|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатики и систем управления

КАФЕДРА Теоретической информатики и компьютерных технологий

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4**

**по курсу: «Разработка параллельных и распределенных программ»**

Студент: Яровикова А. С.

Группа: ИУ9-41Б

Преподаватель: Царев А.С..

*.*

*2022 г.*

# Условие задачи

Лабораторная работа №4: Сделать любой из двух вариантов на выбор, на любом языке.

Вариант 1. Спровоцировать и пронаблюдать проблемы, возникающие в многопоточных программах при отсутствии синхронизации потоков или неправильном её выполнении.

1. Создать программу из пяти потоков — основного, А, B, C и D:

1.1. Основной поток запускает остальные четыре и ожидает их завершения.

1.2. Поток A с определёнными временными интервалами добавляет в список S числа 1, 2, 3 и т. д.

1.3. Поток B извлекает из списка S последний элемент, возводит его в квадрат и помещает в список R. Если в списке S нет элементов, поток B ожидает одну секунду функцией Sleep().

1.4. Поток C извлекает из списка S последний элемент, делит его на 3 и помещает в список R. Если в списке S нет элементов, поток C ожидает одну секунду.

1.5. Поток D извлекает из списка R последний элемент и печатает его. Если в списке R нет элементов, поток D печатает сообщение об этом и ожидает одну секунду.

Синхронизацию потоков производить на данном этапе не нужно. Запустить программу несколько раз, пронаблюдать результаты и занести их в отчет. Стабильной работы программы, очевидно, не ожидается.

2. Обеспечить корректную синхронизацию потоков.

2.1. Каждое обращение к спискам S и R из любого потока защищать критической областью (два разных объекта синхронизации, по одному на каждый из списков)

2.2. Запустить программу несколько раз, пронаблюдать результаты, занести в отчет результаты наблюдений.

Ожидается, что программа будет работать стабильно, не только не завершаясь аварийным образом, но и не «зависая».

3. Спровоцировать взаимоблокировку вследствие состязания.

3.1. Изменить алгоритм потока B на следующий:

1. вход в критическую область для списка S, вход в критическую область для списка R;
2. извлечение элемента из S, вычисление, добавление элемента в R;
3. выход из критической области для списка R, выход из критической области для списка S.

3.2. Аналогично изменить и алгоритм потока C, но порядок входа и выхода из критических областей.

3.3. Повторить пункт 2.2.

В целях наглядности можно между обращениями к КО в обоих потоках вставить задержки функцией Sleep(). Так можно проверить сценарии с разным порядком входа и выхода из КО. Результаты внести в отчёт и пояснить. Ожидается зависание потоков A, B и С, что будет проявляться в постоянно пустом списке R.

4. Спровоцировать блокировку при неконтролируемом удержании КО.

4.1. В версии программы, полученной в п. 2, внести изменение в алгоритм потока B: после захвата критической области для списка R с некоторой вероятностью следует произвести аварийное завершение работы потока.

4.2. Запустить программу, занести в отчет и объяснить результаты

# 2. Характеристики устройства и входных данных

* Процессор Intel(R) Core(TM) i5-4310U CPU @ 2.00GHz, 2601 МГц, ядер: 2, логических процессоров: 4. 8 ГБ ОЗУ;

## Язык Python 3.8.;

**3. Листинг текста программы**

1. **Lab41.py**

# без синхронизации потоков

import threading

import time

s = []

r = []

#lists length

n = 10

def a():

    print('A started')

    global s

    for i in range(n):

        s.append(i + 1)

        print('A added', i + 1)

        time.sleep(0.2)

    print('A finished')

def b():

    print('B started')

    global s, r

    for i in range(n):

        if len(s) == 0:

            time.sleep(1)

        else:

            el = s.pop()

            print('B got', el)

            r.append(el \* el)

        print('B finished')

def c():

    print('C started')

    global s, r

    for i in range(n):

        if len(s) == 0:

            time.sleep(1)

        else:

            el = s.pop()

            print('C got', el)

            r.append(el // 3)

        print('C finished')

def d():

    print('D started')

    global s, r

    for i in range(n):

        if len(r) == 0:

            print('list R is empty')

            time.sleep(1)

        else:

            el = r.pop()

            print(f'last element from D is {el}')

            # if el == n//3 or el == n\*\*2:

            #     print('D finished')

            #     os.\_exit(0)

        print('D finished')

threads = [ threading.Thread(target=a), threading.Thread(target=b), threading.Thread(target=c), threading.Thread(target=d)]

# запуск потоков

for t in threads:

    t.start()

# ожидание завершения всех потоков

for t in threads:

    t.join()

1. **Lab42.py**

# синхронизация поток с помощью критической области

import threading

import time

s = []

r = []

#lists length

n = 10

# lock oblect

locking = threading.Lock()

locking1 = threading.Lock()

def a():

    print('A started')

    global s

    for i in range(n):

        locking.acquire()

        s.append(i + 1)

        locking.release()

        print('A added', i + 1)

        time.sleep(0.2)

    print('A finished')

def b():

    print('B started')

    global s, r

    for i in range(n):

        if len(s) == 0:

            time.sleep(1)

        else:

            locking.acquire()

            el = s.pop()

            locking.release()

            print('B got', el)

            locking1.acquire()

            r.append(el \* el)

            locking1.release()

    print('B finished')

def c():

    print('C started')

    global s, r

    for i in range(n):

        if len(s) == 0:

            time.sleep(1)

        else:

            locking.acquire()

            el = s.pop()

            locking.release()

            print('C got', el)

            locking1.acquire()

            r.append(el // 3)

            locking1.release()

    print('C finished')

def d():

    print('D started')

    global s, r

    for i in range(n):

        if len(r) == 0:

            print('list R is empty')

            time.sleep(1)

        else:

            locking1.acquire()

            el = r.pop()

            locking1.release()

            print(f'last element from D is {el}')

    print('D finished')

threads = [ threading.Thread(target=a), threading.Thread(target=b), threading.Thread(target=c), threading.Thread(target=d)]

# запуск потоков

for t in threads:

    t.start()

# ожидание завершения всех потоков

for t in threads:

    t.join()

1. **Lab43.py**

# спровоцирована блокировка вследствие состязания

# зависание потоков А, В, С, проявляется в постоянно пустом списке К

import threading

import time

s = []

r = []

#lists length

n = 10

# lock oblect

locking = threading.Lock()

locking1 = threading.Lock()

def a():

    print('A started')

    global s

    for i in range(n):

        locking.acquire()

        s.append(i + 1)

        locking.release()

        print('A added', i + 1)

        time.sleep(0.4)

    print('A finished')

def b():

    print('B started')

    global s, r

    for i in range(n):

        if len(s) == 0:

            time.sleep(1)

        else:

            locking.acquire()

            time.sleep(0.5)

            locking1.acquire()

            el = s.pop()

            print('B got', el)

            r.append(el \* el)

            locking1.release()

            time.sleep(0.5)

            locking.release()

    print('B finished')

def c():

    print('C started')

    global s, r

    for i in range(n):

        if len(s) == 0:

            time.sleep(1)

        else:

            locking1.acquire()

            time.sleep(0.5)

            locking.acquire()

            el = s.pop()

            print('C got', el)

            r.append(el // 3)

            locking.release()

            time.sleep(0.5)

            locking1.release()

    print('C finished')

def d():

    print('D started')

    global s, r

    for i in range(n):

        if len(r) == 0:

            print('list R is empty')

            time.sleep(1)

        else:

            locking1.acquire()

            el = r.pop()

            print(f'last element from D is {el}')

            locking1.release()

            # if el == n//3 or el == n\*\*2:

            #     print('D finished')

            #     os.\_exit(0)

    print('D finished')

threads = [ threading.Thread(target=a), threading.Thread(target=b), threading.Thread(target=c), threading.Thread(target=d)]

# запуск потоков

for t in threads:

    t.start()

# ожидание завершения всех потоков

for t in threads:

    t.join()

1. **Lab44.py**

# спровоцировать блокировку при неконтролируемом удержании критической области

# после захвата КО в В для списка R с некоторой вероятностью следует произвести аварийное завершение работы потока

import threading

import time

from random import choice

s = []

r = []

#lists length

n = 10

# lock oblect

locking = threading.Lock()

locking1 = threading.Lock()

class UncontroledRetentionError(Exception):

    def \_\_init\_\_(self, probability, msg="Emergency shutdown of B thread"):

        self.probability = probability

        self.message = msg

        super().\_\_init\_\_(self.message)

def a():

    print('A started')

    global s

    for i in range(n):

        locking.acquire()

        s.append(i + 1)

        locking.release()

        print('A added', i + 1)

        time.sleep(0.2)

    print('A finished')

def b():

    print('B started')

    global s, r

    for i in range(n):

        if len(s) == 0:

            time.sleep(1)

        else:

            locking.acquire()

            el = s.pop()

            locking.release()

            print('B got', el)

            locking1.acquire()

            ch = choice([0, 1])

            if ch == 1:

                raise UncontroledRetentionError(ch)

            r.append(el \* el)

            locking1.release()

    print('B finished')

def c():

    print('C started')

    global s, r

    for i in range(n):

        if len(s) == 0:

            time.sleep(1)

        else:

            locking.acquire()

            el = s.pop()

            locking.release()

            print('C got', el)

            locking1.acquire()

            r.append(el // 3)

            locking1.release()

    print('C finished')

def d():

    print('D started')

    global s, r

    for i in range(n):

        if len(r) == 0:

            print('list R is empty')

            time.sleep(1)

        else:

            locking1.acquire()

            el = r.pop()

            locking1.release()

            print(f'last element from D is {el}')

    print('D finished')

threads = [ threading.Thread(target=a), threading.Thread(target=b), threading.Thread(target=c), threading.Thread(target=d)]

# запуск потоков

for t in threads:

    t.start()

# ожидание завершения всех потоков

for t in threads:

    t.join()

**4. Результаты работы**

1. Программа была запущена несколько раз, результаты каждый раз были разными (см. Таблица 1). Стабильной работы программы нет.

Таблица 1 - Результаты запуска программы lab41.py ( последние 3 элемента списка R)

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| last element from D is 2  D finished  last element from D is 4  D finished  last element from D is 9  D finished  B finished  C finished  C finished  C finished  C finished  C finished | last element from D is 3  D finished  last element from D is 4  D finished  last element from D is 1  D finished  A finished  C finished  C finished  C finished  C finished  C finished  C finished | last element from D is 3  B finished  D finished  B got 6  B finished  last element from D is 64  D finished  C got 7  last element from D is 36  C finished  D finished  A finished  C finished  B finished  B finished  C finished  C finished  C finished |

1. Программа работает практически стабильно, не завершаясь аварийным образом и не «зависая». Результаты программы все еще различные (см. Таблица 2) . Предположительно, работа без «зависаний» обеспечивается Python Global Interpreter Lock (GIL). GIL — это своеобразная блокировка, позволяющая только одному потоку управлять интерпретатором Python. Это означает, что в любой момент времени будет выполняться только один конкретный поток.

Таблица 2 - Результаты запуска программы lab42.py

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| A started  A added 1  B started  C started  B got 1  D started  last element from D is 1  list R is empty  A added 2  A added 3  A added 4  A added 5  B got 5  C got 4  list R is empty  B got 3  C got 2  A added 6  A added 7  A added 8  A added 9  A added 10  B got 10  C got 9  C got 7  B got 8  last element from D is 0  last element from D is 64  last element from D is 2  C got 6  last element from D is 3  last element from D is 2  last element from D is 100  last element from D is 9  D finished  A finished  C finished  B finished | A started  A added 1  B started  C started  B got 1  D started  last element from D is 1  list R is empty  A added 2  A added 3  A added 4  A added 5  B got 5  B got 3  B got 2  C got 4  A added 6  A added 7  A added 8  A added 9  A added 10  last element from D is 1  last element from D is 4  last element from D is 9  last element from D is 25  list R is empty  C got 10  B got 9  C got 8  C got 6  B got 7  A finished  last element from D is 49  last element from D is 2  D finished  B finished  C finished | A started  A added 1  B started  B got 1  C started  D started  last element from D is 1  list R is empty  A added 2  A added 3  A added 4  A added 5  list R is empty  C got 3  B got 4  C got 2  A added 6  A added 7  A added 8  A added 9  A added 10  B got 10  C got 9  C got 7  C got 6  last element from D is 0  last element from D is 2  B got 8  last element from D is 2  last element from D is 64  last element from D is 3  last element from D is 100  last element from D is 16  D finished  A finished  C finished  B finished |

1. Спровоцирована блокировка вследствие состязания. В результате происходит зависание потоков A, B и С, что проявляется в постоянно пустом списке R (см. Листинг 1).

Листинг 1 - Результаты запуска программы lab43.py

A started

A added 1

B started

C started

D started

list R is empty

list R is empty

list R is empty

list R is empty

list R is empty

list R is empty

list R is empty

list R is empty

list R is empty

list R is empty

D finished

1. После захвата критической области для списка R с некоторой вероятностью производится аварийное завершение работы потока В. Результаты программы все еще различные (см. Таблица 3) .

Таблица 3 - Результаты запуска программы lab44.py

|  |  |
| --- | --- |
| 1 (производится аварийное завершение работы потока В – ошибка UncontroledRetentionError) | 2 (ничего не происходит) |
| A started  A added 1  B started  B got 1  Exception in thread Thread-2:  C started  Traceback (most recent call last):  D started  File "C:\Users\soyar\AppData\Local\Programs\Python\Python38\lib\threading.py", line 932, in \_bootstrap\_inner  list R is empty  self.run()  File "C:\Users\soyar\AppData\Local\Programs\Python\Python38\lib\threading.py", line 870, in run  self.\_target(\*self.\_args, \*\*self.\_kwargs)  File "c:/my github/bmstu-iu9/parallel and distributed programs/lab4/lab44.py", line 49, in b  raise UncontroledRetentionError(ch)  \_\_main\_\_.UncontroledRetentionError: Emergency shutdown of B thread  A added 2  A added 3  A added 4  A added 5  A added 6  list R is empty  C got 6  A added 7  A added 8  A added 9  A added 10  list R is empty  A finished  list R is empty  list R is empty  list R is empty  list R is empty  list R is empty  list R is empty  list R is empty  D finished | A started  A added 1  B started  B got 1  C started  D started  last element from D is 1  list R is empty  A added 2  A added 3  A added 4  A added 5  A added 6  list R is empty  C got 5  C got 4  C got 3  B got 6  C got 2  A added 7  A added 8  A added 9  A added 10  A finished  B got 10  C got 9  B got 8  C got 7  last element from D is 0  last element from D is 2  last element from D is 64  last element from D is 3  last element from D is 100  last element from D is 36  last element from D is 1  D finished  C finished  B finished |