Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития

Кафедра инфокоммуникаций

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ №2.24**

**дисциплины «Анализ данных»**

**Вариант 13**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | Выполнил:  Иващенко Олег Андреевич  2 курс, группа ИВТ-б-о-22-1,  09.03.02 «Информационные и вычислительные машины», направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | |  |
|  | | Руководитель практики:  Воронкин Роман Александрович, доцент кафедры инфокоммуникаций  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | |  |
|  |  | |  | |

Отчет защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_ Дата защиты\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ставрополь, 2024 г.

**Тема**: «Синхронизация потоков в языке программирования Python»

**Цель**: Приобретение навыков использования примитивов синхронизации в языке программирования Python версии 3.x.

Порядок выполнения работы

Индивидуальное задание: для своего индивидуального задания лабораторной работы 2.23 необходимо организовать конвейер, в котором сначала в отдельном потоке вычисляется значение первой функции, после чего результаты вычисления должны передаваться второй функции, вычисляемой в отдельном потоке. Потоки для вычисления значений двух функций должны запускаться одновременно.

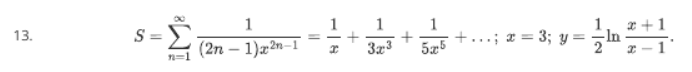


Рисунок 1 – Исходная формула

Листинг 1 – Код individual.py

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*-  """  Для своего индивидуального задания лабораторной работы 2.23 необходимо  организовать конвейер, в котором сначала в отдельном потоке вычисляется  значение первой функции, после чего результаты вычисления должны передаваться  второй функции, вычисляемой в отдельном потоке. Потоки для вычисления значений  двух функций должны запускаться одновременно.  """  import math  import threading  e = 10e-7  stepArray = [1]  def calculateY(x):  return 0.5 \* math.log((x + 1) / (x - 1))  def first\_function(x, n, results, barrier):  result = (2 \* n - 1) \* x\*\*(2 \* n - 1)  results[n] = result  barrier.wait()  def second\_function(step, index, results, barrier):  barrier.wait()  result = 1 / results[index]  step[index] = result  def main():  x = 3  index = 0  results = {}  barrier = threading.Barrier(2)  while abs(stepArray[index]) > e:  stepArray.append(0)  firstThread = threading.Thread(target=first\_function, args=(x, index + 1, results, barrier))  secondThread = threading.Thread(target=second\_function, args=(stepArray, index + 1, results, barrier))  firstThread.start()  secondThread.start()  firstThread.join()  secondThread.join()  index += 1  S = sum(stepArray) - 1  y = calculateY(x)  print(f"\nРезультат при x = {x}")  print(f"Сумма = {round(S, 4)}")  print(f"Y = {round(y, 4)}")  print(f"Разница между S и Y: {abs(round(S - y, 4))}\n")  if \_\_name\_\_ == "\_\_main\_\_":  main() |

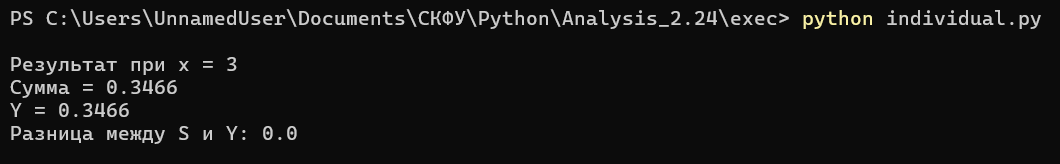


Рисунок 2 – Результат выполнения программы

Контрольные вопросы

1. Каково назначение и каковы приемы работы с Lock-объектом.

Lock-объекты используется для обеспечения эксклюзивного доступа к ресурсу в многопоточной среде.

Приёмы работы:

* acquire() – блокирует доступ к ресурсу. Если ресурс занят, поток будет ждать, пока он не освободится.
* release() – освобождает ресурс, позволяя другим потокам получить к нему доступ.

1. В чем отличие работы с RLock-объектом от работы с Lock-объектом.

RLock позволяет одному и тому же потоку захватывать блокировку несколько раз без блокировки. Требуется столько же realease(), сколько было acquire(), чтобы полностью освободить ресурс.

Для Lock – если поток, который уже владеет блокировкой, попытается снова её захватить, это приведёт к блокировке (deadlock).

1. Как выглядит порядок работы с условными переменными?

Порядок работы с условными переменными:

* Создать условную переменную с помощью Condition;
* Использовать методы acquire() и release() для управления блокировками;
* Методы wait(), notify(), notify\_all() используются для управления ожиданием и уведомлением потоков.

1. Какие методы доступны у объектов условных переменных?

* wait() – поток ожидает, пока не будет вызван notify() или notify\_all();
* notify() – пробуждает один из ожидающих потоков;
* notify\_all() – пробуждает все ожидающие потоки;
* acquire() – захватывает блокировку;
* release() – освобождает блокировку.

1. Каково назначение и порядок работы с примитивом синхронизации “семафор”?

Назначение семафора – управление доступом к ограниченному числу ресурсов. Порядок работы:

* acquire() – уменьшает значение семафора. Если значение семафора равно нулю, поток блокируется.
* release() – увеличивает значение семафора, разблокировывая, если нужно, ожидающий поток.

1. Каково назначение и порядок работы с примитивом синхронизации “событие”?

Используется для уведомления потоков о наступлении определённого состояния. Порядок работы:

* set() – устанавливает событие, переводя его в сигнальное состояние;
* clear() – сбрасывает событие, переводя его в несигнальное состояние;
* wait() – поток ожидает установки события.

1. Каково назначение и порядок работы с примитивом синхронизации “таймер”?

Таймер вызывает функцию через определённое время. Порядок работы:

* Создание таймера: timer = threading.Timer(interval, function, args=None, kwargs=None);
* Запуск таймера: timer.start();
* Отмена таймера: timer.stop();

1. Каково назначение и порядок работы с примитивом синхронизации “барьер”?

Барьер синхронизирует заданное количество потоков, ожидая их достижения определённой точки. Порядок работы:

* Создание барьера: barrier = threading.Barrier(parties);
* Ожидание барьера: barrier.wait();
* Сброс барьера: barrier.reset().

1. Сделайте общий вывод о применении тех или иных примитивов синхронизации в зависимости от решаемой задачи.

Lock и RLock используются для исключения гонок данных при доступе к общим ресурсам.

Условные переменные подходят для более сложных синхронизаций, где требуется уведомление о состоянии.

Семафор ограничивает количество потоков, которые могут одновременно использовать ресурс.

Событие применяется для уведомления потоков о наступлении какого-либо события.

Таймер используется для выполнения функций через определённые интервалы времени.

Барьер подходит для синхронизации точек сборки между несколькими потоками, где необходимо дождаться всех участников.

**Выводы**: В процессе выполнения лабораторной работы были приобретены навыки написания многопоточных приложений на языке программирования Python, а также было выполнено индивидуальное задание.