UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL

RELATÓRIO DO TRABALHO SOBRE LISTA ENCADEADA CONCORRENTE

NATAL/RN
2022

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL

ARTURO FONSECA DE SOUZA - 20190043631
DIEGO FILGUEIRAS BALDERRAMA - 20190035676

RELATÓRIO DO TRABALHO SOBRE LISTA ENCADEADA CONCORRENTE

Trabalho referente à nota da Unidade II da disciplina DIM0124 - Programação Concorrente - T01, orientado pelo Prof. Dr. Everton Ranielly de Sousa Cavalcante.

NATAL/RN
2022

Para a realização do trabalho proposto, inicialmente foi implementada uma solução utilizando a linguagem de programação C++. Entretanto, tal solução estava incompleta pois havia exclusão mútua entre as diferentes operações, mas somente uma *thread* de busca podia ser executada por vez. Foi extremamente difícil de continuá-la para que fosse possível distinguir os tipos de operações/*threads* a serem executadas, permitindo assim, a ação de várias *threads* de busca simultaneamente.

Após revisões das aulas gravadas e algumas pesquisas, foi reparado que a solução mais viável para o problema poderia ser feita de forma simples utilizando a linguagem Java, que possui a classe *ReentrantReadWriteLock*, capaz de definir *locks* de leitura (para a busca) e escrita (para a inserção e remoção), possibilitando enfim a exclusão mútua entre as diferentes operações ao mesmo tempo que permite a execução de múltiplas *threads* simultâneas para a operação de busca.

Dessa forma, a solução em C++ foi substituída por uma solução em Java utilizando *locks* como mecanismo de sincronização, mais especificamente o *ReentrantReadWriteLock*, citado no parágrafo anterior. Assim, o fluxo de execução do programa se dá da seguinte forma: o *Main* instancia um vetor de *threads* para cada tipo de operação a ser realizada, em seguida instancia cada *thread* dos vetores, preparando seus parâmetros de construção e, logo após, as *threads* são colocadas em execução. Por fim, o *Main* aguarda o final da execução de todas as *threads* criadas e finaliza o programa.

A respeito do gerenciamento de acesso à região crítica pela classe da lista encadeada, o fluxo de execução ocorre da seguinte maneira para cada um dos métodos implementados:

- Inserção: assim que esse método é invocado, caso nenhuma outra *thread* esteja acessando o objeto da lista no momento (seja para leitura ou escrita), o *lock* de **escrita** é bloqueado e, a partir desse momento, nenhuma outra *thread* pode efetuar buscas, inserções ou remoções na lista. Quando o conjunto de instruções de inserção é finalizado, o método libera o *lock* de escrita e as outras *threads* já podem ser escaladas para realizar suas respectivas operações, respeitando as regras de exclusão mútua implementadas;
- Remoção: análogo ao de inserção, operando também sobre o *lock* de **escrita**, mas ao invés de adicionar elementos à lista, uma remoção de elemento é efetuada;

- Busca: quando o método de busca é invocado, o *lock* de **leitura** é bloqueado (caso o *lock* de escrita esteja liberado) e as instruções de busca são realizadas. Após o final da busca, o *lock* de leitura é liberado de volta. Apesar desse *lock* ser bloqueado no método, outras *threads* de busca podem efetuar as consultas na lista simultaneamente dada a natureza do *lock* utilizado, mas as *threads* de inserção e remoção terão de esperar a finalização das de leitura para que possam realizar suas respectivas operações, respeitando também as regras de exclusão mútua implementadas.

A garantia da corretude da solução com relação a concorrência, em termos de resultado, ou seja, a ausência de valores inconsistentes, corrompidos ou indefinidos na lista ao final da execução, se dá principalmente pelos fatos: (i) duas ou mais *threads* de escrita nunca são executadas ao mesmo tempo; (ii) a execução de múltiplas *threads* de leitura não é capaz de alterar o estado dos valores da lista; (iii) a linguagem Java não permite a finalização forçada da execução de *threads*.

Sobre as garantias de exclusão mútua e ausência de condições de *deadlock* e *starvation*, se dão principalmente pelos fatos: (i) toda vez que um método de acesso à região crítica é iniciado, é feita uma chamada de solicitação de bloqueio do *lock* correspondente, e essa solicitação só é atendida quando nenhum outro método de acesso estiver em execução (exceto na de busca, que a solicitação é atendida somente se outra *thread* estiver efetuando a mesma operação); (ii) todo *lock* adquirido é liberado no mesmo método; (iii) as instruções de acesso à lista encadeada dos métodos estão inseridas em *try blocks*, portanto mesmo que uma exceção seja capturada durante sua execução, o *lock* adquirido previamente será sempre liberado pois está sendo realizado em um *finally block*; (iv) a liberação de um *lock* não está vinculada à liberação de nenhum outro *lock*; (v) o *ReentrantReadWriteLock* foi instanciado passando um *boolean* com valor verdadeiro como parâmetro no seu construtor, indicando que a política de justiça será aplicada.

Por fim, para compilar o programa, basta abrir o projeto nomeado “ListaEncadeadaConcorrente” no Eclipse[1] e clicar no botão de executar da IDE. Todos os valores de inserção, remoção e busca na lista encadeada são definidos

[1] Eclipse IDE utilizada na versão 2021-12. Para compilação em outras IDEs, pode ser necessário exportar o projeto ou compilar via linha de comando.

arbitrariamente no código, portanto não há necessidade de inserir valores de entrada. A saída padrão exibirá os registros de (tentativas de) bloqueio e liberação dos *locks*, bem como informações referentes às operações realizadas.