Faculdade de Ciências da Universidade do Porto

Implementação de HashSets Concorrentes

Programação Concorrente (CC3040)- 2021/2022

Mateus Almeida 201805265

Introdução

No âmbito da Unidade Curricular de Programação Concorrente, foram desenvolvidas quatro implementações diferentes de Hash Sets Concorrentes, utilizando diferentes métodos de garantir tal concorrência. O projeto foi desenvolvido na linguagem Java.

Desenvolvimento

Todas as classes implementaram a interface IHSet.java, tendo consequentemente sido implementadas os métodos size(), add(e), remove(e), contains(e), rehash() e waitFor(). Seguindo o exemplo da implementação com blocos **synchronized** no ficheiro HSet0.java, todos os ficheiros implementam também o método getEntry(e), que devolve a lista associada ao elemento *e*.

Todas as seguintes implementações foram submetidas aos testes unitários do código base, tendo passado a todos.

HSet1.java

A classe HSet1.java substituiu os blocos synchronized pelo uso de uma instância de **ReentrantLock**, com uma condição **Condition** associada para notificar a presença de um certo elemento desejado.

Antes da realização de qualquer leitura ou escrita no array de LinkedList que representa o HashSet, é bloqueado o **ReentrantLock**, sendo apenas desbloqueado no fim das modificações. O método waitFor() espera que a condição **Condition** receba o sinal **Condition.signalAll()** proveniente do método add(e), caso o elemento ainda não se encontre presente no HashSet.

HSet2.java

Análogo à classe anterior, HSet2.java substituti o ReentrantLock por um **ReentrantReadWriteLock** com uma condição **Condition** associada ao ReentrantReadWriteLock.writeLock().

Os métodos size() e contains(e) necessitam apenas do ReentrantReadWriteLock.readLock() enquanto que os métodos add(e), remove(e), waitFor(e) e rehash() necessitam do ReentrantReadWriteLock.writeLock(). O método waitFor(e) espera a notificação da condição **Condition.signalAll()** para confirmar a presença do elemento desejado.

HSet3.java

A classe HSet3.java mantém agora um array de **ReentrantReadWriteLock,** com um array de condições associado, ambos do tamanho à tabela original (sem rehash()). Garante então que a ação de escrita e leitura são bloqueadas apenas para a entrada na tabela em que se realizará as modificações.

Implementou-se um método getIndex(e) que devolve o índice de entrada a que corresponde o argumento. O método retorna o resultado de Math.abs(e.hashCode() % lockSize), sendo *lockSize* o tamanho original da tabela.

No rehash da tabela, os array **ReentrantReadWriteLock** e array **Condition** mantêm-se inalterados. Para a realização da operação de rehash da tabela, são adquiridos todos os locks de escrita para evitar deadlocks.

HSet4.java

A classe HSet4.java é implementada usando a API Java-friendly para ScalaSTM, uma implementação de Software Transactional Memory.

Foram implementados os métodos getIndex(), semelhante à classe anterior, que devolve o índice de entrada a que corresponde o argumento, e nodeContains(node, e) que dado um nó inicial da lista a que corresponde *e*, retorna verdadeiro ou falso caso esse elemento esteja presenta na lista.

Todos os elementos retorna o resultado dentro de um bloco STM.atomic(). Os métodos add(e) e remove(e) percorrem a lista a que corresponde *e* manualmente, adicionando ou removendo-o no local correcto, respetivamente.

O método waitFor() espera um retorno verdadeiro de contains(e), recorrendo a STM.retry() até obter tal resultado.

O método rehash() cria uma nova instância de STM.newTArray() de tamanho o dobro da tabela anterior e percorre as listas da tabela original, adicionando todos os seus nós com o método add(e).

Conclusão

O projeto demonstrou-se integral para a compreensão dos diferentes métodos de garantia de concorrência em objetos na linguagem Java e para a melhor aprendizagem de programação com threads.

Todas as implementações foram submetidas aos testes unitários do código base, tendo passado a todos. Concluí-se então que foram atingidos os objetivos pretendindos.