

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет науки і технологій**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №3**

**з дисципліни «Операційні системи»**

**на тему:**

**«Купи. Робота з віртуальною пам'яттю.»**

Виконав:

Студент гр. ПЗ2011

Кулик С. В.

Прийняв:

Андрющенко В. О.

Дніпро, 2022

**Тема.** Купи. Робота з віртуальною пам'яттю.

**Теоретичні відомості**

Віртуальна пам’ять — схема адресації пам’яті комп’ютера, при якій пам’ять для запущеної програми реалізується однорідним масивом, в той час як насправді операційна система виділяє пам’ять блоками в різних видах пам’яті, включаючи короткочасну (оперативну) і довгочасну.

Сторінкова пам’ять — це підхід до організації віртуальної пам яті, при якому одиницею відображення віртуальних адрес на фізичні є регіон константного розміру (так звана сторінка).Її головна ідея — розподіл пам'яті блоками фіксованої довжини, що називають сторінками. Логічну пам'ять розбивають на блоки такої самої довжини — сторінки. Коли процес починає виконуватися, його сторінки завантажуються в доступні фрейми фізичної пам'яті з диска або іншого носія. Сторінкова організація пам'яті має апаратну підтримку. Кожна адреса, яку генерує процесор, ділиться на дві частини: номер сторінки і зсув сторінки. Номер сторінки використовують як індекс у таблиці сторінок.

За допомогою функції VirtualAlloc є можливість виділити або зарезервувати сторінки віртуальної пам’яті. Між виділенням і резервуванням є різниця. При виділенні фізично виділяється пам’ять і природно збільшується файл підкачки. Для резервування цього не відбувається.

Купа - це регіон зарезервованого адресного простору. Спочатку більшої його частини фізична пам’ять не передається. У міру того, як програма займає цю область під дані, спеціальний диспетчер, керуючий купами (heap manager), посторінково передасть їй фізичну пам’ять (з сторінкового файлу). А при звільненні блоків в купі диспетчер повертає системі відповідні сторінці фізичної пам’яті.

**Текст програми**

**Файл Memory.h**

#pragma once

#include <windows.h>

#include <iostream>

#include <iomanip>

#include <string>

using namespace std;

struct Data {

int index;

char arr[1024];

};

class Memory {

public:

void\* \_data = NULL;

int \_count = 0;

void print()

{

MEMORY\_BASIC\_INFORMATION memory;

int info\_type;

Data\* info = (Data\*)\_data;

cout << setw(5) << "# |" << setw(18) << " Base address |" << setw(18) << " Alloc base |" << setw(15) << " Region size |"

<< setw(14) << " Alloc prot |" << setw(10) << " Protect |" << setw(10) << " State |" << setw(8) << " Type " << endl;

for (int l = 0; l < 100; l ++) {

info\_type = 0;

VirtualQuery((void\*)info, &memory, sizeof(memory));

cout << setw(3) << l << " |" << setw(13) << memory.BaseAddress << " |" << setw(13) << memory.AllocationBase << " |" << setw(13)

<< memory.RegionSize << " |" << setw(12) <<memory.AllocationProtect << " |" << setw(8) << memory.Protect << " |" << setw(8)

<< getState(memory.State) << " |" << setw(8) << getType(memory.Type) << endl;

if (\_data != NULL)

info++;

}

}

string getState(DWORD state) {

switch (state) {

case 0x1000: return "COMMIT";

case 0x10000: return "FREE";

case 0x2000: return "RESERVE";

default: return "";

}

}

string getType(DWORD type) {

switch (type) {

case 0x1000000: return "IMAGE";

case 0x40000: return "MAPPED";

case 0x20000: return "PRIVATE";

default: return "";

}

}

void memory\_info() {

MEMORYSTATUSEX memInfo;

memInfo.dwLength = sizeof(MEMORYSTATUSEX);

GlobalMemoryStatusEx(&memInfo);

cout << "Memory information:\n" << endl;

cout << "Memory load: " << memInfo.dwMemoryLoad << endl;

cout << "Total pshysical: " << memInfo.ullTotalPhys / (1024 \* 1024) << " Mb" << endl;

cout << "Avail pshysical: " << memInfo.ullAvailPhys / (1024 \* 1024) << " Mb" << endl;

cout << "Total Page File: " << memInfo.ullTotalPageFile / (1024 \* 1024) << " Mb" << endl;

cout << "Avail Page File: " << memInfo.ullAvailPageFile / (1024 \* 1024) << " Mb" << endl;

cout << "Total virtual: " << memInfo.ullTotalVirtual / (1024 \* 1024) << " Mb" << endl;

cout << "Avail virtual: " << memInfo.ullAvailVirtual / (1024 \* 1024) << " Mb" << endl;

}

void system\_info() {

SYSTEM\_INFO sysInfo;

GetSystemInfo(&sysInfo);

cout << "System information:\n" << endl;

cout << "Page size: " << sysInfo.dwPageSize << endl;

cout << "Allocation granularity: " << sysInfo.dwAllocationGranularity << endl;

cout << "Minimum application addres:" << sysInfo.lpMinimumApplicationAddress << endl;

cout << "Maximum application addres:" << sysInfo.lpMaximumApplicationAddress << endl;

}

void allocation() {

\_data = VirtualAlloc(NULL, sizeof(Data) \* 256, MEM\_RESERVE, PAGE\_READWRITE);

\_count++;

if (\_count >= 3) {

cout << "\nMemory is already allocated!\n";

return;

}

\_data = VirtualAlloc(NULL, sizeof(Data) \* 256, MEM\_RESERVE, PAGE\_READWRITE);

Data\* data\_copy = (Data\*)\_data;

for (int i = 0; i < 64; i++) {

data\_copy += rand() % 256 + 1;

VirtualAlloc(data\_copy, sizeof(Data), MEM\_COMMIT, PAGE\_READWRITE);

if (data\_copy->index == 0)

data\_copy->index = 1 + rand() % 6;

else i--;

data\_copy = (Data\*)\_data;

}

print();

}

void clear() {

if (\_data != NULL) {

VirtualFree(\_data, 0, MEM\_RELEASE);

print();

\_data = NULL;

}

else cout << "\nMemory cleared \n";

}

};

**Файл Heap.h**

#pragma once

#include <windows.h>

class Heap {

static HANDLE heap;

static int heap\_size;

public:

static int getHeapSize() { return heap\_size; }

void\* operator new(size\_t size) {

if (heap == NULL) {

heap = HeapCreate(HEAP\_NO\_SERIALIZE, 0, 0);

}

if (heap == NULL)

return (NULL);

void\* pointer = HeapAlloc(heap, 0, size);

if (pointer != NULL)

heap\_size++;

return(pointer);

}

void operator delete(void\* pointer){

if (HeapFree(heap, 0, pointer))

heap\_size--;

if (heap\_size == 0 && HeapDestroy(heap))

heap = NULL;

}

};

HANDLE Heap::heap = NULL;

int Heap::heap\_size = 0;

**Файл main.cpp**

#include "Heap.h"

#include "Memory.h"

#include <windows.h>

#include <iostream>

using namespace std;

int main() {

Heap\* array[10];

Memory memory;

Heap heap;

int choice = 0;

while (true) {

cout << "1 - System informatoin\n"

<< "2 - Memory informatoin\n"

<< "3 - Alloc memory\n"

<< "4 - Clear memory\n"

<< "5 - Operator new\n"

<< "6 - Operator delete\n"

<< "0 - Exit\n";

cin >> choice;

switch (choice)

{

case 1:

memory.system\_info();

break;

case 2:

memory.memory\_info();

break;

case 3:

memory.allocation();

break;

case 4:

memory.clear();

break;

case 5:

array[heap.getHeapSize()] = new Heap();

cout << "New item #" << heap.getHeapSize() << "\n";

break;

case 6:

if (heap.getHeapSize() > 0) {

cout << "Deleted item #" << heap.getHeapSize() << "\n";

delete array[heap.getHeapSize()];

}

else

cout << "There are no item created.\n";

break;

case 0:

system("pause");

return 0;

default:

cout << endl << "Encorrect value! \n";

break;

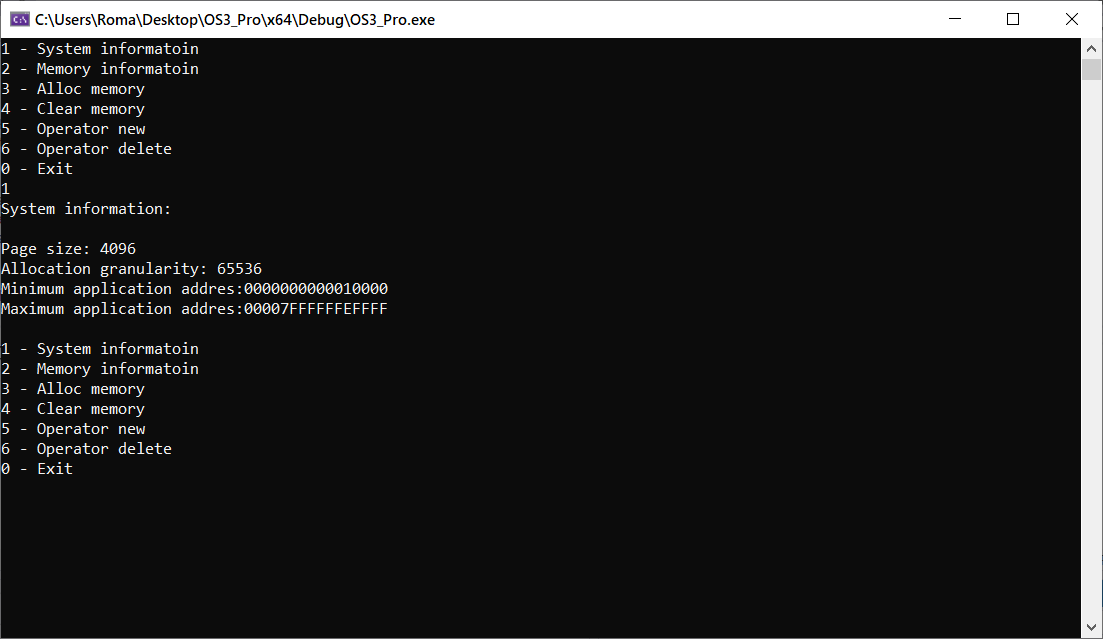
}

cout << endl;

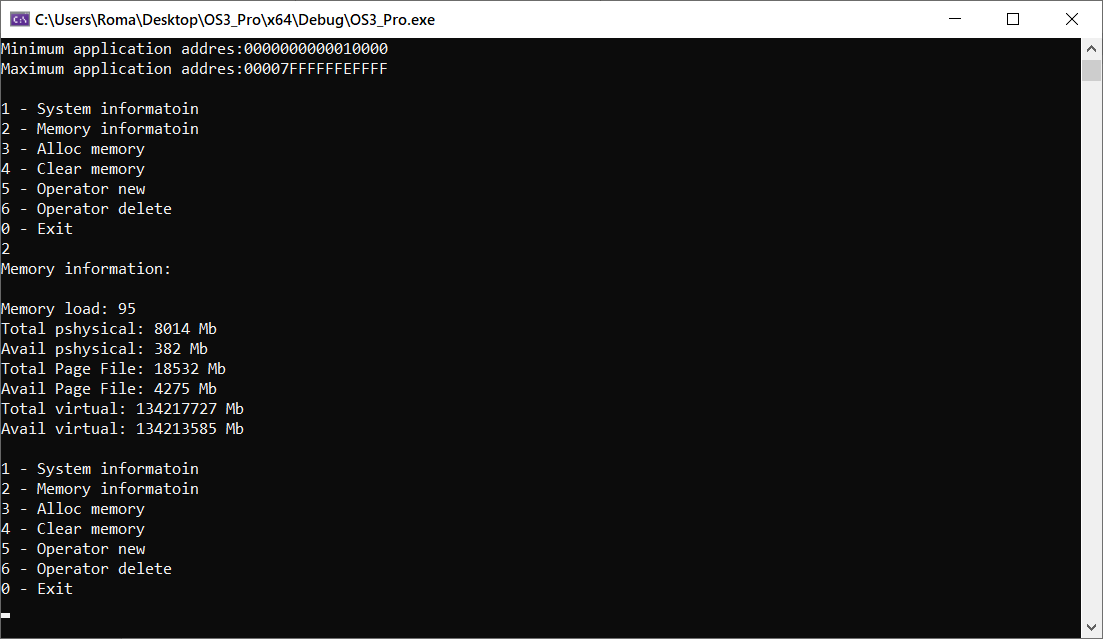
}

}

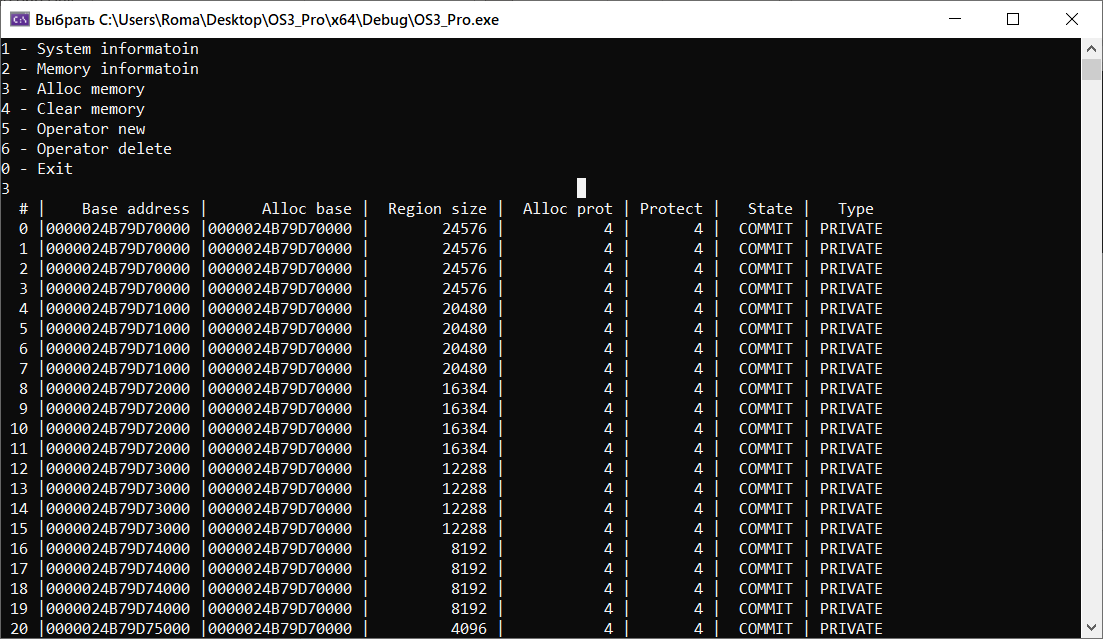
**Результати роботи програми**



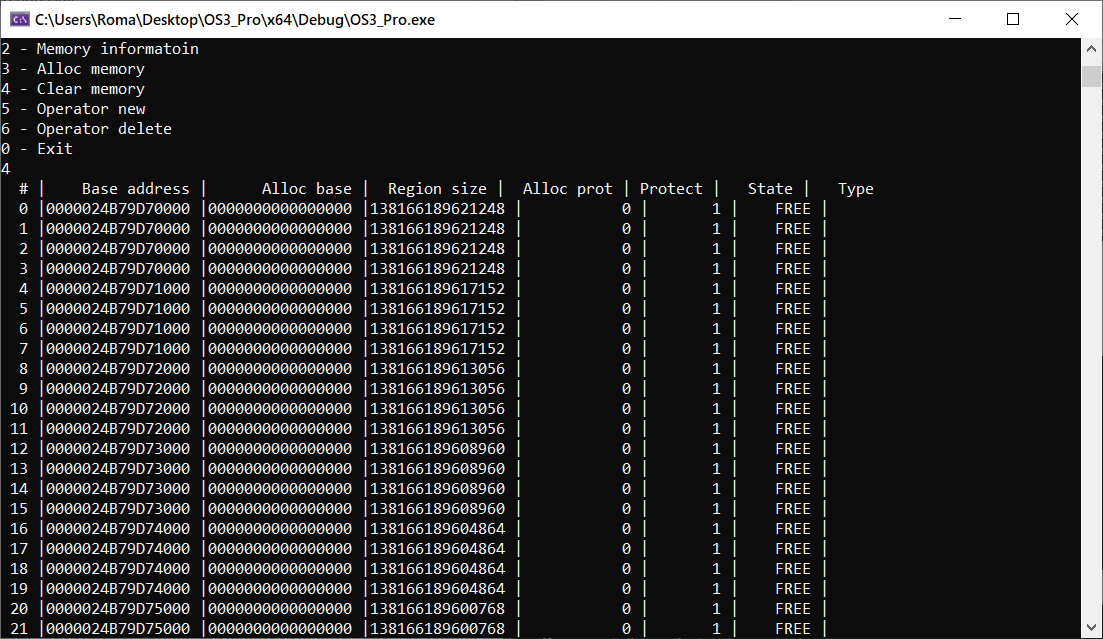
Виведення системної інформації.



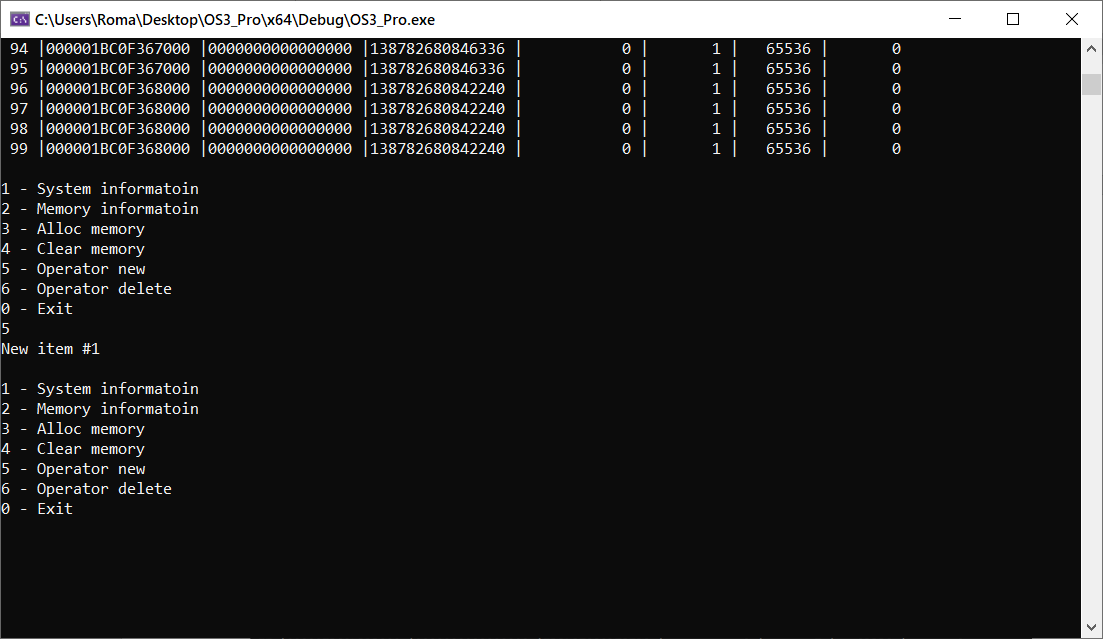
Виведення інформації про пам’ять.



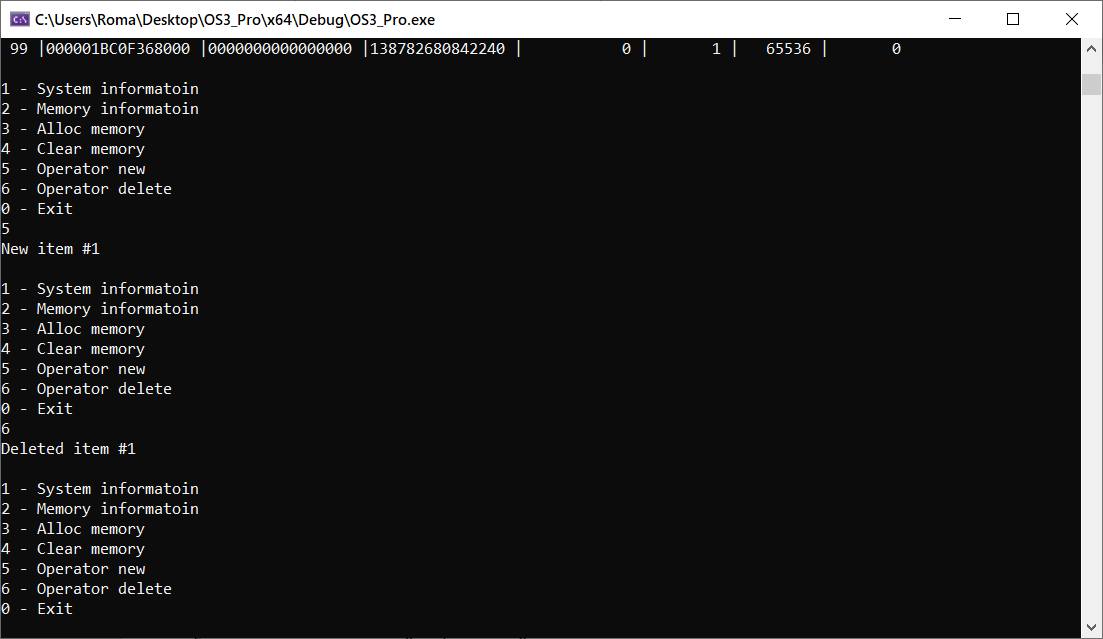
Виділення пам'яті.



Очищення пам'яті.



Перезавантаження оператора new.



Перезавантаження оператора delete.

Можна побачити, що програма працює коректно, виконує поставлені завдання.

**Висновок**

Під час виконання лабораторної роботи було набуто практичні навички роботи з віртуальною пам’ятю та аспектами її використання. Також була продемонстрована робота з купами в мові програмування С++.