Институт цифрового развития
Кафедра инфокоммуникаций

«Синхронизация потоков в языке программирования Python»

ОТЧЕТ
по лабораторной работе №24
дисциплины
«Основы программной инженерии»

Выполнил:
Мизин Глеб Егорович
2 курс, группа ПИЖ-б-о-21-1,
011.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

_____
(подпись)

Проверил:
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

```python
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
```

Рисунок 1 – Пример 1

```python
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()

```

Рисунок 2 – Пример 2

```python
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()
```

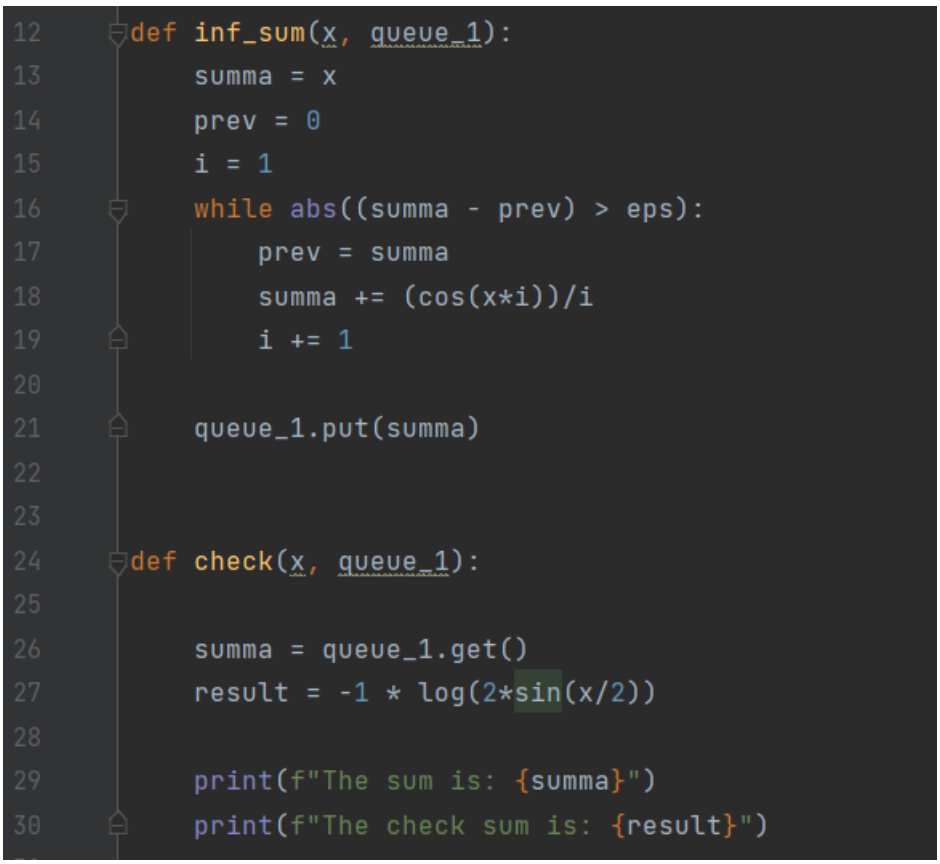
Рисунок 3 – Пример 3

```python
from threading import Timer
from time import sleep, time

timer = Timer(interval=3, function=lambda: print("Message from Timer!"))
timer.start()

```

Рисунок 4 – Пример 4

```python
# Произведем умножение матриц стандартным методом и запишем в out
def original_dot_func(a, b, out):
    out[:] = np.dot(a, b)


# Умножаем матрицы a и b, используя nblocks и mblocks потоков.
def parallel_dot(a, b, nblocks, mblocks, original_dot_func):
    n_jobs = nblocks * mblocks
    print(f'Running {n_jobs} jobs in parallel')

    out = np.empty((a.shape[0], b.shape[1]), dtype=a.dtype)

    # Разбиваем матрицы на блоки функцией blockshaped
    out_blocks = blockshaped(out, nblocks, mblocks)
    a_blocks = blockshaped(a, nblocks, 1)
    b_blocks = blockshaped(b, 1, mblocks)

    """Умножаем каждый блок функцией original_dot
```

Рисунок 5 – Индивидуальное задание №1



```
def inf_sum(x, queue_1):
    summa = x
    prev = 0
    i = 1
    while abs((summa - prev) > eps):
        prev = summa
        summa += (cos(x*i))/i
        i += 1

    queue_1.put(summa)


def check(x, queue_1):

    summa = queue_1.get()
    result = -1 * log(2*sin(x/2))

    print(f"The sum is: {summa}")
    print(f"The check sum is: {result}")
```

Рисунок 6 – Индивидуальное задание №2

Контрольные вопросы

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

Lock-объект используется для синхронизации доступа к общим ресурсам из нескольких потоков. Приемы работы с Lock-объектом включают вызов метода acquire() для получения блокировки и release() для освобождения блокировки.

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

RLock-объект представляет собой рекурсивную блокировку и может быть захвачен несколько раз одним и тем же потоком. В отличие от Lock-объекта, приемы работы с RLock-объектом включают вызов метода acquire() и release() в тех же потоках в любом количестве.

3. Как выглядит порядок работы с условными переменными?

Для работы с условными переменными необходимо создать объект класса Condition. Затем потоки могут вызывать методы wait(), notify() и notify_all(), чтобы ожидать определенного условия и уведомлять другие потоки о его выполнении.

4. Какие методы доступны у объектов условных переменных?

У объектов условных переменных доступны методы wait(), notify() и notify_all().

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

Семафор используется для синхронизации доступа к ограниченному количеству ресурсов. Для создания семафора в Python используется класс Semaphore. При получении блокировки при помощи метода acquire() счетчик

семафора уменьшается, а при освобождении блокировки методом release() увеличивается.

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

Событие используется для синхронизации потоков на выполнении какого-то события, например, завершении задания. В Python для этого используется класс Event. Потоки могут ждать на выполнение события при помощи метода wait(), а событие можно установить методом set() и снять методом clear().

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

Таймер используется для выполнения задач через определенный промежуток времени. В Python для этого можно использовать класс Timer. При создании объекта таймера необходимо указать интервал времени, после которого выполнится задача. Путем вызова метода start() таймер запускается, а метод cancel() останавливает выполнение задачи.

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

Барьер используется для ожидания завершения выполнения задач несколькими потоками. Для этого в Python используется класс Barrier. Создается объект барьера с указанием количества потоков и методами wait() для ожидания завершения задач и reset() для возврата барьера в исходное состояние.

9. Сделайте общий вывод о применении тех или иных примитивов синхронизации в зависимости от решаемой задачи.

Выбор конкретного примитива синхронизации зависит от решаемой задачи. Lock-объекты и RLock-объекты используются для синхронизации

доступа к общим ресурсам, семафоры - для синхронизации ограниченного количества доступа к ресурсам, события - для уведомления потоков об определенных событиях, таймеры - для отложенного выполнения задач, а барьеры - для синхронизации одновременного завершения работы нескольких потоков.