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 №10
дисциплины «Анализ данных»

Вариант №2

Выполнила:
Беседина Инга Олеговна
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р. А., канд. технических наук, доцент, 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.

Ход работы

Пример 1. Пример работы с условной переменной.

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


from threading import Condition, Thread
from queue import Queue
from time import sleep


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)


if __name__ == "__main__":
    # Run order processors
    Thread(target=order_processor, args=("thread 1",)).start()
    Thread(target=order_processor, args=("thread 2",)).start()
    Thread(target=order_processor, args=("thread 3",)).start()
```

```
    # Put data into queue
    for i in range(10):
        q.put(f"order {i}")

    # Put stop-commands for consumers
    for _ in range(3):
        q.put("stop")

    # Notify all consumers
    with cv:
        cv.notify_all()
```

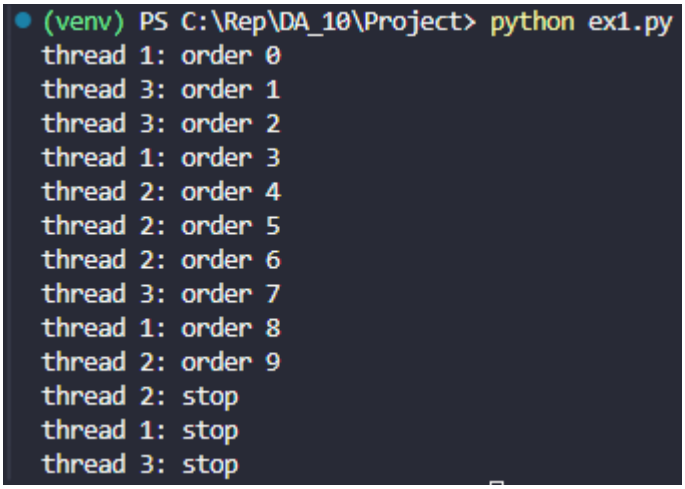

Рисунок 1. Результат выполнения программы

Пример 2. Программа, моделирующую продажу билетов: обслуживание одного клиента занимает одну секунду, касс всего три, клиентов пять.

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


from threading import Thread, BoundedSemaphore
from time import sleep, time

ticket_office = BoundedSemaphore(value=3)


def ticket_buyer(number):
    start_service = time()
    with ticket_office:
```

```
        sleep(1)
        print(f"client {number}, service time: {time() - start_service}")


if __name__ == "__main__":
    buyer = [Thread(target=ticket_buyer, args=(i,)) for i in range(5)]

    for b in buyer:
        b.start()
```

```
(venv) PS C:\Rep\DA_10\Project> python ex2.py
client 0, service time: 1.0156662464141846
client 2, service time: 1.0217087268829346
client 1, service time: 1.0217087268829346
client 3, service time: 2.0238678455352783
client 4, service time: 2.0238678455352783
```

Рисунок 2. Вывод программы

Пример 3. Пример работы с Event-объектом.

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


from threading import Thread, Event
from time import sleep, time


event = Event()


def worker(name: str):
    event.wait()
    print(f"Worker: {name}")


if __name__ == "__main__":
    # Clear event
    event.clear()

    # Create and start workers
    workers = [Thread(target=worker, args=(f"wrk {i}",)) for i in range(5)]
    for w in workers:
        w.start()
```

```
    print("Main thread")
    event.set()
```

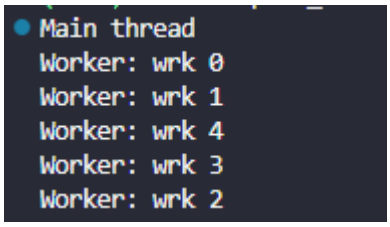

Рисунок 3. Результат работы программы

Пример 4. Пример работы с классом Barrier.

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


from threading import Barrier, Thread
from time import sleep, time


br = Barrier(3)
store = []


def f1(x):
    print("Calc part1")
    store.append(x**2)
    sleep(0.5)
    br.wait()


def f2(x):
    print("Calc part2")
    store.append(x*2)
    sleep(1)
    br.wait()


if __name__ == "__main__":
    Thread(target=f1, args=(3,)).start()
    Thread(target=f2, args=(7,)).start()

    br.wait()
```

```
    print("Result: ", sum(store))
```

```
(venv) PS C:\Rep\DA_10\Project> python ex4.py
Calc part1
Calc part2
Result:  23
```

Рисунок 4. Результат работы программы

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.

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


"""
Вариант 2.
С использованием многопоточности для заданного значения x
найти сумму ряда S с точностью члена ряда по абсолютному значению e=10^-7
и произвести сравнение полученной суммы с контрольным значением функции
для двух бесконечных рядов. 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.
"""


from threading import Lock, Thread


lock = Lock()


def calc_sum1(x, eps, s):
    sum1 = 0
    n = 0
    while True:
        el = x**n
        if abs(el) < eps:
            break
        sum1 += el
        n += 1
```

```
    with lock:
        s["s1"] = sum1


def calc_sum2(x, eps, s):
    sum2 = 0
    n = 0
    while True:
        t = n + 1
        el = (-1) ** n * x**n / 2**t
        if abs(el) < eps:
            break
        else:
            sum2 += el
            n += 1
    with lock:
        s["s2"] = sum2


def res(s, y1, y2):
    while True:
        with lock:
            if "s1" in s and "s2" in s:
                s1 = s["s1"]
                s2 = s["s2"]

                print(f"Вариант №2. Сумма ряда S: {s1}")
                print(
                    f"Контрольное значение функции для бесконечного ряда:
{y1}")

                print(f"Вариант №3. Сумма ряда S: {s2}")
                print(
                    f"Контрольное значение функции для бесконечного ряда:
{y2}")
                break


def main():
    s = {}
    e = 1e-7

    x1 = 0.7
    y1 = 1 / (1 - x1)
```

```
    x2 = 1.2
    y2 = 1 / (2 + x2)

    thread1 = Thread(target=calc_sum1, args=(x1, e, s))
    thread2 = Thread(target=calc_sum2, args=(x2, e, s))
    thread3 = Thread(target=res, args=(s, y1, y2))

    thread1.start()
    thread2.start()
    thread3.start()


if __name__ == "__main__":
    main()
```

```
Вариант №2. Сумма ряда S: 3.333333083650555
Контрольное значение функции для бесконечного ряда: 3.333333333333333
Вариант №3. Сумма ряда S: 0.31250004145136007
Контрольное значение функции для бесконечного ряда: 0.3125
```

Рисунок 5. Результат работы программы

Контрольные вопросы:

1. Lock-объект:

- Назначение: Обеспечение многопоточной безопасности путем блокировки доступа к общим ресурсам. Когда поток блокирует объект Lock, другие потоки должны ожидать, пока первый поток не освободит блокировку.

- Приемы работы:

- Получение блокировки с помощью метода acquire().

- Освобождение блокировки с помощью метода release().

2. RLock-объект:

- Отличие: RLock (рекурсивный Lock) позволяет потоку многократно захватывать блокировку, что полезно в некоторых сценариях, где один и тот же поток может захватывать блокировку несколько раз.

3. Условные переменные:

- Порядок работы:

- Создание объекта условной переменной с помощью threading.Condition().

- Ожидание условия с помощью метода wait().

- Уведомление о событии с помощью метода notify() или notify_all().

4. Методы доступные у объектов условных переменных:

- wait(): Поток ожидает, пока не будет получен сигнал или уведомление.

- notify(n=1): Посылает сигнал одному из потоков, ожидающих на этой условной переменной.

- notify_all(): Посылает сигнал всем потокам, ожидающим на этой условной переменной.

5. Семафор:

- Назначение: Семафор используется для управления доступом к общему ресурсу. Он предоставляет фиксированное количество разрешений на доступ к ресурсу. - Порядок работы:

- Создание объекта семафора с помощью threading.Semaphore(value).

- Запрос разрешения на доступ к ресурсу с помощью acquire().

- Освобождение разрешения с помощью release().

6. Событие:

- Назначение: Событие представляет собой флаг, который потоки могут устанавливать или сбрасывать. Остальные потоки могут ожидать, когда событие будет установлено.

- Порядок работы:

- Создание объекта события с помощью threading.Event().

- Установка события с помощью set().

- Сброс события с помощью clear().

- Ожидание события с помощью wait().

7. Таймер:

- Назначение: Таймер используется для запуска функции через определенный интервал времени.

- Порядок работы:

- Создание объекта таймера с помощью threading.Timer(interval, function).

- Запуск таймера с помощью start().

8. Барьер:

- Назначение: Барьер позволяет нескольким потокам встречаться в одной точке выполнения перед продолжением работы.

- Порядок работы:

- Создание объекта барьера с помощью threading.Barrier(parties).

- Ожидание всех потоков на барьере с помощью wait().

9. Общий вывод:

- Выбор примитива синхронизации зависит от конкретной задачи. Например, Lock и RLock используются для обеспечения безопасного доступа к общим ресурсам. Условные переменные удобны для координации между потоками. Семафоры могут использоваться для управления доступом к ресурсам определенным количеством потоков. События полезны для уведомления о состоянии или завершении задачи. Таймеры могут использоваться для планирования выполнения функций в будущем. Барьеры позволяют координировать выполнение нескольких потоков в определенной точке программы.

Вывод: В ходе выполнения лабораторной работы были приобретены навыки использования примитивов синхронизации в языке программирования Python версии 3.x.