МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ

Отчет о лабораторной работе №2.23 по дисциплине основы программной инженерии

Выполнил:
Шальнев Владимир Сергеевич,
2 курс, группа ПИЖ-б-о-20-1,

Проверил:
Доцент кафедры
прикладной математики и
компьютерной безопасности,
Воронкин Р.А.

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2022 г.

Выполнение:

Пример №1:

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from threading import Thread
from time import sleep


def func():
    for i in range(5):
        print(f"from child thread: {i}")
        sleep(0.5)


if __name__ == '__main__':
    th = Thread(target=func)
    th.start()

    for i in range(5):
        print(f"from main thread: {i}")
    sleep(1)
```

Результат работы примера №1:

```
from child thread: 0
from main thread: 0
from main thread: 1
from main thread: 2
from main thread: 3
from main thread: 4
from child thread: 1
from child thread: 2
from child thread: 3
from child thread: 4

Process finished with exit code 0
```

Пример №2:

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from threading import Thread
from time import sleep


def func():
    for i in range(5):
        print(f"from child thread: {i}")
        sleep(0.5)
```

```
if __name__ == '__main__':
    th = Thread(target=func)
    print(f"thread status: {th.is_alive()}")
    th.start()
    print(f"thread status: {th.is_alive()}")
    sleep(5)
    print(f"thread status: {th.is_alive()}")
```

Результат работы примера №2:

```
thread status: False
from child thread: 0
thread status: True
from child thread: 1
from child thread: 2
from child thread: 3
from child thread: 4
thread status: False

Process finished with exit code 0
```

Пример №3:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from threading import Thread
from time import sleep


class CustomThread(Thread):
    def __init__(self, limit):
        Thread.__init__(self)
        self.limit_ = limit

    def run(self):
        for i in range(self.limit_):
            print(f"from CustomThread: {i}")
            sleep(0.5)


if __name__ == '__main__':
    cth = CustomThread(3)
    cth.start()
```

Результат работы примера №3:

```
from CustomThread: 0
from CustomThread: 1
from CustomThread: 2

Process finished with exit code 0
```

Пример №4:

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from threading import Thread, Lock
from time import sleep


lock = Lock()
stop_thread = False


def infinit_worker():
    print("Start infinit_worker()")
    while True:
        print("--> thread work")
        lock.acquire()
        if stop_thread is True:
            break
        lock.release()
        sleep(0.1)
    print("Stop infinit_worker()")


if __name__ == '__main__':
    # Create and start thread
    th = Thread(target=infinit_worker)
    th.start()
    sleep(2)
    # Stop thread
    lock.acquire()
    stop_thread = True
    lock.release()
```

Результат работы примера №4:

```
--> thread work
--> thread work
--> thread work
--> thread work
--> thread work
--> thread work
--> thread work
--> thread work
--> thread work
--> thread work
--> thread work
--> thread work
--> thread work
--> thread work
--> thread work
--> thread work
--> thread work
Stop infinit_worker()

Process finished with exit code 0
```

Пример №5:

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from threading import Thread
from time import sleep


def func():
    for i in range(5):
        print(f"from child thread: {i}")
        sleep(0.5)


if __name__ == '__main__':
    th = Thread(target=func, daemon=True)
    th.start()
    print("App stop")
```

Результат работы примера №5:

```
from child thread: 0
App stop

Process finished with exit code 0
```

Индивидуальное задание:

```
        #!/usr/bin/env python3
# -*- coding: utf-8 -*-


from threading import Thread
from math import sqrt


def infinite_sum(x, check, num):
    eps = 10 ** -7
    a = 1
    summa = 1
    i = 1
    prev = 0
    while abs(summa - prev) > eps:
        prev = summa
        numerator = 1
        denominator = 1
        for j in range(1, i + 1):
            if 2 * j - 3 > 0:
                numerator *= 2 * j - 3
            denominator *= 2 * j
        temp = (numerator / denominator) * x**i
        if (-1) ** (i + 1) > 0:
            summa += temp
        else:
            summa -= temp
        i += 1
    print(f"The sum number: {num} is: {summa}")
    print(f"Check sum: sqrt(1+({x})) = {check}")


if __name__ == '__main__':
    checksum1 = sqrt(1 - 0.8)
    thread1 = Thread(target=infinite_sum, args=(-0.8, checksum1, 1))
    thread1.start()
    checksum2 = sqrt(1 - 0.4)
    thread2 = Thread(target=infinite_sum, args=(-0.4, checksum2, 2))
    thread2.start()
    checksum3 = sqrt(1 - 0.5)
    thread2 = Thread(target=infinite_sum, args=(-0.5, checksum3, 3))
    thread2.start()
```

Пример работы индивидуального задания:

```
Calculated sum: 1 is: 0.44721390018995993
Verification sum: sqrt(1+(-0.8)) = 0.44721359549995787
Calculated sum: 2 is: 0.7745966925056
Verification sum: sqrt(1+(-0.4)) = 0.7745966692414834
Calculated sum: 3 is: 0.7071068393493292
Verification sum: sqrt(1+(-0.5)) = 0.7071067811865476

Process finished with exit code 0
```

Ответы на вопросы:

1. Что такое синхронность и асинхронность?

Синхронное выполнение программы подразумевает последовательное выполнение операций.

Асинхронное – предполагает возможность независимого выполнения задач

2. Что такое параллелизм и конкурентность?

Конкурентность предполагает выполнение нескольких задач одним исполнителем.

Параллельность предполагает параллельное выполнение задач разными исполнителями.

3. Что такое GIL? Какое ограничение накладывает GIL?

GIL — это аббревиатура от Global Interpreter Lock – глобальная блокировка интерпретатора. Он является элементом эталонной реализации языка Python, которая носит название CPython. Суть GIL заключается в том, что выполнять байт код может только один поток.

Если вы запустили в одном интерпретаторе несколько потоков, которые в основном используют процессор, то скорее всего получите общее замедление работы, а не прирост производительности. Пока выполняется одна задача, остальные простаивают (из-за GIL), переключение происходит через определенные промежутки времени. Таким образом, в каждый конкретный момент времени, будет выполняться только один поток, несмотря на то, что у вас может быть многоядерный процессор (или многопроцессорный сервер), плюс ко всему, будет тратиться время на переключение между задачами. Если код в потоках в основном выполняет операции ввода-вывода, то в этом случае ситуация будет в вашу пользу.

4. Каково назначение класса Thread ?

За создание, управление и мониторинг потоков отвечает класс Thread из модуля threading.

5. Как реализовать в одном потоке ожидание завершения другого потока?

Если необходимо дождаться завершения работы потока(-ов) перед тем как начать выполнять какую-то другую работу, то воспользуйтесь методом join().У join() есть параметр timeout, через который задается время ожидания завершения работы потоков.

6. Как проверить факт выполнения потоком некоторой работы?

Для того, чтобы определить выполняет ли поток какую-то работу или завершился используется метод is_alive().

7. Как реализовать приостановку выполнения потока на некоторый промежуток времени?

sleep(мс)

8. Как реализовать принудительное завершение потока?

В Python у объектов класса Thread нет методов для принудительного завершения работы потока. Один из вариантов решения этой задачи – это создать специальный флаг, через который потоку будет передаваться сигнал остановки. Доступ к такому флагу должен управляться объектом синхронизации.

9. Что такое потоки-демоны? Как создать поток-демон?

Для того, чтобы потоки не мешали остановке приложения (т.е. чтобы они останавливались вместе с завершением работы программы) необходимо при создании объекта Thread аргументу daemon присвоить значение True, либо после создания потока, перед его запуском присвоить свойству deamon значение True.

th = Thread(target=func, daemon=True)