

**Софийски Университет „Св. Климент Охридски”**

**Факултет по математика и информатика**

**Курсов проект**

**По Разпределени софтуерни архитектури**

**Задача:**

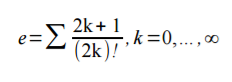
*Пресмятане на e, v1*

**Изготвил: Иванка Златкова Зайчева, фак.№: 61779, 3 курс, Софтуерно инженерство**

**Дата: 27.06.2015г. Проверил: ………………….. (подпис)**

1. **Условие на задачатa.**

Едно важно за математиката число е Неперовото число (Ойлеровото число), тоест числото e. Използвайки сходящи редове, можем да сметнем стойността на e с произволно висока точност. Един от сравнително бързо сходящите към e редове е:



***Изискванията към програмата са следните***:

1. Команден параметър задава точността на пресмятанията. Точността се изразява в брой цифри след десетичната запетая. Командният параметър задаващ точността има вида: “-p 10240”
2. Команден параметър задава максималния брой нишки (задачи) на които разделяме работата по пресмятането на e – например “–t 1” или “–tasks 3”;
3. Програмата извежда подходящи съобщения на различните етапи от работата си, както и времето отделено за изчисление и резултата от изчислението (стойността на e
4. Записва резултата от работа си (стойността на e) във изходен файл, зададен с подходящ параметър, например “-o result.txt”. Ако този параметър е изпуснат, се избира име по подразбиране
5. Да се осигури възможност за „quiet“ режим на работа на програмата, при който се извежда само времето отделено за изчисление на e, отново чрез подходящо избран друг команден параметър - например “-q”;
6. Да се използват паралелни процеси (нишки) и осигуряване пресмятането на e със зададена от потребителя точност.
7. **Описание на реализацията.**

Проекта е реализиран на Java, като са използвани следните инструменти за многонишкова синхронизация:

* Command pattern - постига се като класа отговорен за изчисленията на отделните суми от сходящия ред наследява класа Runnable. По този начин се позволява на thread pool-a да извърши дадената задача, въпреки че самият той е бил написан така, че да няма знание за конкретните команди.
* Fixed Size Thread Pool – Създава thread poool, който използва повторно определен брой нишки. Във всеки момент най-много nThreads нишки ще бъдат активни задачи за обработка. Ако се подават допълнителни задачи, когато всички задачи са активни, те ще чакат в опашката, докато има нишка. Ако някоя нишка се прекъсне поради неизправност по време на изпълнение преди изключване, нова ще заеме мястото си, ако е необходимо, за да изпълни следващите задачи. Задачите в thread pool-a ще съществуват, докато не бъде изрично изключено.
* Synchronized statement – използва се за синхронизиране като се използва конкретен обект за заключване на кода в блока маркиран за синхронизиране. По този начин две нишки нямат достъп едновременно до един и същи блок код.

При стартиране на системата, се изпълнява следната последователност от действия:

* Проверява входните параметри за:
* P
* t
* q
* o
* Прави се проверка дали максималния брой нишки е най-ефективния начин или може да използваме по-малко на брой нишки.
* Прави се проверка и броя нишки се определя така, че всяка нишка да прави изчисленията за 4 суми от сходящия ред,ако максималния брой нишки го позволява.
* Инициализира се Executors.newFixedThreadPool() с цел използване на максимално въведения брой нишки.
* Чрез цикъл се инициализират нишките и им се подава задача на всяка една по отделно – калкулация на отделна сума от сходящия ред
* Класът CustomMath наследява класа Runnable
* Методът run изпълнява следната последователност от дейности:

- Извежда на конзолата коя нишка стартира

- Изчислява определената сума от сходящия ред използвайки метода calculatePartOfSum

- Извежда на конзолата отнелото време на нишката за завърши своята работа в милисекунди

- Извежда на конзолата края на изпълнението на дадена нишка

* Методът calculatePartOfSum прави необходимите изчисления за пресмятане на член от сходящия ред. За тази цел използва Apfloat и метода fact, който изчислява факториел. За добавянето на стойността на дадения член от сходящия ред към крайната сума се използва synchronized statement като по този начин се осигурява, че две нишки няма да извършват операции върху обекта sum едновременно.
* Чрез shutdownNow осигуряваме, че executorService няма да приеме нови задачи и при приключване на задача ще се опита да приключи дадена нишка
* Чрез awaitTermination се осигурява, че всички задачи ще бъдат приключени преди да се продължи с изпълнението на кода.
* Прави се запис във файла с крайната стойност на числото е
* Прави се запис в конзолата за броя използвани нишки и отнелото време за изпълнение на програмата

1. **Причини за избраната реализация**

* Използвайки ExecutorService, защото: предоставя начин за лесно:
* стартиране;
* спиране;
* изпращане;
* блокиране;
* и други;
* Използвайки Runnable:
* не специализираме какво трябва да изпълни нишката
* използваме един обект, който се споделя от всички нишки и използваме синхронизация само на местата, където е необходимо
* Използваме newFixedThreadPool, а не newCachedThreadPool защото:
* Потребителя задава максималния брой нишки, които да изпозва програмата. newFixedThreadPool осигурява това, че ще се използва фиксиран брой нишки.
* newCachedThreadPool преизползва създадените нишки и създава нови при нужда
* чрез newFixedThreadPool ограничаваме броя на задачите, които се изпълняват едновременно
* Използвайки awaitTermination:
* Изчакваме executorService-a да приключи, за да продължим с изпълнението на кода
* Преди него извикваме shutDownNow, за да сме сигурни, че изпълнението ще приключи достатъчно бързо

1. **Резултати от измерването на ефективността, ускорението и времето**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Threads | Number of Terms | Time | Acc | Efficiency |
| 1 | 1000 | 608 | 1 | 1 |
| 2 | 1000 | 444 | 1,369369369 | 0,684684685 |
| 3 | 1000 | 434 | 1,400921659 | 0,466973886 |
| 4 | 1000 | 369 | 1,647696477 | 0,411924119 |
| 5 | 1000 | 366 | 1,661202186 | 0,332240437 |
| 6 | 1000 | 366 | 1,661202186 | 0,276867031 |
| 7 | 1000 | 356 | 1,707865169 | 0,243980738 |
| 8 | 1000 | 345 | 1,762318841 | 0,220289855 |
| 9 | 1000 | 341 | 1,782991202 | 0,198110134 |
| 10 | 1000 | 335 | 1,814925373 | 0,181492537 |

При изпълнение програмата с повече от една нишка, изчисленията се ускоряват значително до даден момент. След него разликата между времето необходимо за извършване на изчисленията варира около 100 ms.