Aula Prática – Introdução a Threads em Java

Nesta aula, iremos exercitar a criação de threads em Java.

Criação de Threads

O código *HelloWorld.java* cria 10 threads através da implementação da interface *Runnable*. As threads criadas se apresentam e saúdam o usuário. Compile e execute o código algumas vezes e veja que a ordem das impressões das saudações é aleatória. Por que? Dê explicações para esses comportamentos.

O que poderia ser feito para que as saudações aparecerem na ordem de criação das threads, ou seja, *"Olá Mundo! Saudações da thread 0, Olá Mundo! Saudações da thread 1, Olá Mundo! Saudações da thread 2..."* e assim por diante? Como a solução adotada afeta a concorrência entre as threads?

Tempo de Criação de Threads e Processos

O código *CreationTime.java* tem por objetivo comparar o tempo de criação de threads vs processos tradicionais. Para isso, o programa cronometra o tempo de duração para a criação de 10000 threads que não fazem nenhum trabalho. Em seguida, cronometra a criação de 10000 processos que também não fazem trabalho algum.

Compile e execute o código. Qual das criações é a mais demorada? Justifique fazendo referência às informações contidas na tabela de threads e na tabela de processos.

OBS.: note que o processo pai não espera pelo término das threads e processos filhos criados.

Threads e o Uso da CPU

No programa *CPUThreads.java,* threads são criadas de acordo com o número inteiro *n* passado na linha de comando. Se, por exemplo, *n=10*, então o seu programa cria *10* threads e as deixa executando **concorrentemente**.

Execute o programa com um número crescente de *n.* Comece com *n=1* e verifique o montante de CPU que o programa/thread está usando (comando *top*). Depois execute com *n=2* e, em seguida, *n=3*, *n=4*... O que acontece com o uso de CPU por thread? Por que? Dica: ao usar o *top*, ative a opção para monitorar threads. Para isso, com o top ativo, pressione a tecla '*H*'. Ative também a opção para monitorar todos os núcleos do processador, pressionando a tecla '*1*'.

Prioridade em Java

O programa *ThreadExplore.java* lê o número de núcleos da máquina e lança duas threads CPU-intensiva para cada um deles. A prioridade da última thread criada é elevada ao máximo possível na JVM. Há alteração no comportamento desta thread? Justifique.

Compartilhamento de Dados

No programa *IncrementTest.java*, duas threads invocam o método *incValue* 1000 vezes cada uma. Este método incrementa o valor de um objeto acumulador que inicialmente é

setado para 0. Dessa maneira, seria esperado que as duas threads juntas incrementassem o acumulador até 2000, mas não é isso o que acontece! Execute o código várias vezes e veja. Por que o comportamento da execução não sai sempre como esperado?
O problema está no método *incValue*. Uma thread executando neste método pode ser *preemptada* justamente no momento em que ela acabou de ler *this.value* em *tmp* (linha 9). Suponha, por exemplo, que *value = 5* já tenha sido salvo em *tmp*. A próxima thread lerá o mesmo valor dessa variável e somará 1 a ela, ou seja, fará *value = 6*. Quando a thread anterior voltar a executar, ela incrementará o valor antigo de *value*, ou seja, 5, sobreescrevendo *value = 6* (não houve incremento).

Exemplifique a execução que gera o menor valor possível de *value*. Que valor é esse?
Como podemos evitar que uma thread seja preemptada dentro do método *incValue*?

Threads e o Processamento Paralelo

Este exercício supõe que a sua máquina tenha ao menos 2 núcleos de processamento. No Linux, você pode verificar a quantidade de núcleos a partir do campo de saída "CPU(s):" do comando
$ lscpu

Crie um programa que inicializa uma thread para um trabalho CPU-intensivo. Por exemplo, encontrar o fatorial de um grande número. Meça o tempo para a thread realizar toda a tarefa.

Em seguida, modifique o programa para criar duas threads, uma que busca o fatorial final a partir de números pares e outra a partir de números ímpares. Meça o tempo para realizar todas as tarefas. O tempo é menor que a versão monothread? Justifique.

Execute a versão de duas threads em um único núcleo usando o comando
$ taskset -c 1 nome_programa
onde *nome_programa* deve ser substituído pelo seu executável Java. Qual o tempo de execução final? Houve ganho de desempenho em relação às execuções anteriores? Justifique.