МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

ОТЧЕТ

по лабораторной работе № 2.1

по дисциплине «Операционные системы»

ТЕМА: «Управление памятью»

Студент гр. 3311 Баймухамедов Р. Р.

Преподаватель Тимофеев А. В.

Санкт-Петербург

2025

Цель работы

Исследовать механизмы управления виртуальной
памятью Win32.

Задание

Постановка задачи и описание решения

Для выполнения данной лабораторной работы необходимо разработать консольное приложение, которое позволяет:

- Получить информацию о вычислительной системе.
- Определить текущее состояние виртуальной памяти.
- Проанализировать статус заданного участка памяти.
- Выполнить резервирование и выделение физической памяти в разных режимах.
- Записывать данные по заданным адресам.
- Управлять защитой доступа к памяти.

В рамках эксперимента необходимо:

1. Определить характеристики системы
    - Получить информацию о вычислительной системе (функция GetSystemInfo).
    - Определить текущее состояние виртуальной памяти (GlobalMemoryStatus).
2. Анализ виртуальной памяти
    - Определить статус конкретного участка памяти по введённому адресу (VirtualQuery).
3. Работа с выделением памяти
    - Резервирование как в автоматическом режиме, так и с вводом адреса (VirtualAlloc, VirtualFree).
    - Одновременное резервирование и передача физической памяти в автоматическом режиме и по введённому адресу (VirtualAlloc).
4. Операции с памятью
    - Запись данных в память по введённому адресу.
    - Изменение защиты памяти на указанном участке (VirtualProtect).

В ходе выполнения работы было исследовано управление виртуальной памятью в Windows. Данный механизм предоставляет широкий набор возможностей, включая резервирование областей памяти, выделение физической памяти, изменение прав доступа и мониторинг состояния системы.

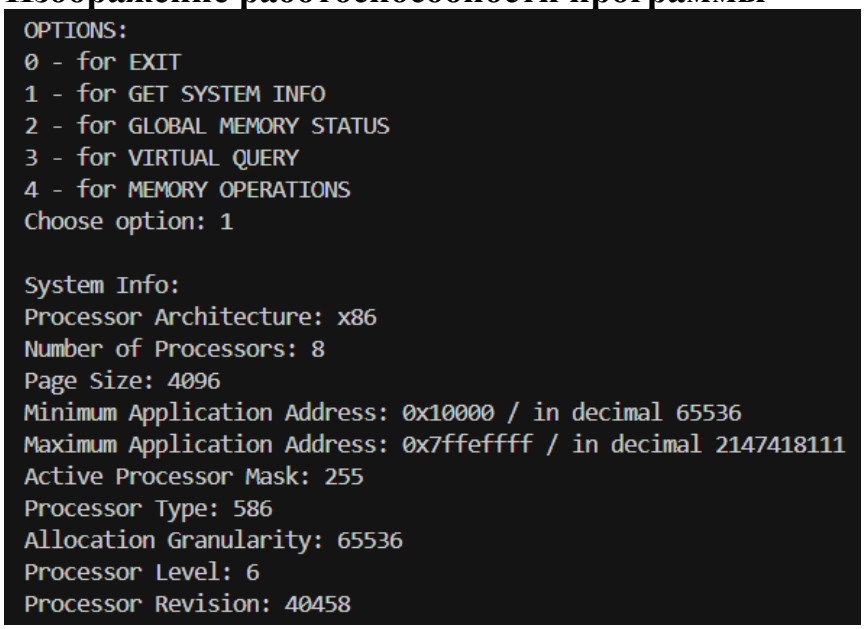
Одним из ключевых аспектов является возможность гибкого управления выделением памяти: можно заранее зарезервировать область и затем передавать ей физические ресурсы по мере необходимости. Это позволяет оптимизировать использование памяти и повышает предсказуемость работы приложения.

Дополнительно была изучена защита памяти, которая позволяет ограничивать доступ к определённым регионам, что может быть полезно как для безопасности, так и для контроля над работой приложения.

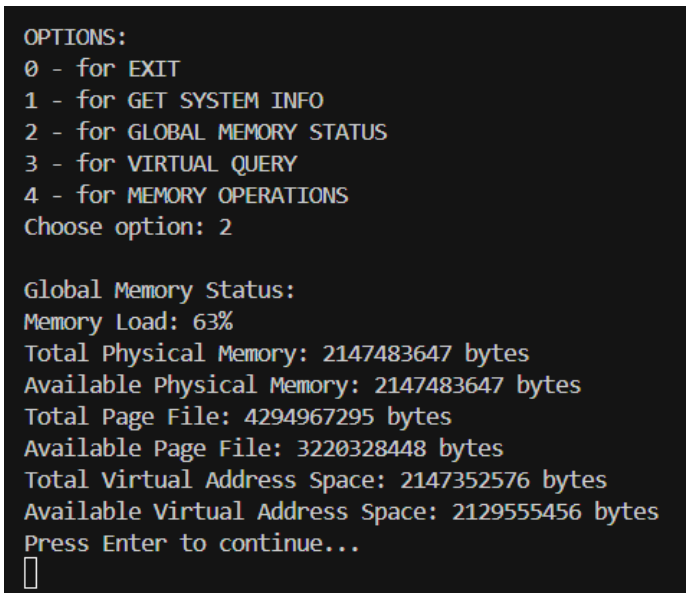
Также было исследовано получение системной информации, связанной с памятью, что позволяет анализировать доступные ресурсы и принимать более обоснованные решения при выделении памяти.

В целом, работа с виртуальной памятью через Win32 API даёт полный контроль над её использованием, что может быть полезно в различных сценариях, включая оптимизацию работы программ и обеспечение безопасности.

Изображение работоспособности программы



Изображение 1 – GET SYSTEM INFO



Изображение 2 – GLOBAL MEMORY STATUS

```
OPTIONS:
0 - for EXIT
1 - for GET SYSTEM INFO
2 - for GLOBAL MEMORY STATUS
3 - for VIRTUAL QUERY
4 - for MEMORY OPERATIONS
Choose option: 3
Enter the adress in range [65536, 2147418111]: 65536

Virtual Query:
Base Address: 0x10000 / in decimal 65536
Allocation Base: 0x10000 / in decimal 65536
Allocation Protect: PAGE_READONLY
Region Size: 69632
State: MEM_COMMIT
Protect: PAGE_READONLY
Type: MEM_MAPPED
Press Enter to continue...
```

Изображение 3 – VIRTUAL QUERY

```
MEMORY OPERATIONS:
0 - for EXIT
1 - for AUTOMATIC MEMORY RESERVATION
2 - for CUSTOM MEMORY RESERVATION
3 - for AUTOMATIC MEMORY RESERVATION AND COMMIT
4 - for CUSTOM MEMORY RESERVATION AND COMMIT
5 - for WRITE DATA TO MEMORY
6 - for SET MEMORY PROTECTION
7 - for FREE MEMORY BLOCK
Choose option: 1
Enter memory size in bytes: 7900
Size rounded to the nearest multiple of 4096: 8192 bytes
Memory successfully reserved:
Address: 0x7f0000 / in decimal 8323072
Size: 8192 bytes
Press Enter to continue...
```

Изображение 4.1 – Автоматическое резервирование памяти

```
OPTIONS:
0 - for EXIT
1 - for GET SYSTEM INFO
2 - for GLOBAL MEMORY STATUS
3 - for VIRTUAL QUERY
4 - for MEMORY OPERATIONS
Choose option: 3
Enter the adress in range [65536, 2147418111]: 8323072

Virtual Query:
Base Address: 0x7f0000 / in decimal 8323072
Allocation Base: 0x7f0000 / in decimal 8323072
Allocation Protect: PAGE_READWRITE
Region Size: 8192
State: MEM_RESERVE
Type: MEM_PRIVATE
Press Enter to continue...
```

Изображение 4.2 – Проверка адреса памяти

```
MEMORY OPERATIONS:
0 - for EXIT
1 - for AUTOMATIC MEMORY RESERVATION
2 - for CUSTOM MEMORY RESERVATION
3 - for AUTOMATIC MEMORY RESERVATION AND COMMIT
4 - for CUSTOM MEMORY RESERVATION AND COMMIT
5 - for WRITE DATA TO MEMORY
6 - for SET MEMORY PROTECTION
7 - for FREE MEMORY BLOCK
Choose option: 2
Enter the starting address (in range [65536, 2147418111]): 40960000
Address rounded to the nearest multiple of 4096. Address: 40960000 / in decimal 40960000
Enter memory size in bytes: 24000
Size rounded to the nearest multiple of 4096: 24576 bytes
Memory successfully reserved:
Requested address: 0x2710000
Allocated address: 0x2710000
Size: 24576 bytes
Press Enter to continue...
```

Изображение 5.1 – Резервирование адреса, вводимого пользователем

```
OPTIONS:
0 - for EXIT
1 - for GET SYSTEM INFO
2 - for GLOBAL MEMORY STATUS
3 - for VIRTUAL QUERY
4 - for MEMORY OPERATIONS
Choose option: 3
Enter the adress in range [65536, 2147418111]: 40960000

Virtual Query:
Base Address: 0x2710000 / in decimal 40960000
Allocation Base: 0x2710000 / in decimal 40960000
Allocation Protect: PAGE_READWRITE
Region Size: 24576
State: MEM_RESERVE
Type: MEM_PRIVATE
Press Enter to continue...
```

Изображение 5.2 – Проверка адреса памяти, резервируемого пользователем

```
MEMORY OPERATIONS:
0 - for EXIT
1 - for AUTOMATIC MEMORY RESERVATION
2 - for CUSTOM MEMORY RESERVATION
3 - for AUTOMATIC MEMORY RESERVATION AND COMMIT
4 - for CUSTOM MEMORY RESERVATION AND COMMIT
5 - for WRITE DATA TO MEMORY
6 - for SET MEMORY PROTECTION
7 - for FREE MEMORY BLOCK
Choose option: 5

Available memory blocks:
1. Address: 0x7f0000 / decimal 8323072, Size: 8192 bytes
2. Address: 0x2710000 / decimal 40960000, Size: 24576 bytes
3. Address: 0x770000 / decimal 7798784, Size: 12288 bytes
Choose memory block number: 3
Current memory protection: PAGE_READWRITE
Enter offset from the start of the block (0 to 12287): 0
Enter data to write (integer): 52
Data successfully written:
Address: 0x770000 / decimal 7798784
Value: 52
Press Enter to continue...
```

Изображение 6.1 – Запись данных по адресу памяти

```
2 - for CUSTOM MEMORY RESERVATION
3 - for AUTOMATIC MEMORY RESERVATION AND COMMIT
4 - for CUSTOM MEMORY RESERVATION AND COMMIT
5 - for WRITE DATA TO MEMORY
6 - for SET MEMORY PROTECTION
7 - for FREE MEMORY BLOCK
Choose option: 6

Available memory blocks:
1. Address: 0x7f0000 / decimal 8323072, Size: 8192 bytes
2. Address: 0x2710000 / decimal 40960000, Size: 24576 bytes
3. Address: 0x770000 / decimal 7798784, Size: 12288 bytes
Choose memory block number: 3

Available protection options:
1 - PAGE_READONLY
2 - PAGE_READWRITE
3 - PAGE_WRITECOPY
4 - PAGE_EXECUTE_READ
5 - PAGE_EXECUTE_READWRITE
6 - PAGE_EXECUTE_WRITECOPY
7 - PAGE_NOACCESS
8 - PAGE_EXECUTE
Choose protection type (1-8): 1

Protection successfully changed:
Address: 0x770000 / decimal 7798784
Size: 12288 bytes
Old protection: PAGE_READWRITE
New protection: PAGE_READONLY

Current protection (verified by VirtualQuery): PAGE_READONLY
```

Изображение 6.2 – Изменение прав доступа к адресу памяти

```
MEMORY OPERATIONS:
0 - for EXIT
1 - for AUTOMATIC MEMORY RESERVATION
2 - for CUSTOM MEMORY RESERVATION
3 - for AUTOMATIC MEMORY RESERVATION AND COMMIT
4 - for CUSTOM MEMORY RESERVATION AND COMMIT
5 - for WRITE DATA TO MEMORY
6 - for SET MEMORY PROTECTION
7 - for FREE MEMORY BLOCK
Choose option: 5

Available memory blocks:
1. Address: 0x7f0000 / decimal 8323072, Size: 8192 bytes
2. Address: 0x2710000 / decimal 40960000, Size: 24576 bytes
3. Address: 0x770000 / decimal 7798784, Size: 12288 bytes
Choose memory block number: 3
Current memory protection: PAGE_READONLY
Error: Memory is not writable. Current protection does not allow writing
Press Enter to continue...
```

Изображение 6.3 – Проверка изменения данных памяти по адресу с измененными правами доступа

Заключение

Работа с виртуальной памятью в Windows позволяет управлять выделением и защитой памяти на низком уровне. Можно как резервировать области памяти, так и передавать им физические ресурсы, а также менять права доступа, ограничивая чтение и запись. Кроме того, доступен инструмент для просмотра состояния памяти и общей загрузки системы, что помогает лучше понимать, как используются ресурсы. В целом, возможности управления памятью достаточно гибкие и позволяют более точно контролировать работу приложения.

Код программы

```
#include <string>
#include <iostream>
#include <windows.h>
#include <vector>
#include <limits>
using namespace std;

struct MemoryBlock {
   LPVOID address;
   SIZE_T size;
   bool is_committed;
};

vector<MemoryBlock> allocated_memory;

int enter_integer(const string& message, int a, int b) {
   int number;

   while (true) {
      cout << message;

      if (!(cin >> number)) {
         if (cin.eof()) {
            cout << "\nInput interrupted. Exiting...\n";
            exit(1);
         }
         cout << "Invalid input! Please enter a number\n";
         cin.clear();
         cin.ignore(numeric_limits<streamsize>::max(), '\n');
      } else if (number >= a && number <= b) {
         return number;
      } else {
         cout << "Entered value is out of range [" << a << ", " << b << "]. Try again!\n";
      }
   }
}

void clear_screen() {
#if defined(_WIN32) || defined(_WIN64)
   system("cls");
#else
   system("clear");
#endif
}

int main_menu() {
   cout << "\nOPTIONS:" << endl;
   cout << "0 - for EXIT" << endl;
   cout << "1 - for GET SYSTEM INFO" << endl;
   cout << "2 - for GLOBAL MEMORY STATUS" << endl;
   cout << "3 - for VIRTUAL QUERY" << endl;
   cout << "4 - for MEMORY OPERATIONS" << endl;
```

```
    return enter_integer("Choose option: ", 0, 5);
}

void get_system_info(){
    SYSTEM_INFO info;
    GetSystemInfo(&info);
    cout << "\nSystem Info:" << endl;
    int processor_architecture = info.wProcessorArchitecture;
    if (processor_architecture == PROCESSOR_ARCHITECTURE_INTEL) cout << "Processor Architecture: x86" << endl;
    if (processor_architecture == PROCESSOR_ARCHITECTURE_AMD64) cout << "Processor Architecture: x64" << endl;
    if (processor_architecture == PROCESSOR_ARCHITECTURE_ARM) cout << "Processor Architecture: ARM" << endl;
    if (processor_architecture == 12) cout << "Processor Architecture: ARM64" << endl;
    if (processor_architecture == PROCESSOR_ARCHITECTURE_IA64) cout << "Processor Architecture based on Itanium" <<
endl;
    if (processor_architecture == PROCESSOR_ARCHITECTURE_UNKNOWN) cout << "Unknown Processor Architecture" <<
endl;

    cout << "Number of Processors: " << info.dwNumberOfProcessors << endl;
    cout << "Page Size: " << info.dwPageSize << endl;
    cout << "Minimum Application Address: " << info.lpMinimumApplicationAddress << " / in decimal " <<
(DWORD)info.lpMinimumApplicationAddress << endl;
    cout << "Maximum Application Address: " << info.lpMaximumApplicationAddress << " / in decimal " <<
(DWORD)info.lpMaximumApplicationAddress << endl;
    cout << "Active Processor Mask: " << info.dwActiveProcessorMask << endl;
    cout << "Processor Type: " << info.dwProcessorType << endl;
    cout << "Allocation Granularity: " << info.dwAllocationGranularity << endl;
    cout << "Processor Level: " << info.wProcessorLevel << endl;
    cout << "Processor Revision: " << info.wProcessorRevision << endl;

    cout << "Press Enter to continue..." << endl;
    getchar();
    getchar();
}

void global_memory_status(){
    MEMORYSTATUS status;
    GlobalMemoryStatus(&status);
    cout << "\nGlobal Memory Status:" << endl;
    cout << "Memory Load: " << status.dwMemoryLoad << "%" << endl;
    cout << "Total Physical Memory: " << status.dwTotalPhys << " bytes" << endl;
    cout << "Available Physical Memory: " << status.dwAvailPhys << " bytes" << endl;
    cout << "Total Page File: " << status.dwTotalPageFile << " bytes" << endl;
    cout << "Available Page File: " << status.dwAvailPageFile << " bytes" << endl;
    cout << "Total Virtual Address Space: " << status.dwTotalVirtual << " bytes" << endl;
    cout << "Available Virtual Address Space: " << status.dwAvailVirtual << " bytes" << endl;

    cout << "Press Enter to continue..." << endl;
    getchar();
    getchar();
}

void get_virtual_query(){
    SYSTEM_INFO sys_info;
    GetSystemInfo(&sys_info);
    DWORD min_adress = (DWORD)sys_info.lpMinimumApplicationAddress;
    DWORD max_adress = (DWORD)sys_info.lpMaximumApplicationAddress;

    int adress;
    adress = enter_integer("Enter the adress in range [" + to_string(min_adress) + ", " + to_string(max_adress) + "]: ",
min_adress, max_adress);
    MEMORY_BASIC_INFORMATION info;
    if (VirtualQuery((void*)adress, &info, sizeof(info)) == 0) {
        cout << "Error! Invalid adress" << endl;
        return;
    }

    cout << "\nVirtual Query:" << endl;
    cout << "Base Address: " << info.BaseAddress << " / in decimal " << (DWORD)info.BaseAddress << endl;
    cout << "Allocation Base: " << info.AllocationBase << " / in decimal " << (DWORD)info.AllocationBase << endl;
```

```
    int protection = info.AllocationProtect;
    if (protection == PAGE_READONLY) cout << "Allocation Protect: PAGE_READONLY" << endl;
    if (protection == PAGE_READWRITE) cout << "Allocation Protect: PAGE_READWRITE" << endl;
    if (protection == PAGE_WRITECOPY) cout << "Allocation Protect: PAGE_WRITECOPY" << endl;
    if (protection == PAGE_EXECUTE_READ) cout << "Allocation Protect: PAGE_EXECUTE_READ" << endl;
    if (protection == PAGE_EXECUTE_READWRITE) cout << "Allocation Protect: PAGE_EXECUTE_READWRITE" << endl;
    if (protection == PAGE_EXECUTE_WRITECOPY) cout << "Allocation Protect: PAGE_EXECUTE_WRITECOPY" << endl;
    if (protection == PAGE_NOACCESS) cout << "Allocation Protect: PAGE_NOACCESS" << endl;
    if (protection == PAGE_EXECUTE) cout << "Allocation Protect: PAGE_EXECUTE" << endl;
    if (protection == 0x40000000) cout << "Allocation Protect: PAGE_TARGETS_INVALID" << endl;
    if (protection == 0x80000000) cout << "Allocation Protect: PAGE_TARGETS_NO_UPDATE" << endl;

    cout << "Region Size: " << info.RegionSize << endl;
    int state = info.State;
    if (state == MEM_COMMIT) cout << "State: MEM_COMMIT" << endl;
    if (state == MEM_FREE) cout << "State: MEM_FREE" << endl;
    if (state == MEM_RESERVE) cout << "State: MEM_RESERVE" << endl;

    int protect = info.Protect;
    if (protect == PAGE_READONLY) cout << "Protect: PAGE_READONLY" << endl;
    if (protect == PAGE_READWRITE) cout << "Protect: PAGE_READWRITE" << endl;
    if (protect == PAGE_WRITECOPY) cout << "Protect: PAGE_WRITECOPY" << endl;
    if (protect == PAGE_EXECUTE_READ) cout << "Protect: PAGE_EXECUTE_READ" << endl;
    if (protect == PAGE_EXECUTE_READWRITE) cout << "Protect: PAGE_EXECUTE_READWRITE" << endl;
    if (protect == PAGE_EXECUTE_WRITECOPY) cout << "Protect: PAGE_EXECUTE_WRITECOPY" << endl;
    if (protect == PAGE_NOACCESS) cout << "Protect: PAGE_NOACCESS" << endl;
    if (protect == PAGE_EXECUTE) cout << "Protect: PAGE_EXECUTE" << endl;
    if (protect == 0x40000000) cout << "Protect: PAGE_TARGETS_INVALID" << endl;
    if (protect == 0x80000000) cout << "Protect: PAGE_TARGETS_NO_UPDATE" << endl;

    int type = info.Type;
    if (type == MEM_IMAGE) cout << "Type: MEM_IMAGE" << endl;
    if (type == MEM_MAPPED) cout << "Type: MEM_MAPPED" << endl;
    if (type == MEM_PRIVATE) cout << "Type: MEM_PRIVATE" << endl;

    cout << "Press Enter to continue..." << endl;
    getchar();
    getchar();
}

void memory_operations_menu() {
    cout << "\nMEMORY OPERATIONS:" << endl;
    cout << "0 - for EXIT" << endl;
    cout << "1 - for AUTOMATIC MEMORY RESERVATION" << endl;
    cout << "2 - for CUSTOM MEMORY RESERVATION" << endl;
    cout << "3 - for AUTOMATIC MEMORY RESERVATION AND COMMIT" << endl;
    cout << "4 - for CUSTOM MEMORY RESERVATION AND COMMIT" << endl;
    cout << "5 - for WRITE DATA TO MEMORY" << endl;
    cout << "6 - for SET MEMORY PROTECTION" << endl;
    cout << "7 - for FREE MEMORY BLOCK" << endl;
}

void automatic_memory_reservation() {
    SYSTEM_INFO sys_info;
    GetSystemInfo(&sys_info);

    cout << "Enter memory size in bytes: ";
    SIZE_T size;
    cin >> size;

    SIZE_T remainder = size % 4096;
    if (remainder > 0) {
        size = size + (4096 - remainder);
        cout << "Size rounded to the nearest multiple of 4096: " << size << " bytes" << endl;
    }

    LPVOID address = VirtualAlloc(NULL, size, MEM_RESERVE, PAGE_READWRITE);

    if (address == NULL) {
```

```
        cout << "Error reserving memory. Error code: " << GetLastError() << endl;
        return;
    }

    MemoryBlock block;
    block.address = address;
    block.size = size;
    block.is_committed = false;
    allocated_memory.push_back(block);

    cout << "Memory successfully reserved:" << endl;
    cout << "Address: " << address << " / in decimal " << (DWORD)address << endl;
    cout << "Size: " << size << " bytes" << endl;

    cout << "Press Enter to continue..." << endl;
    getchar();
    getchar();
}

void custom_memory_reservation() {
    SYSTEM_INFO sys_info;
    GetSystemInfo(&sys_info);
    DWORD min_address = (DWORD)sys_info.lpMinimumApplicationAddress;
    DWORD max_address = (DWORD)sys_info.lpMaximumApplicationAddress;

    cout << "Enter the starting address (in range [" << min_address << ", " << max_address << "]): ";
    DWORD address;
    cin >> address;

    DWORD remainder = address % 4096;
    if (remainder > 2048) {
        address = address + (4096 - remainder);
    } else {
        address = address - remainder;
    }
    cout << "Address rounded to the nearest multiple of 4096. Address: " << address << " / in decimal " << DWORD(address) <<
endl;

    if (address < min_address || address > max_address) {
        cout << "Error: address is out of valid range" << endl;
        return;
    }

    cout << "Enter memory size in bytes: ";
    SIZE_T size;
    cin >> size;

    remainder = size % 4096;
    if (remainder > 0) {
        size = size + (4096 - remainder);
        cout << "Size rounded to the nearest multiple of 4096: " << size << " bytes" << endl;
    }

    LPVOID allocated_address = VirtualAlloc((LPVOID)address, size, MEM_RESERVE, PAGE_READWRITE);

    if (allocated_address == NULL) {
        cout << "Error reserving memory. Error code: " << GetLastError() << endl;
        return;
    }

    MemoryBlock block;
    block.address = allocated_address;
    block.size = size;
    block.is_committed = false;
    allocated_memory.push_back(block);

    cout << "Memory successfully reserved:" << endl;
    cout << "Requested address: " << (void*)address << endl;
    cout << "Allocated address: " << allocated_address << endl;
```

```
    cout << "Size: " << size << " bytes" << endl;

    cout << "Press Enter to continue..." << endl;
    getchar();
    getchar();
}

void automatic_memory_reservation_and_commit() {
    cout << "Enter memory size in bytes: ";
    SIZE_T size;
    cin >> size;

    SIZE_T remainder = size % 4096;
    if (remainder > 0) {
        size = size + (4096 - remainder);
        cout << "Size rounded to the nearest multiple of 4096: " << size << " bytes" << endl;
    }

    LPVOID address = VirtualAlloc(NULL, size, MEM_RESERVE | MEM_COMMIT, PAGE_READWRITE);

    if (address == NULL) {
        cout << "Error allocating memory. Error code: " << GetLastError() << endl;
        return;
    }

    MemoryBlock block;
    block.address = address;
    block.size = size;
    block.is_committed = true;
    allocated_memory.push_back(block);

    cout << "Memory successfully reserved and committed:" << endl;
    cout << "Address: " << address << "/ decimal " << DWORD(address) << endl;
    cout << "Size: " << size << " bytes" << endl;

    cout << "Press Enter to continue..." << endl;
    getchar();
    getchar();
}

void custom_memory_reservation_and_commit() {
    SYSTEM_INFO sys_info;
    GetSystemInfo(&sys_info);
    DWORD min_address = (DWORD)sys_info.lpMinimumApplicationAddress;
    DWORD max_address = (DWORD)sys_info.lpMaximumApplicationAddress;

    cout << "Enter the starting address (in range [" << min_address << ", " << max_address << "]): ";
    DWORD address;
    cin >> address;

    DWORD remainder = address % 4096;
    if (remainder > 2048) {
        address = address + (4096 - remainder);
    } else {
        address = address - remainder;
    }
    cout << "Address rounded to the nearest multiple of 4096. Address: " << address << " / in decimal " << DWORD(address) <<
endl;

    if (address < min_address || address > max_address) {
        cout << "Error: address is out of valid range" << endl;
        return;
    }

    cout << "Enter memory size in bytes: ";
    SIZE_T size;
    cin >> size;

    remainder = size % 4096;
```

```
    if (remainder > 0) {
        size = size + (4096 - remainder);
        cout << "Size rounded to the nearest multiple of 4096: " << size << " bytes" << endl;
    }

    LPVOID allocated_address = VirtualAlloc((LPVOID)address, size, MEM_RESERVE | MEM_COMMIT, PAGE_READWRITE);

    if (allocated_address == NULL) {
        cout << "Error allocating memory. Error code: " << GetLastError() << endl;
        return;
    }

    MemoryBlock block;
    block.address = allocated_address;
    block.size = size;
    block.is_committed = true;
    allocated_memory.push_back(block);

    cout << "Memory successfully reserved and committed:" << endl;
    cout << "Requested address: " << (void*)address << endl;
    cout << "Allocated address: " << allocated_address << endl;
    cout << "Size: " << size << " bytes" << endl;

    cout << "Press Enter to continue..." << endl;
    getchar();
    getchar();
}

void write_data_to_memory() {
    if (allocated_memory.empty()) {
        cout << "Error: No memory blocks allocated. Please allocate memory first" << endl;
        cout << "Press Enter to continue..." << endl;
        getchar();
        getchar();
        return;
    }

    cout << "\nAvailable memory blocks:" << endl;
    for (size_t i = 0; i < allocated_memory.size(); i++) {
        cout << i + 1 << ". Address: " << allocated_memory[i].address << " / decimal " << (DWORD)allocated_memory[i].address << ",
Size: " << allocated_memory[i].size << " bytes" << endl;
    }

    int block_index = enter_integer("Choose memory block number: ", 1, allocated_memory.size()) - 1;
    MemoryBlock& block = allocated_memory[block_index];

    MEMORY_BASIC_INFORMATION mbi;
    if (VirtualQuery(block.address, &mbi, sizeof(mbi))) {
        cout << "Current memory protection: ";
        switch (mbi.Protect) {
            case PAGE_READONLY: cout << "PAGE_READONLY"; break;
            case PAGE_READWRITE: cout << "PAGE_READWRITE"; break;
            case PAGE_WRITECOPY: cout << "PAGE_WRITECOPY"; break;
            case PAGE_EXECUTE_READ: cout << "PAGE_EXECUTE_READ"; break;
            case PAGE_EXECUTE_READWRITE: cout << "PAGE_EXECUTE_READWRITE"; break;
            case PAGE_EXECUTE_WRITECOPY: cout << "PAGE_EXECUTE_WRITECOPY"; break;
            case PAGE_NOACCESS: cout << "PAGE_NOACCESS"; break;
            case PAGE_EXECUTE: cout << "PAGE_EXECUTE"; break;
            default: cout << "Unknown";
        }
        cout << endl;

        if (mbi.Protect == PAGE_NOACCESS || mbi.Protect == PAGE_READONLY || mbi.Protect == PAGE_EXECUTE_READ) {
            cout << "Error: Memory is not writable. Current protection does not allow writing" << endl;
            cout << "Press Enter to continue..." << endl;
            getchar();
            getchar();
            return;
        }
```

```
    }

    cout << "Enter offset from the start of the block (0 to " << block.size - 1 << "): ";
    SIZE_T offset;
    cin >> offset;

    if (offset >= block.size) {
        cout << "Error: Offset is out of bounds!" << endl;
        cout << "Press Enter to continue..." << endl;
        getchar();
        getchar();
        return;
    }

    cout << "Enter data to write (integer): ";
    int data;
    cin >> data;

    if (!block.is_committed) {
        cout << "Error: Memory is not committed. Please commit memory first" << endl;
        cout << "Press Enter to continue..." << endl;
        getchar();
        getchar();
        return;
    }

    int* target_address = (int*)((BYTE*)block.address + offset);
    *target_address = data;

    cout << "Data successfully written:" << endl;
    cout << "Address: " << target_address << " / decimal " << (DWORD)target_address << endl;
    cout << "Value: " << *target_address << endl;

    cout << "Press Enter to continue..." << endl;
    getchar();
    getchar();
}

void free_all_memory() {
    cout << "\nFreeing all allocated memory..." << endl;
    for (const auto& block : allocated_memory) {
        if (VirtualFree(block.address, 0, MEM_RELEASE)) {
            cout << "Successfully freed memory at address: " << block.address << " / in decimal " << DWORD(block.address) << endl;
        } else {
            cout << "Failed to free memory at address: " << block.address << " / in decimal " << DWORD(block.address) << ". Error
code: " << GetLastError() << endl;
        }
    }
    allocated_memory.clear();
}

void set_memory_protection() {
    if (allocated_memory.empty()) {
        cout << "Error: No memory blocks allocated. Please allocate memory first" << endl;
        cout << "Press Enter to continue..." << endl;
        getchar();
        getchar();
        return;
    }

    cout << "\nAvailable memory blocks:" << endl;
    for (size_t i = 0; i < allocated_memory.size(); i++) {
        cout << i + 1 << ". Address: " << allocated_memory[i].address
            << " / decimal " << (DWORD)allocated_memory[i].address
            << ", Size: " << allocated_memory[i].size << " bytes" << endl;
    }

    int block_index = enter_integer("Choose memory block number: ", 1, allocated_memory.size()) - 1;
    MemoryBlock& block = allocated_memory[block_index];
```

```
cout << "\nAvailable protection options:" << endl;
cout << "1 - PAGE_READONLY" << endl;
cout << "2 - PAGE_READWRITE" << endl;
cout << "3 - PAGE_WRITECOPY" << endl;
cout << "4 - PAGE_EXECUTE_READ" << endl;
cout << "5 - PAGE_EXECUTE_READWRITE" << endl;
cout << "6 - PAGE_EXECUTE_WRITECOPY" << endl;
cout << "7 - PAGE_NOACCESS" << endl;
cout << "8 - PAGE_EXECUTE" << endl;

int protection_choice = enter_integer("Choose protection type (1-8): ", 1, 8);
DWORD new_protection;

switch (protection_choice) {
   case 1: new_protection = PAGE_READONLY; break;
   case 2: new_protection = PAGE_READWRITE; break;
   case 3: new_protection = PAGE_WRITECOPY; break;
   case 4: new_protection = PAGE_EXECUTE_READ; break;
   case 5: new_protection = PAGE_EXECUTE_READWRITE; break;
   case 6: new_protection = PAGE_EXECUTE_WRITECOPY; break;
   case 7: new_protection = PAGE_NOACCESS; break;
   case 8: new_protection = PAGE_EXECUTE; break;
   default: new_protection = PAGE_READWRITE;
}

DWORD old_protection;
if (VirtualProtect(block.address, block.size, new_protection, &old_protection)) {
   cout << "\nProtection successfully changed:" << endl;
   cout << "Address: " << block.address << " / decimal " << (DWORD)block.address << endl;
   cout << "Size: " << block.size << " bytes" << endl;

   cout << "Old protection: ";
   switch (old_protection) {
      case PAGE_READONLY: cout << "PAGE_READONLY"; break;
      case PAGE_READWRITE: cout << "PAGE_READWRITE"; break;
      case PAGE_WRITECOPY: cout << "PAGE_WRITECOPY"; break;
      case PAGE_EXECUTE_READ: cout << "PAGE_EXECUTE_READ"; break;
      case PAGE_EXECUTE_READWRITE: cout << "PAGE_EXECUTE_READWRITE"; break;
      case PAGE_EXECUTE_WRITECOPY: cout << "PAGE_EXECUTE_WRITECOPY"; break;
      case PAGE_NOACCESS: cout << "PAGE_NOACCESS"; break;
      case PAGE_EXECUTE: cout << "PAGE_EXECUTE"; break;
      default: cout << "Unknown";
   }
   cout << endl;

   cout << "New protection: ";
   switch (new_protection) {
      case PAGE_READONLY: cout << "PAGE_READONLY"; break;
      case PAGE_READWRITE: cout << "PAGE_READWRITE"; break;
      case PAGE_WRITECOPY: cout << "PAGE_WRITECOPY"; break;
      case PAGE_EXECUTE_READ: cout << "PAGE_EXECUTE_READ"; break;
      case PAGE_EXECUTE_READWRITE: cout << "PAGE_EXECUTE_READWRITE"; break;
      case PAGE_EXECUTE_WRITECOPY: cout << "PAGE_EXECUTE_WRITECOPY"; break;
      case PAGE_NOACCESS: cout << "PAGE_NOACCESS"; break;
      case PAGE_EXECUTE: cout << "PAGE_EXECUTE"; break;
      default: cout << "Unknown";
   }
   cout << endl;

   MEMORY_BASIC_INFORMATION mbi;
   if (VirtualQuery(block.address, &mbi, sizeof(mbi))) {
      cout << "\nCurrent protection (verified by VirtualQuery): ";
      switch (mbi.Protect) {
         case PAGE_READONLY: cout << "PAGE_READONLY"; break;
         case PAGE_READWRITE: cout << "PAGE_READWRITE"; break;
         case PAGE_WRITECOPY: cout << "PAGE_WRITECOPY"; break;
         case PAGE_EXECUTE_READ: cout << "PAGE_EXECUTE_READ"; break;
         case PAGE_EXECUTE_READWRITE: cout << "PAGE_EXECUTE_READWRITE"; break;
```

```
            case PAGE_EXECUTE_WRITECOPY: cout << "PAGE_EXECUTE_WRITECOPY"; break;
            case PAGE_NOACCESS: cout << "PAGE_NOACCESS"; break;
            case PAGE_EXECUTE: cout << "PAGE_EXECUTE"; break;
            default: cout << "Unknown";
        }
        cout << endl;
    }
  } else {
    cout << "Error changing protection. Error code: " << GetLastError() << endl;
  }

  cout << "Press Enter to continue..." << endl;
  getchar();
  getchar();
}

void free_memory_block() {
  if (allocated_memory.empty()) {
    cout << "Error: No memory blocks allocated. Please allocate memory first" << endl;
    cout << "Press Enter to continue..." << endl;
    getchar();
    getchar();
    return;
  }

  cout << "\nAvailable memory blocks:" << endl;
  for (size_t i = 0; i < allocated_memory.size(); i++) {
    cout << i + 1 << ". Address: " << allocated_memory[i].address
        << " / decimal " << (DWORD)allocated_memory[i].address
        << ", Size: " << allocated_memory[i].size << " bytes" << endl;
  }

  int block_index = enter_integer("Choose memory block number to free (1-" + to_string(allocated_memory.size()) + "): ", 1,
allocated_memory.size()) - 1;
  MemoryBlock& block = allocated_memory[block_index];

  if (VirtualFree(block.address, 0, MEM_RELEASE)) {
    cout << "Successfully freed memory block:" << endl;
    cout << "Address: " << block.address << " / decimal " << (DWORD)block.address << endl;
    cout << "Size: " << block.size << " bytes" << endl;

    allocated_memory.erase(allocated_memory.begin() + block_index);

    cout << "Memory block removed from the list" << endl;
  } else {
    cout << "Failed to free memory block. Error code: " << GetLastError() << endl;
  }

  cout << "Press Enter to continue..." << endl;
  getchar();
  getchar();
}

void memory_operations() {
  int option;
  do {
    memory_operations_menu();
    option = enter_integer("Choose option: ", 0, 7);
    switch (option) {
      case 1:
        automatic_memory_reservation();
        break;
      case 2:
        custom_memory_reservation();
        break;
      case 3:
        automatic_memory_reservation_and_commit();
        break;
      case 4:
```

```
            custom_memory_reservation_and_commit();
            break;
        case 5:
            write_data_to_memory();
            break;
        case 6:
            set_memory_protection();
            break;
        case 7:
            free_memory_block();
            break;
    }
  } while (option != 0);
}

int main(){
  int option;
  do {
    option = main_menu();
    switch (option) {
    case 1:
      get_system_info();
      break;
    case 2:
      global_memory_status();
      break;
    case 3:
      get_virtual_query();
      break;
    case 4:
      memory_operations();
      break;
    default:
      cout << "Invalid option. Please choose a valid option" << endl;
      break;
    }
  } while (option!=0);

  free_all_memory();

  cout << "Goodbye";
  return 0;
}
```