МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное

учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

«Управление потоками в Python»

Отчет по лабораторной работе № 2.23

по дисциплине «Основы программной инженерии»

Выполнил студент группы ПИЖ-б-о-21-1

Пуценко И.А._____« » 2023г.

Подпись студента_____

Работа защищена « »_____20__г.

Проверил Воронкин Р.А. _____

(подпись)

Ставрополь 2023

Цель работы: приобретение навыков написания многопоточных приложений на языке программирования Python версии 3.x.

Ход работы:

1. Изучить теоретический материал работы.
2. Проработайте примеры лабораторной работы.

```
#!/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)

th = Thread(target=func) th.start()
 for i in
range(5):
    print(f'from main thread: {i}')
sleep(1)
```

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

Process finished with exit code 0
```

Рисунок 1 – Результат выполнения работы

```
#!/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()}")
```

```
thread status: False
from child thread: 0thread 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
```

Рисунок 2 – Результат выполнения работы

```
#!/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()
```

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

Process finished with exit code 0
```

Рисунок 3 – Результат выполнения работы

```
#!/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()
```

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

Process finished with exit code 0
```

Рисунок 4 – Результат выполнения работы

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


from threading import Thread, Lock 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")
```

Рисунок 5 – Результат выполнения работы

```
from child thread: 0App stop


Process finished with exit code 0
```

3. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.

4. Выполните клонирование созданного репозитория.

5. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.

6. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.

7. Создайте проект PyCharm в папке репозитория.

8. Выполните индивидуальное задание. Приведите в отчете скриншоты работы программы решения индивидуального задания.

С использованием многопоточности для заданного значения найти сумму ряда с точностью члена ряда по абсолютному значению и произвести сравнение полученной суммы с контрольным значением функции для двух бесконечных рядов. Вариант 12.

$$S = \sum_{n=1}^{\infty} \frac{\cos nx}{n} = \cos x + \frac{\cos 2x}{2} + \ldots; \; x = \pi; \; y = -\ln\left(2 \sin \frac{x}{2}\right).$$

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

from threading import Thread import
math

EPS = 10**(-7)
  def sum_func(x):      summa = 1.0
temp = 0      n = 1      while
abs(summa - temp) > EPS:
temp = summa
        summa += math.sin(n*x) / n
n += 1

    print(f"Sum is {summa}")
  def
check_func(x):
    res = - math.log(2 * math.sin(0.5 * x))
print(f"Check: {res}")
  if__name__==
'__main__':
    x = math.pi
    th1 = Thread(target=sum_func, args=(x,))
th2 = Thread(target=check_func, args=(x,))
th1.start()      th2.start()
```

```
Sum is 1.0000000000000002
Check: -0.6931471805599453

Process finished with exit code 0
```

Рисунок 6 – Результат выполнения работы

Контрольные 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?

Для того, чтобы двигаться дальше необходимо сказать несколько слов о 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. Это нужно для того, чтобы упростить работу с памятью (на уровне интерпретатора) и сделать комфортной разработку модулей на языке C. Это приводит к некоторым особенностям, о которых необходимо помнить. Условно, все задачи можно разделить на две большие группы: в первую входят те, что преимущественно используют процессор для своего выполнения, например, математические, их ещё называют CPU-bound, во вторую – задачи работающие с вводом выводом (диск, сеть и т.п.), такие задачи называют IO-bound. 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. В CPython все стандартные библиотечные функций, которые выполняют блокирующий ввод-вывод, освобождают GIL, это дает возможность поработать другим потокам, пока ожидается ответ от ОС.

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?

С помощью метода sleep() из модуля time.

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.

```
lock.acquire()
if stop_thread is True:
break
lock.release()
```

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

Есть такая разновидность потоков, которые называются демоны (терминология взята из мира Unix-подобных систем). Python- приложение не будет закрыто до тех пор, пока в нем работает хотя бы один недемонический поток.