**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №8

по дисциплине: **«**[**Операционные системы**](https://www.edu.gstu.by/course/view.php?id=557)**»**

**Простейшие схемы управления памятью**

Выполнил: студент гр. ИТП-11

Горкунов А.О.

Принял: Преподаватель

Карась О.В.

Гомель 2023

**Цель работы**: изучение алгоритмов управления памятью, разработка программы менеджера памяти*.*

**Ход работы**

***Задание.* Разработать программу, реализующую заданный алгоритм выделения памяти**

**1. Создание файла script.py реализующего менеджер памяти**

**2. Тестирование работы скрипта**

На рисунке 1 показано создание структуры памяти

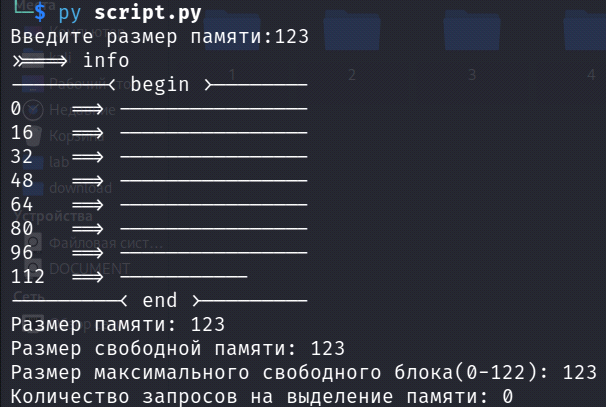
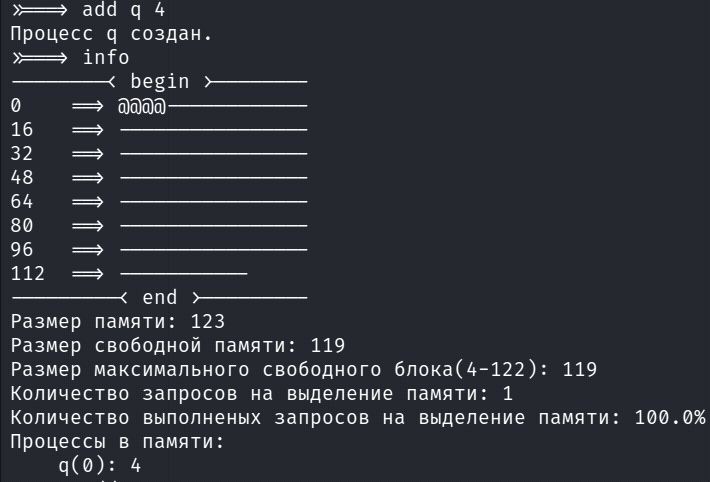


Рисунок 1 - создание структуры памяти

На рисунке 2 показано создание процесса q

Рисунок 2 - создание процесса q

На рисунке 3 показано создание процесса e

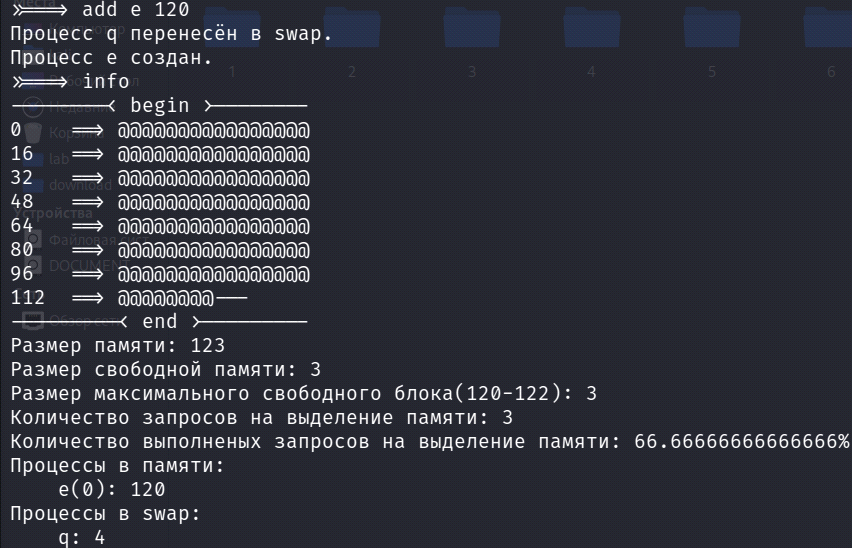
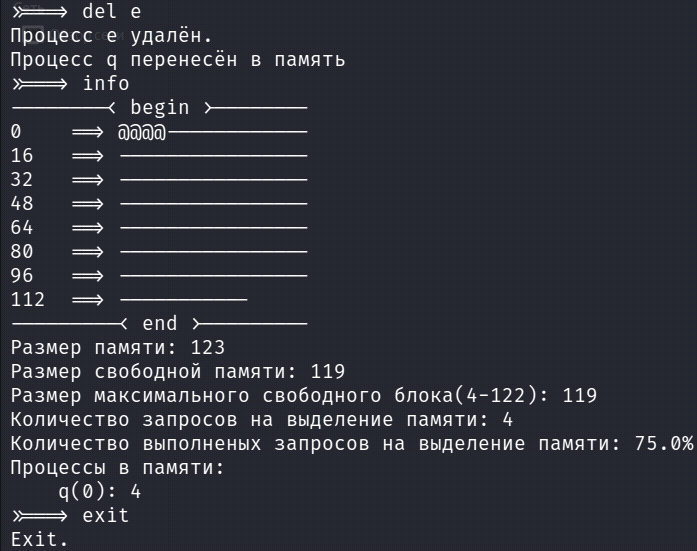


Рисунок 3 - создание процесса e

На рисунке 4 показано удаление процесса e



*Рисунок 4 - удаление процесса e*

**Вывод: мы разработали программу, реалищующую заданный алгоритм выделения памяти.**

Листинг:

from more\_itertools import chunked

from dataclasses import dataclass

class MemoryException(Exception): pass

@dataclass

class Process:

name: str

position: int

size: int

class Memory:

process: list[Process] = []

swap\_process = []

count\_success\_request = 0

count\_request = 0

def \_\_init\_\_(self, size\_memory: int) -> None:

self.size\_memory = size\_memory

self.block\_list = [False for \_ in range(size\_memory)]

self.size\_free\_memory = size\_memory

def print\_state(self) -> None:

print("--------< begin >--------")

for i, blocks\_row in enumerate(chunked(self.block\_list, 16)):

row\_state\_memory = "".join((

"@" if block else "-" for block in blocks\_row

))

print(f"{i\*16:<5}==> {row\_state\_memory}")

print("---------< end >---------")

print(f"Размер памяти: {self.size\_memory}")

print(f"Размер свободной памяти: {self.size\_free\_memory}")

max\_free = 0

index\_max\_free = 0

size\_free\_flat = 0

for i, block in enumerate(self.block\_list):

if not block:

size\_free\_flat += 1

else:

size\_free\_flat = 0

if size\_free\_flat > max\_free:

max\_free = size\_free\_flat

index\_max\_free = i

if max\_free:

print(f"Размер максимального свободного блока({index\_max\_free-max\_free+1}-{index\_max\_free}): {max\_free}")

print(f"Количество запросов на выделение памяти: {self.count\_request}")

if self.count\_success\_request:

print(f"Количество выполненых запросов на выделение памяти: {self.count\_success\_request/self.count\_request\*100}%")

if len(self.process):

print("Процессы в памяти:")

for process in self.process:

print(f" {process.name}({process.position}): {process.size}")

if len(self.swap\_process):

print("Процессы в swap:")

for process in self.swap\_process:

print(f" {process.name}: {process.size}")

def delete\_process\_to\_memory(self, name: str) -> None:

process\_for\_delete = None

for process in self.process:

if process.name == name:

process\_for\_delete = process

if not process\_for\_delete:

raise MemoryException(f"Процесс {name} не найден.")

for i in range(process\_for\_delete.position, process\_for\_delete.position+process\_for\_delete.size):

self.block\_list[i] = False

self.process.remove(process\_for\_delete)

self.size\_free\_memory += process\_for\_delete.size

def add\_process\_to\_memory(self, name: str, size: int) -> bool:

self.count\_request += 1

if size > self.size\_memory:

raise MemoryException("Процесс слишком большой")

for process in self.process:

if process.name == name:

raise MemoryException(f"Процесс с именем {name} уже существует.")

size\_free\_flat = 0

suitable\_position = -1

for i, block in enumerate(self.block\_list):

if not block:

size\_free\_flat += 1

else:

size\_free\_flat = 0

if size\_free\_flat == size:

suitable\_position = i - size + 1

break

if suitable\_position == -1:

return False

for i in range(suitable\_position, suitable\_position+size):

self.block\_list[i] = True

self.process.append(Process(name, suitable\_position, size))

self.size\_free\_memory -= size

self.count\_success\_request += 1

return True

def situated\_move\_swap\_process\_to\_memory(self) -> None:

while True:

list\_to\_delete\_in\_swap = []

for process in self.swap\_process:

if self.add\_process\_to\_memory(process.name, process.size):

list\_to\_delete\_in\_swap.append(process)

print(f"Процесс {process.name} перенесён в память")

if not len(list\_to\_delete\_in\_swap):

break

for process in list\_to\_delete\_in\_swap:

self.swap\_process.remove(process)

def delete\_process\_for\_name(self, name: str) -> None:

self.delete\_process\_to\_memory(name)

print(f"Процесс {name} удалён.")

self.situated\_move\_swap\_process\_to\_memory()

def add\_process(self, name: str, size: int) -> None:

while not self.add\_process\_to\_memory(name, size):

largest\_process = max(self.process, key=lambda k: k.size)

self.delete\_process\_to\_memory(largest\_process.name)

self.swap\_process.append(largest\_process)

print(f"Процесс {largest\_process.name} перенесён в swap.")

print(f"Процесс {name} создан.")

def main() -> None:

size\_memory = int(input("Введите размер памяти:"))

memory = Memory(size\_memory)

while True:

user\_input = input(">>==> ").split(" ")

try:

match user\_input:

case ["info",]:

memory.print\_state()

case ["add", name, size\_str]:

if size\_str.isdigit():

memory.add\_process(name, int(size\_str))

case ["del", name]:

memory.delete\_process\_for\_name(name)

case ["exit",]:

print("Exit.")

break

except MemoryException as e:

print(f"Ошибка: {e}")

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

try:

main()

except KeyboardInterrupt:

print("\nExit.")