Tischler Florian
Mat. Nr.: 1315066
Gruppenmitglieder: Mathias Hölzl
gelöste Aufgaben: 4/4
resultierende Punkte: 100%

Aufgabe 1

a)

In einer For - Schleife werden 8 erzeugt und in einem Array gespeichert. Anschließend wird auf deren Beendung mittels join() gewartet.

b

Anmerkungen zur Implementierung

Damit die Threads lange genug "am Leben" sind wurde ein Thread.sleep(100000) verwendet um sicherzustellen dass kein Thread freigegeben wird während noch neue hinzugefügt werden.

Maximale Threadanzahl

Die maximale Anzahl an Threads ist beschränkt durch die Größe des Stacks der jedem Thread zugewiesen wird und ob das Program für 32 bit oder 64 bit Systeme kompiliert wird. Daher erhalten wir bei exzessiven Thread Generierung irgendwann einen Out-Of-Memory Fehler da kein Speicher für den Thread Stack vorhanden ist.

Da die Standardgröße des Stacks plattformabhängig ist kann keine allgmeine Aussage getroffen werden, wieviele Threads maximal erzeugt werden können.

Auf meinem Testsystem:

- Windows 10 Home

- 8 GB RAM

- Java SE Runtime Environment 1.8.0_66-b17

konnte die Testimplementierung bei Standardeinstellungen ca. 2800 Threads erzeugen. Nachdem die Stackgröße auf den kleinst möglichen Wert (64K) konfiguriert wurde, konnten ca. 4950 Threads erzeugt werden. Die Stackgröße wurde per JVM Option "-XX:ThreadStackSize=64" angepasst.

Ich kann aber leider nicht ganz nachvollziehen warum auf meinem System nur so wenige Threads erzeugt werden können da theoretisch (bei Stackgröße 64K und 32 Bit also 2 GB virtuellem Speicher) 32768 Threads möglich sein sollten (die

Anzahl ist natürlich geringer da nicht die ganzen 2GB für die Threads zur Verfügung stehen aber die Größenordnung sollte trotzdem zutreffend sein).

Aufgabe 2)

Anmerkungen zu den Klassen

Pi: statische öffentliche Klasse welche die calc(iterationen, threadCount) Funktion zur Verfügung stellt die Pi berechnet

- ThreadPool: Erzeugt und verwaltet die Threads
- ThreadPoolRunnable: spezielle Runnable Implementierung die ein anderes Runnable ausführt und nach Beendigung dies dem ThreadPool mitteilt, damit dieser die nächste Aufgabe starten kann
- PiCalcRunnable: Enthält die eigentliche Berechnung der Teilsummen
- Accumulator: Synchronisierter Punkt an dem die Teilsummen zusammengefasst werden

Folgende Werte wurden bei der Ausführung auf meinem System

- Windows 10 Home

- Intel Core i7 Q4710MQ @ 2,5 GHz

- 8 GB RAM

- Java SE Runtime Environment 1.8.0_66-b17

erzielt:

Iteration Count: 500000

[1 Threads] PI: 3,142, Result: 3,142, Elapsed: 738,50 ms
[2 Threads] PI: 3,142, Result: 3,142, Elapsed: 384,55 ms
[3 Threads] PI: 3,142, Result: 3,142, Elapsed: 287,93 ms
[4 Threads] PI: 3,142, Result: 3,142, Elapsed: 225,39 ma
[5 Threads] PI: 3,142, Result: 3,142, Elapsed: 233,52 ms
[6 Threads] PI: 3,142, Result: 3,142, Elapsed: 221,98 ms
[7 Threads] PI: 3,142, Result: 3,142, Elapsed: 211,29 ms
[8 Threads] PI: 3,142, Result: 3,142, Elapsed: 208,14 ms
[9 Threads] PI: 3,142, Result: 3,142, Elapsed: 208,10 ms
[10 Threads] PI: 3,142, Result: 3,142, Elapsed: 212,86 ms
[11 Threads] PI: 3,142, Result: 3,142, Elapsed: 208,94 ms
[12 Threads] PI: 3,142, Result: 3,142, Elapsed: 212,11 ms
[13 Threads] PI: 3,142, Result: 3,142, Elapsed: 215,82 ms
[14 Threads] PI: 3,142, Result: 3,142, Elapsed: 212,36 ms
[15 Threads] PI: 3,142, Result: 3,142, Elapsed: 213,77 ms
[16 Threads] PI: 3,142, Result: 3,142, Elapsed: 216,38 ms
[17 Threads] PI: 3,142, Result: 3,142, Elapsed: 225,34 ms
[18 Threads] PI: 3,142, Result: 3,142, Elapsed: 221,26 ms
[19 Threads] PI: 3,142, Result: 3,142, Elapsed: 216,49 ms
[20 Threads] PI: 3,142, Result: 3,142, Elapsed: 217,19 ms

fastest: 208,10 (9 Threads)

Aufgabe 3: Java Thread States

Die Klasse ThreadStates.java erstellt und startet 5 Threads im abstand von ~3sec. Ein weiterer Thread gibt Informationen über die Threads und iheren Status. Haben alle Threads den Status Terminated erreicht, beendet sich das programm.

Reihenfolge:

- NEW: Thread constructor wurde aufgerufen
- RUNNABLE: Thread.start()
- TIMED_WAITING Thread.sleep() innerhalb von Thread.run()
- WAITING Objects.wait() innerhalb von Thread.run()
- RUNNBALE(Resume) Objects.notify()
- BLOCKED Thread Object locked
- TERMINATED Thread.run() beendet

NEW
Thread wurde erstellt

RUNNABLE
Thread Läuft und wird vom Scheduler verarbeited

WAITING/TIMED_WAITING
Thread wartet bis sleep() abgelaufen bzw notify aufgerufen wird

BLOCKED
Thread objekt wartet bis lock aufgelöst

Terminated
Thread.run() terminiert

Java Thread States: (http://www.c-jump.com/bcc/c257c/Week13/Week13.html)