

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ**

**Отчет о лабораторной работе №12 по дисциплине основы программной
инженерии**

Выполнил:

Выходцев Егор Дмитриевич,
2 курс, группа ПИЖ-б-о-20-1,

Проверил:

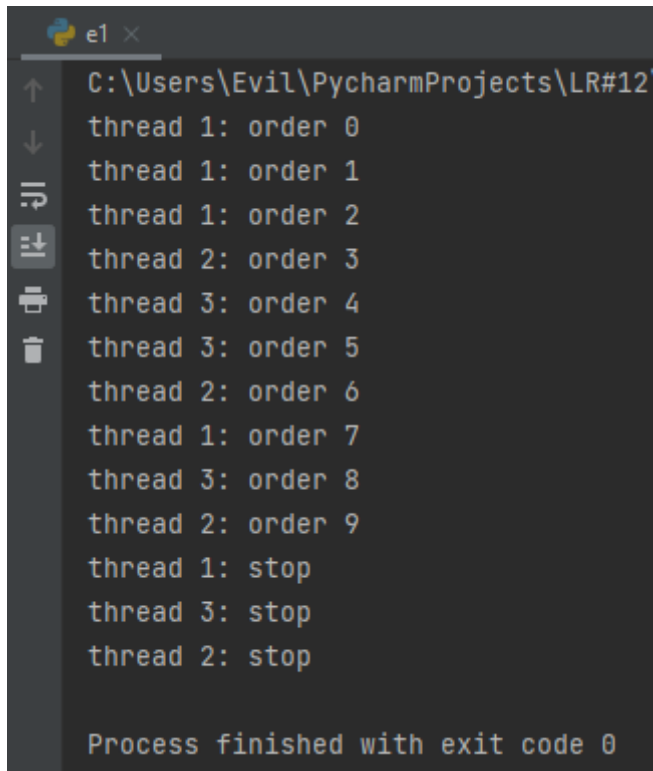
Доцент кафедры инфокоммуникаций,
Воронкин Р.А.

Ставрополь, 2022 г

1. Примеры из методических указаний

```
e1.py x
1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4      from threading import Condition, Thread
5          from queue import Queue
6      from time import sleep
7
8      cv = Condition()
9      q = Queue()
10
11
12      # Consumer function for order processing
13  def order_processor(name):
14      while True:
15          with cv:
16              # Wait while queue is empty
17              while q.empty():
18                  cv.wait()
19              try:
20                  # Get data (order) from queue
21                  order = q.get_nowait()
22                  print(f"{name}: {order}")
23                  # If get "stop" message then stop thread
24                  if order == "stop":
25                      break
26          except:
27              pass
28          sleep(0.1)
29
30
31  ▶  if __name__ == "__main__":
32      # Run order processors
33      Thread(target=order_processor, args=("thread 1",)).start()
34      Thread(target=order_processor, args=("thread 2",)).start()
35      Thread(target=order_processor, args=("thread 3",)).start()
36      # Put data into queue
37      for i in range(10):
38          q.put(f"order {i}")
39      # Put stop-commands for consumers
40      for _ in range(3):
41          q.put("stop")
42
43
order_processor() > while True > with cv > except
```

```
# 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()
```



e1 x

C:\Users\Evil\PycharmProjects\LR#12

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

Process finished with exit code 0

```
e1.py x e2.py x
1 ▶ #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 from threading import Thread, BoundedSemaphore
6 from time import sleep, time
7
8 ticket_office = BoundedSemaphore(value=3)
9
10
11 def ticket_buyer(number):
12     start_service = time()
13     with ticket_office:
14         sleep(1)
15         print(f"client {number}, service time: {time() - start_service}")
16
17
18 ▶ if __name__ == "__main__":
19     buyer = [Thread(target=ticket_buyer, args=(i,)) for i in range(5)]
20     for b in buyer:
21         b.start()
22
```

```
e2 x
↑ C:\Users\Evil\PycharmProjects\LR#12\venv\Scripts\python.exe C:/Users/Evil/Pycharm
↓ client 1, service time: 1.000920057296753client 0, service time: 1.00092005729675
|| client 2, service time: 1.001873254776001
|| client 4, service time: 2.00083065032959client 3, service time: 2.00083065032959
||
|| Process finished with exit code 0
```

```
e1.py x e2.py x e3.py x
1 ▶ #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 from threading import Thread, Event
6 from time import sleep, time
7
8
9 event = Event()
10
11
12 def worker(name: str):
13     event.wait()
14     print(f"Worker: {name}")
15
16
17 ▶ if __name__ == "__main__":
18     # Clear event
19     event.clear()
20     # Create and start workers
21     workers = [Thread(target=worker, args=(f"wrk {i}",)) for i in range(5)]
22     for w in workers:
23         w.start()
24
25     print("Main thread")
26     event.set()
27
```

```
e3 x
↑ C:\Users\Evil\PycharmProjects\LR#12\venv\Scripts\python.exe C:/Use
↓ Main thread
Worker: wrk 0Worker: wrk 2Worker: wrk 1Worker: wrk 4Worker: wrk 3
```

```
e1.py × e2.py × e3.py × e4.py ×
1 ▶ #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 from threading import Timer
6
7
8 ▶ if __name__ == "__main__":
9     timer = Timer(interval=3, function=lambda: print("Message from Timer!"))
10     timer.start()
11
```

```
e4 ×
↑ C:\Users\Evil\PycharmProjects\LR#1
↓ Message from Timer!
⏮
⏭ Process finished with exit code 0
```

```
e1.py x e2.py x e3.py x e4.py x e5.py x
1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4
5      from threading import Barrier, Thread
6      from time import sleep
7
8
9      br = Barrier(3)
10     store = []
11
12
13     def f1(x):
14         print("Calc part1")
15         store.append(x**2)
16         sleep(0.5)
17         br.wait()
18
19
20     def f2(x):
21         print("Calc part2")
22         store.append(x*2)
23         sleep(1)
24         br.wait()
25
26
27     if __name__ == "__main__":
28         Thread(target=f1, args=(3,)).start()
29         Thread(target=f2, args=(7,)).start()
30         br.wait()
31         print("Result: ", sum(store))
32
```

```
C:\Users\Evil\PycharmProjects\LR#12
Calc part1
Calc part2
Result:  23

Process finished with exit code 0
```

2. Индивидуальное задание №1 (рис. 1-3).

Задача: использовать многопоточность для ускорения операции произведения матриц.

```
idz1.py x
1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4
5      import numpy as np
6          from numpy.testing import assert_array_equal
7          from threading import Thread
8      from time import time
9
10
11     def blockshaped(arr, nrows, ncols):
12         h, w = arr.shape
13         n, m = h // nrows, w // ncols
14         return arr.reshape(nrows, n, ncols, m).swapaxes(1, 2)
15
16
17     def original_dot(a, b, out):
18         out[:] = np.dot(a, b)
19
20
21     def parallel_dot(a, b, nblocks, mblocks, dot_func=original_dot):
22         n_jobs = nblocks * mblocks
23         print(f'Running {n_jobs} jobs in parallel')
24
25         out = np.empty((a.shape[0], b.shape[1]), dtype=a.dtype)
26
27         out_blocks = blockshaped(out, nblocks, mblocks)
28         a_blocks = blockshaped(a, nblocks, 1)
29         b_blocks = blockshaped(b, 1, mblocks)
30
31         threads = []
32         for i in range(nblocks):
33             for j in range(mblocks):
34                 th = Thread(target=dot_func,
35                             args=(a_blocks[i, 0, :, :],
36                                   b_blocks[0, j, :, :],
37                                   out_blocks[i, j, :, :])
38                               )
39                 th.start()
```

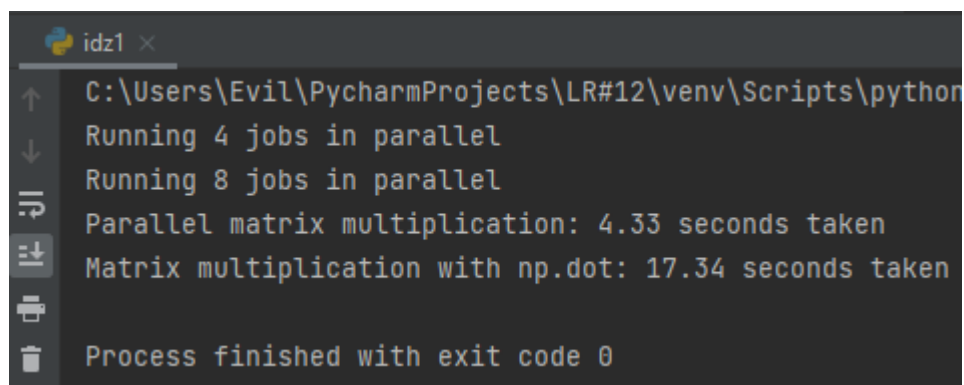
Рисунок 1 – Код программы

```

39         th.start()
40         threads.append(th)
41
42     for th in threads:
43         th.join()
44
45     return out
46
47
48 ▶ if __name__ == '__main__':
49     a = np.ones((4, 3), dtype=int)
50     b = np.arange(18, dtype=int).reshape(3, 6)
51     assert_array_equal(parallel_dot(a, b, 2, 2), np.dot(a, b))
52
53     a = np.random.randn(1500, 1500).astype(int)
54
55     start = time()
56     parallel_dot(a, a, 2, 4)
57     time_par = time() - start
58     print('Parallel matrix multiplication: {:.2f} seconds taken'
59           .format(time_par)
60           )
61
62     start = time()
63     np.dot(a, a)
64     time_dot = time() - start
65     print('Matrix multiplication with np.dot: {:.2f} seconds taken'
66           .format(time_dot)
67           )
68

```

Рисунок 2 – Код программы, продолжение



```

idz1 x
C:\Users\Evil\PycharmProjects\LR#12\venv\Scripts\python
Running 4 jobs in parallel
Running 8 jobs in parallel
Parallel matrix multiplication: 4.33 seconds taken
Matrix multiplication with np.dot: 17.34 seconds taken
Process finished with exit code 0

```

Рисунок 3 – Результат работы программы

3. Индивидуальное задание №2

```
idz2.py x
1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4
5      from threading import Thread, Lock
6      from math import cos
7      from queue import Queue
8
9
10     eps = .0000001
11     q = Queue()
12     lock = Lock()
13
14
15     def inf_sum(x):
16         lock.acquire()
17         a = 1
18         summa = 1
19         i = 1
20         prev = 0
21         while abs(summa - prev) > eps:
22             a = a * x ** 2 / ((2 * i) * (2 * i - 1))
23             prev = summa
24             if i % 2 == 0:
25                 summa += a
26             else:
27                 summa += -1 * a
28             i += 1
29         q.put(summa)
30         lock.release()
31
32
33     def check_ans(inf_res, d_res):
34         print(f"The sum of an infinite series is: {inf_res}")
35         print(f"The calculated answer is: {d_res}")
36
37
38     ▶  if __name__ == '__main__':
39         num = int(input("Enter the number to calculate: "))
40
41         check_ans()
```

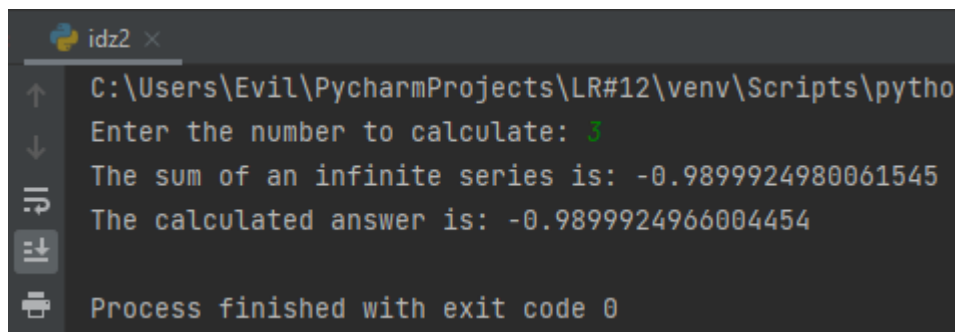
Рисунок 4 – Код программы

```

39     num = int(input("Enter the number to calculate: "))
40     check = cos(num)
41     thread1 = Thread(target=inf_sum, args=(num,)).start()
42     thread2 = Thread(target=check_ans, args=(q.get(), check)).start()
43

```

Рисунок 5 – Код программы, продолжение

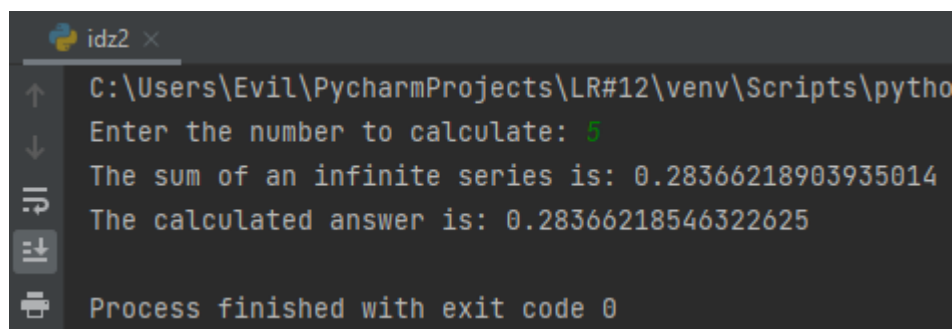


```

idz2 x
C:\Users\Evil\PycharmProjects\LR#12\venv\Scripts\python
Enter the number to calculate: 3
The sum of an infinite series is: -0.9899924980061545
The calculated answer is: -0.9899924966004454
Process finished with exit code 0

```

Рисунок 6 – Результат работы программы для числа 3



```

idz2 x
C:\Users\Evil\PycharmProjects\LR#12\venv\Scripts\python
Enter the number to calculate: 5
The sum of an infinite series is: 0.28366218903935014
The calculated answer is: 0.28366218546322625
Process finished with exit code 0

```

Рисунок 7 – Результат работы программы для числа 5

4. Ответы на контрольные вопросы

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

Lock-объект может находиться 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. Повторный

вызов `acquire()` приведет к блокировке инициировавшего это действие потока до тех пор, пока Lock не будет разблокирован каким-то другим

потоком с помощью метода `release()`. Вывоз метода `release()` на свободном Lock-объекте приведет к вы抛су исключения `RuntimeError`.

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

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()`, у RLock нет метода `locked()`.

3. Как выглядит порядок работы с условными переменными?

Порядок работы с условными переменными выглядит так:

1. 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.

2. 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?

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`.

`wait_for(predicate, timeout=None)` – метод позволяет сократить количество кода, которое нужно написать для контроля готовности ресурса и ожидания оповещения.

`notify(n=1)` – снимает блокировку с остановленного методом `wait()` потока. Если необходимо разблокировать несколько потоков, то для этого следует передать их количество через аргумент `n`.

`notify_all()` – снимает блокировку со всех остановленных методом `wait()` потоков.

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

Суть его идеи заключается в том, при каждом вызове метода `acquire()` происходит уменьшение счетчика семафора на единицу, а при вызове `release()` – увеличение. Значение счетчика не может быть меньше нуля, если на момент вызова `acquire()` его значение равно нулю, то происходит блокировка потока до тех пор, пока не будет вызван `release()`. Семафоры поддерживают протокол менеджера контекста.

Для работы с семафорами в Python есть класс `Semaphore`, при создании его объекта можно указать начальное значение счетчика через параметр `value`. `Semaphore` предоставляет два метода:

`acquire(blocking=True, timeout=None)` – если значение внутреннего счетчика больше нуля, то счетчик уменьшается на единицу и метод возвращает `True`. Если значение счетчика равно нулю, то вызвавший данный метод поток блокируется, до тех пор, пока не будет кем-то вызван метод `release()`. Дополнительно при вызове метода можно указать параметры

`blocking` и `timeout`, их назначение совпадает с `acquire()` для `Lock`.

`release()` – увеличивает значение внутреннего счетчика на единицу.

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

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.

Методы класса `Event`:

`is_set()` – возвращает `True` если флаг находится в взведенном состоянии.

`set()` – переводит флаг в взведенное состояние.

`clear()` – переводит флаг в сброшенное состояние.

`wait(timeout=None)` – блокирует вызвавший данный метод поток если флаг соответствующего `Event`-объекта находится в сброшенном состоянии. Время нахождения в состоянии блокировки можно задать через параметр `timeout`.

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

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()`.

Конструктор класса `Timer`:

```
Timer(interval, function, args=None, kwargs=None)
```

Параметры:

`interval` – количество секунд, по истечении которых будет вызвана функция `function`.

`function` – функция, вызов которой нужно осуществить по таймеру.

`args, kwargs` – аргументы функции `function`.

Методы класса `Timer`:

`cancel()` – останавливает выполнение таймера

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.

Конструктор класса:

```
Barrier(parties, action=None, timeout=None)
```

Параметры:

`parties` – количество потоков, которые будут работать в рамках барьера.

`action` – определяет функцию, которая будет вызвана, когда потоки будут освобождены (достигнут барьера).

`timeout` – таймаут, который будет использовать как значение по умолчанию для методов `wait()`.

Свойства и методы класса:

`wait(timeout=None)` – блокирует работу потока до тех пор, пока не будет получено уведомление либо не пройдет время указанное в `timeout`.

`reset()` – переводит `Barrier` в исходное (пустое) состояние. Потокам, ожидающим уведомления, будет передано исключение `BrokenBarrierError`.

`abort()` – останавливает работу барьера, переводит его в состояние “разрушен” (`broken`). Все текущие и последующие вызовы метода `wait()` будут завершены с ошибкой с выбросом исключения `BrokenBarrierError`.

`parties` – количество потоков, которое нужно для достижения барьера.

`n_waiting` – количество потоков, которое ожидает срабатывания барьера.

`broken` – значение флага равное `True` указывает на то, что барьер находится в “разрушенном” состоянии.

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

Для решения определенного вида задач удобным будет каждый из способов, в зависимости от условий задачи, универсального способа нет.