МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное

образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Отчет по лабораторной работе №2.24

Работа с переменными окружения в Python3

по дисциплине «Синхронизация потоков в Python»

Выполнил студент группы ИВТ-б-о-20-1

Хашиев Х.М. « »_____20__г.

Подпись студента_____

Работа защищена « »_____20__г.

Проверил Воронкин Р.А. _____

(подпись

Ставрополь 2022

Цель работы: приобретение навыков с управлением потоками с помощью языка программирования Python версии 3.x.

Ход работы: Примеры

https://github.com/Mirror-Shard/L2.24

1. Изучил теоретический материал и приступил к выполнению примеров:

```
cv = Condition()
q = Queue()


# Consumer function for order processing
def order_processor(name):
    while True:
        with cv:
            # Wait while queue is empty
            while q.empty():
                cv.wait()
            try:
                # Get data (order) from queue
                order = q.get_nowait()
                print(f"{name}: {order}")
                # If get "stop" message then stop thread
                if order == "stop":
                    break
            except:
                pass
            sleep(0.1)
```

Рисунок 1 – Код примера

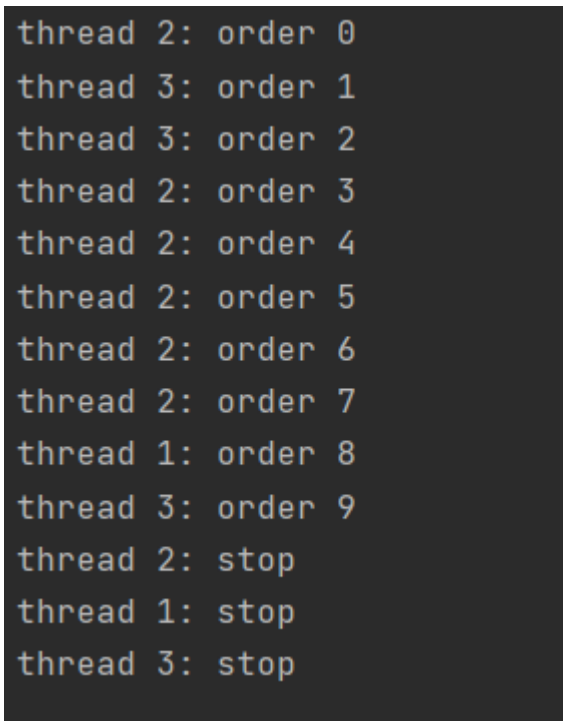

Рисунок 2 – Результат работы примера

Задание 1

Разработать приложение, в котором выполнить решение вычислительной задачи (например, задачи из области физики, экономики, математики, статистики и т. д.) с помощью паттерна “Производитель-Потребитель”.

1. Написал программу с двумя функциями для выполнения задачи с использованием функций Lock() и Queue():

```python
def producer(ls):
    lock.acquire()
    ls = ls
    for i in range(6):
        idx = random.randint(0, 3)
        exp = random.randint(1, 1000)
        q.put([ls[idx], exp])
    lock.release()


def consumer():
    lock.acquire()
    ls = []
    while not q.empty():
        s = q.get()
        r = random.choice(["Не подходит",
                           "Необходима стажировка",
                           "Принят"])
        print(f"Работник с id: {s[1]} "
              f"На должность: {s[0]}, "
              f"Результат: {r}")
        ls.append(
            {
                "id": s[1],
                "Должность": s[0],
                "Результат": r
            }
        )
    for i in ls:
        if i["Результат"] == "Принят":
            print(f"Работник с id {i['id']} принят на работу")
    lock.release()
```

Рисунок 3 – Код задания

```
Работник с id: 393 На должность: руководящие должности, Результат: Не подходит
Работник с id: 902 На должность: руководящие должности, Результат: Необходима стажировка
Работник с id: 613 На должность: руководящие должности, Результат: Не подходит
Работник с id: 433 На должность: специальности, для которых нужно высшее образование, Результат: Не подходит
Работник с id: 723 На должность: специальности, для которых нужно высшее образование, Результат: Принят
Работник с id: 394 На должность: руководящие должности, Результат: Принят
Работник с id 723 принят на работу
Работник с id 394 принят на работу
```

Рисунок 4 – Результат выполнения задания

Задание 2

Для своего индивидуального задания лабораторной работы 2.23 необходимо организовать конфейер, 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.

1. Написал программу для вычисления значений функций.

```
def func_1(x=0.5):
    lock.acquire()
    EPS = 1e-07
    n, s, m, curr = 0, 0, 0, 0
    while True:
        pre = math.pow(x, 2*n) / math.factorial(2*n)
        n += 1
        curr = math.pow(x, 2*n) / math.factorial(2*n)
        if abs(curr - pre) < EPS:
            break
        s += curr
        q.put(s)
    lock.release()


def func_2():
    x = q.get()
    result = (math.exp(x) + math.exp(-x)) / 2
    print(result)
```

Рисунок 5 – Функции для вычисления

```
 "C:/Users/1/Desktop/Алгоритмизация/Лабораторная 2.24/ind_2.py"
1.0078226778257109
```

Рисунок 6 – Результат вычисления

Контрольные вопросы:

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

Lock-объект может находится в двух состояниях: захваченное (заблокированное) и не захваченное (не заблокированное, свободное). После создания он находится в свободном состоянии. Для работы с Lock-объектом используются методы acquire() и release(). Если Lock свободен, то вызов метода acquire() переводит его в заблокированное состояние.

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

В отличии от рассмотренного выше Lock-объекта RLock может освободить только тот поток, который его захватил. Повторный захват потоком уже захваченного RLock-объекта не блокирует его. RLock-объекты поддерживают возможность вложенного захвата, при этом освобождение происходит только после того, как был выполнен release() для внешнего acquire().

3. Как выглядит порядок работы с условными переменными?

На стороне Consumer'a: проверить доступен ли ресурс, если нет, то перейти в режим ожидания с помощью метода wait(), и ожидать оповещение от Producer'a о том, что ресурс готов и с ним можно работать. Метод wait() может быть вызван с таймаутом, по истечении которого поток выйдет из состояния блокировки и продолжит работу.
На стороне Producer'a: произвести работы по подготовке ресурса, после того, как ресурс готов оповестить об этом ожидающие потоки с помощью методов notify() или notify_all(). Разница между ними в том, что notify() разблокирует только один поток (если он вызван без параметров), а notify_all() все потоки, которые находятся в режиме ожидания.

4. Какие методы доступны у объектов условных переменных?

При создании объекта Condition вы можете передать в конструктор объект Lock или RLock, с которым хотите работать. Перечислим методы объекта Condition с кратким описанием:

– acquire(*args) – захват объекта-блокировки.

– release() – освобождение объекта-блокировки.

– wait(timeout=None) – блокировка выполнения потока до оповещения о снятии блокировки. Через параметр timeout можно задать время ожидания оповещения о снятии блокировки. Если вызвать wait() на Условной переменной, у которой предварительно не был вызван acquire(), то будет выброшено исключение RuntimeError.

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

С помощью семафоров удобно управлять доступом к ресурсу, который имеет ограничение на количество одновременных обращений к нему (например, количество подключений к базе данных и т.п.).

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

События по своему назначению и алгоритму работы похожи на рассмотренные ранее условные переменные. Основная задача, которую они решают – это взаимодействие между потоками через механизм оповещения. Объект класса Event управляет внутренним флагом, который сбрасывается с помощью метода clear() и устанавливается методом set(). Потоки, которые используют объект Event для синхронизации блокируются при вызове метода wait(), если флаг сброшен.

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

Модуль threading предоставляет удобный инструмент для запуска задач по таймеру – класс Timer. При создании таймера указывается функция, которая будет выполнена, когда он сработает. Timer реализован как поток, является наследником от Thread, поэтому для его запуска необходимо вызвать start(), если необходимо остановить работу таймера, то вызовите cancel().

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

Он позволяет реализовать алгоритм, когда необходимо дождаться завершения работы группы потоков, прежде чем продолжить выполнение задачи.

Вывод: в ходе работы приобрёл навыки по работе с синхронизацией потоков при написании программ с помощью языка программирования Python версии 3.