Trabalho 05

Disciplina: Programação Paralela **Data: 02 de maio de 2014.** **Prof.:** Fernando Castor

1. Um deque (abreviatura para *Double-Ended Queue*, do inglês) é uma estrutura de dados similar a uma fila, com a diferença fundamental de que é possível tanto inserir elementos quanto removê-los a partir dos dois lados. Consequentemente, ao invés de ter apenas as operações `push` e `pop`, um deque tem quatro operações: `push_left`, `pop_left`, `push_right` e `pop_right`. Essas operações realizam, respectivamente, inserção de um elemento pelo lado esquerdo, remoção de um elemento a partir do lado esquerdo, inserção de um elemento pelo lado direito e remoção de um elemento a partir do lado direito.

Implemente em Java uma estrutura de dados deque que oferece as quatro operações mencionadas acima e que, quando utilizada por múltiplas *threads* simultaneamente, deve satisfazer as seguintes propriedades:

P1 Nunca ocorre de duas *threads* realizarem uma operação (inserção ou remoção) em um mesmo lado ao mesmo tempo.

P2 Operações de inserção e remoção inserem e removem, respectivamente, exatamente um elemento do deque, exceto quando ele está vazio. Neste caso, a remoção de um elemento não modifica a estrutura de dados.

P3 Remoções sempre se aplicam aos elementos nas pontas do deque. Da mesma forma, inserções sempre ocorrem nas pontas do deque (propriedade básica de um deque).

P4 O programa não entra em deadlock.

P5 Sempre é possível para duas *threads* distintas realizar operações de inserção (`push_left()` e `push_right()`) simultaneamente nos dois lados do deque.

P6 Se duas threads distintas tentam realizar operações (remoção ou inclusão) em um mesmo lado, uma delas consegue.

Além disso, sua estrutura de dados deve, tanto quanto possível, permitir que remoções sejam realizadas a partir dos dois lados simultaneamente.

Não é permitido o uso de nenhuma estrutura de dados da biblioteca `java.util.concurrent`. É permitido, porém, o uso de travas explícitas do pacote `java.util.concurrent.locks`. É explicitamente permitido usar a classe `java.util.LinkedList<E>`, que implementa uma lista duplamente ligada em Java. Essa classe tem operações `addFirst()`, `addLast()`, `removeFirst()` e `removeLast()`, responsáveis por inserir um item no início da lista, inserir um elemento no fim, remover o último elemento da lista e remover o primeiro, respectivamente. `LinkedList` não é uma classe segura para *threads*.

2. Implemente uma solução para a questão 2 que: (i) empregue hashing conforme descrito na Seção 6.1.2.3; **ou** (ii) necessite de apenas uma lista ligada.

3. Analise cada um dos programas construídos nos trabalhos 04 e 05 (contadores paralelos e deques), excetuando-se variações simples envolvendo mudanças de tipos de variáveis, em termos dos cinco critérios de projeto descritos no Capítulo 6: (i) speedup; (ii) contenção; (iii) taxa entre trabalho e sincronização; (iv) taxa entre leituras e escritas; (v) complexidade.

4. Que fatores de hardware e software impedem que as memórias dos nossos computadores funcionem de forma sequencialmente consistente? Considere um programa concorrente em Java onde todos os atributos sejam declarados com o modificador `volatile`. É possível dizer que ele é sequencialmente consistente? Justifique sua resposta e apresente exemplos que apoiem essa justificativa. E se todos os atributos fossem declarados com o tipo `java.util.concurrent.atomic.AtomicReference`? Neste último caso você pode considerar que os atributos não são `volatile`.