Segundo Exercício-Programa

Norton Trevisan Roman

27 de outubro de 2023

1 Implementação de Readers e Writers

Este trabalho tem por objetivo apresentar ao aluno o uso de threads, ilustrando a importância prática das soluções para o problema de Leitores e Escritores. Para tal, vocês devem inicialmente organizar-se em grupos de 4 (quatro) ou 5 (cinco) integrantes. Em seguida, devem criar uma estrutura de dados (arranjo, lista ligada ou qualquer outra lista de Strings) contendo, em cada um de seus elementos, uma palavra do arquivo "bd.txt" (essa estrutura deve permanecer em memória). Esse arquivo corresponde ao texto "A Treatise Concerning the Principles of Human Knowledge", de George Berkeley (1710), formatado de modo a que haja apenas uma palavra (e pontuação satélite) por linha, num total de 36.242 linhas (é bastante recomendável a leitura desse texto filosófico). Assim, sua estrutura terá 36.242 elementos, com uma palavra por elemento.

Uma vez feito isso, a ideia é transformar a estrutura em uma região crítica, permitindo acesso concorrente. Você deve então criar um arranjo de objetos de thread (classes que estendem Thread ou implementam Runnable). Os objetos serão de dois tipos distintos (podem ser objetos de uma mesma classe, mas com comportamentos distintos): leitores, que acessarão a base somente para leitura, e escritores, que não farão nada além de escrever na base.

O arranjo de threads deve conter exatamente 100 (cem) objetos de leitura ou escrita, organizados aleatoriamente[1]. Note que são objetos já criados, ou seja, feito já o *new*. Uma vez populado o arranjo, você deve rodar cada thread na sequência (a aleatoriedade na posição do objeto dentro do arranjo deve ajudar a dissipar quaisquer efeitos determinísticos nesse experimento.).

Assim, o processo de criação e execução das threads pode ser resumido como:

1. Repita 100 vezes:
   (a) Crie um objeto reader ou writer (sua proporção será definida mais adiante) – será uma thread
   (b) Escolha aleatoriamente uma posição ainda vazia do arranjo de threads
   (c) Guarde esse objeto nessa posição

[1]Para saber como trabalhar com números aleatórios em java consulte `http://www.javapractices.com/topic/TopicAction.do?Id=62`

2. Para i ← 1 até 100:

   (a) Rode a i-ésima thread

Obs: Para o passo 1, alternativamente, pode-se popular o arranjo com os objetos e então embaralhá-lo.

Cada objeto no arranjo, seja leitor ou escritor, deve entrar na base e fazer 100 acessos a posições aleatórias da base (ou seja, a posição deve vir do gerador de números aleatórios). Se o objeto for um leitor, ele deve tão somente ler a palavra na posição desejada, armazenando-a em alguma variável local (e não mais usando aquele valor). Se, contudo, o objeto for um escritor, ele deve escrever "MODIFICADO" na posição correspondente na base. Terminados os 100 acessos, o processo deve dormir por 1ms, ainda dentro da base, simulando alguma validação que esteja fazendo dos dados. Só então ele sairá da base.

Durante o período desses 100 acessos (mais a soneca de 1ms), os processos estarão usando a base como um todo. Muito embora pudéssesmos bloquear o acesso a registros individuais, para fins didáticos vamos transformar a base inteira em uma única região crítica. Ou seja, antes do primeiro acesso, o objeto entra na região crítica, saindo dela somente após ter acordado de seu sono de 1ms (que se dá após seu 100º acesso).

O restante da tarefa é dividida em duas partes:

1. Implemente ou baixe uma solução (sim, é permitido o uso de solução não própria), em Java, para o problema de Leitores e Escritores. Aplique então essa solução à estrutura de dados e ao arranjo de threads concorrentes, marcando o tempo total de execução do sistema[2]. O tempo deve ser marcado entre o final do povoamento do arranjo de threads (ou seja, após todos os objetos terem sido criados mas não terem iniciado sua execução) e o término da última thread. Atenção! Atente para o fato de que a thread principal, que comanda as demais e toma o tempo do sistema, não pode terminar antes da última thread terminar (dica: use join()).

2. Implemente ou baixe uma versão do sistema que não faça uso de Leitores e Escritores, ou seja, uma com o sistema bloqueando todo e qualquer acesso à base toda vez que alguém, seja Leitor ou Escritor, estiver dentro dela.

Em ambos os casos, você deverá rodar o sistema para diferentes proporções de Readers e Writers: 0 Reader e 100 Writers, 1 Reader e 99 Writers, 2 Readers e 98 Writers . . . 99 Readers e 1 Writer, 100 Readers e 0 Writer. Para cada proporção dessas, rode o sistema 50 vezes, calculando então o tempo médio gasto nesse processo. É muito importante que o arranjo de threads não seja reutilizado a cada nova rodada do sistema, ou seja, que, a cada proporção diferente, ele seja reconstruído de maneira aleatória, conforme descrito acima.

Na implementação de Readers e Writers (item 1 acima), a política empregada pelo seu sistema será a de priorizar sempre os leitores. Assim, Escritores esperam até que não haja mais

[2]Para isso, use currentTimeMilils, de `http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/System.html`

leitores na base. A implementação sem Readers e Writers não precisa apresentar prioridade específica.

1.1 Material para Entrega

A entrega será feita unica e exclusivamente via eDisciplinas. Você deve criar um arquivo "No_USP.zip" (em que No_USP é seu número USP) contendo o seguinte material:

- Código java do programa
- Relatório de avaliação do sistema (em PDF)

O relatório deve dizer (por meio de tabelas e gráficos), para cada proporção usada, qual o tempo médio gasto pelo sistema (média das 50 repetições), tanto para a implementação com Readers e Writers quanto para a implementação sem eles. Deve também indicar claramente se houve ganho no uso de Leitores e Escritores e, se for o caso, em que situação (proporção de Leitores e Escritores) isso ocorreu. Ele deve explicar suas conclusões com base em gráficos do tempo de execução médio para cada uma das proporções, em ambas implementações.

Atenção!

- É de suma importância o relatório, já que ele corresponderá a 70% de sua nota.
- Apenas um integrante do grupo deve fazer a entrega. Não esqueçam de, no relatório, listar os integrantes.

2 Observações

- Ao modificar a base na memória, não escrevam no disco. Todas as modificações ficam em RAM. Escrever no disco só faria o sistema ficar mais lento.
- Na hora de tomar os tempos, eliminem todas as chamadas ao SO (println, escritas a logfile etc) que não sejam absolutamente necessárias (como sleep, por exemplo), pois isso irá ativar o escalonador e a tomada de tempo pode ser prejudicada.
- Se quiserem usar nanoTime() em vez de currentTimeMillis() tudo bem. O ponto é que não necessariamente haverá uma melhora. Então, na prática, tanto faz, pois as 100 threads dormindo 1ms cobrem o problema.

3 Data de Entrega

26 de Novembro de 2023.