Introdução

O objetivo deste trabalho consiste em resolver uma série de exercícios propostos numa ficha prática das aulas de Programação Concorrente.
Os exercícios estão relacionados com a implementação de multi-threading.
Existem três exercícios com o mesmo objetivo: somar valores dentro de um array.

1º Exercício

É-nos pedido no primeiro exercício para somarmos os valores de um array através da criação de uma classe. A minha implementação foi feita da seguinte forma: criei uma classe result com um in value e duas funções synchronized: uma que soma um valor e outra que retorna o valor.
Na função mt1, inicializo uma variável da classe result, um array de threads e um n que é o número de partes em que o array é dividido. Faço um ciclo com o n de threads. Crio duas variáveis que vão ser as posições iniciais e finais do array, as quais vão ser somadas pela thread no momento. De seguida, crio a thread e num ciclo de start a end (as variáveis de cima) uso a classe criada para somar os valores do array recebido na função. No fim retorno com a função de retorno da classe.

2º Exercício

A implementação do segundo é muito semelhante à do primeiro, Ciclos variáveis auxiliares, etc são exatamente as mesmas. A única diferença da implementação é a forma como guardo e somo o resultado.
O ponto chave é o uso de um AtomicInteger, implementado já no java. Segui a mesma implementação de cima exceto que ao somar uso a função addAndGet( ) para retornar o valor uso a função get( ).

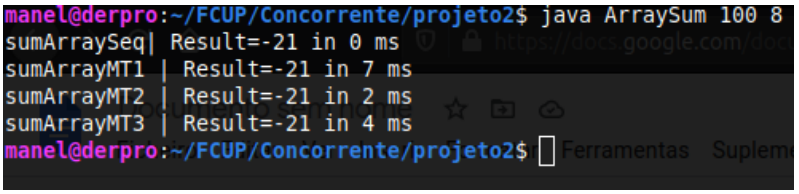
3º Exercício

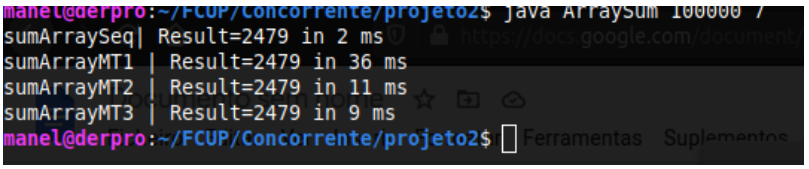
O terceiro exercício é mais do mesmo. Como a função tem o mesmo objetivo, a forma de como a fiz foi praticamente igual às outras duas. A única diferença é, novamente, a forma de como são somados e retornados os valores. Desta vez foi criado um array de int com o mesmo tamanho do array de threads e cada posição dos dois estava diretamente relacionada à outra. Assim, cada posição do array de ints tinha a soma das posições do array recebido de input. No final, faz-se um ciclo a somar todos os valores do array da soma, e retorna-se isso.

Resolução do resto da divisão > 0

Para os três casos acima, foi necessário ter em conta os casos em que a divisão do array não atribui o número de posições iguais às threads. Esse problema foi resolvido de igual forma nos três casos.
A solução é relativamente simples: criei uma variável auxiliar aux=0, a qual é somada sempre à variável start (posição do array onde a thread começa a somar).
Sempre que é detetado um resto da divisão maior que 0, a variável auxiliar é incrementada para na próxima iteração ver a posição seguinte sempre, o end também se torna uma posição à frente e a variável que guarda o resto da divisão icial decrementa um valor.

Análise de Resultados e conclusão





Concluindo, num modo geral, a segunda opção é a preferível para grande partes dos casos, visto que aparenta ser a mais rápida. No entanto, para casos excessivamente grandes, a terceira opção seria um caminho a seguir. Independentemente disso, o conceito de multi-threading traz uma série de vantagens, tanto a nível de conhecimento como a nível de eficiência do código. Sem dúvida uma ferramenta a implementar sempre que possível.

Manuel Sá
up201805273