ФЕДЕРАЛЬНОЕ АГЕНСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПУТЕЙ СООБЩЕНИЯ Императора Александра I»

Кафедра «Информационные и вычислительные системы»

Дисциплина «Программирование на языках высокого уровня (Python)»

ОТЧЁТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 7

Выполнил студент Шефнер А.
Факультет: АИТ
Группа: ИВБ-211

Проверил: Баталов Д.И.

Санкт-Петербург

2023

Оценочный лист результатов ЛР № 7

Ф.И.О. студента _____Шефнер Альберт_____

Группа _____ИВБ-211_____

№ п/п	Материалы необходимые для оценки знаний, умений и навыков	Показатель оценивания	Критерии Оценивания	Шкала оценивания	Оценка
1	Лабораторная работа№	Соответствие методике выполнения	Соответствует	7	
			Не соответствует	0	
		Срок выполнения	Выполнена в срок	2	
			Выполнена с опозданием на 2 недели	0	
		оформление	Соответствует требованиям	1 0	
			Не соответствует		
	ИТОГО количество баллов			10	

Доцент кафедры

«Информационные и вычислительные

системы» Баталов Д.И. «__»

_____2023 г.

1. Напишите программу, которая осуществляет чтение данных из файла посредством одного потока и запись этих данных в другом потоке в файл. Названия файлов должны отличаться.

```python
from queue import Queue
from threading import Event, Thread


class InputThread(Thread):
    __event: Event
    __path: str
    __queue: Queue

    def __init__(self, path: str, event: Event, queue: Queue) -> None:
        super().__init__()

        self.__event = event
        self.__path = path
        self.__queue = queue

    def run(self) -> None:
        print("Started reading")
        self.__event.set()
        with open(self.__path, "r") as file:
            for line in file.readlines():
                self.__queue.put(line)
        self.__event.clear()
        print("Done reading")
```

```python
class OutputThread(Thread):
    __event: Event
    __path: str
    __queue: Queue

    def __init__(self, path: str, event: Event, queue: Queue) -> None:
        super().__init__()

        self.__event = event
        self.__path = path
        self.__queue = queue

    def run(self) -> None:
        self.__event.wait()
        print("Started writing")
        with open(self.__path, "w") as file:
            while self.__event.is_set() or not self.__queue.empty():
                while not self.__queue.empty():
                    file.write(self.__queue.get())
        print("Done writing")
```

```python
if __name__ == "__main__":
    input_path = "task01-input.txt"
    output_path = "task01-output.txt"

    event = Event()
    event.clear()
    queue = Queue()

    input_thread = InputThread(input_path, event, queue)
    output_thread = OutputThread(output_path, event, queue)

    input_thread.start()
    output_thread.start()
```

2. Напишите программу, которая осуществляет чтение из файла посредством одного процесса и запись этих данных в другом процессе в файл. Названия файлов должны отличаться.

```python
from multiprocessing import Event, Process, Queue


class InputProcess(Process):
    __event: Event
    __path: str
    __queue: Queue

    def __init__(self, path: str, event: Event, queue: Queue) -> None:
        super().__init__()

        self.__event = event
        self.__path = path
        self.__queue = queue

    def run(self) -> None:
        print("Started reading")
        self.__event.set()
        with open(self.__path, "r") as file:
            for line in file.readlines():
                self.__queue.put(line)
        self.__event.clear()
        print("Done reading")
```

```python
class OutputProcess(Process):
    __event: Event
    __path: str
    __queue: Queue

    def __init__(self, path: str, event: Event, queue: Queue) -> None:
        super().__init__()

        self.__event = event
        self.__path = path
        self.__queue = queue

    def run(self) -> None:
        self.__event.wait()
        print("Started writing")
        with open(self.__path, "w") as file:
            while self.__event.is_set() or not self.__queue.empty():
                while not self.__queue.empty():
                    file.write(self.__queue.get())
        print("Done writing")


if __name__ == "__main__":
    input_path = "task02-input.txt"
    output_path = "task02-output.txt"

    event = Event()
    event.clear()
    queue = Queue()

    input_process = InputProcess(input_path, event, queue)
    output_process = OutputProcess(output_path, event, queue)

    input_process.start()
    output_process.start()
```

3. Напишите программу, которая, используя механизм асинхронного программирования, осуществляет чтение данных из одного файла и их запись в другой файл.

```python
import asyncio

async def read_line(file):
    return file.readline();
```

```python
async def copy_file_async(f_from, f_to):
    with open(f_from, "r") as readfile, open(f_to, 'w') as writefile:
        line = await read_line(readfile)
        while line:
            line_future = read_line(readfile)
            writefile.write(line)
            line = await line_future

asyncio.run(copy_file_async("input.txt", "output.txt"))
```

4. Напишите программу, осуществляющую перемножение двух матриц с использованием потоков и процессов. Каждая строка новой матрицы должна высчитываться в отдельном потоке (процессе).

```python
from collections import namedtuple
from multiprocessing import Process, Queue
import array
from typing import List


Result = namedtuple("Result", "thread row")


def mul_row(
        queue: Queue,
        arr_a: List[array.array],
        arr_b: List[array.array],
        row: int
    ) -> None:
    result = array.array("d")
    for col in range(len(arr_a)):
        s = 0
        for i in range(len(arr_b)):
            s += arr_a[row][i] * arr_b[i][col]
        result.append(s)

    queue.put(Result(row, result))
```

```python
def mat_mul(a, b):
    queue = Queue()
    processes = []

    for i in range(len(a)):
        process = Process(target=mul_row, args=(queue, a, b, i))
        processes.append(process)
        process.start()

    for process in processes:
        process.join()

    results = []
    while not queue.empty():
        results.append(queue.get())

    results.sort(key=lambda r : r.row)
    return list(map(lambda r: r.row, results))

if __name__ == "__main__":
    mat_a = [
        array.array("d", [1, 2]),
        array.array("d", [3, 4]),
        array.array("d", [5, 6]),
    ]

    mat_b = [
        array.array("d", [7, 8, 9]),
        array.array("d", [10, 11, 12]),
    ]

    print(mat_mul(mat_a, mat_b))
```

5. Напишите программу, в которой несколько процессов осуществляют увеличение значения общего для них счетчика.

```python
from threading import Thread, RLock
from time import sleep

class Counter:
    value: int

    def __init__(self) -> None:
        self.value = 0
```

```python
def inc(counter, lock, amount, thread):
    for _ in range(amount):
        with lock:
            print(f"[Thread {thread}: {counter.value}")
            sleep(0.03)
            counter.value += 1

threads = []
counter = Counter()
lock = RLock()
thread_count = 10
for i in range(thread_count):
    thread = Thread(target=inc, args=(counter, lock, 10, i))
    threads.append(thread)
    thread.start()

for thread in threads:
    thread.join()

print(counter.value)
```

6. Напишите программу, реализующую задачу про обедающих философов на основе процессов.

```python
from threading import Lock, Thread, Semaphore
import time


class Philosopher(Thread):
    def __init__(self, semaphore, index, left_fork, right_fork):
        Thread.__init__(self)
        self.semaphore = semaphore
        self.index = index
        self.left_fork = left_fork
        self.right_fork = right_fork

    def run(self):
        for _ in range(10000):
            with self.semaphore:
                self.eat()
            self.think()
```

```
21
22      def eat(self):
23          self.left_fork.acquire()
24          self.right_fork.acquire()
25          print(f"Philosopher {self.index} is eating")
26          time.sleep(0.0001)
27          self.left_fork.release()
28          self.right_fork.release()
29          print(f"Philosopher {self.index} is done eating")
30
31      def think(self):
32          print(f"Philosopher {self.index} is thinking")
33          time.sleep(0.0001)
34
35
36
37  forks = [Lock() for _ in range(5)]
38  semaphore = Semaphore(2)
39
40  philosophers = [
41      Philosopher(semaphore, i,  forks[i], forks[(i + 1) % 5])
42      for i in range(5)
43  ]
44
45  for philosopher in philosophers:
46      philosopher.start()
47
48  for philosopher in philosophers:
49      philosopher.join()
```

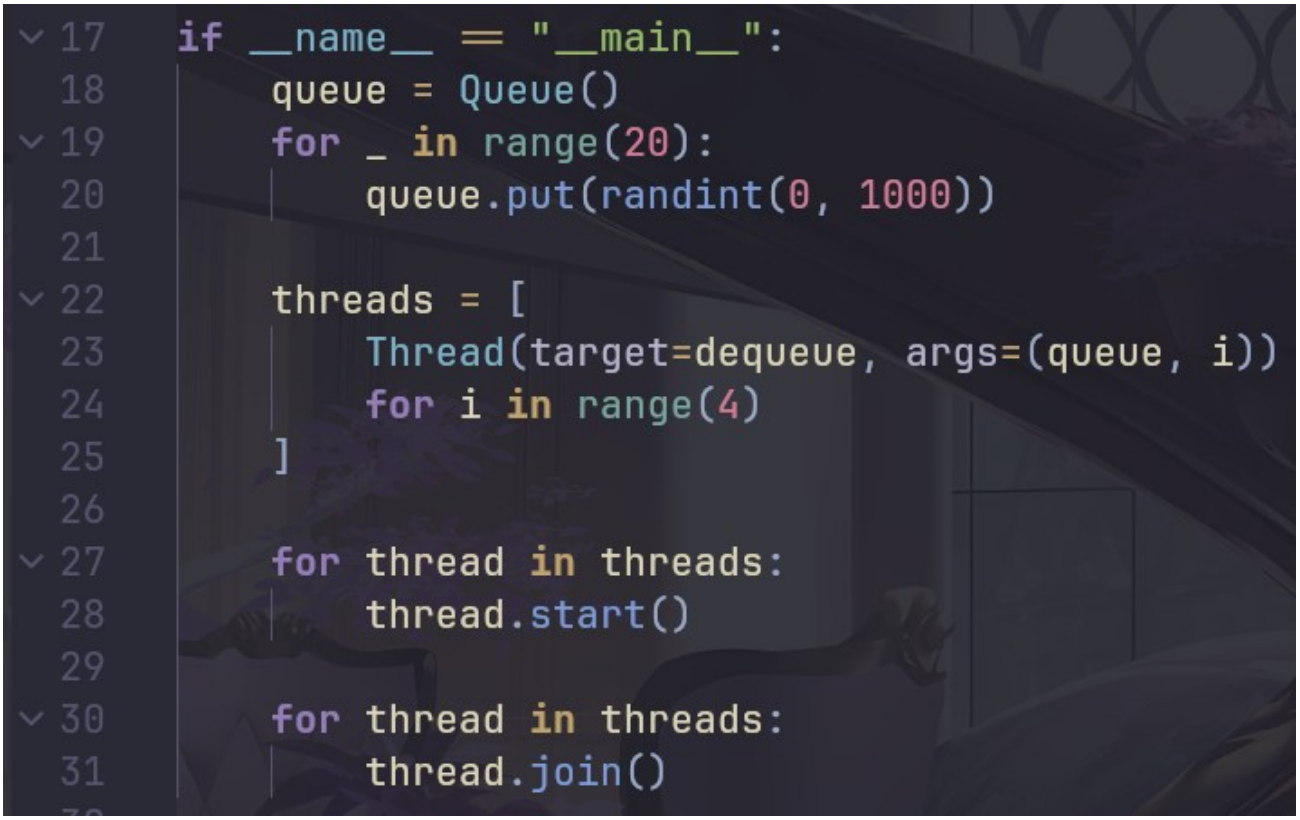
7. Напишите программу, в которой 10 списков заполняются случайными значениями,после чего для каждого из списка в отдельном потоке (процессе) находится медиана.

```
4
5   from collections import namedtuple
6   from multiprocessing import Process, Queue
7   from statistics import median
8   from random import randint
9
10  Result = namedtuple("Result", "median list")
```

```python
def proc_median(queue, lst):
    result = Result(median(lst), lst)
    queue.put(result)

if __name__ == "__main__":
    lists = [
        [randint(0, 100) for _ in range(randint(5, 10))]
        for _ in range(10)
    ]

    processes = []
    queue = Queue()

    for lst in lists:
        process = Process(target=proc_median, args=(queue, lst))
        processes.append(process)
        process.start()

    for process in processes:
        process.join()

    while not queue.empty():
        print(queue.get())
```

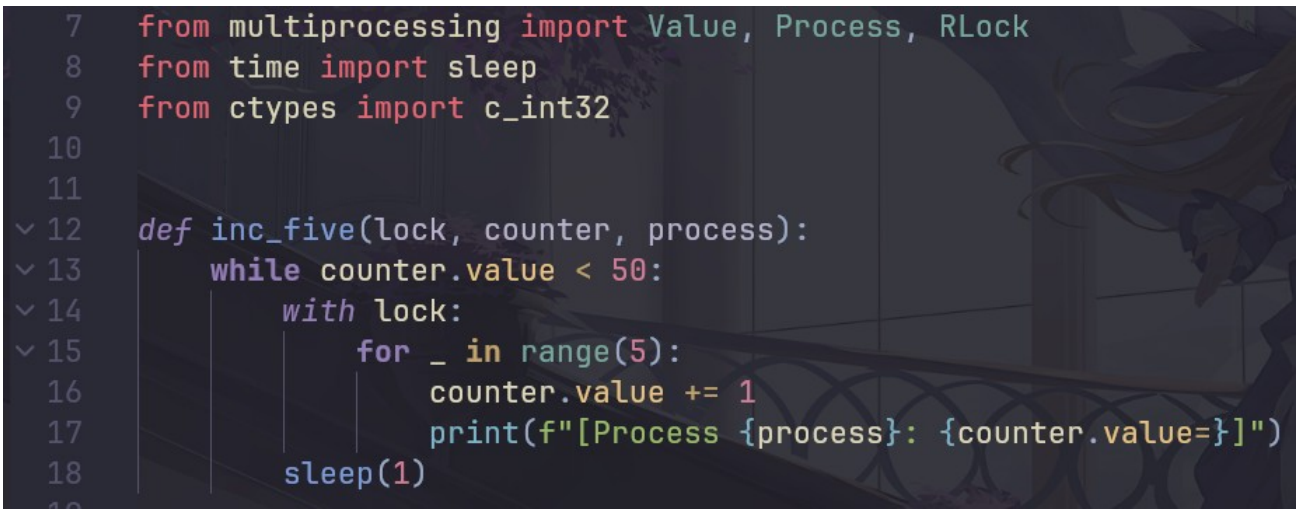
8. Напишите программу, которая заполняет 20 случайными значениями Queue, передаваемую четырем запускаемым потокам (процесса), после чего выведите в консоль извлекаемые из структуры данных значения с информацией о том, в каком потоке (процессе) это произошло.

```python
from threading import Thread
from queue import Queue
from time import sleep
from random import randint


def dequeue(queue: Queue, thread):
    while not queue.empty():
        print(f"[Thread {thread}]: got {queue.get()}")
        sleep(0.1)
```



```python
if __name__ == "__main__":
    queue = Queue()
    for _ in range(20):
        queue.put(randint(0, 1000))

    threads = [
        Thread(target=dequeue, args=(queue, i))
        for i in range(4)
    ]

    for thread in threads:
        thread.start()

    for thread in threads:
        thread.join()
```

9. Напишите программу, которая реализует следующую логику. Имеется общий счетчик для двух потоков (процессов), максимальное значение которого равно 50. Каждый из потоков (процессов) ожидает, пока другой увеличит значение счетчика на 5, после чего тот поток переходит в режим ожидания, а текущий начинает увеличивать значение счетчика.



```python
from multiprocessing import Value, Process, RLock
from time import sleep
from ctypes import c_int32


def inc_five(lock, counter, process):
    while counter.value < 50:
        with lock:
            for _ in range(5):
                counter.value += 1
                print(f"[Process {process}: {counter.value=}]")
        sleep(1)
```

```python
if __name__ == "__main__":
    counter = Value(c_int32, 0)
    lock = RLock()

    processes = [
        Process(target=inc_five, args=(lock, counter, i))
        for i in range(2)
    ]

    for process in processes:
        process.start()

    for process in processes:
        process.join()
```

10. Напишите программу, в которой один поток (процесс) осуществляет чтение данных из файла, а три других потока (процесса) осуществляют их запись в файл. У каждого потока (процесса) свой файл для записи данных. При этом учтите, что файлы, в которые производится запись, не должны содержать в себе один и тот же набор данных, то есть прочитанная строка из файла может быть единожды записана в один из трех файлов.

```python
from threading import Thread, Event
from queue import Queue
from time import sleep


def read_file_proc(file, queue, start_event, end_event):
    print("Started reading file")
    start_event.set()
    for line in file.readlines():
        queue.put(line)
    print("Ended reading file")
    end_event.set()
```

```python
def write_file_proc(process, file, queue, start_event, end_event):
    print(f"[Process {process}] Waiting until file reading")
    start_event.wait()
    print(f"[Process {process}] Started writing")
    while not (end_event.is_set() and queue.empty()):
        if not queue.empty():
            line = queue.get()
            print(f"[Process {process}] Writing '{line[0:-1]}'")
            file.write(line)
            sleep(0.01)
    print(f"[Process {process}] Ended writing")

with open("task10-input.txt", 'r') as inputfile, \
     open("task10-output.txt", 'w') as outputfile:
    start_event = Event()
    start_event.clear()
    end_event = Event()
    end_event.clear()

    queue = Queue()
    read_thread = Thread(
        target=read_file_proc,
        args=(inputfile, queue, start_event, end_event)
    )

    write_thread = [
        Thread(
            target=write_file_proc,
            args=(i, outputfile, queue, start_event, end_event)
        )
        for i in range(3)
    ]

    read_thread.start()
    for thread in write_thread:
        thread.start()

    read_thread.join()
    for thread in write_thread:
        thread.join()
```