Правительство Санкт-Петербурга

Комитет по науке и высшей школе

Санкт-Петербургское государственное бюджетное

профессиональное образовательное учреждение

«Политехнический колледж городского хозяйства»

**ПРАКТИЧЕСКАЯ РАБОТА № 21**

**Тема:** Работа с виртуальной памятью в Windows

**Специальность: 09.02.07 Информационные системы и программирование,**

**по квалификации «Программист»**

Группа ИП-21-7К\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент БобриковАА\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ф. И. О

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

Санкт-Петербург

2023 г

// Листинг 21.1. Распределение виртуальной памяти процессу

#include <windows.h>

#include <iostream>

using namespace std;

int main()

{

int\* a; // указатель на массив целых чисел

const int size = 1000; // размерность массива

// распределяем виртуальную память

a = (int\*)VirtualAlloc(

NULL,

size \* sizeof(int),

MEM\_COMMIT,

PAGE\_READWRITE);

if (!a)

{

cout << "Virtual allocation failed." << endl;

return GetLastError();

}

cout << "Virtual memory address: " << a << endl;

// освобождаем виртуальную память

if (!VirtualFree(a, 0, MEM\_RELEASE))

{

cout << "Memory release failed." << endl;

return GetLastError();

}

return 0;

}



// Листинг 21.2. Распределение виртуальной памяти по конкретному адресу

#include <windows.h>

#include <iostream>

using namespace std;

int main()

{

LPVOID lp;

const int size = 1000;

// распределяем виртуальную память

lp = VirtualAlloc(

(LPVOID)0x00890002,

size,

MEM\_RESERVE | MEM\_COMMIT,

PAGE\_READWRITE);

if (!lp)

{

cout << "Virtual allocation failed." << endl;

return GetLastError();

}

cout << "Virtual memory address: " << lp << endl;

// освобождаем виртуальную память

if (!VirtualFree(lp, 0, MEM\_RELEASE))

{

cout << "Memory release failed." << endl;

return GetLastError();

}

return 0;

}



// Листинг 21.3. Резервирование и распределение виртуальной памяти

#include <windows.h>

#include <iostream>

using namespace std;

int main()

{

LPVOID lpr, lpc;

const int Kb = 1024;

const int size = 100;

// резервируем виртуальную память

lpr = VirtualAlloc(

(LPVOID)0x00880000,

size \* Kb,

MEM\_RESERVE,

PAGE\_READWRITE);

if (!lpr)

{

cout << "Virtual memory reservation failed." << endl;

return GetLastError();

}

cout << "Virtual memory address: " << lpr << endl;

// распределяем виртуальную память

lpc = VirtualAlloc(

(LPVOID)0x00888000,

Kb,

MEM\_COMMIT,

PAGE\_READWRITE);

if (!lpc)

{

cout << "Virtual memory allocation failed." << endl;

return GetLastError();

}

cout << "Virtual memory address: " << lpc << endl;

// отменяем распределение

if (!VirtualFree(lpc, Kb, MEM\_DECOMMIT))

{

cout << "Memory decommit failed." << endl;

return GetLastError();

}

// освобождаем виртуальную память

if (!VirtualFree(lpr, 0, MEM\_RELEASE))

{

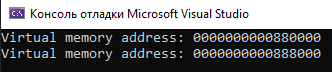
cout << "Memory release failed." << endl;

return GetLastError();

}

return 0;

}



// Листинг 21.4. Доступ к охраняемой странице виртуальной памяти

#include <windows.h>

#include <iostream>

using namespace std;

int main()

{

int\* a; // указатель на массив целых чисел

const int size = 1024; // размерность массива

// распределяем виртуальную память

a = (int\*)VirtualAlloc(

NULL,

size \* sizeof(int),

MEM\_COMMIT,

PAGE\_READWRITE | PAGE\_GUARD);

if (!a)

{

cout << "Virtual allocation failed." << endl;

return GetLastError();

}

\_\_try

{

a[10] = 10;

}

\_\_except (EXCEPTION\_EXECUTE\_HANDLER)

{

DWORD ecode = GetExceptionCode();

if (ecode == EXCEPTION\_GUARD\_PAGE)

cout << "Access to a guard virtual page." << endl;

else

cout << "Some exception." << endl;

}

cout << "a[10] = " << a[10] << endl;

a[10] = 10;

cout << "a[10] = " << a[10] << endl;

// освобождаем виртуальную память

if (!VirtualFree(a, 0, MEM\_RELEASE))

{

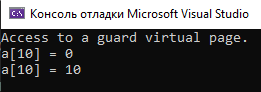
cout << "Memory release failed." << endl;

return GetLastError();

}

return 0;

}



// Листинг 21.5. Блокирование и разблокирование виртуальных страниц

#include <windows.h>

#include <iostream>

using namespace std;

int main()

{

LPVOID vm; // указатель на виртуальную память

SIZE\_T size = 4096; // размер памяти

// распределяем виртуальную память

vm = VirtualAlloc(

NULL,

size,

MEM\_COMMIT,

PAGE\_READWRITE);

if (!vm)

{

cout << "Virtual allocation failed." << endl;

return GetLastError();

}

// блокируем виртуальную память

if (!VirtualLock(vm, size))

{

cout << "Virtual lock failed." << endl;

return GetLastError();

}

// разблокируем виртуальную память

if (!VirtualUnlock(vm, size))

{

cout << "Virtual unlock failed." << endl;

return GetLastError();

}

// освобождаем виртуальную память

if (!VirtualFree(vm, 0, MEM\_RELEASE))

{

cout << "Memory release failed." << endl;

return GetLastError();

}

return 0;

}



// Листинг 21.6. Чтение и изменение рабочего множества виртуальных страниц процесса

#include <windows.h>

#include <iostream>

using namespace std;

int main()

{

DWORD dwOldProtect; // для старых атрибутов защиты

int\* a; // указатель на массив целых чисел

const int size = 1000; // размерность массива

// распределяем виртуальную память

a = (int\*)VirtualAlloc(

NULL,

size \* sizeof(int),

MEM\_COMMIT,

PAGE\_READONLY);

if (!a)

{

cout << "Virtual allocation failed." << endl;

return GetLastError();

}

// попробуем записать в виртуальную память

\_\_try

{

a[10] = 10;

}

\_\_except (EXCEPTION\_EXECUTE\_HANDLER)

{

DWORD ecode = GetExceptionCode();

if (ecode == EXCEPTION\_ACCESS\_VIOLATION)

cout << "Access to write protected page." << endl;

else

cout << "Some exception." << endl;

}

// изменим атрибуты доступа

if (!VirtualProtect(a, size, PAGE\_READWRITE, &dwOldProtect))

{

cout << "Virtual protect failed." << endl;

return GetLastError();

}

// теперь можно писать в виртуальную память

a[10] = 10;

cout << "a[10] = " << a[10] << endl;

// освобождаем виртуальную память

if (!VirtualFree(a, 0, MEM\_RELEASE))

{

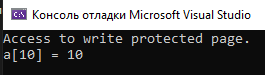
cout << "Memory release failed." << endl;

return GetLastError();

}

return 0;

}



// Листинг 21.7. Чтение и изменение рабочего множества виртуальных страниц процесса

#include <windows.h>

#include <iostream>

using namespace std;

int main()

{

const int size = 4096; // размер страницы

HANDLE hProcess; // дескриптор процесса

SIZE\_T min, max; // мин. и макс. размеры рабочего множества страниц

SIZE\_T\* pMin = &min; // указатель на минимальный размер

SIZE\_T\* pMax = &max; // указатель на максимальный размер

// получить дескриптор текущего процесса

hProcess = GetCurrentProcess();

// прочитать границы рабочего множества

if (!GetProcessWorkingSetSize(hProcess, pMin, pMax))

{

cout << "Get process working set size failed." << endl;

return GetLastError();

}

else

{

cout << "Min = " << (min / size) << endl;

cout << "Max = " << (max / size) << endl;

}

// установить новые границы рабочего множества

if (!SetProcessWorkingSetSize(hProcess, min - 10, max - 10))

{

cout << "Set process working set size failed." << endl;

return GetLastError();

}

// прочитать новые границы рабочего множества

if (!GetProcessWorkingSetSize(hProcess, pMin, pMax))

{

cout << "Get process working set size failed." << endl;

return GetLastError();

}

else

{

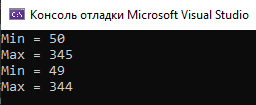
cout << "Min = " << (min / size) << endl;

cout << "Max = " << (max / size) << endl;

}

return 0;

}



// Листинг 21.8. Инициализация и копирование блоков виртуальной памяти

#include <windows.h>

#include <iostream>

using namespace std;

int main()

{

BYTE\* a, \* b; // указатели на блоки памяти

const int size = 1000; // размерность массива

// распределяем виртуальную память

a = (BYTE\*)VirtualAlloc(NULL, size, MEM\_COMMIT, PAGE\_READWRITE);

if (!a)

{

cout << "Virtual allocation failed." << endl;

return GetLastError();

}

b = (BYTE\*)VirtualAlloc(NULL, size, MEM\_COMMIT, PAGE\_READWRITE);

if (!b)

{

cout << "Virtual allocation failed." << endl;

return GetLastError();

}

// инициализируем символом X

FillMemory(a, size, 'X');

// копируем блок A в блок B

CopyMemory(b, a, size);

// распечатываем результат

cout << "b[10] = " << b[10] << endl;

// освобождаем виртуальную память

if (!VirtualFree(a, 0, MEM\_RELEASE))

{

cout << "Memory release failed." << endl;

return GetLastError();

}

return 0;

}



// Листинг 21.9. Инициализация и копирование блоков виртуальной памяти

#include <windows.h>

#include <iostream>

using namespace std;

int main()

{

BYTE\* a, \* b; // базовый адрес области и подобласти

const int size = 10000; // размер области

const int shift = 5000; // смещения для подобласти

MEMORY\_BASIC\_INFORMATION mbi; // структура для информации

// о виртуальной памяти

DWORD mbi\_size = sizeof(MEMORY\_BASIC\_INFORMATION);

// распределяем виртуальную память

a = (BYTE\*)VirtualAlloc(NULL, size, MEM\_COMMIT, PAGE\_READWRITE);

if (!a)

{

cout << "Virtual allocation failed." << endl;

return GetLastError();

}

// устанавливает адрес подобласти

b = a + shift;

// определяем информацию о виртуальной памяти

if (mbi\_size != VirtualQuery(b, &mbi, mbi\_size))

{

cout << "Virual query failed." << endl;

return GetLastError();

}

// распечатываем эту информацию

cout << "Base address: " << mbi.BaseAddress << endl;

cout << "Allocation base: " << mbi.AllocationBase << endl;

cout << "Allocation protect: " << mbi.AllocationProtect << endl;

cout << "Region size: " << mbi.RegionSize << endl;

cout << "State: " << mbi.State << endl;

cout << "Protect: " << mbi.Protect << endl;

cout << "Type: " << mbi.Type << endl;

// освобождаем виртуальную память

if (!VirtualFree(a, 0, MEM\_RELEASE))

{

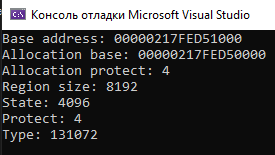
cout << "Memory release failed." << endl;

return GetLastError();

}

return 0;

}



// Листинг 21.10. Доступ к виртуальной памяти другого процесса

#include <windows.h>

#include <iostream>

#include "Lis 21.10.h"

using namespace std;

int main()

{

char c; // служебный символ

char lpszCommandLine[80]; // командная строка

char send[] = "This is a message."; // строка для пересылки

char buffer[80]; // буфер для ответа

LPVOID v = (LPVOID)0x00880000; // указатель на область памяти

HANDLE hWrite, hRead; // события для синхронизации

// записи-чтения в виртуальную память

char WriteEvent[] = "WriteEvent";

char ReadEvent[] = "ReadEvent";

// создаем события

hWrite = CreateEvent(NULL, FALSE, FALSE, LPWSTR(WriteEvent));

hRead = CreateEvent(NULL, FALSE, FALSE, LPWSTR(ReadEvent));

STARTUPINFO si;

PROCESS\_INFORMATION pi;

ZeroMemory(&si, sizeof(STARTUPINFO));

si.cb = sizeof(STARTUPINFO);

// формируем командную строку

wsprintf(LPWSTR(lpszCommandLine), L"X:\\МДК.01.04 Систем. прог\\Lis 21\\Lis 21.11\\x64\\Debug\\Lis 21.11.exe %d", (int)v);

// создаем новый консольный процесс

if (!CreateProcess(NULL, LPWSTR(lpszCommandLine), NULL, NULL, FALSE,

CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi))

{

cout << "Create process failed." << endl;

return GetLastError();

}

// распределяем виртуальную память в этом процессе

v = VirtualAllocEx(

pi.hProcess,

v,

sizeof(send),

MEM\_RESERVE | MEM\_COMMIT,

PAGE\_READWRITE);

if (!v)

{

cout << "Virtual allocation failed." << endl;

return GetLastError();

}

// записываем в виртуальную память сообщение

WriteProcessMemory(pi.hProcess, v, (void\*)send, sizeof(send), NULL);

// оповещаем о записи

SetEvent(hWrite);

// ждем сигнала на чтение

WaitForSingleObject(hRead, INFINITE);

// читаем ответ

ReadProcessMemory(pi.hProcess, v, (void\*)buffer, sizeof(buffer), NULL);

// выводим ответ

cout << buffer << endl;

// освобождаем виртуальную память

if (!VirtualFreeEx(pi.hProcess, v, 0, MEM\_RELEASE))

{

cout << "Memory release failed." << endl;

return GetLastError();

}

cout << "Input any char to exit: ";

cin >> c;

return 0;

}

// Листинг 21.11. Работа с виртуальной памятью, захваченной другим процессом

#include <windows.h>

#include <iostream>

using namespace std;

int main(int argc, char\* argv[])

{

char c;

char answer[] = "This is an answer.";

HANDLE hWrite, hRead; // события для синхронизации

char WriteEvent[] = "WriteEvent";

char ReadEvent[] = "ReadEvent";

char\* v; // для адреса виртуальной памяти

// открываем события

hWrite = OpenEvent(EVENT\_MODIFY\_STATE, FALSE, LPWSTR(WriteEvent));

hRead = OpenEvent(EVENT\_MODIFY\_STATE, FALSE, LPWSTR(ReadEvent));

// преобразуем параметр в адрес

v = argv[1];

// выводим сообщение

cout << v << endl;

// ждем разрешения на запись

WaitForSingleObject(hWrite, INFINITE);

// записываем ответ

//strcpy(v, "This is an answer.");

// разрешаем чтение

SetEvent(hRead);

// закрываем дескрипторы

CloseHandle(hWrite);

CloseHandle(hRead);

// ждем команды на завершение

cout << "Input any char to exit: ";

cin >> c;

return 0;

}

