

**МИНОБРНАУКИ РОССИИ**

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«МИРЭА – Российский технологический университет»**

|  |
| --- |
| **РТУ МИРЭА** |
|  |
| **Институт кибербезопасности и цифровых технологий (ИКБ)** |
|  |
| КБ-2 «Прикладные информационные технологии» |

**ОТЧЕТ О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ №4**

**В РАМКАХ ДИСЦИПЛИНЫ «ОПЕРАЦИОННЫЕ СИСТЕМЫ»**

Выполнил:

Студент 2-ого курса

Учебной группы БИСО-02-22

Зубарев В.С.

Оглавление

[Задание 3](#_Toc163922152)

[Листинг программы 5](#_Toc163922153)

# Задание

Разработать программу, моделирующую один из алгоритмов управления памятью в соответствии с вариантом задания. При моделировании считать что:

• объем моделируемой «памяти» составляет 64К;

• поступаемые на выполнение задачи содержатся в отдельных структурах, содержащих количество килобайт либо страниц (для страничной организации памяти), необходимых для выполнения программы (моделируем размер задачи);

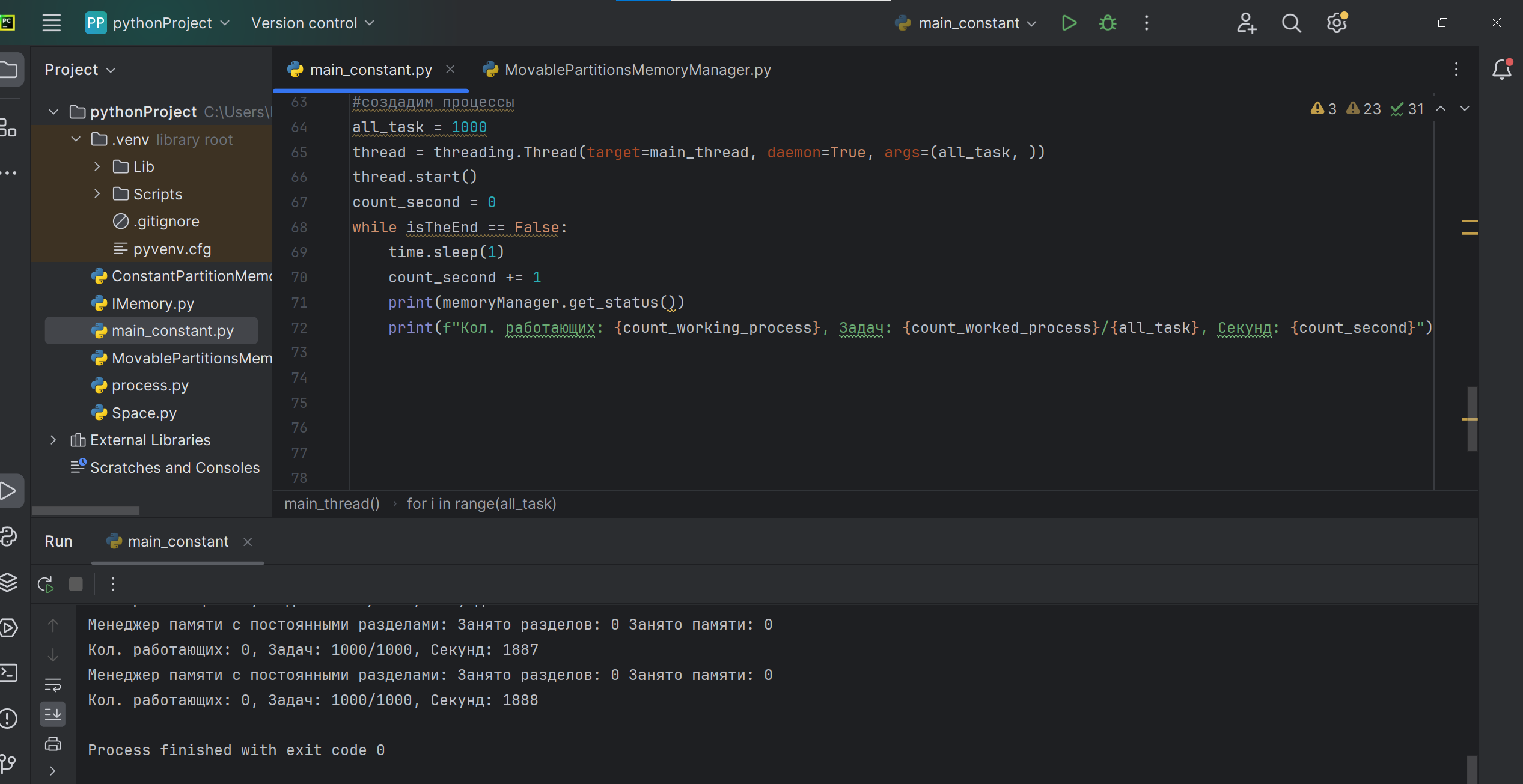
• размер задачи в диапазоне от 1 до 8 кбайт (кратно 1 кбайту), либо от 1 до 8 страниц для страничной организации памяти.

• для простоты моделирования разрешено заранее сформировать список доступных разделов в Варианте 1 и 2.

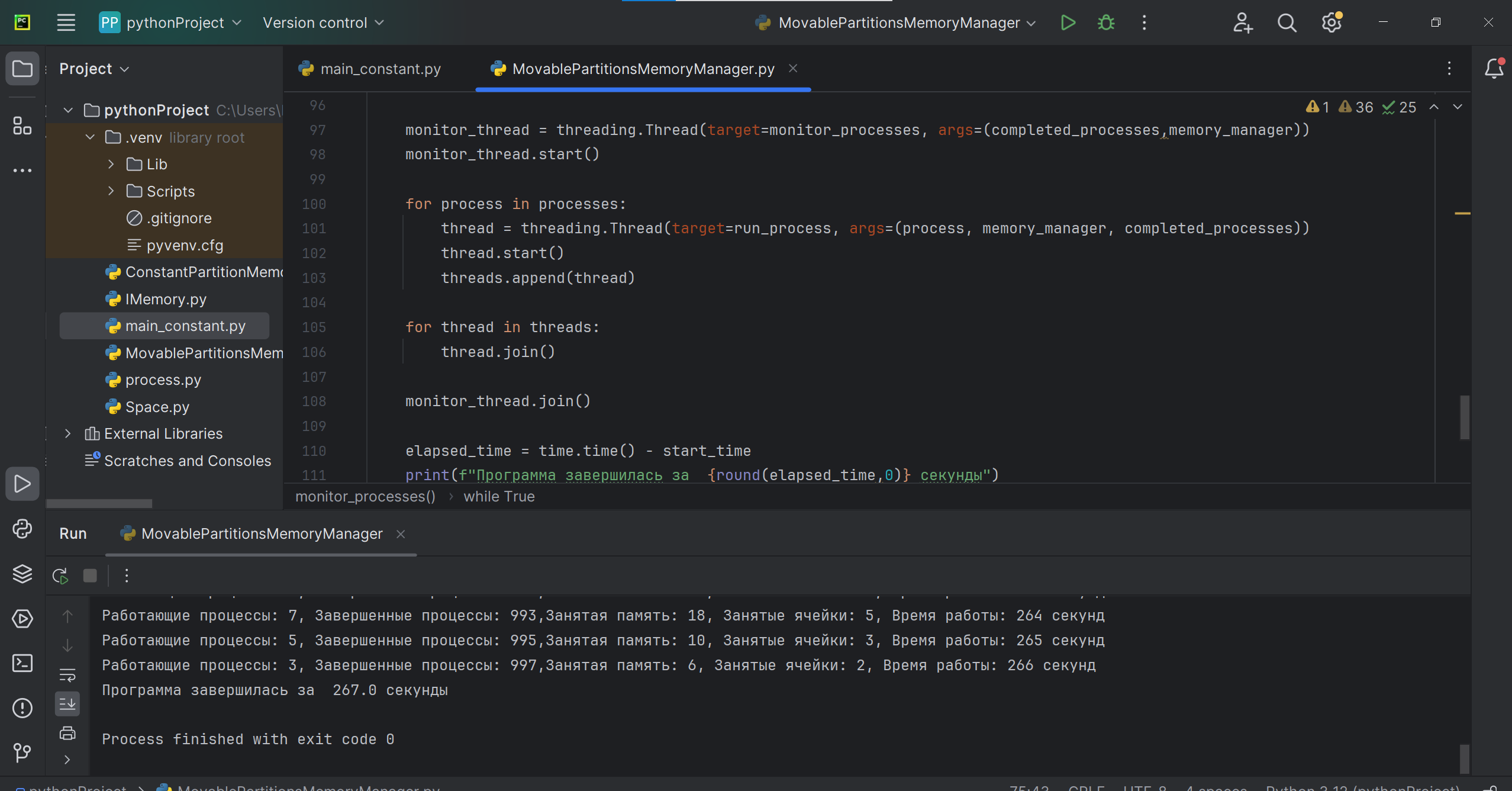
• Провести сравнение производительности реализованного алгоритма управления памятью относительно алгоритма управления памятью с постоянными разделами на примере n-количества задача (100 и больше). Результат сравнения представить в процентах. Программа должна иметь возможность просмотра состояния моделируемой «памяти»

Вариант задания 1(Распределение памяти перемещаемыми разделами)

Вариант реализации статического распределения памяти



Вариант реализации распределения памяти перемещаемыми разделами



Прирост производительности ~ 707%

# Листинг программы

import threading  
import time  
import random  
  
class Process:  
 def \_\_init\_\_(self, pid, size, time\_to\_run):  
 self.pid = pid  
 self.size = size  
 self.time\_to\_run = time\_to\_run  
  
class MemoryManager:  
 def \_\_init\_\_(self, total\_memory):  
 self.memory\_map = []  
  
 def initialize\_memory(self):  
 block\_sizes = [8, 8, 6, 6, 6, 6, 4, 4, 4, 4, 2, 2, 2, 2]  
 pid,p\_size = 0,0  
 for index, size in enumerate(block\_sizes, start=1):  
 self.memory\_map.append([index, size, "free", pid,p\_size])  
  
 def allocate\_memory(self, process):  
 for block in self.memory\_map:  
 index, size, status, pid,p\_size = block  
 if status == "free" and size >= process.size:  
 block[2] = "blocked"  
 block[3] = process.pid  
 block[4]=process.size  
 return index  
 # Если подходящей ячейки не найдено, пытаемся переместить другие процессы  
 for block1 in self.memory\_map:  
 index1, size1, status1, pid1,p\_size1= block1  
 for block2 in self.memory\_map:  
 index2, size2, status2, pid2, p\_size2 = block2  
 if status=="free" and size2>=p\_size1 and index1!=index2:  
 block2[2]="blocked"  
 block2[3]=pid1  
 block2[4]=p\_size1  
 block1[3]=process.pid  
 block1[4]=process.size  
 return block1[0]  
 return -1 # Allocation failed  
  
 def free\_memory(self, index):  
 self.memory\_map[index-1][2] = "free"  
 self.memory\_map[index-1][3] = 0  
 self.memory\_map[index-1][4] = 0  
  
 def active\_memory(self):  
 count\_memory=0  
 active\_cells=0  
 for block in self.memory\_map:  
 count\_memory+=block[4]  
 if block[2]=="blocked":  
 active\_cells+=1  
 return count\_memory,active\_cells  
  
  
def run\_process(process, memory\_manager, completed\_processes):  
 while True:  
 start\_time = time.time()  
 allocated\_memory = memory\_manager.allocate\_memory(process)  
 if allocated\_memory != -1:  
 time.sleep(process.time\_to\_run)  
 memory\_manager.free\_memory(allocated\_memory)  
 completed\_processes.append(process)  
 break  
 else:  
 time.sleep(1)  
  
  
def monitor\_processes(completed\_processes,memory\_manager):  
 t=0  
 while True:  
 mem,cells=memory\_manager.active\_memory()  
 print(f"Работающие процессы: {1000-len(completed\_processes)}, Завершенные процессы: {len(completed\_processes)},"  
 f"Занятая память: {mem}, Занятые ячейки: {cells}, Время работы: {t} секунд")  
 t+=1  
 time.sleep(1)  
 # Проверяем, завершились ли все процессы  
 if len(completed\_processes) == 1000:  
 break  
  
def main():  
 total\_memory = 64  
  
 memory\_manager = MemoryManager(total\_memory)  
 memory\_manager.initialize\_memory()  
  
 completed\_processes = []  
  
 processes = [Process(pid + 1, random.choice([2, 4, 6,8]), random.randint(5, 10)) for pid in range(1000)]  
  
 threads = []  
  
 start\_time = time.time()  
  
 monitor\_thread = threading.Thread(target=monitor\_processes, args=(completed\_processes,memory\_manager))  
 monitor\_thread.start()  
  
 for process in processes:  
 thread = threading.Thread(target=run\_process, args=(process, memory\_manager, completed\_processes))  
 thread.start()  
 threads.append(thread)  
  
 for thread in threads:  
 thread.join()  
  
 monitor\_thread.join()  
  
 elapsed\_time = time.time() - start\_time  
 print(f"Программа завершилась за {round(elapsed\_time,0)} секунды")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()