JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Protokoll

A06 Summenberechnung

Softwareentwicklung
4CHIT 2016/17

Filip Scopulovic

Version 1.0
Note:
Begonnen am 13. November 2016
Betreuer: W. Rafeiner-Magor
Beendet am 13. November 2016

Inhaltsverzeichnis

1 Code **1**

1.1 Vorgehensweise 1

1.2 Aufwand 1

1.3 Resultate 1

1.4 Schwierigkeiten 2

1.5 Gesamter Code 3

2 Beobachtung **5**

2.1 Laufzeitmessung 5

1 Code

1.1 Vorgehensweise

Die Vorgehensweise war schlicht. Ich habe mir kurz überlegt wie ich es anstellen könnte und habe drauf los programmiert. Die Initialisierung von der Thread-Klasse habe ich aus dem PDF von dem Herr Professor Rafeiner-Magor entnommen.

1.2 Aufwand

Der Aufwand war nicht recht groß. Selbst `Sphinx` hat keinen großen Aufwand erfordert.

1.3 Resultate

Die Summierung funktioniert zwar, jedoch muss ich wegen der Partitionsmethode die eingegebene Zahl manuell einfügen. Hier einige Beispiele:

```
D:\Programme\Python\python.exe C:/Users/Filip/Desktop/Daten/Schule/4CHIT/SEW_4CHIT/SEW-16-17/
    A06_Thread_Summenberechnung/summenberechnung.py
Write a number that you want to sum!
>>4
[[0], [1, 2], [3, 4]]
Current number: 0
Current number: 1
Current number: 3
Current number: 6
Current number: 10
Sum of the numbers: 10

Process finished with exit code 0
```

Listing 1: Resultat mit der Eingabe 4

```
D:\Programme\Python\python.exe C:/Users/Filip/Desktop/Daten/Schule/4CHIT/SEW_4CHIT/SEW-16-17/
    A06_Thread_Summenberechnung/summenberechnung.py
Write a number that you want to sum!
>>20
[[0, 1, 2, 3, 4, 5, 6], [7, 8, 9, 10, 11, 12], [13, 14, 15, 16, 17, 18, 19, 20]]
Current number: 0
Current number: 1
Current number: 3
Current number: 6
Current number: 10
Current number: 15
Current number: 21
Current number: 28
Current number: 36
Current number: 45
Current number: 55
Current number: 66
Current number: 78
Current number: 91
Current number: 105
Current number: 120
Current number: 136
Current number: 153
Current number: 171
Current number: 190
Current number: 210
Sum of the numbers: 210
```



```
Process finished with exit code 0
```

Listing 2: Resultat mit der Eingabe 20

```
D:\Programme\Python\python.exe C:/Users/Filip/Desktop/Daten/Schule/4CHIT/SEW_4CHIT/SEW-16-17/
    A06_Thread_Summenberechnung/summenberechnung.py
Write a number that you want to sum!
>>100
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27,
    28, 29, 30, 31, 32], [33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
    52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66], [67, 68, 69, 70, 71, 72, 73, 74, 75,
    76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99,
    100]]
Current number: 0
Current number: 1
...
Current number: 4919
Current number: 4984
Current number: 5050
Sum of the numbers: 5050

Process finished with exit code 0
```

Listing 3: Resultat mit der Eingabe 100

1.4 Schwierigkeiten

Ich hatte ein Problem mit der Methode, die meine Inputliste in 3 circa gleich große Teile teilt. Deshalb habe ich gegoogelt und einen gute Idee von der Seite stackoverflow.com[1] geholt. Ich habe diese Methode an das Projekt angepasst und implementiert. Das ist die angepasste Methode:

```
def partition(lst):
    """
    Method idea from http://stackoverflow.com/questions/2659900/python-slicing-a-list-into-n-nearly-
        equal-length-partitions
    :param lst:
    :return [lst[int(round(division * i)): int(round(division * (i + 1)))] for i in range(3)]: return a
        list, that has lists in it, those lists are seperated in the number what division got
    """
    div_three = len(lst) / 3
    return [lst[int(round(div_three * i)): int(round(div_three * (i + 1)))] for i in range(3)]
```

Listing 4: Partitionmethode

[1] http://stackoverflow.com/questions/2659900/python-slicing-a-list-into-n-nearly-equal-length-partitions

1.5 Gesamter Code

```
"""
@author: Filip Scopulovic
@date: 11-11-2016
@use: Threading Klasse; Benutzer gibt eine Zahl ein und diese werden dann aufaddiert
"""
import threading, time

class Summenberechnung(threading.Thread):
    """
    User gives a input and then it sums the numbers from 1 to the input number
    It works with three threads
    class-attributes: counter, lock
    :inheritance threading.Thread:
    """
    #counter is here for counting up and knowing what number a thread is using
    __counter = 0

    #lock is here to lock the attribute counter so the threads won't interrupt themself
    __lock = threading.Lock()


    def __init__(self, sum_list):
        """
        Constructor that calls the super-constructor from the threading.Thread class
        """
        threading.Thread.__init__(self)
        self.start_time = time.time()
        self.sum_list = sum_list

    def run(self):
        """
        run-method from the constructor
        Here is everything that the threads will do
        :return None:
        """
        for i in range(len(self.sum_list)):
            with Summenberechnung.__lock:
                Summenberechnung.__counter += self.sum_list[i]
                #print("Current number: %s" % (Summenberechnung.__counter))
        end_time = time.time()
        print(end_time - self.start_time)

    def get_counter(self):
        """
        Returns the counter
        :return Summenberechnung.__counter:
        """
        return Summenberechnung.__counter

def make_input_list():
    """
    Method that takes care of the input
    Checks what inputs are in and if one is not okay it runs the method another time
    Initialise a list that has every number of the input number in it
    :return None:
    """
    input_list = []
    input_number = input("Write a number that you want to sum!\n>>")
    try:
        for i in range(int(input_number)):
            input_list += [i]
        if int(input_number) < 0:
            print("No negative numbers")
            make_input_list()
        else:
            input_div_three(input_list, input_number)
    except ValueError:
        print("The input is not a number!")
        make_input_list()

```

```
def partition(lst):
    """
    Method idea from http://stackoverflow.com/questions/2659900/python-slicing-a-list-into-n-nearly-equal-length-partitions
    :param lst:
    :return [lst[int(round(division * i)): int(round(division * (i + 1)))] for i in range(3)]: return a list, that has lists in it, those lists are seperated in the number what division got
    """
    div_three = len(lst) / 3
    return [lst[int(round(div_three * i)): int(round(div_three * (i + 1)))] for i in range(3)]

def input_div_three(input_number_list, last_number):
    """
    Takes the input number list and cuts this through three
    :param input_number:
    :param last_number:
    :return None:
    """
    div_three_list = partition(input_number_list)
    #Need to add the input number, because the partition method does not put the input number in the list
    div_three_list[2] += [int(last_number)]
    print(div_three_list)
    start_threads(div_three_list)

def start_threads(div_sum_list):
    """
    Starts the threads and gives them their list
    :param div_sum_list:
    :return None:
    """
    threads = []
    #Initialize 3 threads of the class Summenberechung and starts them
    for i in range(3):
        thread = Summenberechnung(div_sum_list[i])
        threads += [thread]
        thread.start()

    #waits for the children-threads
    for x in threads:
        x.join()

    #prints the summed number out
    print("Sum of the numbers: %s" %(str(Summenberechnung.get_counter(Summenberechnung))))

make_input_list()
```

Listing 5: Gesamter Code

2 Beobachtung

2.1 Laufzeitmessung

Zuerst messe ich die Laufzeit mit kleineren Zahlen. Später mit mittelgroßen Zahlen und zum Schluss mit ganz großen Zahlen. Die Einheit der Zeit ist in Sekunden.

Thread 1	Thread 2	Thread 3	Eingabe
0.0004992485046386719	0.0004982948303222656	0.000484466552734375	9
0.0005011558532714844	0.00049972534179687 5	0.0004999637603759766	18
0.0004990100860595703	0.0	0.0004999637603759766	27
0.0005004405975341797	0.0005054473876953125	0.0004947185516357422	100
0.00099945068359375	0.0005004405975341797	0.0005023479461669922	300
0.0010013580322265625	0.0010018348693847656	0.0004999637603759766	500
3.316378355026245	3.339895486831665	3.3479015827178955	1000000
10.093743085861206	10.091236114501953	10.10425353050232	3000000
15.984353065490723	16.162463665008545	16.189979553222656	5000000

Tabelle 1: Laufzeit mit drei Threads

Thread 1	Thread 2	Eingabe
0.0005009174346923828	0.0005002021789550781	9
0.001001119613647461	0.0004994869232177734	18
0.001001119613647461	0.0004999637603759766	27
0.0005009174346923828	0.0005011558532714844	100
0.0004999637603759766	0.0005004405975341797	300
0.0005006790161132812	0.0004978179931640625	500
2.1929757595062256	2.2344863414764404	1000000
6.305695295333862	6.39276123046875	3000000
10.990462303161621	11.038997888565063	5000000

Tabelle 2: Laufzeit mit zwei Threads

Die Laufzeit mit zwei Threads ist bei größeren Zahlen besser, da man weniger Threads verwendet und das das Programm behindert.

Tabellenverzeichnis

1 Laufzeit mit drei Threads 5
2 Laufzeit mit zwei Threads 5

Listings

1 Resultat mit der Eingabe 4 1
2 Resultat mit der Eingabe 20 1
3 Resultat mit der Eingabe 100 2
4 Partitionmethode 2
5 Gesamter Code 3

Abbildungsverzeichnis

