Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ***ИУК «Информатика и управление»***

КАФЕДРА ***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

ЛАБОРАТОРНАЯ РАБОТА №5

«Многопоточное программирование на Python»

ДИСЦИПЛИНА: «Перспективные языки программирования»

Выполнил: студент гр. ИУК4-32Б _____ (_____Зудин Д.В._____)
(Подпись) (Ф.И.О.)

Проверил: _____ (___Пчелинцева Н.И.___)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка: 86
- Оценка: хорошо

Калуга, 2022 г.

Цель: формирование практических навыков многопоточного программирования, разработки и отладки программ, овладение методами и средствами разработки.

Задачи:

1. Изучить особенности создания потоков и процессов;
2. Научиться создавать многопоточные программы;
3. Изучить типовые алгоритмы решения задач с использованием принципов многопоточного программирования.

Вариант №1

Формулировка задания №1

Контрольная сумма. Для нескольких файлов (разного размера) требуется вычислить контрольную сумму (сумму кодов всех символов файла). Обработка каждого файла выполняется в отдельном процессе (потоке).

Листинг программы для задания №1

```
import random
import string
import threading
import time


def fill_file_with_random_characters(file_name: str, string_len: int) ->
None:
    with open(file_name, 'w') as f:
        f.write(''.join(random.choice(string.ascii_letters) for i in
range(string_len)))


def get_string_from_file(file_name: str) -> str:
    with open(file_name, 'r') as f:
        data = f.read()
        return data


def get_checksum(file_name: str) -> int:
    with open(file_name, 'r') as f:
        return sum([ord(i) for i in f.read()])


def task(file: str, string_len: int, results: dict[str, int]) -> None:
    fill_file_with_random_characters(file, string_len)
    results[file] = get_checksum(file)


def foo_without_threads(files: list[str], string_len: int) -> None:
    results = dict()
    start_time = time.time()
    for f in files:
        task(f, string_len, results)
```

```
    for f in files:
        print(f'{f}:{results[f]}')
    print(f'Execution time foo1: {time.time() - start_time}s')


def foo_with_threads(files: list[str], string_len: int) -> None:
    results = dict()
    start_time = time.time()
    threads = list()
    for f in files:
        new_thread = threading.Thread(target=task, args=(f, string_len,
results))
        threads.append(new_thread)
        new_thread.start()
    for curr_thread in threads:
        curr_thread.join()
    for f in files:
        print(f'{f}:{results[f]}')
    print(f'Execution time foo2: {time.time() - start_time}s')


def main():
    files = ['data' + str(i) + '.txt' for i in range(20)]
    foo_without_threads(files, 100)
    foo_with_threads(files, 100)


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

Результат выполнения программы №1

```
data0.txt:9264
data1.txt:9449
data2.txt:9292
data3.txt:9433
data4.txt:8993
data5.txt:9209
data6.txt:9427
data7.txt:9399
data8.txt:9184
data9.txt:9251
data10.txt:9148
data11.txt:9278
data12.txt:9228
data13.txt:9215
data14.txt:9201
data15.txt:9283
data16.txt:8943
data17.txt:9346
data18.txt:9112
data19.txt:9369
```

```
Execution time foo2: 0.03017878532409668s
```

Формулировка задания №2

Имеются один или несколько производителей, генерирующих данные некоторого типа (записи, символы и т.п.) и помещающих их в буфер, а также единственный потребитель, который извлекает помещенные в буфер элементы по одному. Требуется защитить систему от перекрытия операций с буфером, т.е. обеспечить, чтобы одновременно получить доступ к буферу мог только один процесс (производитель или потребитель). Решить задачу с помощью условных переменных.

Листинг программы для задания №2

```
import queue
import threading
from threading import Condition, Thread
from queue import Queue
from time import sleep
import random
import string


cond = Condition()
buffer = Queue()


def put_data():
    # Производитель
    global buffer
    new_string = ''.join(random.choice(string.ascii_letters) for _ in
range(5))
    buffer.put(new_string)
    print(threading.current_thread().name, new_string)


def get_data():
    # Потребитель
    global buffer
    stop = False
    while not stop:
        with cond:
            while buffer.empty():
                cond.wait()
            order = buffer.get()
            # print(threading.current_thread().name, order)
            print(order)
            if order == 'stop':
                stop = True


threads = list()
for i in range(10):
    new_thread = Thread(target=put_data, args=())
    threads.append(new_thread)
for i in threads:
```

```
    i.start()

thread_get_data = Thread(target=get_data, args=())
thread_get_data.start()

for i in range(10):
    buffer.put('stop')
with cond:
    cond.notify_all()
```

Результат выполнения программы №2

```
Thread-1 iZbxg
Thread-2 pyGTI
Thread-3 cZEXy
Thread-4 ZCoHU
Thread-5 EeFEj
Thread-6 RFJiA
Thread-7 SjuoN
Thread-8 qMidT
Thread-9 xFmSW
Thread-10 MjSok
iZbxg
pyGTI
cZEXy
ZCoHU
EeFEj
RFJiA
SjuoN
qMidT
xFmSW
MjSok
stop
```

Выводы:

В ходе работы были сформированы практические навыки многопоточного программирования, разработки и отладки программ, овладения методами и средствами разработки.