|  |  |
| --- | --- |
| **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. ИУК4-32Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_Зудин Д.В.\_\_\_\_\_)  (Подпись) (Ф.И.О.) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_Пчелинцева Н.И.\_\_\_)  (Подпись) (Ф.И.О.) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |
| 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. | | |