МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ



**Дніпровський національний університет  
залізничного транспорту імені академіка В. Лазаряна**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №3**

**з дисципліни «Операційні системи»**

**на тему: «Купи. Робота з віртуальною пам’яттю.»**

Виконав: студент гр. ПЗ1911

Сіньков Г.О.

Прийняла: ас. каф. КІТ

Адрющенко В. О.

Дніпро, 2020

**Лабораторна робота №3**

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

**Завдання.** Розробити програми, які моделюють рішення задачі синхронізації з використанням заданих системних ресурсів. Процес обмінюються даними через проекцію файлу. Програма повина бути реалізована як графічний додаток.

**Завдання:**

1. Скласти «карту пам’яті» - вивести інформацію про регіони адресного простору процесу (стан, доступ, розмір).

2. Перевірити дію атрибутів доступу до сторінок та гранулярність виділення пам’яті.

3. Створити додаткові купи процесу та використовувати їх, перевантаживши операції new та delete для динамічних структур даних.

**Текст програми**

main.cpp

#include <Windows.h>

#include <sysinfoapi.h>

#include <iostream>

#include <heapapi.h>

#include <memoryapi.h>

#include <errhandlingapi.h>

using namespace std;

HANDLE hHeap;

//Initialization of heap that returns pointer to dedicated part of memory

LPVOID heap\_initialize() {

cout << "-HEAP\_INITIALIZATION-" << endl;

LPVOID pHeap = HeapAlloc(GetProcessHeap(), 1024, HEAP\_ZERO\_MEMORY);

if (pHeap == 0) {

cout << "Fail to allocate heap" << endl;

return 0;

}

else {

hHeap = HeapCreate(1, 1024, 5000);

cout <<"Heap size " << HeapSize(hHeap, HEAP\_ZERO\_MEMORY, pHeap) << endl;

return pHeap;

}

cout << '\n';

//HeapFree(GetProcessHeap(), HEAP\_ZERO\_MEMORY, HeapFree);

}

//output information about regions of the memory address space

void arrangeMemory(SYSTEM\_INFO &sysinf) {

cout << "Memory card" << endl;

//GetSystemInfo(sysinf);

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

cout << "Number of processors: " << sysinf.dwNumberOfProcessors << endl;

cout << "Processor type: " << sysinf.dwProcessorType << endl;

cout << "Processor level: " << sysinf.wProcessorLevel << endl;

cout << "Processor revision: " << sysinf.wProcessorRevision << endl;

cout << '\n';

}

//check granularity

//if we dedicated particular amount of bytes

//it should dedicate define amount of virtual memory pages

void CheckGranularity(SYSTEM\_INFO &inf) {

cout << "-CHECK\_GRANULARITY-" << endl;

LPVOID all = VirtualAlloc(&inf.lpMinimumApplicationAddress, inf.dwPageSize / 2 + 1 , inf.dwAllocationGranularity, PAGE\_READWRITE);

if (all == NULL) {

SetLastError(GetLastError());

}

else {

PMEMORY\_BASIC\_INFORMATION lpBuffer = NULL;

SIZE\_T size = VirtualQuery(all, lpBuffer, 1024);

cout << "We allocated " << inf.dwPageSize / 2 + 1 << " and get" << size << endl;

}

}

//check attributes of acces

//attribute of access

void CheckAtribute(SYSTEM\_INFO& inf) {

cout << "-CHECK\_ACCESS\_ATRIBUTE-" << endl;

LPVOID all = VirtualAlloc(&inf.lpMinimumApplicationAddress, inf.dwPageSize / 2 + 1, inf.dwAllocationGranularity, PAGE\_READWRITE);

cout << "Trying to check access atributes(NO\_ACCESS)" << endl;

if (all == NULL) {

SetLastError(GetLastError());

}

else {

PMEMORY\_BASIC\_INFORMATION lpBuffer = NULL;

SIZE\_T size = VirtualQuery(all, lpBuffer, 1024);

if (lpBuffer->AllocationProtect == 0)

cout << "the caller does not have access" << endl;

else

cout << "We succeeded!" << endl;

}

}

//operator new/delete overloading

void\* operator new(std::size\_t size, int a, const char\* b)

{

LPVOID pheap = HeapAlloc(hHeap, 1, 1024);

if (pheap == NULL)

throw exception();

else {

cout << "X new\n";

return ::operator new(size);

}

}

void operator delete(void\* ptr)

{

LPVOID pheap = HeapAlloc(1, 1, 1024);

if (HeapFree(hHeap, 1, pheap)) {

cout << "X delete\n";

::operator delete(ptr);

}

else

cout << "Fail to delete" << endl;

}

class Exmpl{};

int main()

{

LPVOID pHeap;

pHeap = heap\_initialize();

SYSTEM\_INFO sysinfo;

arrangeMemory(sysinfo);

CheckGranularity(sysinfo);

CheckAtribute(sysinfo);

size\_t sizeExmpl = 1024;

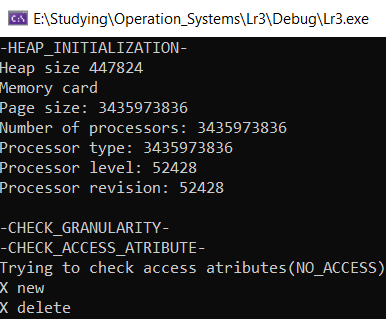
Exmpl\* p = new(sizeExmpl, "oh") Exmpl();

delete p;

system("pause");

return 0;

}



Висновок: В ході виконання лабораторної роботи ми ознайомилися з поняттям купи та деякими специфіками роботи з віртуальної пам’яті. На практиці ж перевірили організацію віртуальної пам’яті, куп та перевантаження операцій з використанням створення додаткових куп процесу.