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»

по теме «Управление памятью»

|  |  |  |
| --- | --- | --- |
| Студентки гр. 2307 |  | Левушкина С.И. |
| Преподаватель |  | Тимофеев А.В. |

Санкт-Петербург, 2024

**СОДЕРЖАНИЕ**

[ЦЕЛЬ РАБОТЫ 3](#_Toc162889298)

[ЗАДАНИЯ 3](#_Toc162889299)

[1. Исследование виртуального адресного пространства процессора 3](#_Toc162889300)

[2. Приложения для клиента и сервера на Linux 4](#_Toc162889301)

[ЗАДАНИЕ 1 5](#_Toc162889302)

[Описание и примеры работы программы 5](#_Toc162889303)

[Меню (функция choice): 5](#_Toc162889304)

[Информация о вычислительной системе (функция system\_info): 5](#_Toc162889305)

[Определение статуса виртуальной памяти (функция memory\_status): 6](#_Toc162889306)

[Определение статуса участка памяти (функция block\_info): 7](#_Toc162889307)

[Резервирование региона (функция reserve\_block\_without\_memory): 7](#_Toc162889308)

[Резервирование региона и выделение физической памяти (функция reserve\_block): 8](#_Toc162889309)

[Запись данных в ячейку по заданному адресу (функция write\_in\_block) 9](#_Toc162889310)

[Освобождение ячейки памяти (функция free\_region) 10](#_Toc162889311)

[Установка защиты доступа для региона и проверка (функция est\_protect) 10](#_Toc162889312)

[Вывод 11](#_Toc162889313)

[ЗАДАНИЕ 2 12](#_Toc162889314)

[Описание и примеры работы программы 12](#_Toc162889315)

[Проецирование файла (функция project) 12](#_Toc162889316)

[Запись данных (функция write в программе сервера) 13](#_Toc162889317)

[Чтение данных (функция read в программе клиента) 15](#_Toc162889318)

[Завершение работы (функция stop) 15](#_Toc162889319)

[Вывод 17](#_Toc162889320)

[ИСХОДНЫЕ КОДЫ ПРОГРАММ 18](#_Toc162889321)

[Задание 1 18](#_Toc162889322)

[Задание 2 32](#_Toc162889323)

[1. Сервер 32](#_Toc162889324)

[2. Клиент 34](#_Toc162889325)

ЦЕЛЬ РАБОТЫ

Исследовать механизмы управления виртуальной

памятью Win32, механизмы взаимодействия клиент-сервер в сетевых приложениях.

ЗАДАНИЯ

1. Исследование виртуального адресного пространства процессора

Создать 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);
* Резервирование региона в автоматическом режиме и в режиме ввода адреса начала региона (функция Win32 API – VirtualAlloc);
* Резервирование региона и передача ему физической памяти в автоматическом режиме и в режиме ввода адреса начала региона (функция Win32 API – VirtualAlloc);
* 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);
* Возврат физической памяти и освобождение региона адресного

пространства заданного (с клавиатуры) региона памяти (функция

Win32 API –VirtualFree).

1. Приложения для клиента и сервера на Linux

Сервер создает файл на диске и проецирует его в память, далее записывает информацию и ждет,

пока клиент не прочтет, затем отменяет проецирование и удаляет файл.

Клиент открывает файл, проецирует и ждет доступности чтения, затем читает и выводит результат,

в завершении работы отменяет проецирование.

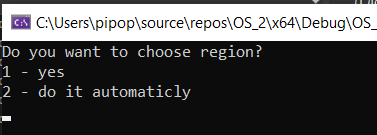
Сделать меню в каждой программе:

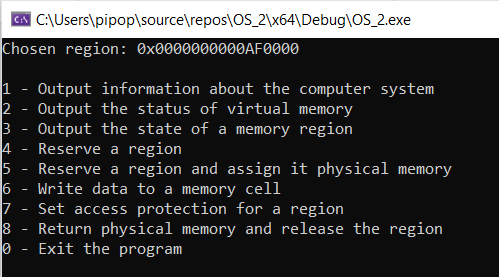
* пункты меню Сервера – «выполнить проецирование», «записать данные», «завершить работу»;
* пункты меню Клиента – «выполнить проецирование», «прочитать данные», «завершить работу».

ЗАДАНИЕ 1

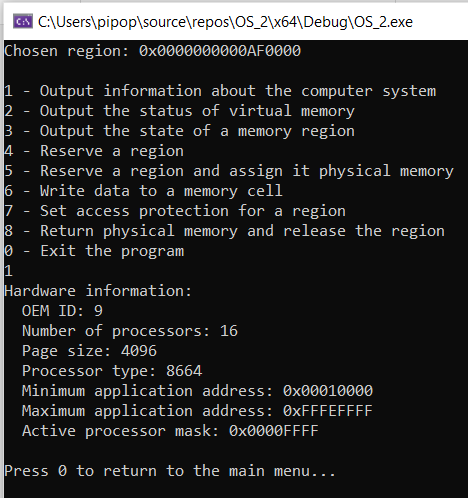
Описание и примеры работы программы

Меню (функция choice):



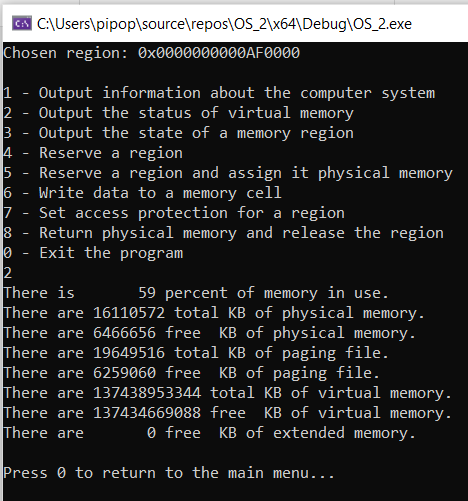


Информация о вычислительной системе (функция system\_info):



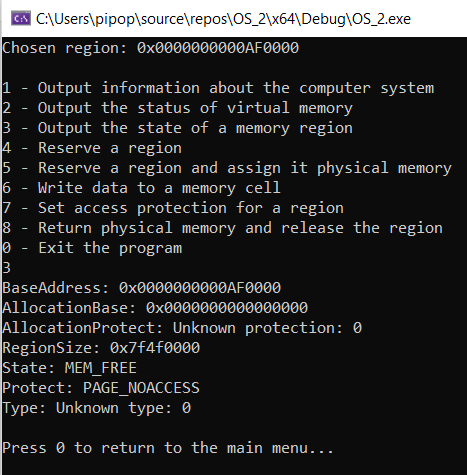
Выводит информацию о системной конфигурации компьютера (функция из win32 api “GetSystemInfo”), такую как количество процессоров, размер страницы памяти, тип процессора, минимальный и максимальный адреса доступные для приложений, и активную маску процессоров.

Определение статуса виртуальной памяти (функция memory\_status):



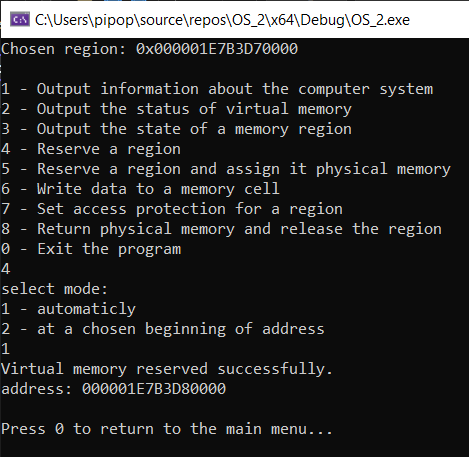
Выводит информацию о состоянии памяти в системе (функция из win32 api “GlobalMemoryStatusEx”), такую как количество использованной памяти, общее количество физической памяти, свободное количество физической памяти, общее количество виртуальной памяти, свободное количество виртуальной памяти, и свободное количество расширенной виртуальной памяти.

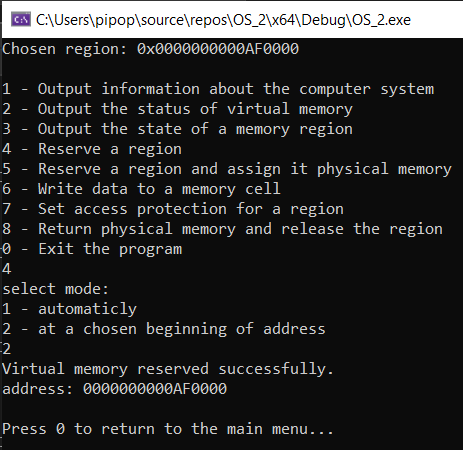
Определение статуса участка памяти (функция block\_info):



Выводит информацию о блоке памяти, на который указывает переданный указатель addr (функция из win32 api “VirtualQuery”). Информация включает в себя базовый адрес, адрес выделения, размер блока, тип блока, защиту блока и состояние блока.

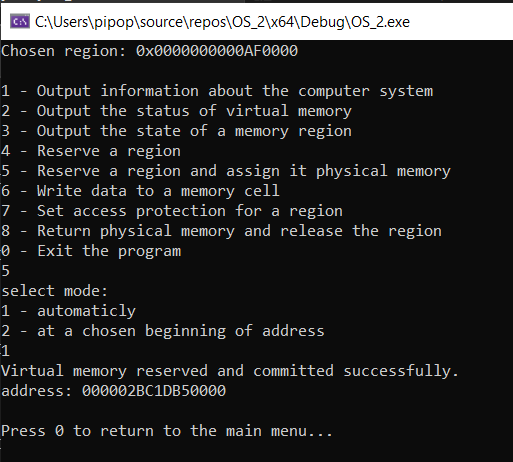
Резервирование региона (функция reserve\_block\_without\_memory):

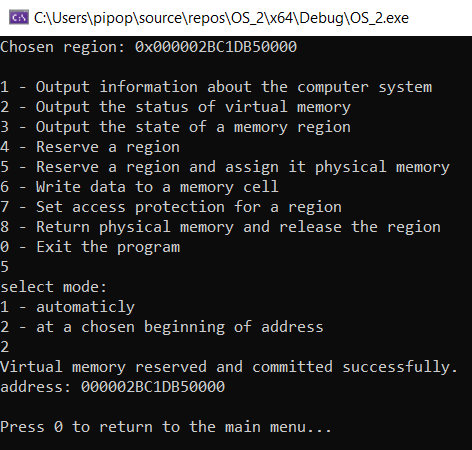




Выделяет блок памяти размером 4096 байт без инициализации физической памяти (функция из win32 api “VirtualAlloc” с флагом MEM\_RESERVE)

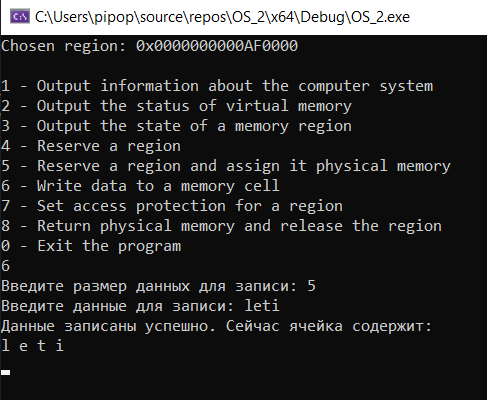
Резервирование региона и выделение физической памяти (функция reserve\_block):





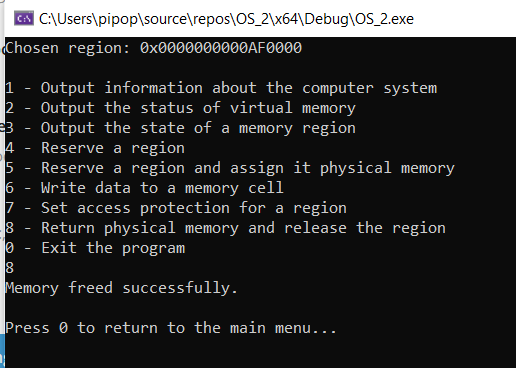
Выделяет блок памяти размером 4096 байт и инициализирует его значением 0 (функция из win32 api “VirtualAlloc” с флагами MEM\_RESERVE | MEM\_COMMIT)

Запись данных в ячейку по заданному адресу (функция write\_in\_block)



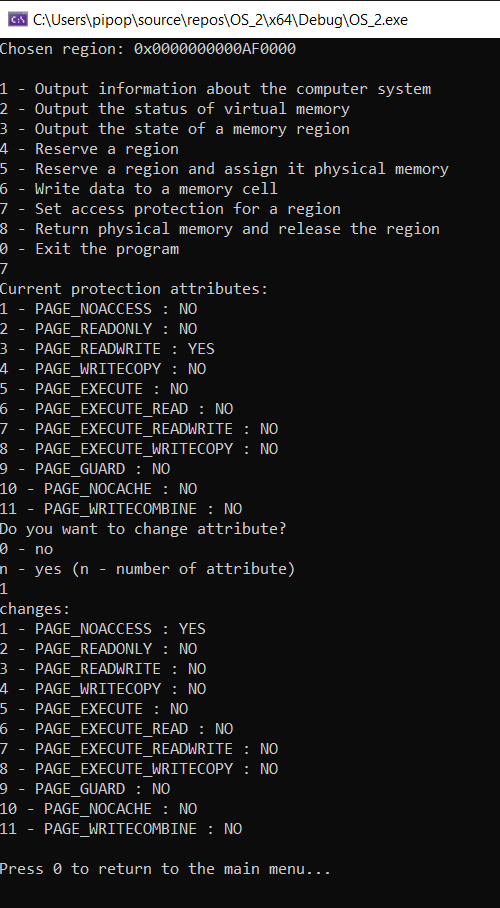
Записывает данные в блок памяти, на который указывает переданный указатель addr. Функция запрашивает пользовательский ввод размера данных и самих данных, а затем использует функцию WriteProcessMemory для записи данных в блок памяти.

Освобождение ячейки памяти (функция free\_region)



Освобождает блок памяти, на который указывает переданный указатель addr с помощью VirtualFree с флагом MEM\_RELEASE.

Установка защиты доступа для региона и проверка (функция est\_protect)



Изменяет атрибуты защиты памяти в вызывающем процессе. Она принимает один параметр - указатель на адрес памяти, для которого будут изменены атрибуты защиты.

Функция использует функцию VirtualQuery для получения текущих атрибутов защиты для указанного адреса памяти. Затем она выводит текущие их на консоль и запрашивает у пользователя, хочет ли он изменить один из этих атрибутов. Если пользователь вводит 0, функция возвращается к главному меню. Если пользователь вводит число от 1 до 11, функция устанавливает соответствующий атрибут защиты с помощью функции VirtualProtect.

Вывод

В ходе выполнения этого задания были получены навыки работы с памятью с помощью функций Win32 API и выявлены преимущества и недостатки этого подхода. Из преимуществ стоит отметить возможность работы с виртуальной памятью, что позволяет приложениям использовать больше памяти, чем доступно физически, возможность установки атрибутов защиты доступа к памяти, что позволяет предотвратить несанкционированный доступ к ней, большое разнообразие функций, которое позволяет реализовать множество задач и поэтому делает его мощным инструментом для разработки системного программного обеспечения. Но при этом функции Win32 API требуют внимательности и понимания устройства памяти, так как при разработке можно встретить ряд ошибок, связанных с попытками доступа к неверному адресу или использованию неправильных флагов в функциях, утечек памяти, что делает процесс разработки кропотливым.

ЗАДАНИЕ 2

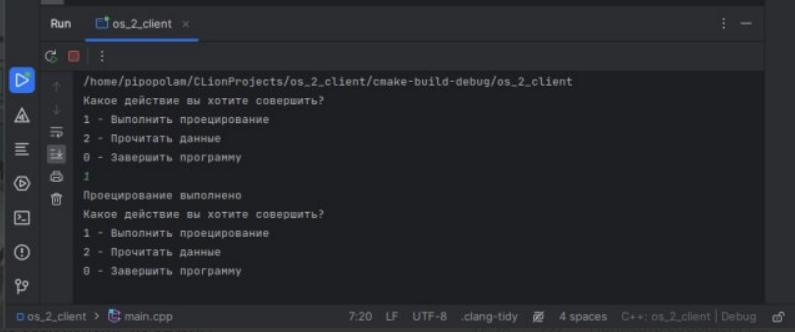
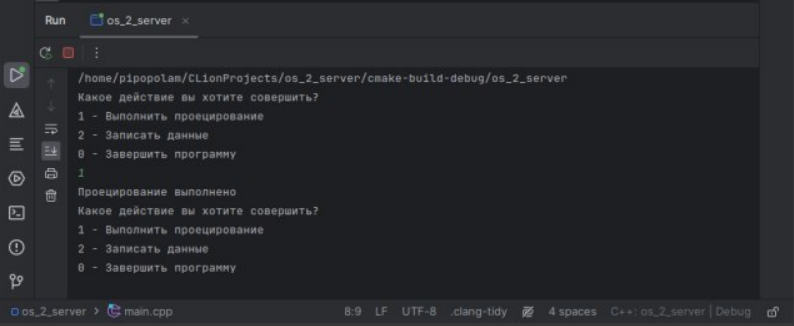
Описание и примеры работы программы

Проецирование файла (функция project)

Проецирование файла в память с помощью функции mmap позволяет клиенту и серверу работать с одним и тем же файлом в памяти, как будто это обычный массив байт. Когда клиент или сервер вызывает функцию project, она проецирует файл в виртуальное адресное пространство процесса, создавая область памяти, которая будет содержать содержимое файла.

Со стороны сервера, функция project вызывается с дескриптором файла, открытого для чтения и записи, и создает проекцию файла в памяти. Затем сервер может писать данные в эту область памяти, и клиент может читать эти данные из своей проекции файла в памяти.

Со стороны клиента, функция project также вызывается с дескриптором файла, открытого для чтения и записи, и создает проекцию файла в памяти. Затем клиент может читать данные из этой области памяти, и сервер может писать данные в свою проекцию файла в памяти.

Запись данных (функция write в программе сервера)

Функция write используется для записи строки "Hello shared memory!" в область памяти, которая была проецирована с помощью функции mmap.

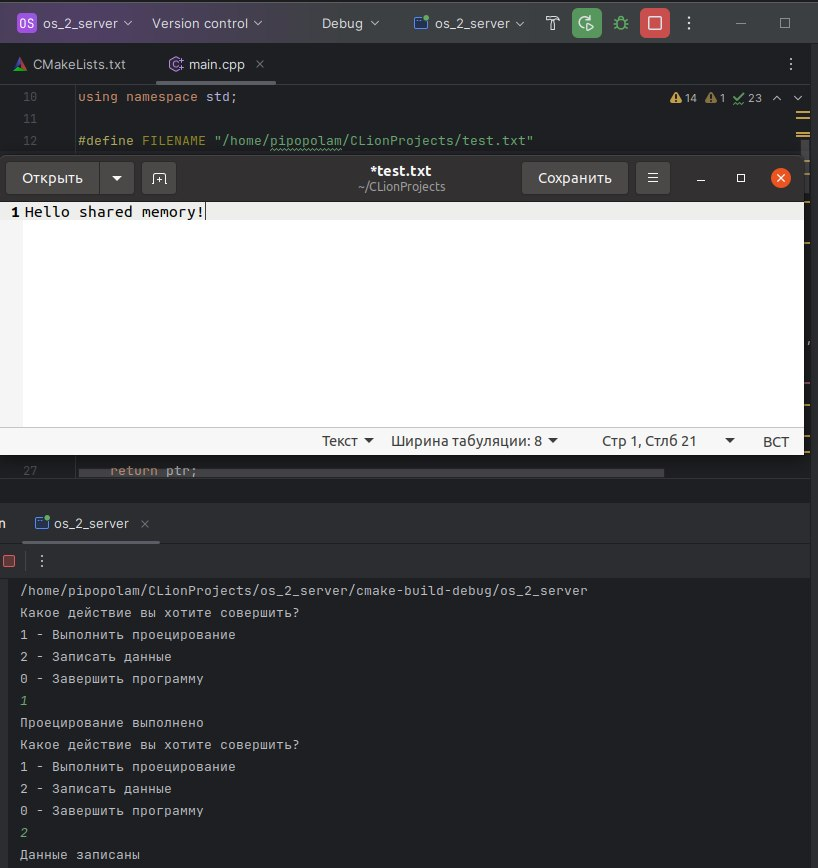
Первым делом функция проверяет, имеет ли указатель ptr значение MAP\_FAILED, которое возвращается функцией mmap в случае ошибки. Если значение указателя MAP\_FAILED, то функция выводит сообщение об ошибке и завершает работу программы с кодом возврата 1.

Затем функция проверяет, является ли указатель ptr нулевым. Если указатель ptr равен нулю, то функция выводит сообщение об ошибке и завершает работу программы с кодом возврата 1.

Если проверки пройдены успешно, то функция использует функцию sprintf для записи строки "Hello shared memory!" в область памяти, на которую указывает указатель ptr. Функция sprintf записывает строку в область памяти, начиная с адреса, на который указывает указатель ptr, и возвращает количество записанных символов.

После записи строки в область памяти функция не возвращает никакого значения и завершает свою работу.

Функция write должна быть вызвана после успешного вызова функции project, которая проецирует файл в память, и до вызова функции munmap, которая освобождает проекцию файла в памяти. Если функция write будет вызвана до вызова функции project или после вызова функции munmap, то программа выведет соответствующее сообщение и будет ожидать нового выбора действия.



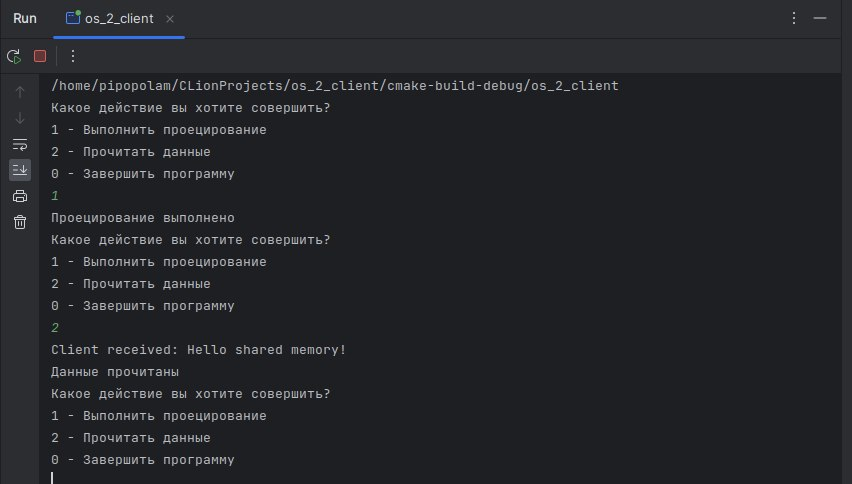
Чтение данных (функция read в программе клиента)

Функция read используется для чтения строки из области памяти, которая была проецирована с помощью функции mmap.

Первым делом функция устанавливает флаг read\_flag в значение 1, чтобы указать, что чтение из области памяти было успешно выполнено.

Затем функция использует функцию printf для вывода строки, которая была прочитана из области памяти, на консоль. Функция printf читает строку из области памяти, начиная с адреса, на который указывает указатель ptr, и выводит ее на консоль.

После вывода строки на консоль функция не возвращает никакого значения и завершает свою работу.



Завершение работы (функция stop)

Функция stop используется для освобождения проекции файла в памяти и закрытия файла, который был открыт с помощью функции open.

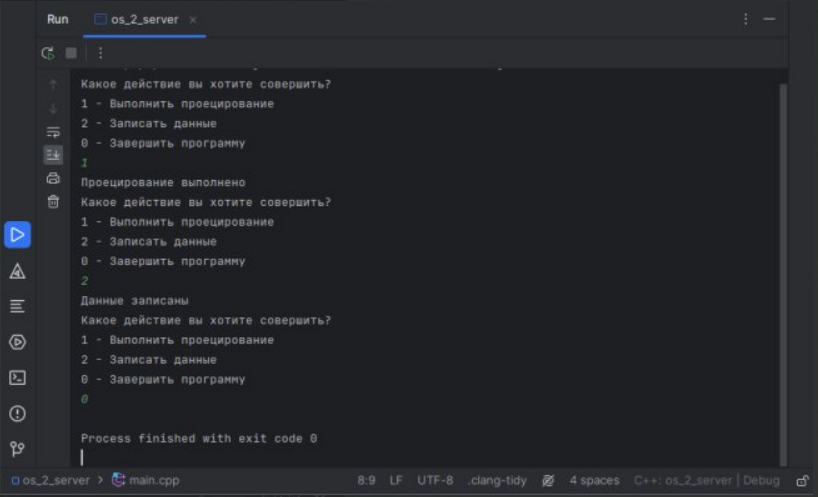
Первым делом функция вызывает функцию munmap для освобождения проекции файла в памяти. Функция munmap принимает в качестве аргументов указатель на начало проекции в памяти и размер проекции, и освобождает память, которая была выделена для проекции.

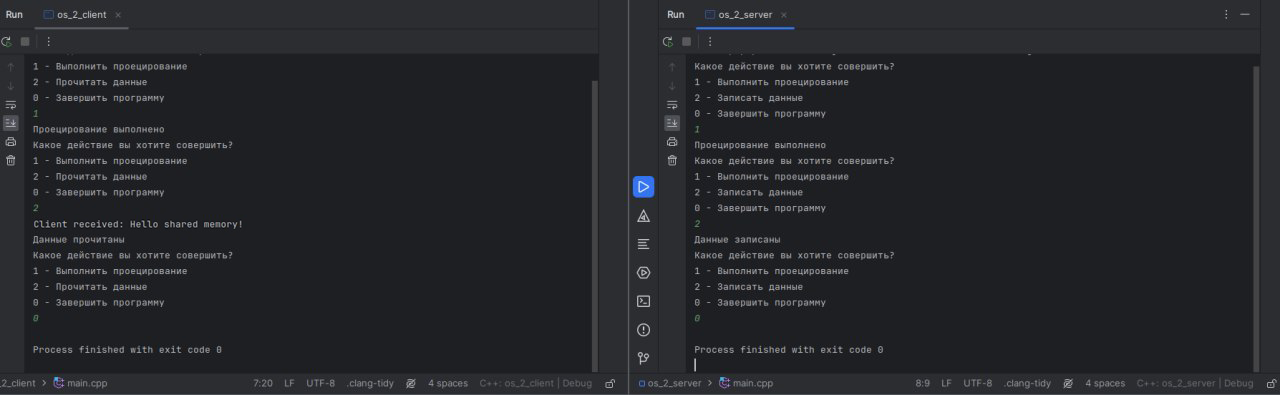
Если функция munmap возвращает значение -1, то это означает, что освобождение проекции в памяти не удалось, и функция выводит сообщение об ошибке и завершает работу программы с кодом возврата 1.

Затем функция вызывает функцию close для закрытия файла, который был открыт с помощью функции open. Функция close принимает в качестве аргумента дескриптор файла, который был возвращен функцией open, и закрывает файл.

В приложении клиента stop после закрытия файла функция устанавливает флаг stop\_flag в значение 1, чтобы указать, что работа с файлом была успешно завершена.

В приложении сервера функции stop после закрытия файла функция вызывает функцию unlink для удаления файла из файловой системы. Функция unlink принимает в качестве аргумента имя файла, которое было передано в функцию open, и удаляет файл из файловой системы.





Вывод

В ходе выполнения этого задания был реализован механизм обмена данными между клиентом и сервером в Linux с использованием проецируемых файлов. Проецирование файла в память позволяет обеспечить быстрый доступ к данным в файле, поскольку они находятся в оперативной памяти и могут быть обработаны как переменные. Кроме того, использование проецируемых файлов позволяет избежать необходимости копирования данных между процессами и уменьшает нагрузку на файловую систему.

В программах были реализованы функции для проецирования файла в память, записи и чтения данных в проецированный файл, а также освобождения проекции файла из памяти. Клиент и сервер используют эти функции для обмена данными через проецируемый файл.

ИСХОДНЫЕ КОДЫ ПРОГРАММ

Задание 1

#include <iostream>

#include <stdlib.h>

#include <windows.h>

#include <atlbase.h>

#include <atlconv.h>

#include <string>

#include <vector>

#include <map>

#include <locale>

using namespace std;

#define DIV 1024

#define WIDTH 7

DWORD dwSize = 4096;

void choose(LPVOID addr);

int auto\_or\_not()

{

int flag = 0;

cout << "Do you want to choose region?\n1 - yes\n2 - do it automaticly\n";

cin >> flag;

return flag;

}

void system\_info(LPVOID addr)

{

SYSTEM\_INFO siSysInfo;

GetSystemInfo(&siSysInfo);

printf("Hardware information: \n");

printf(" OEM ID: %u\n", siSysInfo.dwOemId);

printf(" Number of processors: %u\n",

siSysInfo.dwNumberOfProcessors);

printf(" Page size: %u\n", siSysInfo.dwPageSize);

printf(" Processor type: %u\n", siSysInfo.dwProcessorType);

printf(" Minimum application address: 0x%08X\n",

siSysInfo.lpMinimumApplicationAddress);

printf(" Maximum application address: 0x%08X\n",

siSysInfo.lpMaximumApplicationAddress);

printf(" Active processor mask: 0x%08X\n",

siSysInfo.dwActiveProcessorMask);

cout << "\nPress 0 to return to the main menu...";

int choice = -1;

cin >> choice;

while (choice != 0)

{

cout << "Error. Please enter 0 to return to the main menu.\n";

cin >> choice;

}

choose(addr);

}

void memory\_status(LPVOID addr)

{

MEMORYSTATUSEX statex;

statex.dwLength = sizeof(statex);

GlobalMemoryStatusEx(&statex);

\_tprintf(TEXT("There is %\*ld percent of memory in use.\n"),

WIDTH, statex.dwMemoryLoad);

\_tprintf(TEXT("There are %\*I64d total KB of physical memory.\n"),

WIDTH, statex.ullTotalPhys / DIV);

\_tprintf(TEXT("There are %\*I64d free KB of physical memory.\n"),

WIDTH, statex.ullAvailPhys / DIV);

\_tprintf(TEXT("There are %\*I64d total KB of paging file.\n"),

WIDTH, statex.ullTotalPageFile / DIV);

\_tprintf(TEXT("There are %\*I64d free KB of paging file.\n"),

WIDTH, statex.ullAvailPageFile / DIV);

\_tprintf(TEXT("There are %\*I64d total KB of virtual memory.\n"),

WIDTH, statex.ullTotalVirtual / DIV);

\_tprintf(TEXT("There are %\*I64d free KB of virtual memory.\n"),

WIDTH, statex.ullAvailVirtual / DIV);

\_tprintf(TEXT("There are %\*I64d free KB of extended memory.\n"),

WIDTH, statex.ullAvailExtendedVirtual / DIV);

cout << "\nPress 0 to return to the main menu...";

int choice = -1;

cin >> choice;

while (choice != 0)

{

cout << "Error. Please enter 0 to return to the main menu.\n";

cin >> choice;

}

choose(addr);

}

void block\_info(LPVOID addr)

{

LPVOID reservedMemory = NULL;

MEMORY\_BASIC\_INFORMATION info;

VirtualQuery(addr, &info, sizeof(info));

cout << "BaseAddress: 0x" << hex << info.BaseAddress << dec << endl;

cout << "AllocationBase: 0x" << hex << info.AllocationBase << dec << endl;

cout << "AllocationProtect: ";

switch (info.AllocationProtect)

{

case PAGE\_NOACCESS:

cout << "PAGE\_NOACCESS";

break;

case PAGE\_READONLY:

cout << "PAGE\_READONLY";

break;

case PAGE\_READWRITE:

cout << "PAGE\_READWRITE";

break;

case PAGE\_WRITECOPY:

cout << "PAGE\_WRITECOPY";

break;

case PAGE\_EXECUTE:

cout << "PAGE\_EXECUTE";

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\_GUARD:

cout << "PAGE\_GUARD";

break;

case PAGE\_NOCACHE:

cout << "PAGE\_NOCACHE";

break;

case PAGE\_WRITECOMBINE:

cout << "PAGE\_WRITECOMBINE";

break;

default:

cout << "Unknown protection: " << info.AllocationProtect;

break;

}

cout << endl;

cout << "RegionSize: 0x" << hex << info.RegionSize << dec << endl;

cout << "State: ";

switch (info.State)

{

case MEM\_COMMIT:

cout << "MEM\_COMMIT";

break;

case MEM\_FREE:

cout << "MEM\_FREE";

break;

case MEM\_RESERVE:

cout << "MEM\_RESERVE";

break;

default:

cout << "Unknown state: " << info.State;

break;

}

cout << endl;

cout << "Protect: ";

switch (info.Protect)

{

case PAGE\_NOACCESS:

cout << "PAGE\_NOACCESS";

break;

case PAGE\_READONLY:

cout << "PAGE\_READONLY";

break;

case PAGE\_READWRITE:

cout << "PAGE\_READWRITE";

break;

case PAGE\_WRITECOPY:

cout << "PAGE\_WRITECOPY";

break;

case PAGE\_EXECUTE:

cout << "PAGE\_EXECUTE";

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\_GUARD:

cout << "PAGE\_GUARD";

break;

case PAGE\_NOCACHE:

cout << "PAGE\_NOCACHE";

break;

case PAGE\_WRITECOMBINE:

cout << "PAGE\_WRITECOMBINE";

break;

default:

cout << "Unknown protection: " << info.Protect;

break;

}

cout << endl;

cout << "Type: ";

switch (info.Type)

{

case MEM\_IMAGE:

cout << "MEM\_IMAGE";

break;

case MEM\_MAPPED:

cout << "MEM\_MAPPED";

break;

case MEM\_PRIVATE:

cout << "MEM\_PRIVATE";

break;

default:

cout << "Unknown type: " << info.Type;

break;

}

cout << endl;

cout << "\nPress 0 to return to the main menu...";

int choice = -1;

cin >> choice;

while (choice != 0)

{

cout << "Error. Please enter 0 to return to the main menu.\n";

cin >> choice;

}

choose(addr);

}

void reserve\_block\_without\_memory(LPVOID addr)

{

DWORD dwSize = 4096;

DWORD dwAllocationType = MEM\_RESERVE;

DWORD dwProtect = PAGE\_READWRITE;

int mode = 0;

LPVOID lpReservedMem=0;

cout << "select mode:\n1 - automaticly\n2 - at a chosen beginning of address\n";

cin >> mode;

if (mode == 1)

{

lpReservedMem = VirtualAlloc(NULL, dwSize, dwAllocationType, dwProtect);

}

else if (mode == 2)

{

lpReservedMem = VirtualAlloc(addr, dwSize, dwAllocationType, dwProtect);

}

if (lpReservedMem == NULL)

{

DWORD error\_code = GetLastError();

cout << "VirtualAlloc failed with error code " << error\_code << endl;

}

else

{

cout << "Virtual memory reserved successfully." << endl;

cout << "address: " << lpReservedMem << endl;

}

cout << "\nPress 0 to return to the main menu...";

int choice = -1;

cin >> choice;

while (choice != 0)

{

cout << "Error. Please enter 0 to return to the main menu.\n";

cin >> choice;

}

choose(lpReservedMem);

}

void reserve\_block(LPVOID addr)

{

DWORD dwSize = 4096;

DWORD dwAllocationType = MEM\_COMMIT;

DWORD dwProtect = PAGE\_READWRITE;

LPVOID lpReservedMem=0;

int mode = 0;

cout << "select mode:\n1 - automaticly\n2 - at a chosen beginning of address\n";

cin >> mode;

if (mode == 1)

{

lpReservedMem = VirtualAlloc(NULL, dwSize, dwAllocationType, dwProtect);

}

else if (mode == 2)

{

lpReservedMem = VirtualAlloc(addr, dwSize, dwAllocationType, dwProtect);

}

if (lpReservedMem == NULL)

{

DWORD error\_code = GetLastError();

cout << "VirtualAlloc failed with error code " << error\_code << endl;

}

else

{

cout << "Virtual memory reserved and committed successfully." << endl;

cout << "address: " << lpReservedMem << endl;

}

cout << "\nPress 0 to return to the main menu...";

int choice = -1;

cin >> choice;

while (choice != 0)

{

cout << "Error. Please enter 0 to return to the main menu.\n";

cin >> choice;

}

choose(lpReservedMem);

}

void free\_region(LPVOID addr)

{

if (VirtualFree(addr, 0, MEM\_RELEASE) == 0)

{

DWORD error\_code = GetLastError();

cout << "VirtualFree failed with error code " << error\_code << endl;

}

else

{

cout << "Memory freed successfully." << endl;

}

cout << "\nPress 0 to return to the main menu...";

int choice = -1;

cin >> choice;

while (choice != 0)

{

cout << "Error. Please enter 0 to return to the main menu.\n";

cin >> choice;

}

choose(addr);

}

void write\_in\_block(LPVOID addr)

{

size\_t size;

cout << "Введите размер данных для записи: ";

cin >> size;

cin.ignore();

char\* buffer = new char[size + 1];

cout << "Введите данные для записи: ";

cin.getline(buffer, size);

if (WriteProcessMemory(GetCurrentProcess(), addr, buffer, size, 0) == 0)

{

DWORD error\_code = GetLastError();

cout << "ERROR :" << error\_code << endl;

delete[] buffer;

}

else

{

cout << "Данные записаны успешно. Сейчас ячейка содержит: " << endl;

if (ReadProcessMemory(GetCurrentProcess(), addr, buffer, size, 0) == 0)

{

DWORD error\_code = GetLastError();

cout << "ERROR :" << error\_code << endl;

delete[] buffer;

}

for (SIZE\_T i = 0; i < size; i++)

{

cout << buffer[i] << " ";

}

cout << endl;

delete[] buffer;

}

Sleep(2000);

choose(addr);

}

void est\_protect(LPVOID addr)

{

MEMORY\_BASIC\_INFORMATION info;

VirtualQuery(addr, &info, sizeof(info));

cout << "Current protection attributes:" << endl;

vector <DWORD> values = { PAGE\_NOACCESS, PAGE\_READONLY, PAGE\_READWRITE,

PAGE\_WRITECOPY, PAGE\_EXECUTE, PAGE\_EXECUTE\_READ, PAGE\_EXECUTE\_READWRITE,

PAGE\_EXECUTE\_WRITECOPY, PAGE\_GUARD, PAGE\_NOCACHE, PAGE\_WRITECOMBINE };

vector <string> names = { "PAGE\_NOACCESS", "PAGE\_READONLY", "PAGE\_READWRITE",

"PAGE\_WRITECOPY", "PAGE\_EXECUTE", "PAGE\_EXECUTE\_READ", "PAGE\_EXECUTE\_READWRITE",

"PAGE\_EXECUTE\_WRITECOPY", "PAGE\_GUARD", "PAGE\_NOCACHE", "PAGE\_WRITECOMBINE" };

map<int, DWORD> protect\_attrs =

{

{ 1, PAGE\_NOACCESS },

{ 2, PAGE\_READONLY },

{ 3, PAGE\_READWRITE },

{ 4, PAGE\_WRITECOPY },

{ 5, PAGE\_EXECUTE },

{ 6, PAGE\_EXECUTE\_READ },

{ 7, PAGE\_EXECUTE\_READWRITE },

{ 8, PAGE\_EXECUTE\_WRITECOPY },

{ 9, PAGE\_GUARD },

{ 10, PAGE\_NOCACHE },

{ 11, PAGE\_WRITECOMBINE }

};

for (const auto& attr : protect\_attrs)

{

cout << attr.first << " - " << names[attr.first - 1];

if (info.Protect & attr.second)

{

cout << " : YES\n";

}

else

{

cout << " : NO\n";

}

}

int new\_protect\_num = 0;

cout << "Do you want to change attribute?\n0 - no\nn - yes (n - number of attribute)\n";

cin >> new\_protect\_num;

if (new\_protect\_num == 0)

{

choose(addr);

}

else if (new\_protect\_num < 1 || new\_protect\_num > 11)

{

cout << "Invalid attribute number. Protection attributes unchanged.\n";

}

else

{

DWORD old\_protect;

if (!VirtualProtect(addr, 4096, values[new\_protect\_num - 1], &old\_protect)) {

DWORD error\_code = GetLastError();

cout << "VirtualProtect failed with error code " << error\_code << endl;

}

}

VirtualQuery(addr, &info, sizeof(info));

cout << "changes:\n";

for (const auto& attr : protect\_attrs)

{

cout << attr.first << " - " << names[attr.first - 1];

if (info.Protect & attr.second)

{

cout << " : YES\n";

}

else

{

cout << " : NO\n";

}

}

cout << "\nPress 0 to return to the main menu...";

int choice = -1;

cin >> choice;

while (choice != 0)

{

cout << "Error. Please enter 0 to return to the main menu.\n";

cin >> choice;

}

choose(addr);

}

void choose(LPVOID addr)

{

system("cls");

cout << "Chosen region: 0x" << hex << addr << dec << endl << endl;

int choice = -1;

cout << "1 - Output information about the computer system\n";

cout << "2 - Output the status of virtual memory\n";

cout << "3 - Output the state of a memory region\n";

cout << "4 - Reserve a region\n";

cout << "5 - Reserve a region and assign it physical memory\n";

cout << "6 - Write data to a memory cell\n";

cout << "7 - Set access protection for a region\n";

cout << "8 - Return physical memory and release the region\n";

cout << "0 - Exit the program\n";

cin >> choice;

while (choice < -1 || choice > 8)

{

cout << "Error. Please re-enter your choice\n";

cin >> choice;

}

switch (choice)

{

case 0: exit(0);

case 1: system\_info(addr); break;

case 2: memory\_status(addr); break;

case 3: block\_info(addr); break;

case 4: reserve\_block\_without\_memory(addr); break;

case 5: reserve\_block(addr); break;

case 6: write\_in\_block(addr); break;

case 7: est\_protect(addr); break;

case 8: free\_region(addr); break;

}

}

int main()

{

DWORD address = 0x000000000AF0000;

string region;

LPVOID addr = 0;

int flag = auto\_or\_not();

SYSTEM\_INFO systemInfo;

GetSystemInfo(&systemInfo);

if (flag == 1)

{

cout << "Enter an address in the range of 0x" << systemInfo.lpMinimumApplicationAddress << " before 0x" << systemInfo.lpMaximumApplicationAddress << ": ";

cin >> address;

addr = reinterpret\_cast<LPVOID>(address);

}

else

{

addr = reinterpret\_cast<LPVOID>(address);

}

setlocale(LC\_ALL, "Russian");

choose(addr);

return 0;

}

Задание 2

1. Сервер

#include <stdio.h>

#include <iostream>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <unistd.h>

#include <sys/select.h>

#include <time.h>

using namespace std;

#define FILENAME "/home/pipopolam/CLionProjects/test.txt"

#define FILESIZE 33

int project\_flag = 0;

int write\_flag = 0;

void\* ptr = NULL;

void\* project(int fd)

{

void\* ptr = mmap(NULL, FILESIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (ptr == MAP\_FAILED) {

cerr << "Error: mmap failed" << endl;

exit(1);

}

project\_flag = 1;

return ptr;

}

void write(void\* ptr)

{

if (ptr == MAP\_FAILED) {

cerr << "Error: ptr has invalid value" << endl;

exit(1);

}

else if (ptr == nullptr) {

cerr << "Error: ptr is null" << endl;

exit(1);

}

sprintf(static\_cast<char\*>(ptr), "Hello shared memory!");

}

void stop(void\* ptr, int fd)

{

if (munmap(ptr, FILESIZE) == -1) {

cerr << "Error: munmap failed" << endl;

exit(1);

}

close(fd);

if (unlink(FILENAME) == -1) {

cerr << "Error: unlink failed" << endl;

exit(1);

}

}

void choose(int fd)

{

int flag=-1;

cout << "Какое действие вы хотите совершить?\n";

cout << "1 - Выполнить проецирование\n2 - Записать данные\n0 - Завершить программу\n";

cin >> flag;

switch (flag)

{

case 1:

ptr = project(fd);

cout << "Проецирование выполнено\n";

break;

case 2:

if (project\_flag == 0)

{

cout << "Сначала выполните проецирование\n";

break;

}

write(ptr);

cout << "Данные записаны\n";

break;

case 0:

stop(ptr, fd);

exit(0);

default:

cout << "Некорректный выбор, попробуйте снова\n";

}

}

int main() {

int fd;

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

if (fd == -1) {

cerr << "Error: open failed" << endl;

exit(1);

}

struct stat st;

if (fstat(fd, &st) == -1) {

cerr << "Error: fstat failed" << endl;

exit(1);

}

if (st.st\_size != FILESIZE) {

cerr << "Error: file size mismatch" << endl;

exit(1);

}

while (true)

{

choose(fd);

}

return 0;

}

1. Клиент

#include <stdio.h>

#include <iostream>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <unistd.h>

#include <sys/select.h>

#include <time.h>

using namespace std;

#define FILENAME "/home/pipopolam/test.txt"

#define FILESIZE 4096

int project\_flag = 0;

int read\_flag = 0;

int stop\_flag = 0;

void\* ptr = NULL;

void\* project(int fd)

{

void\* ptr = mmap(NULL, FILESIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (ptr == MAP\_FAILED) {

cerr << "Error: mmap failed" << endl;

exit(1);

}

project\_flag = 1;

return ptr;

}

void read(void\* ptr)

{

read\_flag = 1;

printf("Client received: %s\n", static\_cast<char\*>(ptr));

}

void stop(void\* ptr, int fd)

{

if (munmap(ptr, FILESIZE) == -1) {

cerr << "Error: munmap failed" << endl;

exit(1);

}

close(fd);

stop\_flag = 1;

}

void choose(int fd)

{

int flag=-1;

int result;

cout << "Какое действие вы хотите совершить?\n";

cout << "1 - Выполнить проецирование\n2 - Прочитать данные\n0 - Завершить программу\n";

cin >> flag;

switch (flag)

{

case 1:

ptr = project(fd);

cout << "Проецирование выполнено\n";

break;

case 2:

if (project\_flag == 0)

{

cout << "Сначала выполните проецирование\n";

break;

}

fd\_set read\_fds;

struct timeval timeout;

timeout.tv\_sec = 1;

timeout.tv\_usec = 0;

FD\_ZERO(&read\_fds);

FD\_SET(fd, &read\_fds);

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

if (result == -1) {

cerr << "Error: select failed" << endl;

exit(1);

} else if (result == 0) {

cout << "Timeout: no data available for reading\n";

break;

}

read(ptr);

cout << "Данные прочитаны\n";

break;

case 0:

stop(ptr, fd);

exit(0);

break;

default:

cout << "Некорректный выбор, попробуйте снова\n";

break;

}

}

int main() {

int fd;

fd = open(FILENAME, O\_RDWR);

if (fd == -1) {

cerr << "Error: open failed" << endl;

exit(1);

}

while (true)

{

choose(fd);

}

return 0;

}