|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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: |

**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:**

1 import threading

2

3

4 class FileChecksumCalculator:

5 def \_\_init\_\_(self, file\_path):

6 self.file\_path = file\_path

7 self.checksum = 0

8

9 def calculate\_checksum(self):

10 """Вычисляет контрольную сумму файла."""

11 with open(self.file\_path, "r", encoding="utf-8") as file:

12 for line in file:

13 self.checksum += sum(ord(char) for char in line)

14

15

16 class ChecksumManager:

17 def \_\_init\_\_(self):

18 self.threads = []

19 self.checksums = []

20

21 def add\_task(self, calculator):

22 """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

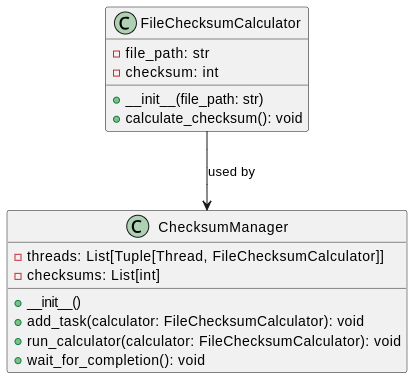
**Результат работы программы 1:**

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:**

1 import threading

2 import time

3 import random

4

5

6 class Buffer:

7 def \_\_init\_\_(self, size):

8 self.size = size

9 self.buffer = []

10 self.condition = threading.Condition()

11

12 def produce(self, item):

13 with self.condition:

14 while len(self.buffer) >= self.size:

15 print("Буфер полон. Производитель ждет...")

16 self.condition.wait()

17 self.buffer.append(item)

18 print(f"Производитель добавил: {item}")

19 self.condition.notify\_all()

20

21 def consume(self):

22 with self.condition:

23 while not self.buffer:

24 print("Буфер пуст. Потребитель ждет...")

25 self.condition.wait()

26 item = self.buffer.pop(0)

27 print(f"Потребитель извлек: {item}")

28 self.condition.notify\_all()

29 return item

30

31

32 class Producer(threading.Thread):

33 def \_\_init\_\_(self, buffer):

34 super().\_\_init\_\_()

35 self.buffer = buffer

36

37 def run(self):

38 for \_ in range(10):

39 item = random.randint(1, 100)

40 self.buffer.produce(item)

41 time.sleep(random.uniform(0.2, 0.6))

42

43

44 class Consumer(threading.Thread):

45 def \_\_init\_\_(self, buffer):

46 super().\_\_init\_\_()

47 self.buffer = buffer

48

49 def run(self):

50 for \_ in range(30):

51 item = self.buffer.consume()

52 time.sleep(random.uniform(0.1, 0.5))

53

54

55 if \_\_name\_\_ == "\_\_main\_\_":

56 buffer\_size = 5

57 buffer = Buffer(buffer\_size)

58

59 producers = [Producer(buffer) for \_ in range(3)]

60 consumer = Consumer(buffer)

61

62 for producer in producers:

63 producer.start()

64

65 consumer.start()

66

67 for producer in producers:

68 producer.join()

69

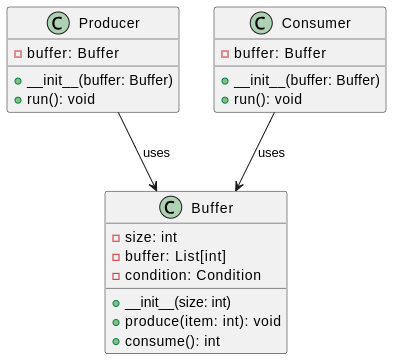
70 consumer.join()

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.