

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

**ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №8
дисциплины
«Объектно-ориентированное программирование»
Вариант № 3**

Выполнил:
Левашев Тимур Рашидович
3 курс, группа ИВТ-б-о-23-2,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

(подпись)

Проверил:
Доцент департамента цифровых,
робототехнических систем и
электроники института перспективной
инженерии Воронкин Р.А

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2025 г.

Тема работы: “Синхронизация потоков в языке программирования python.

Цель работы: приобретение навыков использования примитивов синхронизации в языке программирования Python версии 3.x.

Порядок выполнения работы:

Ссылка на Git репозиторий: https://github.com/mazy99/oop_pract_8

1. Пример 1: Использование условных переменных (threading.Condition).

Листинг программы:

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

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

cv = Condition()
q = Queue()

def order_processor(name):
    while True:
        with cv:
            while q.empty():
                cv.wait()
            try:
                order = q.get_nowait()
                print(f"{name}: {order}")
                if order == "stop":
                    break
            except:
                pass
            sleep(0.1)

if __name__ == "__main__":
    Thread(target = order_processor, args=("thread 1",)).start()
    Thread(target = order_processor, args=("thread 2",)).start()
    Thread(target = order_processor, args=("thread 3",)).start()

    for i in range(10):
```

```
q.put(f"order {i}")

for _ in range(3):
    q.put("stop")

with cv:
    cv.notify_all()
```

```
thread 2: order 0
thread 3: order 1
thread 1: order 2
thread 2: order 3
thread 3: order 4
thread 1: order 5
thread 2: order 6
thread 3: order 7
thread 1: order 8
thread 2: order 9
thread 3: order 4
thread 1: order 5
thread 2: order 6
thread 3: order 7
thread 1: order 8
thread 2: order 9
thread 3: stop
thread 1: stop
thread 2: stop
```

Рисунок 1 – Результат вызова программы

2. Пример 2: Использование семафоров (threading.Semaphore).

Листинг программы:

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

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

ticket_office: BoundedSemaphore = BoundedSemaphore(value=3)

def ticket_buyer(number: int) -> None:
    start_service = time()
    with ticket_office:
        sleep(0.1)
```

```

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

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

    for b in buyer:
        b.start()

```

```

client 0, service time: 0.10026264190673828
client 1, service time: 0.10054540634155273
client 2, service time: 0.10051655769348145
client 3, service time: 0.20070385932922363
client 4, service time: 0.20106220245361328

```

Рисунок 2 – Результат выполнения программы

3. Пример 3: Использование событий (threading. Event).

Листинг программы:

```

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

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

event = Event()

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

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

    worker = [Thread(target=worker, args=(f"wrk {i}",)) for i in range(5)]

    for w in worker:
        w.start()

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

```

```
Main thread
Worker: wrk 0
Worker: wrk 1
Worker: wrk 3
Worker: wrk 2
Worker: wrk 4
```

Рисунок 3 – Результат выполнения программы

4. Пример 4: Использование Барьера threading. Barrier.

Листинг программы:

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

from time import sleep
from threading import Barrier, Thread

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(f"Result: {sum(store)}")
```

```
Calc part1
Calc part2
Result: 23
```

Рисунок 4 – Результат выполнения программы

5. Индивидуальное задание: с использованием многопоточности для заданного значения x необходимо организовать конвейер, в котором сначала в отдельном потоке вычисляется значение первой функции, после чего результат вычисления должны передаваться второй функции, вычисленной в отдельном потоке. Потоки для вычисления значений двух функций должны запускаться одновременно. Значения сумм рядов должны быть вычислены с точностью $\varepsilon = 10^{-7}$.

$$S = \sum_{n=0}^{\infty} (-1)^n \frac{x^n}{2^{n+1}} = \frac{1}{2} - \frac{x}{4} + \frac{x^2}{8} - \dots;$$

$$x = 1, 2; y = \frac{1}{2+x}.$$

Рисунок 5 – Исходный ряд, значение переменной x , функция приблизительно описывающая сумму ряда

Листинг программы:

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

from threading import Thread, Event

class SeriesPipeline:

    def __init__(self, x: float = 1.2, eps: float = 10**-7) -> None:
        self.x: float = x
        self.eps : float = eps

        self.series_result: float | None = None
        self.final_result: float | None = None

        self.ready_event: Event = Event()

    def _term(self, n:int) -> float:
        return ((-1) ** n) * (self.x**n) / (2 ** (n + 1))

    def _first_worker(self, index: int) -> None:
        print(f"[Thread {index}] Начало вычисления ряда")
```

```

s: float = 0.0
n: int = 0

while True:
    a_n = self._term(n)
    if abs(a_n) < self.eps:
        break
    s += a_n
    n += 1

self.series_result = s
print(f"[Thread-1] Сумма ряда = {s}")
self.ready_event.set()

def _second_function(self) -> float:
    return 1 / (2 + self.x)

def _second_worker(self, index: int) -> None:
    print(f"[Thread {index}] Ожидание результата первой функции")
    self.ready_event.wait()

    assert self.series_result is not None
    self.final_result = self._second_function()

    print(f"[Thread {index}] Результат второй функции = {self.final_result}")

def run(self) -> None:
    t1: Thread = Thread(target=self._first_worker, args=(1,))
    t2: Thread = Thread(target=self._second_worker, args=(2,))

    t1.start()
    t2.start()

    t1.join()
    t2.join()

def __str__(self) -> str:
    return (
        "Ряд:\n"
        "S = Σ [(-1)^n * x^n / 2^(n+1)], n = 0 .. ∞\n\n"
        f"x = {self.x}\n"
        f"epsilon = {self.eps}\n\n"
        "Аналитическое выражение:\n"
        "S = 1 / (2 + x)"
    )

```

Листинг вызова программы:

```

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

from pipeline_sum_series import SeriesPipeline

if __name__ == "__main__":
    pipeline = SeriesPipeline(x=1.2)
    print(pipeline)

    pipeline.run()

    print("\n--- Итог ---")
    print(f"Сумма ряда      = {pipeline.series_result}")
    print(f"Аналитическое  = {pipeline.final_result}")
    print(f"Абсолютная погрешность  = {pipeline.series_result - pipeline.final_result}")

```

Ряд:

$S = \sum [(-1)^n * x^n / 2^{(n+1)}], n = 0 \dots \infty$

$x = 1.2$

$\text{epsilon} = 1e-07$

Аналитическое выражение:

$S = 1 / (2 + x)$

[Thread 1] Начало вычисления ряда

[Thread-1] Сумма ряда = 0.31250004145136007

[Thread 2] Ожидание результата первой функции

[Thread 2] Результат второй функции = 0.3125

--- Итог ---

Сумма ряда = 0.31250004145136007

Аналитическое = 0.3125

Абсолютная погрешность = 4.145136006661332e-08 □

Рисунок 6 – Результат выполнения программы

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