**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**ЛАБОРАТОРНАЯ РАБОТА №2**

**по дисциплине «Операционные системы»**

**Тема: “Управление памятью”**

| Студент |  |  |
| --- | --- | --- |
| Преподаватель |  | Тимофеев А. В. |

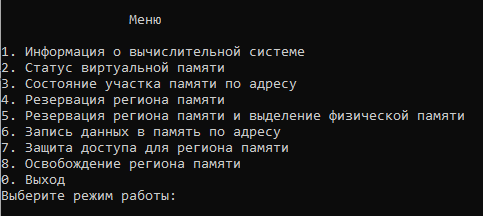
Санкт-Петербург

2023

**Цель работы:** исследовать механизмы управления виртуальной памятью Win32.

**Задание 2.1.** Исследовать виртуальное адресное пространство процесса.

Меню:



1. получение информации о вычислительной системе.

Для получения информации о вычислительной системе используется функция GetSystemInfo().

Параметры, которые выводятся в коде:

wProcessorArchitecture - это свойство структуры SYSTEM\_INFO в операционной системе Windows, которое указывает на архитектуру процессора текущей системы. Значение в этом поле может быть одним из следующих:

- PROCESSOR\_ARCHITECTURE\_INTEL (0) - процессор с архитектурой Intel (x86).

- PROCESSOR\_ARCHITECTURE\_AMD64 (9) - процессор с архитектурой AMD64 (x64).

- PROCESSOR\_ARCHITECTURE\_ARM (5) - процессор с архитектурой ARM.

- PROCESSOR\_ARCHITECTURE\_ARM64 (12) - процессор с архитектурой ARM64.

dwPageSize - это поле структуры SYSTEM\_INFO, которое хранит размер страницы памяти в байтах. Страница памяти является минимальной единицей управления памятью в операционной системе.

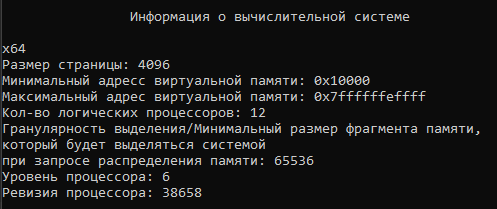
lpMinimumApplicationAddress и lpMaximumApplicationAddress - это поля структуры SYSTEM\_INFO, которые указывают на минимальный и максимальный адреса доступной виртуальной памяти для приложения соответственно. Эти адреса определяют диапазон, в пределах которого приложение может работать со своей виртуальной памятью.

dwNumberOfProcessors - это поле структуры SYSTEM\_INFO, показывающее общее количество процессоров в системе.

dwAllocationGranularity - это поле структуры SYSTEM\_INFO, которое содержит константу, определяющую гранулярность выделения виртуальной памяти. Гранулярность выделения определяет минимальный размер блока, который можно выделить или освободить в виртуальной памяти.

wProcessorLevel - это поле структуры SYSTEM\_INFO, которое указывает на уровень процессора в системе.

wProcessorRevision - это поле структуры SYSTEM\_INFO, которое указывает на ревизию процессора в системе.

Все эти поля предоставляют информацию о процессоре и памяти текущей системы, которую можно использовать для адаптации программного обеспечения к конкретной архитектуре и возможностям процессора.

1. определение статуса виртуальной памяти.

Функция GlobalMemoryStatus является частью Windows API и предоставляет информацию о состоянии памяти в системе. Она принимает указатель на структуру MEMORYSTATUS, чтобы заполнить ее данными о памяти. Вызов GlobalMemoryStatus(&MEMORYSTATUS) заполняет структуру MEMORYSTATUS данными о текущем состоянии памяти

- dwMemoryLoad: Поле dwMemoryLoad в структуре MEMORYSTATUS операционной системы Windows указывает на процентное использование памяти в системе. Значение этого поля может находиться в диапазоне от 0 до 100, где 0 означает, что память в системе практически не используется, а 100 означает, что память полностью загружена.

- dwTotalPhys: Поле dwTotalPhys в структуре MEMORYSTATUS указывает на общий объем физической памяти (в байтах) в системе. Он представляет собой общий объем физической памяти, доступной для работы приложений и операционной системы.

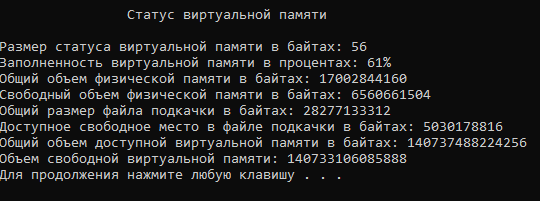
- dwAvailPhys: Поле dwAvailPhys в структуре MEMORYSTATUS указывает на доступный объем физической памяти (в байтах) в системе. Это количество памяти, которое свободно и может быть использовано для загрузки дополнительных данных или процессов.

- dwTotalPageFile: Поле dwTotalPageFile в структуре MEMORYSTATUS указывает на общий объем файла подкачки (в байтах), который доступен в системе. Файл подкачки используется операционной системой для расширения доступной памяти путем выгрузки неиспользуемых данных на жесткий диск.

- dwAvailPageFile: Поле dwAvailPageFile в структуре MEMORYSTATUS указывает на доступный объем файла подкачки (в байтах) в системе. Это количество места в файле подкачки, которое свободно и может быть использовано операционной системой при нехватке физической памяти.

- dwTotalVirtual: Поле dwTotalVirtual в структуре MEMORYSTATUS указывает на общий объем виртуальной памяти (в байтах), доступной в системе. Виртуальная память представляет собой комбинацию используемой физической памяти и файла подкачки.

- dwAvailVirtual: Поле dwAvailVirtual в структуре MEMORYSTATUS указывает на доступный объем виртуальной памяти (в байтах) в системе. Это количество памяти в виртуальном адресном пространстве, которое свободно и может быть выделено процессам при их запуске или расширении.



1. определение состояния конкретного участка памяти по заданному с клавиатуры адресу

Функция VirtualQuery является частью Windows API и предоставляет информацию о виртуальной памяти в процессе. Она принимает адрес виртуальной памяти и указатель на структуру MEMORY\_BASIC\_INFORMATION, чтобы заполнить ее данными о регионе виртуальной памяти.

Функция VirtualQuery возвращает размер структуры MEMORY\_BASIC\_INFORMATION, заполненной данными о регионе виртуальной памяти, в байтах. Если возвращаемое значение равно нулю, это означает, что переданный адрес виртуальной памяти был недействителен, и информация о виртуальной памяти не была получена.

Структура MEMORY\_BASIC\_INFORMATION содержит следующие поля:

- BaseAddress: Это поле содержит базовый адрес (начальный адрес) региона виртуальной памяти.

- AllocationBase: Это поле указывает на базовый адрес выделения региона виртуальной памяти. Если регион является частью выделения памяти, это поле будет указывать на базовый адрес этого выделения.

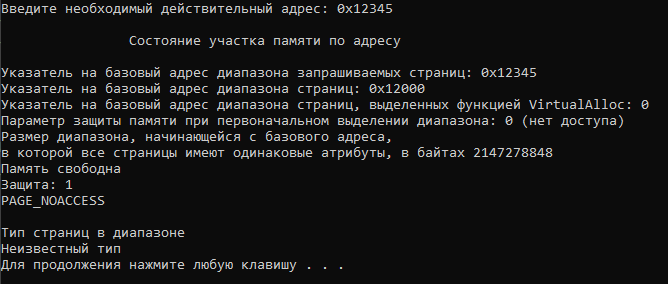
- AllocationProtect: Это поле содержит защиту памяти, примененную к региону выделения.

- RegionSize: Это поле указывает на общий размер региона виртуальной памяти.

- State: Это поле указывает на текущее состояние региона виртуальной памяти, такое как MEM\_COMMIT (выделено), MEM\_RESERVE (зарезервировано) или MEM\_FREE (свободно).

- Protect: Это поле указывает на текущую защиту памяти, примененную к региону.

- Type: Это поле указывает на тип региона виртуальной памяти, такой как MEM\_PRIVATE (частный) или MEM\_MAPPED (соответствующий отображаемому файлу).



1. резервирование региона в автоматическом режиме и в режиме ввода адреса начала региона.

VirtualAlloc(address, SYSTEM\_INFO.dwPageSize, MEM\_RESERVE, PAGE\_READWRITE) используется для резервирования памяти в адресном пространстве процесса с помощью функции VirtualAlloc().

Аргументы этой функции:

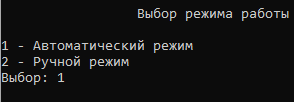
1. address: Это указатель на начальный адрес, по которому требуется зарезервировать память. Если address имеет значение NULL, система сама выберет адрес, с которого начнется резервирование памяти.

2. SYSTEM\_INFO.dwPageSize: Это размер страницы памяти в системе. Доступ к памяти обычно осуществляется путем чтения/записи целых страниц, поэтому резервируемая память будет выровнена по границам страницы.

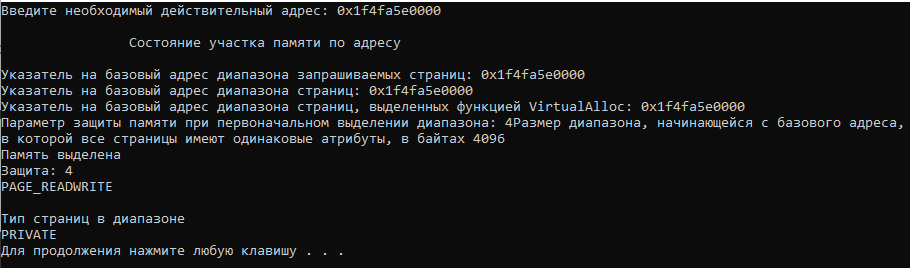
3. MEM\_RESERVE: Этот флаг указывает, что память должна быть зарезервирована, но не заполнена фактическими данными. Резервируется только адресное пространство без выделения реального физического пространства.

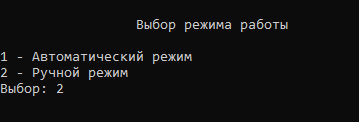
4. PAGE\_READWRITE: Этот флаг указывает, что зарезервированная память должна быть доступна для чтения/записи. Это означает, что процесс будет иметь право читать и записывать данные в зарезервированную память.

Функция VirtualAlloc() возвращает указатель на начало зарезервированной области памяти. Если резервирование не удалось, функция возвращает NULL.

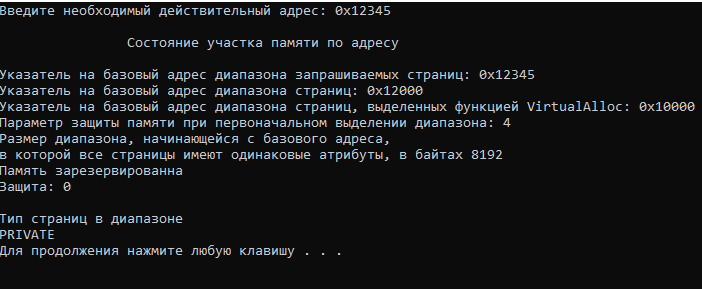












1. резервирование региона и передача ему физической памяти в автоматическом режиме и в режиме ввода адреса начала региона.

VirtualAlloc(address, SYSTEM\_INFO.dwPageSize, MEM\_RESERVE | MEM\_COMMIT, PAGE\_READWRITE) также используется для резервирования памяти с помощью функции VirtualAlloc(), но с добавлением флага MEM\_COMMIT.

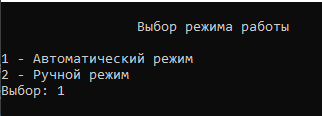
Функция VirtualAlloc используется для резервирования и выделения памяти в системе. Её аргументы включают:

1. Адрес (address): Это указатель на начальный адрес, откуда должна начаться резервация памяти. Если значение NULL, система сама выберет адрес.

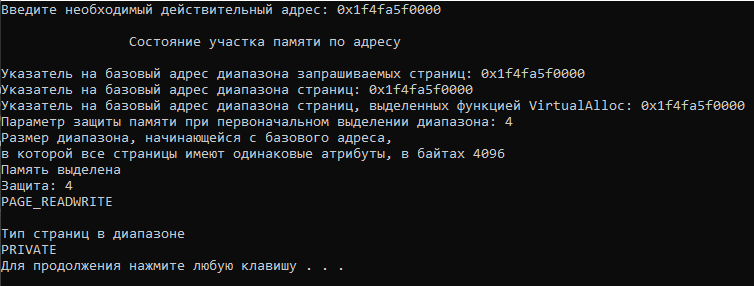
2. Размер страницы (dwPageSize) из структуры SYSTEM\_INFO: Виртуальная память обычно разбивается на фиксированные блоки, называемые страницами, размер которых определяется системными настройками. Резервируемая память будет выровнена по границам страницы.

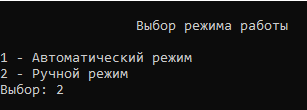
3. Флаги (dwAllocationType): Они указывают режим выделения памяти. Флаг MEM\_RESERVE указывает, что память должна быть зарезервирована без выделения физического пространства. Флаг MEM\_COMMIT указывает, что зарезервированная память должна быть немедленно выделена из физической памяти.

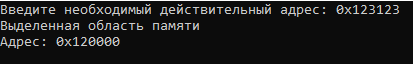
4. Флаг доступа (flProtect): Он указывает режим доступа к памяти. Например, PAGE\_READWRITE позволяет чтение и запись в выделенную память.

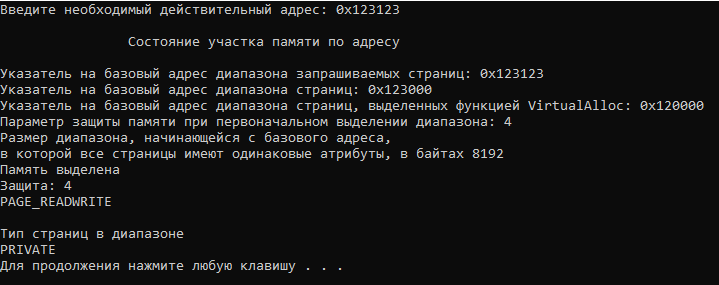












1. запись данных в ячейки памяти по заданным с клавиатуры адресам.

Функция CopyMemory используется для копирования данных из одного места в другое в памяти. Аргументы этой функции включают:

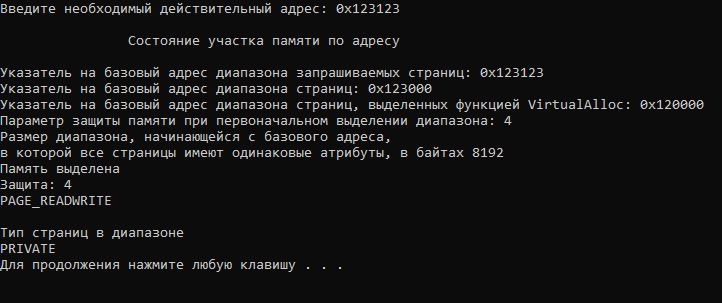
1. Адрес (address): Это указатель на начальный адрес, куда требуется скопировать данные.

2. Исходные данные (data.c\_str()): Данные, которые необходимо скопировать. В данном случае, data.c\_str() используется для получения указателя на внутреннюю часть объекта std::string, содержащего данные.

3. Размер данных (data.length() \* sizeof(char)): Размер данных, которые нужно скопировать, определяется путем умножения длины строки data на размер типа char в байтах. Это даст общий размер данных в байтах для копирования.

Таким образом, использование функции CopyMemory(address, data.c\_str(), data.length() \* sizeof(char)) позволяет скопировать данные из строки data в указанную область памяти, начиная с адреса address. Количество копируемых байтов будет определяться длиной строки data и размером типа char.





1. установку защиты доступа для заданного (с клавиатуры) региона памяти и ее проверку.

- PAGE\_EXECUTE: Этот уровень позволяет выполнение кода в памяти и предназначен для исполняемых страниц.

- PAGE\_EXECUTE\_READ: Этот уровень допускает выполнение и чтение данных в памяти и обычно используется для исполняемых страниц, которые также могут хранить данные.

- PAGE\_EXECUTE\_READWRITE: Этот уровень позволяет выполнение, чтение и запись данных в памяти и может использоваться для страниц, которые содержат код и данные, и которые могут изменяться.

- PAGE\_EXECUTE\_WRITECOPY: Этот уровень допускает выполнение и запись данный в память, но использует отдельные копии страниц при записи, обеспечивая копирование данных перед изменением.

- PAGE\_NOACCESS: Этот уровень запрещает доступ к памяти и обычно используется для неразрешенных или недоступных страниц.

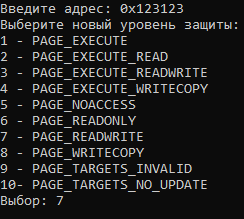
- PAGE\_READONLY: Этот уровень разрешает только чтение данных в памяти и не позволяет запись.

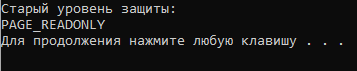
- PAGE\_READWRITE: Этот уровень позволяет чтение и запись данных в памяти и может использоваться для обычных страниц с данными.

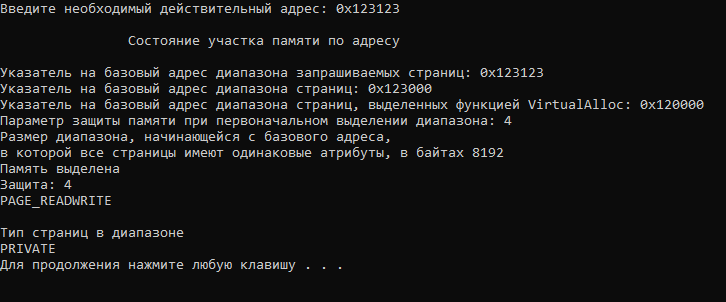
- PAGE\_WRITECOPY: Этот уровень разрешает запись данных в память, при этом используются отдельные копии страниц при записи, обеспечивая копирование данных перед изменением.

- PAGE\_TARGETS\_INVALID: Этот уровень указывает, что страницы памяти имеют недопустимую цель и не могут быть обновлены.

- PAGE\_TARGETS\_NO\_UPDATE: Этот уровень указывает, что страницы памяти не требуют обновления целей.







1. возврат физической памяти и освобождение региона адресного пространства заданного (с клавиатуры) региона памяти.

Функция VirtualFree предназначена для освобождения зарезервированной или выделенной памяти в рамках Windows API. Приведу параметры этой функции:

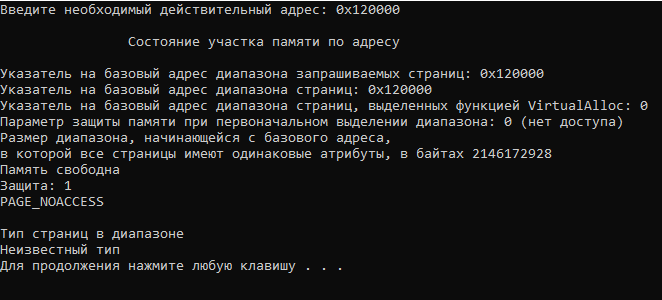
1. lpAddress: Указатель на начало области памяти, которую необходимо освободить. В общем случае, здесь передается адрес, полученный с помощью функции VirtualAlloc при резервировании или выделении памяти.

2. dwSize: Размер области памяти, которую требуется освободить. Если вы планируете освободить всю выделенную память, параметр dwSize должен быть установлен в 0.

3. dwFreeType: Флаг, указывающий, как именно следует освободить память. В данном случае используется флаг MEM\_RELEASE, который указывает на освобождение выделенной памяти и обнуление ее содержимого.

Итак, вызов функции VirtualFree(address, 0, MEM\_RELEASE) осуществляет освобождение всей памяти, начиная с адреса address. Функция освободит память и обнулит ее содержимое.





**Вывод:**

В ходе выполнения данной работы мы смогли углубиться в изучение Win32 API и его функций, предназначенных для работы с памятью. Мы успешно разработали консольное приложение, которое позволяет пользователям проверить функциональность данных функций.

Это приложение предоставляет пользователю возможность резервирования и выделения памяти с использованием функции VirtualAlloc. Затем, с помощью функции VirtualFree, пользователь может освободить зарезервированную или выделенную память.

Данные функции, такие как VirtualAlloc и VirtualFree, позволяют управлять памятью внутри процесса, обеспечивая ее эффективное использование. Они играют важную роль в разработке приложений, требующих динамического выделения и освобождения памяти.

Наши исследования и разработка приложения позволили нам лучше понять, как работает Win32 API и как использовать его функции для управления памятью. Мы получили ценный опыт, который может быть применен в будущих проектах, требующих работу с памятью.

**Задание 2.2.** Использование проецируемых файлов для обмена данными между процессами.

Необходимо создать два консольных приложения с меню (каждая выполняемая функция и/или операция должна быть доступна по отдельному пункту меню).

Первое приложение – это приложение-сервер, которое создает проецируемый

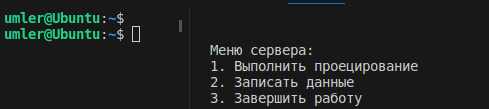
файл, проецирует фрагмент файла в память и осуществляет запись данных в

спроецированный файл.

Второе приложение – это приложение-клиент, которое открывает проецируемый файл, проецирует фрагмент файла в память и считывает содержимое из спроецированного файла и отображает на экран.

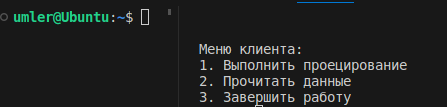
**Приложение-сервер:**

Создаётся файл с помощью функции open. Потом с помощью функции fallocate резервируется пространство на диске для указанного файла. Потом с помощью функции mmap происходит отображение файла в адресное пространство. Если не возникло ошибок, то приложение возвращает адрес спроецированного фрагмента. После записывает данные в проецируемый файл. Далее приложение-сервер после завершения прочтения файла клиентом отменяет отображения и удаляет файл.



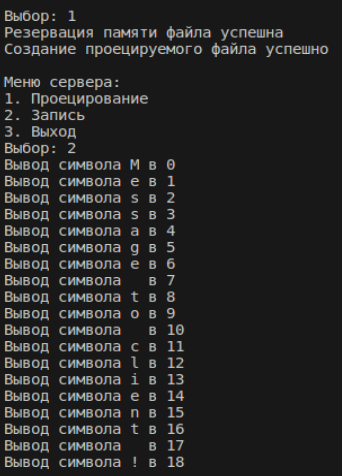
**Приложение-клиент:**

Нужно запустить это приложение, не завершая приложение-сервер. Далее открывает проецируемый файл открывается функцией open, его фрагмент проецируется функцией mmap в память. Данные, записанные в проецируемый файл в приложении-писателе, считываются приложением-читателем и выводятся на экран.

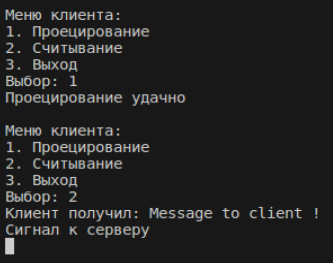


Процесс выполнения:

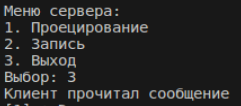
1. В сервере запустить выполнение проецирования, потом записать данные.



1. В клиенте запустить выполнение проецирования, а потом прочитать данные.



1. В сервере завершить работу.



**Вывод:**

Проецируемый файл представляет собой специальный тип файла, который может быть отображен в адресное пространство процесса операционной системы. При этом, содержимое файла становится доступным для чтения и записи, как если бы это была обычная память процесса. Такой механизм позволяет процессам обращаться к данным файла через указатель, без необходимости явно выполнять операции ввода-вывода для чтения или записи файла.

В случае проецируемого файла данные попадают в оперативную память следующим образом:

1. Сначала операционная система открывает файл и создает специальные структуры данных, которые будут использоваться для доступа к файлу. Эти структуры включают в себя таблицу страниц (page table) или дескриптор файловой системы, которые содержат информацию о местонахождении данных в файле.

2. При обращении к определенной области файла, операционная система осуществляет чтение соответствующих данных с диска и загружает их в оперативную память. Она может загрузить только часть файла или целиком, в зависимости от потребностей и доступного свободного места в оперативной памяти.

3. Загруженные данные хранятся в оперативной памяти в виде страниц или блоков памяти. Каждая страница имеет свой уникальный адрес или индекс, по которому можно обратиться к ней.

4. После того как данные загружены в оперативную память, они могут использоваться приложениями или другими процессами, которые требуют доступа к этим данным. Процессор может выполнять операции чтения и записи с загруженными данными, и операционная система обеспечивает доступ к этим данным через указатели страниц или другие методы индексации.

Таким образом, проецируемые файлы позволяют работать с данными, которые находятся на диске, как если бы они находились непосредственно в оперативной памяти. Это позволяет ускорить доступ к данным и упростить работу с файлами для приложений и процессов.

Когда файл проецируется в оперативную память, операционная система выделяет участок виртуальной памяти процесса и создает связь между этой областью в памяти и соответствующим фрагментом файла на диске. Таким образом, данные из файла становятся доступными в виртуальной памяти процесса и могут быть прочитаны или изменены непосредственно в памяти, без необходимости выполнять операции чтения или записи на диск.

Когда несколько программ проецируют один и тот же файл, каждая программа видит свою собственную виртуальную копию отображения файла. Это означает, что изменения, внесенные одной программой, не будут немедленно отражаться в проецированиях других программ. Вместо этого, изменения будут видны только в пределах собственного виртуального адресного пространства каждой программы.

Процесс использования проецируемых файлов может быть полезен для взаимодействия между разными процессами. Например, серверная программа может проецировать файл в память и записывать туда данные, а клиентская программа может отобразить этот же файл и прочитать данные из него. Это позволяет обмениваться информацией между программами без необходимости использовать механизмы межпроцессного взаимодействия, такие как сокеты или разделяемую память.

Реализация взаимодействия двух программ – клиента и сервера, с использованием проецируемого файла, предоставляет пример использования этого механизма. Этот подход обеспечивает быстрый доступ к данным и позволяет программам обмениваться информацией, не заботясь о низкоуровневых операциях чтения и записи на диск. При этом, данные могут быть переданы между процессами без лишней сложности и накладных расходов.

**Код программ:**

#include <iostream>

#include <windows.h>

#include <bitset>

#include <cstring>

#define CLS system("cls")

using namespace std;

void GetSystemInfoFunc(SYSTEM\_INFO&);

void GlobalMemoryStatusFunc();

void AllocatedMemoryStatusFunc();

void VirtualReserveFunc(SYSTEM\_INFO&);

void ReservetionWithCommitFunc(SYSTEM\_INFO&);

void WriteMemoryFunc();

void SetVirtualProtectionFunc();

void VirtualFreeFunc();

void menu(int& option);

int safe\_cin();

void choose\_protection(DWORD&);

void protect\_info(const DWORD&);

int main() {

setlocale(LC\_ALL, "Rus");

int option = -1;

SYSTEM\_INFO SYSTEM\_INFO;

GetSystemInfo(&SYSTEM\_INFO);

do {

menu(option);

switch (option) {

case 0:

break;

case 1:

GetSystemInfoFunc(SYSTEM\_INFO);

break;

case 2:

GlobalMemoryStatusFunc();

break;

case 3:

AllocatedMemoryStatusFunc();

break;

case 4:

VirtualReserveFunc(SYSTEM\_INFO);

break;

case 5:

ReservetionWithCommitFunc(SYSTEM\_INFO);

break;

case 6:

WriteMemoryFunc();

break;

case 7:

SetVirtualProtectionFunc();

break;

case 8:

VirtualFreeFunc();

break;

}

} while (option);

return 0;

}

void menu(int& option) {

option = -1;

while (option < 0 || option > 8) {

cout << "\n\t\tМеню\n\n";

cout << "1. Информация о вычислительной системе\n";

cout << "2. Статус виртуальной памяти\n";

cout << "3. Состояние участка памяти по адресу\n";

cout << "4. Резервация региона памяти\n";

cout << "5. Резервация региона памяти и выделение физической памяти\n";

cout << "6. Запись данных в память по адресу\n";

cout << "7. Защита доступа для региона памяти\n";

cout << "8. Освобождение региона памяти\n";

cout << "0. Выход\n";

cout << "Выберите режим работы: ";

option = safe\_cin();

CLS;

if (option < 0 || option > 8) cout << "Неверный режим работы\n";

}

}

int safe\_cin() {

int choice = -1;

char str[50];

cin >> str;

while (sscanf(str, "%d", &choice) != 1) {

cout << "Неправильный ввод! Используйте только цифры";

cin >> str;

}

return choice;

}

void GetSystemInfoFunc(SYSTEM\_INFO& SYSTEM\_INFO) {

WORD archType;

cout << "\n\t\tИнформация о вычислительной системе\n\n";

archType = SYSTEM\_INFO.wProcessorArchitecture;

if (archType == PROCESSOR\_ARCHITECTURE\_AMD64)

cout << "x64 \n";

else if (archType == PROCESSOR\_ARCHITECTURE\_ARM)

cout << "ARM\n";

else if (archType == PROCESSOR\_ARCHITECTURE\_IA64)

cout << "Intel Itanium-based\n";

else if (archType == PROCESSOR\_ARCHITECTURE\_INTEL)

cout << "x86\n";

else if (archType == PROCESSOR\_ARCHITECTURE\_UNKNOWN)

cout << "Неизвестная архитектура\n";

cout << "Размер страницы: " << SYSTEM\_INFO.dwPageSize << endl;

cout << "Минимальный адресс виртуальной памяти: " << SYSTEM\_INFO.lpMinimumApplicationAddress << endl;

cout << "Максимальный адрес виртуальной памяти: " << SYSTEM\_INFO.lpMaximumApplicationAddress << endl;

cout << "Кол-во логических процессоров: " << SYSTEM\_INFO.dwNumberOfProcessors << endl;

cout << "Гранулярность выделения/Минимальный размер фрагмента памяти,\nкоторый будет выделяться системой \nпри запросе распределения памяти: " << SYSTEM\_INFO.dwAllocationGranularity << endl;

cout << "Уровень процессора: " << SYSTEM\_INFO.wProcessorLevel << endl;

cout << "Ревизия процессора: " << SYSTEM\_INFO.wProcessorRevision << "\n\n";

system("pause");

CLS;

}

void GlobalMemoryStatusFunc() {

cout << "\n\t\tСостояние участка памяти по адресу\n\n";

MEMORYSTATUS MEMORYSTATUS;

GlobalMemoryStatus(&MEMORYSTATUS);

cout << "Размер статуса виртуальной памяти в байтах: " << MEMORYSTATUS.dwLength << endl;

cout << "Заполненность виртуальной памяти в процентах: " << MEMORYSTATUS.dwMemoryLoad << "%" << endl;

cout << "Общий объем физической памяти в байтах: " << MEMORYSTATUS.dwTotalPhys << endl;

cout << "Свободный объем физической памяти в байтах: " << MEMORYSTATUS.dwAvailPhys << endl;

cout << "Общий размер файла подкачки в байтах: " << MEMORYSTATUS.dwTotalPageFile << endl;

cout << "Доступное свободное место в файле подкачки в байтах: " << MEMORYSTATUS.dwAvailPageFile << endl;

cout << "Общий объем доступной виртуальной памяти в байтах: " << MEMORYSTATUS.dwTotalVirtual << endl;

cout << "Объем свободной виртуальной памяти: " << MEMORYSTATUS.dwAvailVirtual << endl;

system("pause");

CLS;

}

void AllocatedMemoryStatusFunc() {

LPVOID address;

MEMORY\_BASIC\_INFORMATION info;

DWORD state, type;

cout << "Введите необходимый действительный адрес: 0x";

cin >> std::hex >> address;

if (VirtualQuery(address, &info, sizeof(info)) == 0) {

cout << "Ошибка 0x" << GetLastError() << "\n";

system("pause");

CLS;

return;

}

cout << "\n\t\tСостояние участка памяти по адресу\n\n";

cout << "Указатель на базовый адрес диапазона запрашиваемых страниц: " << address << endl;

cout << "Указатель на базовый адрес диапазона страниц: " << info.BaseAddress << endl;

cout << "Указатель на базовый адрес диапазона страниц, выделенных функцией VirtualAlloc: " << info.AllocationBase << endl;

cout << "Параметр защиты памяти при первоначальном выделении диапазона: " <<

info.AllocationProtect << ((info.AllocationProtect != 0) ? "" : " (нет доступа)") << endl;

cout << "Размер диапазона, начинающейся с базового адреса, \nв которой все страницы имеют одинаковые атрибуты, в байтах " << info.RegionSize << endl;

state = info.State;

if (state == MEM\_COMMIT)

cout << "Память выделена\n";

else if (state == MEM\_FREE)

cout << "Память свободна\n";

else if (state == MEM\_RESERVE)

cout << "Память зарезервированна\n";

else cout << "Неизвестное состояние\n";

cout << "Защита: " << info.Protect << endl;

//?????? ????? ????? info\_protect(??? ?????? ???? ?????-?????? ?????????)

protect\_info(info.Protect);

cout << endl;

cout << "Тип страниц в диапазоне\n";

type = info.Type;

if (type == MEM\_IMAGE)

cout << "IMAGE\n";

else if (type == MEM\_MAPPED)

cout << "MAPPED\n";

else if (type == MEM\_PRIVATE)

cout << "PRIVATE\n";

else cout << "Неизвестный тип\n";

system("pause");

CLS;

}

void VirtualReserveFunc(SYSTEM\_INFO& SYSTEM\_INFO) {

int option = -1;

void\* address = nullptr;

while (option < 1 || option > 2) {

cout << "\n\t\t Выбор режима работы\n\n";

cout << "1 - Автоматический режим\n";

cout << "2 - Ручной режим\n";

cout << "Выбор: ";

option = safe\_cin();

CLS;

if (option < 1 || option > 2) cout << "Неверный ввод\n";

}

if (option == 2) {

cout << "Введите необходимый действительный адрес: 0x";

cin >> address;

}

address = VirtualAlloc(address, SYSTEM\_INFO.dwPageSize, MEM\_RESERVE, PAGE\_READWRITE);

if (address) {

if (address = VirtualAlloc(address, SYSTEM\_INFO.dwPageSize, MEM\_COMMIT, PAGE\_READWRITE))

cout << "Выделенная область памяти\nАдрес: " << address << endl;

else cout << "Ошибка 0x" << GetLastError() << endl << "Память не была выделена";

}

else cout << "Ошибка 0x" << GetLastError() << endl << "Память не была зарезервированна\n";

}

void ReservetionWithCommitFunc(SYSTEM\_INFO& SYSTEM\_INFO) {

int option = -1;

void\* address = nullptr;

while (option < 1 || option > 2) {

cout << "\n\t\t Выбор режима работы\n\n";

cout << "1 - Автоматический режим\n";

cout << "2 - Ручной режим\n";

cout << "Выбор: ";

option = safe\_cin();

CLS;

if (option < 1 || option > 2) cout << "Неверный ввод\n";

}

if (option == 2) {

cout << "Введите необходимый действительный адрес: 0x";

cin >> address;

}

if (address = VirtualAlloc(address, SYSTEM\_INFO.dwPageSize, MEM\_RESERVE | MEM\_COMMIT, PAGE\_READWRITE))

cout << "Выделенная область памяти\nАдрес: " << address << endl;

else cout << "Ошибка 0x" << GetLastError() << endl << "Память не была выделена\n";

}

void WriteMemoryFunc() {

LPVOID address = nullptr;

string data;

std::cout << "Введите адрес памяти: 0x";

cin >> std::hex >> address;

std::cout << "Введите данные: ";

cin >> data;

if (CopyMemory(address, data.c\_str(), data.length() \* sizeof(char))) cout << "Данные записаны" << endl;

else cout << "Память не была выделена. Ошибка:" << GetLastError() << endl;

}

void SetVirtualProtectionFunc() {

LPVOID address;

DWORD newLevel, oldLevel;

cout << "Введите адрес: 0x";

cin >> address;

if (!address) {

cout << "Адрес пуст\n";

system("pause");

return;

}

cout << "Выберите новый уровень защиты:\n";

choose\_protection(newLevel);

if (VirtualProtect(address, sizeof(DWORD), newLevel, &oldLevel)) {

cout << "Старый уровень защиты:\n";

protect\_info(oldLevel);

}

else cout << "Ошибка 0x" << GetLastError() << "\nДоступ запрещен\n";

system("pause");

}

void choose\_protection(DWORD& newLevel) {

newLevel = -1;

while (newLevel < 1 || newLevel > 10) {

//cout << "Enter the memory protection:\n";

cout << "1 - PAGE\_EXECUTE\n";

cout << "2 - PAGE\_EXECUTE\_READ\n";

cout << "3 - PAGE\_EXECUTE\_READWRITE\n";

cout << "4 - PAGE\_EXECUTE\_WRITECOPY\n";

cout << "5 - PAGE\_NOACCESS\n";

cout << "6 - PAGE\_READONLY\n";

cout << "7 - PAGE\_READWRITE\n";

cout << "8 - PAGE\_WRITECOPY\n";

cout << "9 - PAGE\_TARGETS\_INVALID\n";

cout << "10- PAGE\_TARGETS\_NO\_UPDATE\n";

cout << "Выбор: ";

newLevel = safe\_cin();

CLS;

if (newLevel < 1 || newLevel > 10) cout << "Неверный ввод\n";

}

if (newLevel == 1) newLevel = PAGE\_EXECUTE;

else if (newLevel == 2) newLevel = PAGE\_EXECUTE\_READ;

else if (newLevel == 3) newLevel = PAGE\_EXECUTE\_READWRITE;

else if (newLevel == 4) newLevel = PAGE\_EXECUTE\_WRITECOPY;

else if (newLevel == 5) newLevel = PAGE\_NOACCESS;

else if (newLevel == 6) newLevel = PAGE\_READONLY;

else if (newLevel == 7) newLevel = PAGE\_READWRITE;

else if (newLevel == 8) newLevel = PAGE\_WRITECOPY;

else if (newLevel == 9) newLevel = PAGE\_TARGETS\_INVALID;

else newLevel = PAGE\_TARGETS\_NO\_UPDATE;

}

void protect\_info(const DWORD& pro) {

if (pro & PAGE\_EXECUTE) cout << "PAGE\_EXECUTE\n";

if (pro & PAGE\_EXECUTE\_READ) cout << "PAGE\_EXECUTE\_READ\n";

if (pro & PAGE\_EXECUTE\_READWRITE) cout << "PAGE\_EXECUTE\_READWRITE\n";

if (pro & PAGE\_EXECUTE\_WRITECOPY) cout << "PAGE\_EXECUTE\_WRITECOPY\n";

if (pro & PAGE\_NOACCESS) cout << "PAGE\_NOACCESS\n";

if (pro & PAGE\_READONLY) cout << "PAGE\_READONLY\n";

if (pro & PAGE\_READWRITE) cout << "PAGE\_READWRITE\n";

if (pro & PAGE\_WRITECOPY) cout << "PAGE\_WRITECOPY\n";

if (pro & PAGE\_TARGETS\_INVALID) cout << "PAGE\_TARGETS\_INVALID\n";

if (pro & PAGE\_TARGETS\_NO\_UPDATE) cout << "PAGE\_TARGETS\_NO\_UPDATE\n";

}

void VirtualFreeFunc() {

LPVOID address = NULL;

cout << "Введите адрес: 0x";

cin >> address;

if (VirtualFree(address, 0, MEM\_RELEASE))

cout << "Область памяти освобождена\n";

else std::cerr << "Ошибка: 0x" << GetLastError() << endl;

system("pause");

}

**Server:**

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <sys/time.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/socket.h>

#include <sys/unistd.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#define FILENAME "file.txt"

int client\_read\_status() {

int sockfd = 0, n = 0;

char recvBuff[1024];

struct sockaddr\_in serv\_addr;

memset(recvBuff, '0', sizeof(recvBuff));

if((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

printf("\n Ошибка - невозможно создать сокет \n");

return 2;

}

memset(&serv\_addr, '0', sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(5000);

if(inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr)<=0)

{

printf("\n Ошибка - inet\_pton error occured\n");

return 2;

}

if( connect(sockfd, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0)

{

printf("\n Ошибка подключения\n");

return 2;

}

while ( (n = read(sockfd, recvBuff, sizeof(recvBuff)-1)) > 0)

{

recvBuff[n] = 0;

if(strcmp("readed", recvBuff) == 0) return 1;

printf("Буффер - %s\n", recvBuff);

}

if(n < 0) printf("\n Ошибка чтения \n");

}

int main() {

int choice;

int result;

int size = 4096;

int client\_status;

do {

printf("\nМеню сервера:\n");

printf("1. Проецирование\n");

printf("2. Запись\n");

printf("3. Выход\n");

printf("Выбор: ");

scanf("%d", &choice);

switch (choice) {

case 1:

int file = open(FILENAME, O\_RDWR | O\_CREAT, S\_IRUSR | S\_IWUSR);

int alloc = fallocate(file, 0, 0, size);

if(alloc == -1) printf("Ошибка резервации памяти файла\n");

else printf("Резервация памяти файла успешна\n");

char \*ptr = mmap(NULL, size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, file, 0);

if(ptr == MAP\_FAILED) printf("Ошибка создания проецируемого файла\n");

else printf("Создание проецируемого файла успешно\n");

break;

case 2:

char \*message = "Message to client !";

size = strlen(message);

for (int i = 0; i < size; i++)

{

printf("Вывод символа %c в %d\n", message[i], i);

ptr[i] = message[i];

}

break;

case 3:

client\_status = client\_read\_status();

if(client\_status == 1) printf("Клиент прочитал сообщение\n");

else if(client\_status == 2) printf("Ошибка сокета\n");

else printf("Ошибка кода %d", client\_status);

munmap(ptr, size);

unlink(FILENAME);

choice = 0;

break;

default:

printf("\n Неверное значение\n");

break;

}

} while (choice != 0);

return 0;

}

**Client:**

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <sys/time.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <arpa/inet.h>

#include <fcntl.h>

#define FILENAME "file.txt"

int client\_send\_status() {

int listenfd = 0, connfd = 0;

struct sockaddr\_in serv\_addr;

char sendBuff[1025];

time\_t ticks;

listenfd = socket(AF\_INET, SOCK\_STREAM, 0);

memset(&serv\_addr, '0', sizeof(serv\_addr));

memset(sendBuff, '0', sizeof(sendBuff));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

serv\_addr.sin\_port = htons(5000);

bind(listenfd, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr));

listen(listenfd, 10);

connfd = accept(listenfd, (struct sockaddr\*)NULL, NULL);

snprintf(sendBuff, sizeof(sendBuff), "readed");

write(connfd, sendBuff, strlen(sendBuff));

close(connfd);

}

int main() {

int choice;

int result;

int secret\_file;

char \*ptr;

fd\_set read\_fds;

struct timeval timeout;

struct stat statbuf;

do {

printf("\nМеню клиента:\n");

printf("1. Проецирование\n");

printf("2. Считывание\n");

printf("3. Выход\n");

printf("Выбор: ");

scanf("%d", &choice);

switch (choice) {

case 1:

secret\_file = open(FILENAME, O\_RDWR | O\_CREAT, S\_IRUSR | S\_IWUSR);

fstat(secret\_file, &statbuf);

ptr = mmap(NULL, statbuf.st\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, secret\_file, 0);

if(ptr == MAP\_FAILED) printf("Проецирование неудачно\n");

else printf("Проецирование удачно\n");

break;

case 2:

timeout.tv\_sec = 20;

FD\_ZERO(&read\_fds);

FD\_SET(secret\_file, &read\_fds);

result = select(secret\_file + 1, &read\_fds, NULL, NULL, &timeout);

if (result == -1) {

printf("Ошибка выбора\n");

}

else if (result == 0 ) {

printf("Время выполнения превышено\n");

}

else {

printf("Клиент получил: %s\n", ptr);

}

printf("Сигнал к серверу\n");

client\_send\_status();

break;

case 3:

munmap(ptr, statbuf.st\_size);

choice = 0;

break;

default:

printf("\n Неверное значение\n");

break;

}

} while (choice != 0);

return 0;

}