**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Вычислительной техники**

**ОТЧЕТ**

**по лабораторной работе №2**

**по дисциплине «Операционные системы»**

**Тема: «Управление памятью»**

Студентки гр. 3312 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Поляков А.И.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тимофеев А.В.

Санкт-Петербург

2025

**Цель работы.**

исследовать механизмы управления виртуальной памятью Win32.

**Задание 2.1.**

Исследовать виртуальное адресное пространство

процесса.

Указания к выполнению.

1. Создайте консольное приложение с меню (каждая выполняемая

функция и/или операция должна быть доступна по отдельному пункту

меню), которое выполняет:

− получение информации о вычислительной системе (функция Win32 API – GetSystemInfo);

− определение статуса виртуальной памяти (функция Win32 API – GlobalMemoryStatus);

− определение состояния конкретного участка памяти по заданному

с клавиатуры адресу (функция Win32 API – VirtualQuery); − раздельное резервирование региона и передачу ему физической

памяти в автоматическом режиме и в режиме ввода адреса начала

региона (функция Win32 API – VirtualAlloc, VirtualFree); − одновременное резервирование региона и передача ему физической

памяти в автоматическом режиме и в режиме ввода адреса начала региона (функция Win32 API – VirtualAlloc, VirtualFree);

− запись данных в ячейки памяти по заданным с клавиатуры адресам;

− установку защиты доступа для заданного (с клавиатуры) региона

памяти и ее проверку (функция Win32 API – VirtualProtect).

2. Запустите

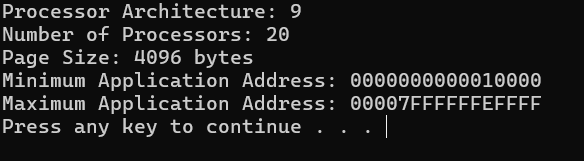
приложение и проверьте его работоспособность на нескольких наборах

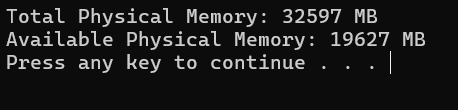
вводимых данных. Запротоколируйте результаты в отчет. Дайте свои

комментарии в отчете относительно выполнения функций Win32 API.

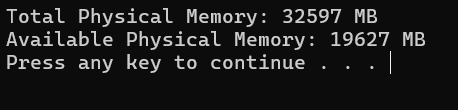
2. Подготовьте итоговый отчет с развернутыми выводами по заданию.

***Тестирование функций Win32 API***

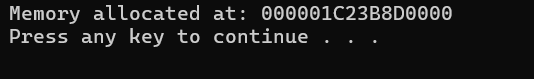
1. ***GetSystemInfoFunc*** Функция успешно получает информацию о системе, включая архитектуру процессора, количество процессоров, размер страницы памяти и диапазон доступных адресов памяти. Однако значение архитектуры процессора выводится в числовом формате, что затрудняет интерпретацию. 
2. ***GetMemoryStatusFunc*** Функция корректно отображает общий и доступный объем физической памяти.



1. **QueryMemoryStatus** Функция получает адрес от пользователя и использует VirtualQuery для запроса информации о памяти. Возможная ошибка — пользователь может ввести некорректный адрес, что приведет к сбою.



1. **ReserveAndCommitMemory** Функция успешно резервирует и коммитит 1 МБ памяти с правами PAGE\_READWRITE.



1. **ProtectMemory** Функция получает от пользователя адрес и пытается изменить его защиту на PAGE\_READONLY с помощью VirtualProtect.

**Вывод:** Функции Win32 API работают корректно, можно улучшить в обработке ошибок и удобстве вывода информации.

**Текст программы**

*Main.cpp*

#include <iostream>

#include <windows.h>

void GetSystemInfoFunc()

{

SYSTEM\_INFO si;

GetSystemInfo(&si);

std::cout << "Processor Architecture: " << si.wProcessorArchitecture << "\n";

std::cout << "Number of Processors: " << si.dwNumberOfProcessors << "\n";

std::cout << "Page Size: " << si.dwPageSize << " bytes\n";

std::cout << "Minimum Application Address: " << si.lpMinimumApplicationAddress << "\n";

std::cout << "Maximum Application Address: " << si.lpMaximumApplicationAddress << "\n";

}

void GetMemoryStatusFunc()

{

MEMORYSTATUS ms;

GlobalMemoryStatus(&ms);

std::cout << "Total Physical Memory: " << ms.dwTotalPhys / (1024 \* 1024) << " MB" << std::endl;

std::cout << "Available Physical Memory: " << ms.dwAvailPhys / (1024 \* 1024) << " MB" << std::endl;

}

void QueryMemoryStatus()

{

void\* address;

MEMORY\_BASIC\_INFORMATION mbi;

std::cout << "Enter memory address (hex): ";

std::cin >> address;

if (VirtualQuery(address, &mbi, sizeof(mbi)))

{

std::cout << "Base Address: " << mbi.BaseAddress << std::endl;

std::cout << "Allocation Protect: " << mbi.AllocationProtect << std::endl;

}

else

std::cerr << "VirtualQuery failed!" << std::endl;

}

void ReserveAndCommitMemory()

{

SIZE\_T size = 1024 \* 1024;

LPVOID mem = VirtualAlloc(NULL, size, MEM\_RESERVE | MEM\_COMMIT, PAGE\_READWRITE);

if (mem)

{

std::cout << "Memory allocated at: " << mem << std::endl;

VirtualFree(mem, 0, MEM\_RELEASE);

}

else

std::cerr << "Memory allocation failed!" << std::endl;

}

void ProtectMemory()

{

void\* address;

DWORD oldProtect;

std::cout << "Enter memory address to protect (hex): ";

std::cin >> address;

if (VirtualProtect(address, 4096, PAGE\_READONLY, &oldProtect))

std::cout << "Memory protection changed successfully!" << std::endl;

else

std::cerr << "VirtualProtect failed!" << std::endl;

}

int main()

{

int choice;

do

{

std::cout << "#================================#" << std::endl;

std::cout << "1 - Get System Info" << std::endl;

std::cout << "2 - Get Memory Status" << std::endl;

std::cout << "3 - Query Memory Status" << std::endl;

std::cout << "4 - Reserve and Commit Memory" << std::endl;

std::cout << "5 - Protect Memory" << std::endl;

std::cout << "0 - Exit" << std::endl;

std::cout << "#================================#" << std::endl;

std::cout << "Enter choice: ";

std::cin >> choice;

switch (choice)

{

case 1:

system("cls");

GetSystemInfoFunc();

system("pause");

system("cls");

break;

case 2:

system("cls");

GetMemoryStatusFunc();

system("pause");

system("cls");

break;

case 3:

system("cls");

QueryMemoryStatus();

system("pause");

system("cls");

break;

case 4:

system("cls");

ReserveAndCommitMemory();

system("pause");

system("cls");

break;

case 5:

system("cls");

ProtectMemory();

system("pause");

system("cls");

break;

case 0:

std::cout << "Exiting...\n";

break;

default:

std::cout << "Invalid choice!\n";

}

}

while (choice != 0);

return 0;

}