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. ИУК4-31Б _____ (Суриков Н. С.)
(подпись) (Ф.И.О.)

Проверил: _____ (Осипова О. В.)
(подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2024

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 5

Задача 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 1:

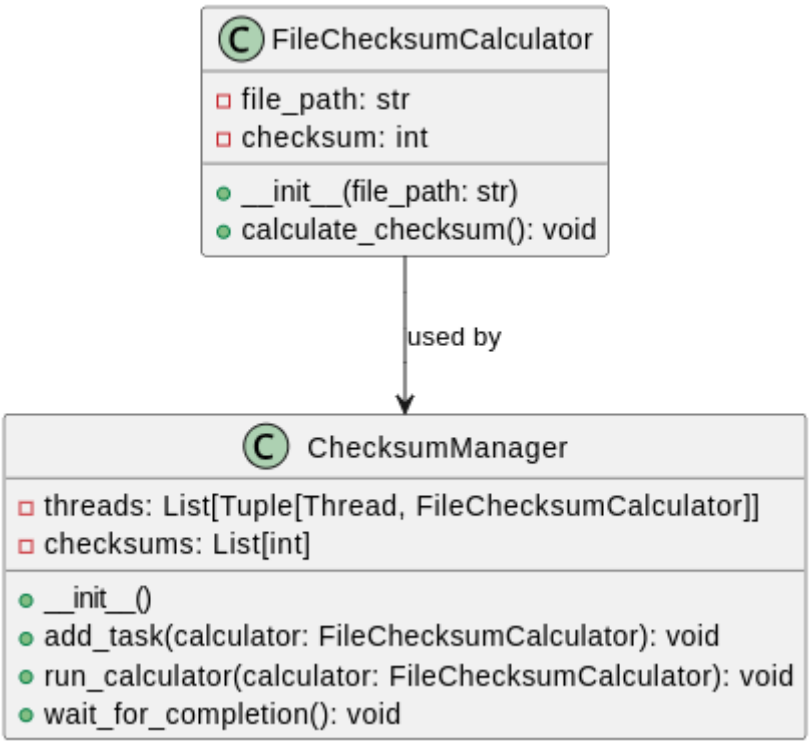
```
import threading


class FileChecksumCalculator:
    def __init__(self, file_path):
        self.file_path = file_path
        self.checksum = 0

    def calculate_checksum(self):
        """Вычисляет контрольную сумму файла."""
        with open(self.file_path, "r", encoding="utf-8") as file:
            for line in file:
                self.checksum += sum(ord(char) for char in line)


class ChecksumManager:
    def __init__(self):
        self.threads = []
        self.checksums = []

    def add_task(self, calculator):
        """Добавляет задачу для выполнения в отдельном потоке."""
```

```
23          thread = threading.Thread(target=self.run_calculator,
args=(calculator,))
24          self.threads.append((thread, calculator))
25          thread.start()
26
27      def run_calculator(self, calculator):
28          """Запускает калькулятор и сохраняет контрольную сумму."""
29          calculator.calculate_checksum()
30          self.checksums.append(calculator.checksum)
31
32      def wait_for_completion(self):
33          """Ожидает завершения всех потоков."""
34          for thread, _ in self.threads:
35              thread.join()
36
37
38  if __name__ == "__main__":
39      files_to_process = ["file1.txt", "file2.txt", "file3.txt"]
40      checksum_manager = ChecksumManager()
41
42      for file_path in files_to_process:
43          calculator = FileChecksumCalculator(file_path)
44          checksum_manager.add_task(calculator)
45
46      checksum_manager.wait_for_completion()
47
48      for i, calculator in enumerate(checksum_manager.threads):
49          print(
50              f"Контрольная сумма для {files_to_process[i]}:
{checksum_manager.checksums[i]}"
51          )
52
```

Результат работы программы 1:

```
Контрольная сумма для file1.txt: 150
Контрольная сумма для file2.txt: 156
Контрольная сумма для file3.txt: 150
```

UML диаграмма программы 1:



Задача 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.

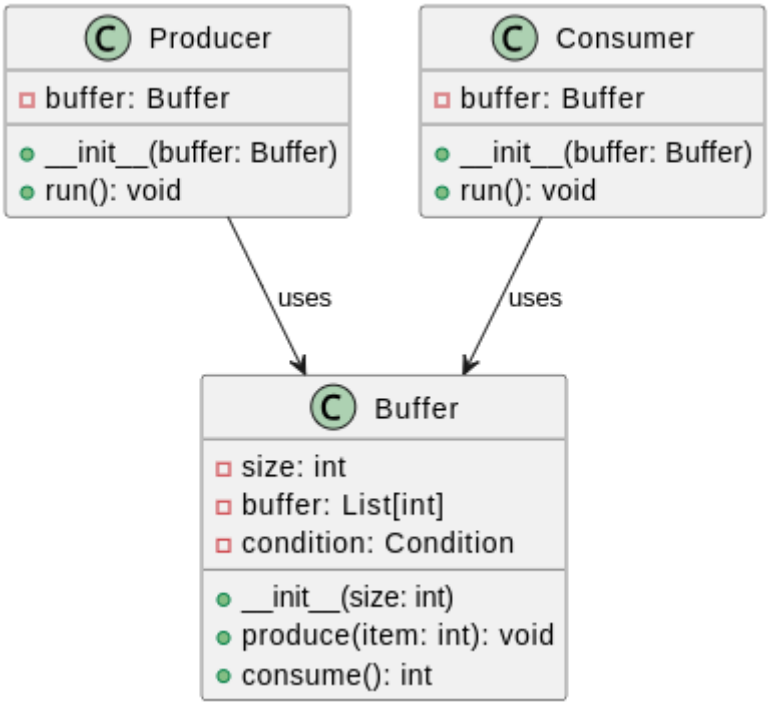
Листинг программы 2:

```
import threading
import time
import random


class Buffer:
    def __init__(self, size):
        self.size = size
        self.buffer = []
        self.condition = threading.Condition()

    def produce(self, item):
        with self.condition:
            while len(self.buffer) >= self.size:
```

```
                print("Буфер полон. Производитель ждет...")
                self.condition.wait()
            self.buffer.append(item)
            print(f"Производитель добавил: {item}")
            self.condition.notify_all()

    def consume(self):
        with self.condition:
            while not self.buffer:
                print("Буфер пуст. Потребитель ждет...")
                self.condition.wait()
            item = self.buffer.pop(0)
            print(f"Потребитель извлек: {item}")
            self.condition.notify_all()
            return item


class Producer(threading.Thread):
    def __init__(self, buffer):
        super().__init__()
        self.buffer = buffer

    def run(self):
        for _ in range(10):
            item = random.randint(1, 100)
            self.buffer.produce(item)
            time.sleep(random.uniform(0.2, 0.6))


class Consumer(threading.Thread):
    def __init__(self, buffer):
        super().__init__()
        self.buffer = buffer

    def run(self):
        for _ in range(30):
            item = self.buffer.consume()
            time.sleep(random.uniform(0.1, 0.5))


if __name__ == "__main__":
    buffer_size = 5
    buffer = Buffer(buffer_size)

    producers = [Producer(buffer) for _ in range(3)]
    consumer = Consumer(buffer)

    for producer in producers:
        producer.start()

    consumer.start()

    for producer in producers:
        producer.join()

```

```
70      consumer.join()
71
72      print("Все операции завершены.")
73
```

UML диаграмма программы 2:



Результат работы программы 2:

```
Производитель добавил: 48
Производитель добавил: 74
Производитель добавил: 72
Потребитель извлек: 48
Производитель добавил: 10
Производитель добавил: 96
Потребитель извлек: 74
Производитель добавил: 20
Производитель добавил: 72
Буфер полон. Производитель ждет...
Буфер полон. Производитель ждет...
Потребитель извлек: 72
Производитель добавил: 41
Буфер полон. Производитель ждет...
Потребитель извлек: 10
Производитель добавил: 40
Потребитель извлек: 96
Потребитель извлек: 20
Потребитель извлек: 72
Потребитель извлек: 41
Потребитель извлек: 40
Все операции завершены.
```

Вывод: в ходе работы были сформированы практические навыки использования многопоточного программирования, разработки и отладки программ.