T-444-USTY

3. Febrúar 2017

Lab 3

Thread Assignment

Teacher: Kári Halldórsson
Reykjavík University
Computer Science
Spring 2017

Students:
Hrafnkell Ívarsson
Raquelita Rós Aguilar

Requirements

The program was supposed to be able to do the following three requirements when running the function Solve.findAndPrintSolution() thirty times:

Requirement 1
,,Sequentially: Don’t run the next instance until the one before has returned”

Requirement 2
,,All at once in separate threads. A new thread is created for each instance. “

Requirement 3
,,A certain number at a time. Threads are run through a thread pool of a certain size (students can try different sizes). New instances aren’t run until a thread is free in the thread pool.”

Given with assignment

Five classes were given with this assignment as base program, named:

- Problem.java
- Problematic.java
- Solver.java
- ThreadAssignment01Main.java
- Tile.java

Implemented

As suggested in the project description a class was created, SomeRunnableThing.java which created a new instance of Thread. Inside this class the function public void run() was implemented which is executed when the start() function is called with some thread. For each requirement a for-loop was created were three different functions was supposed to run NUMBER_OF_PROBLEMS times. A private variable REQUIREMENTS was also created for testing purposes to make a condition so that the program would run the problems correctly through the if-loops.

Code snippet 1 – Requirement 1

```
System.out.println("Requirement 1");
          for(int i = 0; i < NUMBER_OF_PROBLEMS; i++){
        Solver.findAndPrintSolution(Problematic.nextProblem());
      }
```

Here you can see a code snippet for a requirement 1, where Solver.findAndPrintSolution takes in some problem, solves it and returns the solution. Below you can find sample output for this part which always gave us the same result when testing.

```
Requirement 1
Problem: (557,160) - (110,177) - length: 464 - iterations: 217040
Problem: (461,73) - (99,155) - length: 444 - iterations: 204752
Problem: (615,254) - (394,470) - length: 437 - iterations: 188783
Problem: (296,109) - (34,279) - length: 432 - iterations: 243092
Problem: (85,493) - (61,440) - length: 77 - iterations: 11811
Problem: (263,29) - (336,608) - length: 652 - iterations: 329093
Problem: (279,152) - (618,465) - length: 652 - iterations: 383761
Problem: (276,214) - (2,334) - length: 394 - iterations: 263061
Problem: (113,34) - (489,423) - length: 765 - iterations: 342223
Problem: (60,307) - (350,354) - length: 337 - iterations: 149614
Problem: (393,490) - (64,8) - length: 811 - iterations: 406964
Problem: (493,167) - (222,265) - length: 369 - iterations: 183582
Problem: (422,584) - (107,589) - length: 320 - iterations: 125153
Problem: (216,64) - (525,344) - length: 589 - iterations: 304957
Problem: (626,79) - (148,152) - length: 551 - iterations: 200691
Problem: (303,611) - (328,206) - length: 430 - iterations: 191063
Problem: (313,267) - (603,279) - length: 302 - iterations: 181243
Problem: (232,148) - (381,22) - length: 275 - iterations: 133651
Problem: (423,194) - (398,320) - length: 151 - iterations: 45554
Problem: (119,429) - (111,580) - length: 159 - iterations: 48948
Problem: (13,309) - (113,149) - length: 260 - iterations: 74511
Problem: (279,35) - (623,21) - length: 358 - iterations: 147055
Problem: (413,605) - (248,223) - length: 547 - iterations: 260947
Problem: (130,454) - (548,561) - length: 525 - iterations: 296899
Problem: (297,432) - (392,501) - length: 164 - iterations: 53931
Problem: (436,382) - (324,618) - length: 348 - iterations: 213150
Problem: (359,475) - (441,372) - length: 185 - iterations: 68236
Problem: (299,273) - (344,252) - length: 66 - iterations: 8754
Problem: (431,263) - (150,389) - length: 407 - iterations: 270185
Problem: (553,90) - (379,333) - length: 417 - iterations: 160783
```

Code snippet 2 – Requirement 2

```
if(REQUIREMENT == 2 || true){
    System.out.println("Requirement 2");
    Thread[] threads = new Thread[NUMBER_OF_PROBLEMS];

    for(int i = 0; i < NUMBER_OF_PROBLEMS; i++){
            threads[i] = new Thread(new
        SomeRunnableThing(Problematic.nextProblem()));
                    threads[i].start();
    }
    for(int i = 0; i < NUMBER_OF_PROBLEMS; i++){
    //join the threads to get running time
            try { threads[i].join(); }
            catch (InterruptedException e) { e.printStackTrace(); }
    }
```

In requirement 2, each problem has it's own thread, so they are all solved at once. This causes the results to come in a slightly different order with each run, but the shorter problems are solved first and the longer last.

Afterwards, the threads are joined so running time can be calculated correctly. Each thread is of the class SomeRunnableThing.java that takes in a problem and runs Solver.findAndPrintSolution() for it. The class is shown below.

```
public class SomeRunnableThing implements Runnable{

            Problem problem;

            @Override
            public void run() {
                        Solver.findAndPrintSolution(problem);
            }

            public SomeRunnableThing(Problem p) {
                        problem = p;
            }
}
```

Below you can find sample output for this part.

```
Requirement 2
Problem: (17,123) - (35,73) - length: 68 - iterations: 6747
Problem: (199,303) - (201,301) - length: 4 - iterations: 36
Problem: (209,74) - (26,40) - length: 217 - iterations: 73585
Problem: (234,614) - (69,593) - length: 186 - iterations: 43263
Problem: (121,283) - (60,396) - length: 174 - iterations: 57623
Problem: (430,18) - (155,17) - length: 276 - iterations: 82526
Problem: (54,190) - (194,89) - length: 241 - iterations: 78845
Problem: (368,273) - (411,44) - length: 272 - iterations: 148425
Problem: (516,7) - (182,147) - length: 474 - iterations: 167550
Problem: (367,204) - (254,64) - length: 253 - iterations: 125489
Problem: (620,392) - (387,533) - length: 374 - iterations: 143767
Problem: (605,344) - (433,113) - length: 403 - iterations: 177142
Problem: (329,292) - (476,452) - length: 307 - iterations: 188609
Problem: (296,446) - (80,284) - length: 378 - iterations: 243601
Problem: (88,37) - (90,587) - length: 552 - iterations: 220940
Problem: (301,5) - (542,276) - length: 512 - iterations: 228537
Problem: (58,406) - (602,283) - length: 667 - iterations: 346968
Problem: (485,612) - (180,180) - length: 737 - iterations: 343975
Problem: (159,582) - (581,436) - length: 568 - iterations: 272058
Problem: (579,65) - (302,559) - length: 771 - iterations: 336367
Problem: (241,504) - (565,307) - length: 521 - iterations: 311244
Problem: (47,177) - (362,566) - length: 704 - iterations: 345957
Problem: (468,19) - (545,634) - length: 692 - iterations: 326039
Problem: (189,449) - (600,323) - length: 537 - iterations: 333002
Problem: (95,341) - (425,66) - length: 605 - iterations: 341967
Problem: (0,209) - (461,411) - length: 663 - iterations: 309539
Problem: (620,232) - (286,614) - length: 716 - iterations: 351743
Problem: (345,541) - (50,154) - length: 682 - iterations: 376601
Problem: (596,111) - (266,629) - length: 848 - iterations: 371364
Problem: (432,427) - (70,44) - length: 745 - iterations: 403001
```

Code snippet 3 – Requirement 3

```
if(REQUIREMENT == 3 || true){
  System.out.println("Requirement 3");
  ExecutorService threadPool =
  Executors.newFixedThreadPool(POOL_SIZE);
  for(int i = 0; i < NUMBER_OF_PROBLEMS; i++){
    threadPool.execute(new
    SomeRunnableThing(Problematic.nextProblem()));
  }
        threadPool.shutdown();
        try{
                threadPool.awaitTermination(5, TimeUnit.MINUTES);
        }catch(InterruptedException e){
                e.printStackTrace();
        }
}
```

For requirement 3 a threadPool was created to keep track of a specific number of concurrent threads. To do that a private variable POOL_SIZE was created as suggested in the project description. POOL_SIZE is the number of threads running at a time. A new instance of SomeRunnableThing was then sent to threadPool.execute and when the program had gone through the problems a shutdown() function was used to shut down the threads. But to make sure that the threads weren't shutten down before the problems were finished a try-catch was added that waits five minutes before shutting the threads down.

Below you can find sample output for this part.

```
Requirement 3
Problem: (297,269) - (333,484) - length: 251 - iterations: 126433
Problem: (633,524) - (413,257) - length: 487 - iterations: 172137
Problem: (256,408) - (449,256) - length: 345 - iterations: 217551
Problem: (383,303) - (161,170) - length: 355 - iterations: 239078
Problem: (201,209) - (294,519) - length: 403 - iterations: 247092
Problem: (260,403) - (566,551) - length: 454 - iterations: 319412
Problem: (157,94) - (558,274) - length: 581 - iterations: 302173
Problem: (139,102) - (339,524) - length: 622 - iterations: 321680
Problem: (122,279) - (425,279) - length: 303 - iterations: 150306
Problem: (201,340) - (539,610) - length: 608 - iterations: 386581
Problem: (324,355) - (6,141) - length: 532 - iterations: 383508
Problem: (27,547) - (181,326) - length: 375 - iterations: 113294
Problem: (223,361) - (12,518) - length: 368 - iterations: 241446
Problem: (263,45) - (271,486) - length: 449 - iterations: 211200
Problem: (441,485) - (401,256) - length: 269 - iterations: 126673
Problem: (625,203) - (210,6) - length: 612 - iterations: 285062
Problem: (632,423) - (118,634) - length: 725 - iterations: 347142
Problem: (42,402) - (376,498) - length: 430 - iterations: 192962
Problem: (551,13) - (348,203) - length: 393 - iterations: 114455
Problem: (316,194) - (226,374) - length: 270 - iterations: 139997
Problem: (495,211) - (424,274) - length: 134 - iterations: 35772
Problem: (368,63) - (62,70) - length: 313 - iterations: 131646
Problem: (578,491) - (406,635) - length: 316 - iterations: 112530
Problem: (636,535) - (306,447) - length: 418 - iterations: 127319
Problem: (263,407) - (369,496) - length: 195 - iterations: 76229
Problem: (17,35) - (330,341) - length: 619 - iterations: 224039
Problem: (554,528) - (550,174) - length: 358 - iterations: 134459
Problem: (105,64) - (526,275) - length: 632 - iterations: 294057
Problem: (38,334) - (376,592) - length: 596 - iterations: 303667
Problem: (581,167) - (411,540) - length: 543 - iterations: 259525
```

Running Times

Method

Each version of the program was run 5 times and running times noted.
All runs were done on the same computer. This method does not produce a reliable average, but can give a good reference to compare the different methods with.

System Information

Operating System: Linux Mint 18 Cinnamon 64-bit
Linux Kernel: 4.4.0-21
Processor: Intel Core i5-6500 3.20GHz x 4

Requirement 1

1	2	3	4	5	avg
23197 ms	23237 ms	23173 ms	23088 ms	23227 ms	23184 ms

This method is very unefficient and has by far the longest running time. However, the running time seem to deviate very little from the average.

Requirement 2

1	2	3	4	5	avg
4122 ms	3408 ms	3530 ms	3435 ms	3930 ms	3685 ms

An impressive improvement over requirement 1, more than 6 times faster on average than the method used there. The running time seem to deviate from the average a little bit more though.

Requirement 3 (with POOL_SIZE = 10)

1	2	3	4	5	avg
3759 ms	3827 ms	3624 ms	3867 ms	3870 ms	3790 ms

Similar to requirement 2 in speed, but deviates less from the average.
When trying different values of POOL_SIZE it seemed the higher the number the faster the runtime. Lower numbers, however, seemed to cause less deviation from average running time.