НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування та спеціалізованих комп‘ютерних систем

**Лабораторна робота №3  
*З дисципліни: «Операційні системи»***

«Керування пам'яттю»

Студентка групи КВ-42

Брикалова Вікторія

Перевірив(ла):

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Київ 2017

**Завдання**

1. Написатипрограму, що моделює процес управлінняпам’яттю (розподілу

пам’яті для процесів, перетворення віртуальної адреси у фізичну, пошук у пам’яті за запитами процесів, вивільнення пам’яті) при заданому способі організації пам’яті (перелік варіантів представлений нижче у табл. 3.1).

Вхідні дані– розмір пам’яті, що підлягає розподілу, розміри сторінок (розділів, сегментів тощо), розміри потрібної процесам пам’яті та ін. задаються самостійно та у відповідності до завдання.

1. Продемонструвати роботу моделі з виконанням основних операцій з

пам’яттю: надання пам’яті потрібного розміру за запитом процеса, перетворення віртуальної адреси у “фізичну” при зверненні до комірки пам’яті, здійнення запису або читання, вивільнення пам’яті при завершенні процесу. Завдання операцій можна реалізувати за допомогою меню.

**Варіант 2:**

Д**инамічні розділи** (без використання зовнішньої пам’яті).

Кількість розділів меньша, ніж кількість процесів. Процеси утворюють загальну чергу до розділів пам’яті. Використовується **лінійний** адресний простір. Розміри процесів задаються випадково.

**Текст програми**

**Process.py**

class Process:

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

self.size = size

self.num = 0

self.addr = 0

**Lab3.py**

import random as rnd

import process as pr

global MEMORY\_SIZE

global QUEUE

global MEMORY

global PROCESS\_NUM

def SetMemory():

global MEMORY\_SIZE

global FREE\_MEMORY\_SIZE

if MEMORY\_SIZE != -1:

print("Memory size already set")

else:

print("Enter memory size:")

while MEMORY\_SIZE < 0:

MEMORY\_SIZE = int(input())

if MEMORY\_SIZE > 0:

break

else:

print("Enter right size")

def AddProcess():

global PROCESS\_NUM

global MEMORY

global QUEUE

global MEMORY\_SIZE

if MEMORY\_SIZE > 0:

new\_process = pr.Process(rnd.randint(1, 10)\*10)

PROCESS\_NUM +=1

new\_process.num = PROCESS\_NUM

if MEMORY == []:

new\_process.addr = 0;

MEMORY.append(new\_process)

else:

QUEUE.append(new\_process)

buff = MEMORY[0]

i = 0

for element in MEMORY:

if element != buff:

if (element.addr - (buff.addr + buff.size)) >= new\_process.size:

new\_process.addr = buff.addr + buff.size

QUEUE.remove(new\_process)

MEMORY.insert(i, new\_process)

break

else:

buff = element

i+=1

if new\_process in QUEUE:

if (MEMORY\_SIZE - (buff.addr + buff.size)) >= new\_process.size:

new\_process.addr = buff.addr + buff.size

QUEUE.remove(new\_process)

MEMORY.append(new\_process)

def FillMemory():

global QUEUE

global MEMORY

buff\_queue = QUEUE

for process in QUEUE:

buff = MEMORY[0]

i=0

for element in MEMORY:

if element != buff:

if (element.addr - (buff.addr + buff.size)) >= process.size:

process.addr = buff.addr + buff.size

buff\_queue.remove(process)

MEMORY.insert(i, process)

break

else:

buff = element

else:

if buff.addr > 0 and buff.addr >= process.size:

process.addr = 0

buff\_queue.remove(process)

MEMORY.insert(i, process)

break

i+=1

if process in buff\_queue:

if (MEMORY\_SIZE - (buff.addr + buff.size)) >= process.size:

process.addr = buff.addr + buff.size

buff\_queue.remove(process)

MEMORY.append(process)

QUEUE = buff\_queue

def FinishProcess():

global QUEUE

global MEMORY

print("Enter number of process: ")

num = int(input())

flag = 0

for element in MEMORY:

if element.num == num:

flag = 1

process = element

break

if flag == 0:

print("No such process in memory! ")

MEMORY.remove(process)

FillMemory()

def SqueezeMemory():

if MEMORY != []:

i=0

for element in MEMORY:

if i>0:

if element.addr > (MEMORY[i-1].addr + MEMORY[i-1].size):

element.addr = MEMORY[i-1].addr + MEMORY[i-1].size

i+=1

FillMemory()

else:

print("Memory is empty")

def GetAddres():

global QUEUE

global MEMORY

print("Enter number of process: ")

num = int(input())

flag = 0

for element in MEMORY:

if element.num == num:

flag = 1

process = element

break

if flag == 0:

print("No such process in memory! ")

else:

print("Enter virtual address: ")

virtual\_address = int(input())

if virtual\_address > process.size:

print("Wrong virtual addres!")

else:

print("Real addres: ", process.addr + virtual\_address)

def PrintMemoryState():

print()

print("MEMORY: ", MEMORY\_SIZE)

print("----------------------------------------------------")

print("| number | start\_addr | finish\_addr | size |")

print("----------------------------------------------------")

for element in MEMORY:

print("| %6d | %10d | %11d | %4d |" %(element.num, element.addr, element.addr + element.size, element.size))

print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

print()

print("QUEUE: ")

print("----------------------------------------------------")

print("| number | start\_addr | finish\_addr | size |")

print("----------------------------------------------------")

for element in QUEUE:

print("| %6d | %10d | %11d | %4d |" %(element.num, element.addr, element.addr + element.size, element.size))

print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

def Menu():

ch = 0

while ch != '6':

print()

print("Print memory state: 0")

print("Set memory: 1")

print("Add process: 2")

print("Finish process: 3")

print("Squeeze memory: 4")

print("Get real address: 5")

print("Finish program: 6")

print()

ch = input()

if ch == '0':

PrintMemoryState()

elif ch == '1':

SetMemory()

elif ch == '2':

AddProcess()

elif ch == '3':

FinishProcess()

elif ch == '4':

SqueezeMemory()

elif ch == '5':

GetAddres()

elif ch == '6':

break

else:

continue

if \_\_name\_\_ == '\_\_main\_\_':

MEMORY\_SIZE = -1

PROCESS\_NUM = 0

QUEUE = []

MEMORY = []

**Тест**

