0. O trabalho deve ser realizado em grupos de até duas pessoas. Em caso de cópia entre grupos, o componente do trabalho onde houve cópia terá a nota dividida entre todos os grupos envolvidos.

A apresentação cobre tanto a implementação como o relatório. Não deixe de fazer o relatório.

5 Dicas

Esses são alguns termos e links que podem ser relacionados à solução e podem ajudá-los. Esses se referem principalmente à implementação

- Capítulo de escalonadores no seu livro de SO preferido
- Monitores em Java (`sycnhronized`)
- Ordenamento lexicográfico
- Mútiplas filas
- Explorem os javadocs da API padao do Java
- https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html
- https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html
- https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html
- https://google.github.io/guava/releases/21.0/api/docs/com/google/common/base/Stopwatch.html