**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ **«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. В. Г. ШУХОВА»**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**ЛАБОРАТОРНАЯ РАБОТА №2**

**Дисциплина: Операционные системы**

**Тема: Архитектура Windows**

Выполнил: ст. группы ВТ-31

Подкопаев Антон Валерьевич

Проверил: ст. пр. ПОВТиАС

Михелев Владимир Михайлович

**Белгород 2019**

**Цель работы**: Изучение операционной архитектуры Windows.

Приложение (application) Windows - это совокупность исполняемых прог­рамм и вспомогательных файлов. Например, Microsoft Word представляет собой одно из популярных приложений Windows. Процессом называется исполняемый экземпляр приложения. В большинстве случаев пользователь может запускать несколько экземпляров (копий) одного и того же приложения одновре­менно. Каждый исполняемый экземпляр - это отдельный процесс со своей соб­ственной областью памяти.

Если быть более точным, процессом (process) называется исполняемый экзем­пляр (running instance) приложения и комплект ресурсов, отводящийся данному исполняемому приложению.

Поток (thread) - это внутренняя составляющая процесса, которой операци­онная система выделяет процессорное время для выполнения кода. Именно пото­ки исполняют программный код, а не процессы. Каждый процесс должен иметь как минимум один поток. Конечно, основное назначение потоков - дать процессу возможность поддерживать несколько ветвей управления, то есть выполнять больше действий одновременно. В многопроцессорной конфигурации (компью­тер с несколькими процессорами) Windows NT (но не Windows 9x) может распре­делять потоки по процессорам, реально обеспечивая параллельную обработку. В однопроцессорной конфигурации процессор должен выделять кванты времени (time slices) каждому исполняемому в данный момент потоку.

На ниже приведенном рисунке представлена в обобщенном виде архитектура Windows NT. Рассмотрим некоторые из изображенных пунктов.

***Режим ядра и пользовательский режим***

Микропроцессор Pentium имеет четыре уровня привилегий (privilege levels), известных также как кольца (rings), которые управляют, например, доступом к памяти, возможностью использовать некоторые критичные команды процессо­ра (такие как команды, связанные с защитой) и т.д. Каждый поток выполняется на одном из этих уровней привилегий. Кольцо 0 - наиболее привилегированный уровень, с полным доступом ко всей памяти и ко всем командам процессора. Коль­цо 3 - наименее привилегированный уровень.

Для обеспечения совместимости с системами на базе процессоров, отличных от тех, что выпускает компания Intel, Windows поддерживает только два уровня привилегий - кольца 0 и 3. Если поток работает в кольце 0, говорят, что он выпол­няется в режиме ядра (kernel mode). Если поток выполняется в кольце 3, говорят, что он работает в пользовательском режиме (user mode). Низкоуровневый код операционной системы действует в режиме ядра, тогда как пользовательские при­ложения выполняются в основном в пользовательском режиме.

Заметим, что прикладной поток может переключаться из пользовательского режима в режим ядра при вызове некоторых API-функций, которые требуют бо­лее высокого уровня привилегий, например, связанных с доступом к файлам или с выполнением функций, ориентированных на графические операции. В действи­тельности некоторые пользовательские потоки могут работать в режиме ядра даже больше времени, чем в пользовательском режиме.

Но как только завершается выполнение той части кода, которая относится к режиму ядра, пользовательский поток автоматически переключается обратно в пользовательский режим. Такой подход лишает возможности писать код, пред­назначенный для работы в режиме ядра, программист может только вызывать вы­полняющиеся в режиме ядра системные функции (system functions). При работе с Windows NT можно определить, когда поток выполняется в пользовательском режиме, а когда - в режиме ядра. Для этого используется утилита Performance Monitor (Системный монитор) из пункта Administrative Tools (Администрирование) меню Start. (Пуск) (что Вы должны будете и сделать – это будет Ваше первое задание в этой работе).

Интересно, что драйверы устройств работают в режиме ядра. Это обстоятельство имеет два следствия. Во-первых, в отличие от неправильно выполняющегося приложения неправильно работающий драйвер устройства может нарушить работу всей системы, так как он имеет доступ и ко всему системному коду, и ко всей памяти. Во-вторых, прикладной программист может получить доступ к защищенным ресурсам, написав драйвер псевдоустройства (fake device), хотя это и нелегкая задача.



**Сервисы**

Термин сервис (service) имеет в среде Windows множество значений. Ниже представлены некоторые из них, имеющие отношение к рассматриваемой теме:

* Сервис АРI - функция или подпрограмма API, которая реализует некото­рое действие (сервис) операционной системы, такое как создание файла или работа с графикой (рисование линий или окружностей). Например, функ­ция API *CreateProcess* используется в Windows для создания нового про­цесса;
* системный сервис - недокументированная (undocumented) функция, кото­рая может вызываться из пользовательского режима. Эти функции часто ис­пользуются функциями Win32 API для предоставления низкоуровневых сервисов. Например, функция API *CreateProcess* для реального созда­ния процесса вызывает системный сервис *NTCreateProcess*;
* внутренний (internal) сервис - функция или подпрограмма, которая может вызываться только из кода, выполняемого в режиме ядра. Эти функции от­носятся к низкоуровневой части кода Windows: к исполнительной системе Windows NT, к ядру или к слою абстрагирования от аппаратуры (HAL).

**Системные процессы.**

Системные процессы (system processes) - это особые процессы, обслуживаю­щие операционную систему. В системе Windows постоянно задействованы следующие системные процессы (учтите, что все они, кроме процесса system, выполняются в пользовательском режиме):

* процесс *idle*, который состоит из одного потока, управляющего временем простоя процессора;
* процесс *system* - специальный про­цесс, выполняющийся только в режиме ядра. Его потоки называются системными потоками (system threads);
* процесс *Session Manager* (диспетчер сеансов) - SMSS.EXE;
* подсистема *Win32* - CSRSS.EXE;
* процесс регистрации в системе — *WinLogon* (WINLOGON.EXE).

Вы можете убедиться в том, что эти системные процессы действительно вы­полняются в системе, посмотрев на вкладку Processes (Процессы) программы Task Manager (Диспетчер Задач). Рассмотрим вкратце некоторые из этих системных процессов.

**Процесс Session Manager**

Процесс Session Manager (SMSS.EXE) - один из первых процессов, созда­ваемых операционной системой в процессе загрузки. Он выполняет важные функции инициализации, такие как создание, переменных окружения системы; задание имен устройств MS DOS, например, LPT1 и СОМ1; загрузка той части подсистемы Win32, которая относится к режиму ядра; запуск процесса регистрации в систе­ме WinLogon.

**Процесс WinLogon**

Этот системный процесс управляет входом пользователей в систему и выходом из нее. Вызывается специальной комбинацией клавиш Windows Ctrl+Alt+Delete. WinLogon отвечает за загрузку оболочки Windows (обычно это Windows Explorer).

**Процесс system**

Процесс system состоит из системных потоков (system threads), являющих­ся потоками режима ядра. Windows и многие драйверы устройств создают потоки прoцecca system для различных целей. Например, диспетчер памяти формирует системные потоки для решения задач управления виртуальной памятью, диспет­чер кэша использует системные потоки для управления кэш-памятью, а драйвер гибкого диска - для контроля над гибкими дисками.

**Подсистема Win32**

Подсистема Win32 - основной предмет нашего рассмотрения. Она яв­ляется разновидностью подсистемы среды (environment subsystem). Другие под­системы среды Windows (не показаны на рисунке) включают POSIX и OS/2. POSIX является сокращением термина «переносимая операционная система па базе UNIX» (portable operating system based on UNIX) и реализует ограниченную поддержку операционной системы UNIX.

Назначение подсистемы среды - служить интерфейсом между пользовательски­ми приложениями и соответствующей частью исполнительной системы Windows. Каждая подсистема имеет свои функциональные возможности на базе единой исполнительной системы Windows. Любой выполняемый файл неразрывно связан с одной из этих подсистем. Подсистема Win32 содержит Win32 API в виде набора DLL, таких как KERNEL32.DLL, GDI32.DLL и USER32.DLL.

В Windows NT Microsoft перенесла часть подсистемы Win32 из пользовательского режима в режим ядра. В частности, драйвер устройства режима ядра WIN32K.SYS, который управляет отображением окон, вы­водом на экран, вводом данных с клавиатуры или при помощи мыши и передачей сообщений. Он включает также библиотеку интерфейсов графических устройств (Graphical Device Interface library – GDL.DLL), используемую для создания гра­фических объектов и текста.

**Вызов Win32 API-функций**

Когда приложение вызывает API-функцию из подсистемы Win32, может про­изойти одно из нескольких событий:

* если DLL подсистемы (например, USER32.DLL), экспортирующая данную API-функцию, содержит весь код, необходимый для выполнения функции то функция выполняется и возвращает результат;
* API-функции, вызываемой приложением, может потребоваться вызвать для выполнения вспомогательных действий дополнительный модуль, принад­лежащий подсистеме Win32 (но не той DLL, которая экспортирует данную функцию);
* API-функции, вызываемой приложением, могут понадобиться услуги не­документированного системного сервиса. Например, чтобы создать но­вый процесс, API-функция CreateProcess вызывает недокументирован­ный системный сервис NTCreateProcess для реального создания данного процесса. Это делается с помощью функций библиотеки NTDLL.DLL, кото­рая помогает осуществлять переход из пользовательского режима в ре­жим ядра.

**Исполнительная система Windows**

Сервисы исполнительной системы Windows составляют низкоуровневую часть Windows NT режима ядра, включенную в файл NTOSKRNL.EXE.

Сервисы исполнительной системы делят на две группы: исполнительную систему (executive), относящуюся к верхнему уровню, и ядро (kernel). Ядро - это самый нижний уровень операционной системы, реализующий наиболее фун­даментальные сервисы, такие как:

* планирование потоков;
* обработку исключений;
* обработку прерываний;
* синхронизацию процессоров в многопроцессорной системе;
* создание объектов ядра.

Ниже приведены некоторые наиболее важные составляющие исполнительной системы:

* диспетчер процессов и потоков создает и завершает и процессы, и потоки, используя сервисы низкоуровневого ядра;
* диспетчер виртуальной памяти реализует механизм виртуальной памяти;
* диспетчер ввода/вывода реализует аппаратно-независимый ввод/вывод и взаимодействует с драйверами устройств;
* диспетчер КЭШа управляет кэшированием диска;
* диспетчер объектов создает объекты исполнительной системы Windows и управляет ими. Windows использует объекты для представления разнооб­разных ресурсов, таких как процессы и потоки;
* библиотеки времени выполнения содержат такие функции, как обработки строк и арифметическиефункции.

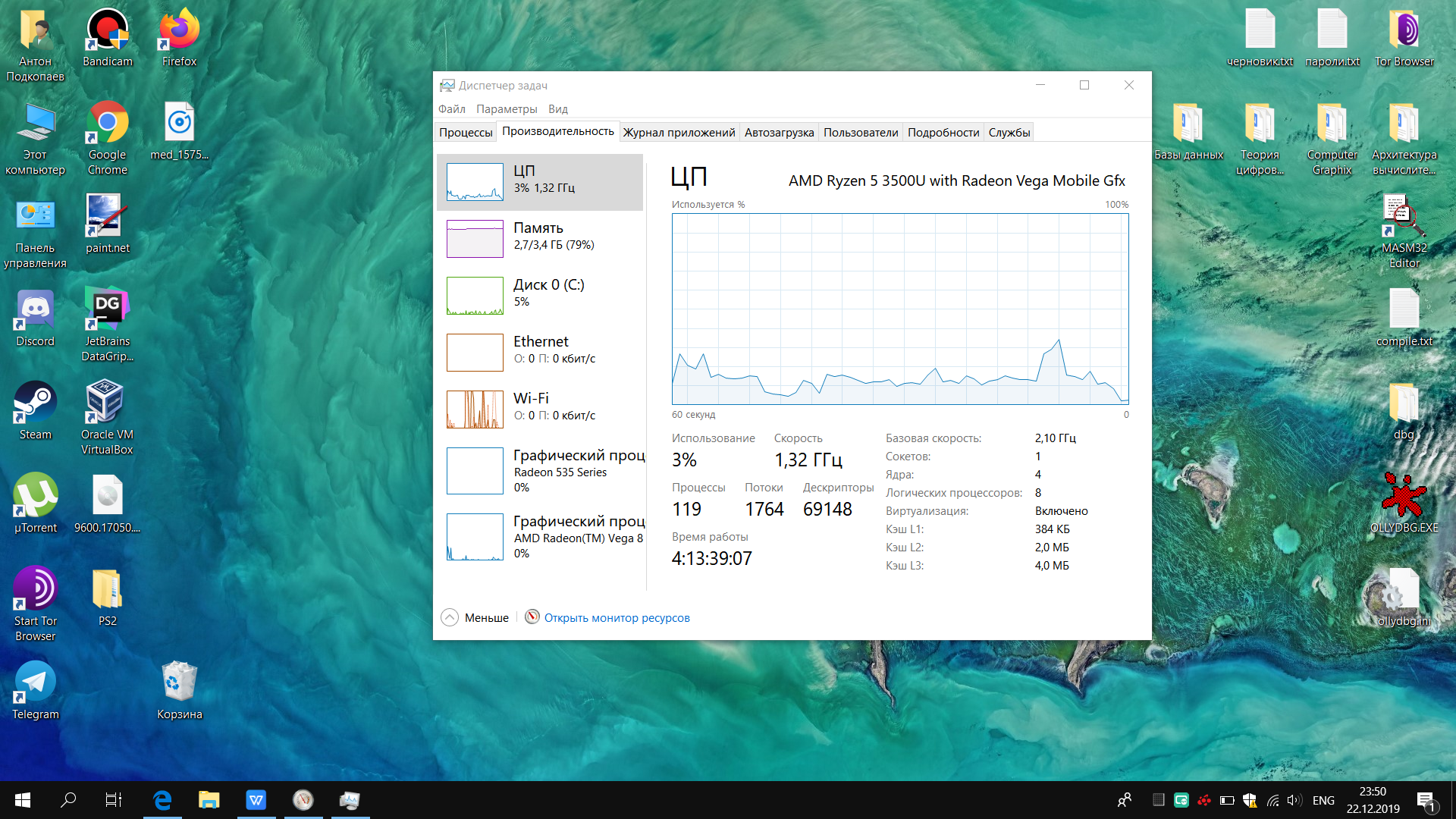
**Уровень абстрагирования от аппаратуры (HAL)**

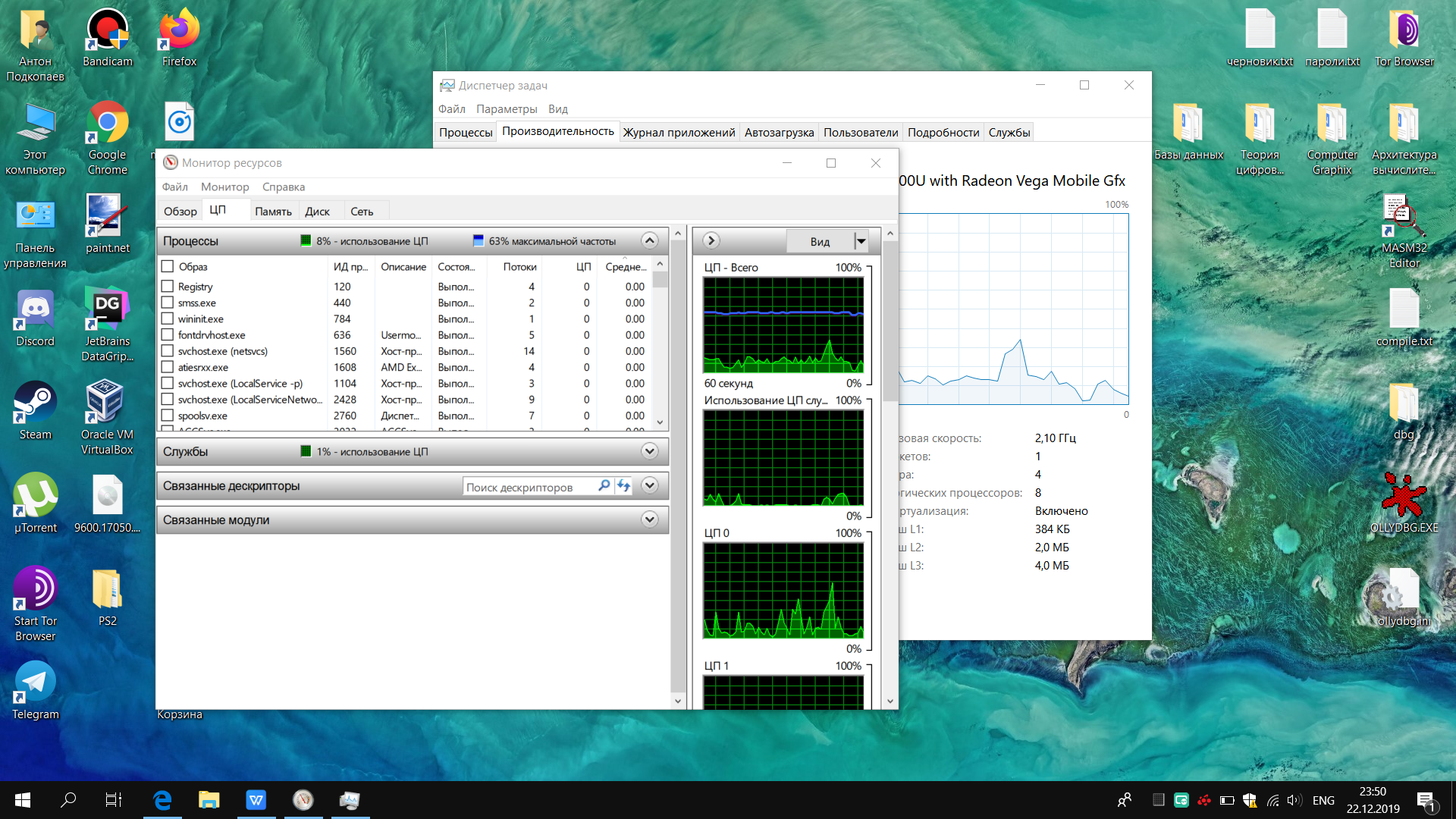
Уровень абстрагирования от аппаратуры (HAL) - это библиотека режима ядра (HAL.DLL), которая реализует низкоуровневый интерфейс с аппаратурой. Компо­ненты Windows и драйверы устройств от других компаний взаимодействуют с аппа­ратурой посредством HAL. Существует много версий HAL под различные аппарат­ные платформы. Подходящий уровень выбирается в процессе установки Windows.

**Отличие Windows 9x и Windows NT**

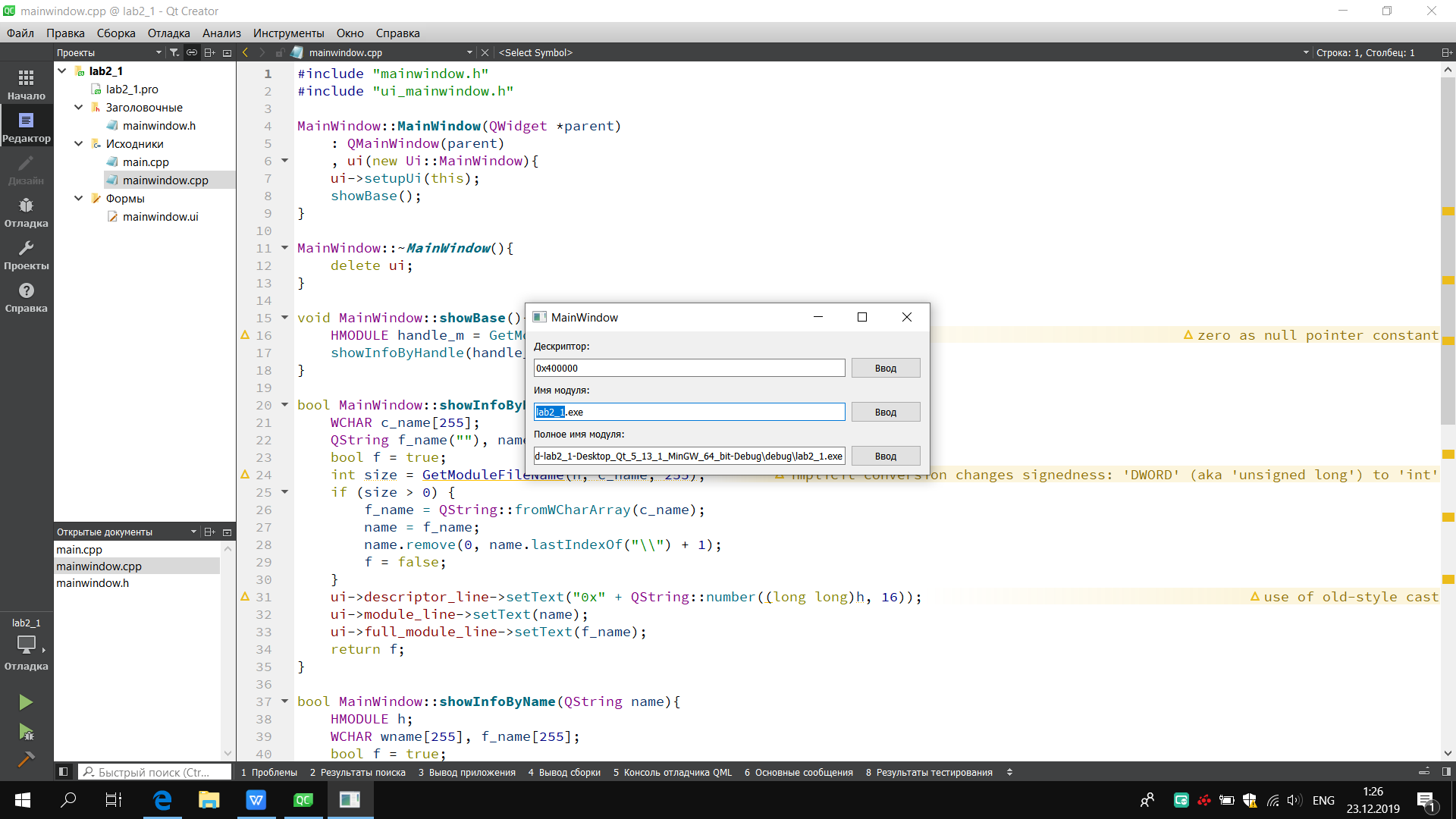
Далее приведены некоторые из наиболее важных отличий между oneрационными системами Windows 9x и Windows NT;

* Windows NT поддерживает симметричную многопроцессорную обработку (SMP), то есть Windows NT может использовать одновременно несколько процессоров. Термин «симметричная» объясняется тем, что Windows NT интерпретирует все процессоры одинаково и распределяет как потоки опе­рационной системы, так и прикладные пользовательские программы по всем процессорам (в противоположность асимметричной схеме, когда один процессор закрепляется исключительно за операционной системой);
* Windows NT способна работать на платформах, построенных не на базе про­цессоров Intel. Например, Windows NT работает на системах с процессором PowerPC;
* Windows NT является истинной 32-разрядной операционной системой, тог­да как Windows 9x содержит значительное количество 16-разрядного кода, который был перенесен из Windows 3.1. Как следствие, в Windows 9x обла­сти памяти, закрепленные за операционной системой, доступны из пользовательского режима, что делает Windows 9x гораздо менее стабильной, чем Windows NT;
* Windows NT реализует надежную защиту файловой системы, которая от­сутствует в Windows 9x;
* и в той, и в другой операционных системах приложения могут совместно использовать память. Однако в Windows NT к этой памяти получают дос­туп только те приложения, которые специально запросили данный ресурс, тогда как в Windows 9x разделяемая память доступна всем исполняемым программам.





**2-1**



mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QString>

#include <windows.h>

#include <QMessageBox>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

**MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

void **showBase**();

bool **showInfoByHandle**(HMODULE handle\_m);

bool **showInfoByName**(QString name);

private slots:

void **on\_descriptor\_button\_clicked**();

void **on\_module\_button\_clicked**();

void **on\_full\_module\_button\_clicked**();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow){

ui->setupUi(this);

showBase();

}

MainWindow::~***MainWindow***(){

delete ui;

}

void MainWindow::**showBase**(){

HMODULE handle\_m = GetModuleHandleA(NULL);

showInfoByHandle(handle\_m);

}

bool MainWindow::**showInfoByHandle**(HMODULE h){

WCHAR c\_name[255];

QString f\_name(""), name("");

bool f = true;

int size = GetModuleFileName(h, c\_name, 255);

if (size > 0) {

f\_name = QString::fromWCharArray(c\_name);

name = f\_name;

name.remove(0, name.lastIndexOf("\\") + 1);

f = false;

}

ui->descriptor\_line->setText("0x" + QString::number((long long)h, 16));

ui->module\_line->setText(name);

ui->full\_module\_line->setText(f\_name);

return f;

}

bool MainWindow::**showInfoByName**(QString name){

HMODULE h;

WCHAR wname[255], f\_name[255];

bool f = true;

int size = name.toWCharArray(wname);

wname[size] = 0;

h = GetModuleHandle(wname);

if (h != NULL){

GetModuleFileName(h, f\_name, 255);

f = false;

}else {

name = "";

f\_name[0] = '\0';

}

ui->descriptor\_line->setText("0x" + QString::number((long long)h, 16));

ui->module\_line->setText(name);

ui->full\_module\_line->setText(QString::fromWCharArray(f\_name));

return f;

}

void MainWindow::**on\_descriptor\_button\_clicked**(){

long long h = ui->descriptor\_line->text().toInt(0,16);

if (showInfoByHandle(HMODULE(h)))

QMessageBox::warning(this, "Ошибка", "Неверное значение");

}

void MainWindow::**on\_module\_button\_clicked**(){

QString name = ui->module\_line->text();

if (showInfoByName(name))

QMessageBox::warning(this, "Ошибка", "Неверное значение");

}

void MainWindow::**on\_full\_module\_button\_clicked**(){

QString name = ui->full\_module\_line->text();

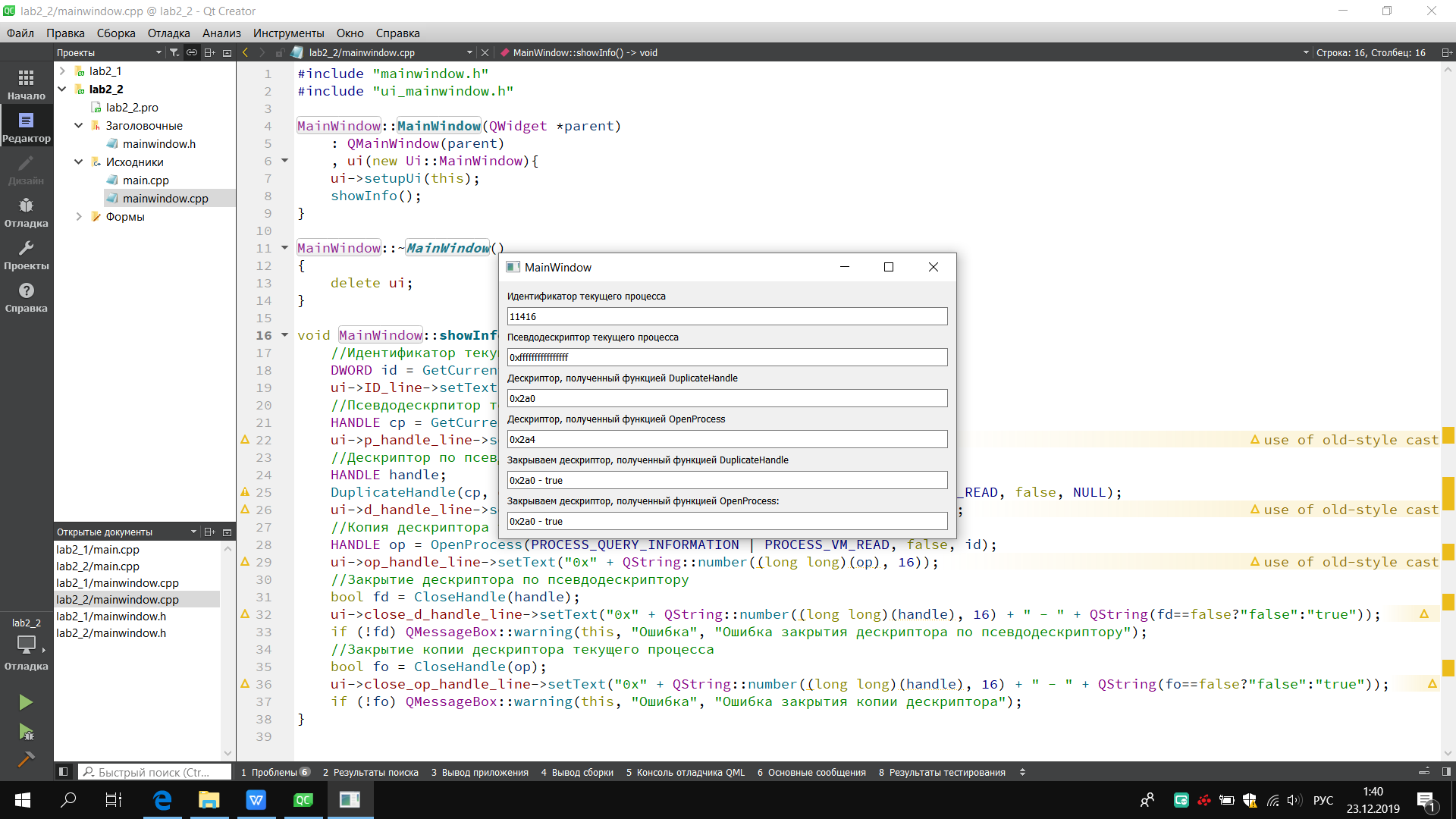
name.remove(0, name.lastIndexOf("\\") + 1);

if (showInfoByName(name))

QMessageBox::warning(this, "Ошибка", "Неверное значение");

}

**2-2**

****

mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <windows.h>

#include <QMessageBox>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

**MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

void **showInfo**();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

maiwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow){

ui->setupUi(this);

showInfo();

}

MainWindow::~***MainWindow***()

{

delete ui;

}

void MainWindow::**showInfo**(){

//Идентификатор текущего процесса

DWORD id = GetCurrentProcessId();

ui->ID\_line->setText(QString::number(id));

//Псевдодескрпитор текущего процесса

HANDLE cp = GetCurrentProcess();

ui->p\_handle\_line->setText("0x" + QString::number((long long)(cp), 16));

//Дескриптор по псевдодескриптору

HANDLE handle;

DuplicateHandle(cp, cp, cp, &handle, PROCESS\_QUERY\_INFORMATION | PROCESS\_VM\_READ, false, NULL);

ui->d\_handle\_line->setText("0x" + QString::number((long long)(handle), 16));

//Копия дескриптора текущего процесса

HANDLE op = OpenProcess(PROCESS\_QUERY\_INFORMATION | PROCESS\_VM\_READ, false, id);

ui->op\_handle\_line->setText("0x" + QString::number((long long)(op), 16));

//Закрытие дескриптора по псевдодескриптору

bool fd = CloseHandle(handle);

ui->close\_d\_handle\_line->setText("0x" + QString::number((long long)(handle), 16) + " - " + QString(fd==false?"false":"true"));

if (!fd) QMessageBox::warning(this, "Ошибка", "Ошибка закрытия дескриптора по псевдодескриптору");

//Закрытие копии дескриптора текущего процесса

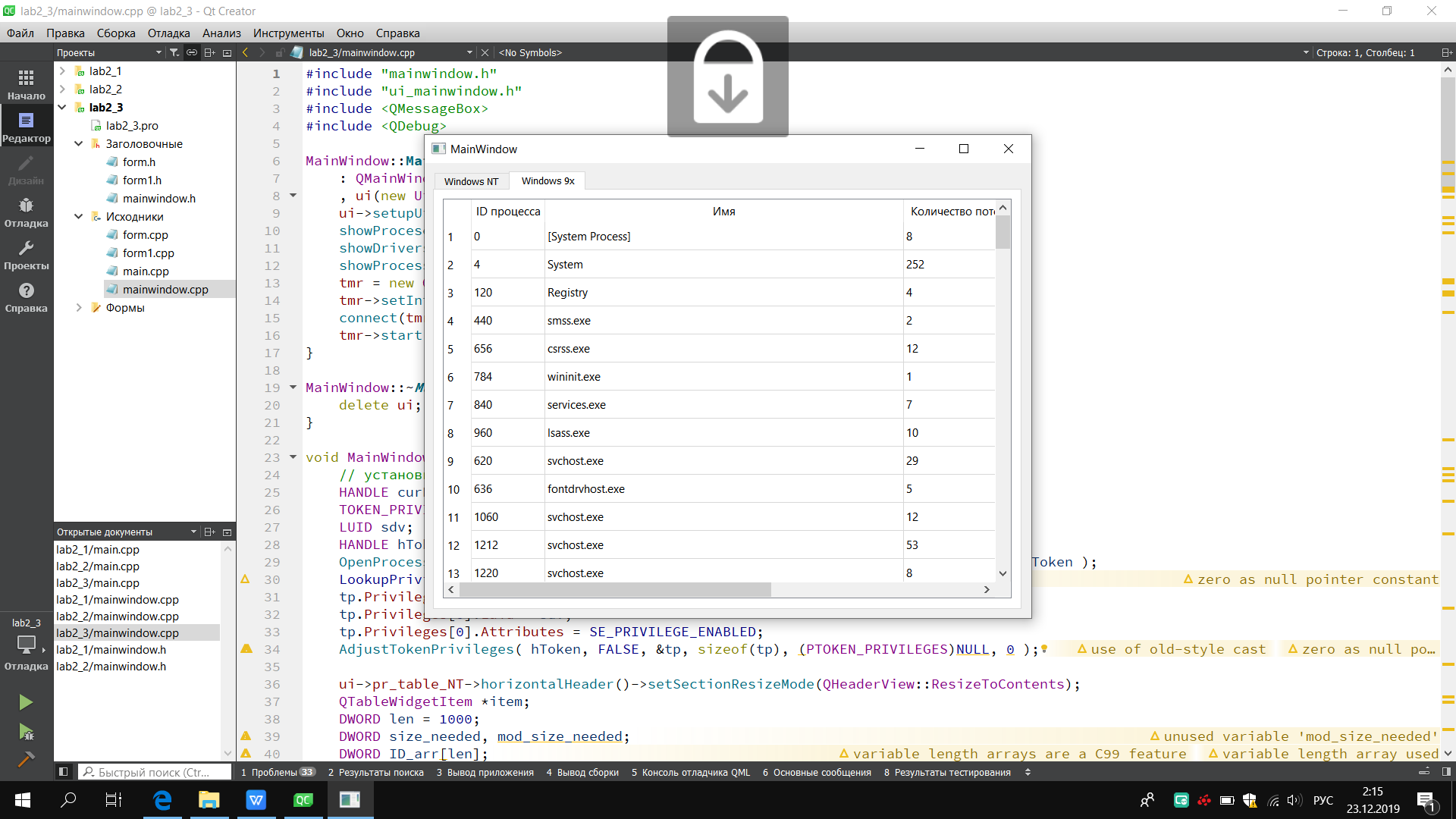
bool fo = CloseHandle(op);

ui->close\_op\_handle\_line->setText("0x" + QString::number((long long)(handle), 16) + " - " + QString(fo==false?"false":"true"));

if (!fo) QMessageBox::warning(this, "Ошибка", "Ошибка закрытия копии дескриптора");

}

**2-3**

****

maiwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QTableWidget>

#include <windows.h>

#include <psapi.h>

#include <tlhelp32.h>

#include <QTimer>

#include "form.h"

#include "form1.h"

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

//Win NT

void showProcesesNT();

void showThreadsNT();

void showDrivers();

void showModulesNT(DWORD proc\_ID, QString proc\_name);

//Win 9x

void showProcesses();

void showThreadsAndModules(DWORD proc\_ID, QString name);

public slots:

void updateTables();

private slots:

void on\_pr\_table\_NT\_cellClicked(int row, int column);

void on\_proc\_table\_cellClicked(int row, int column);

private:

Ui::MainWindow \*ui;

QTimer \*tmr;

Form\* window;

Form1\* window1;

};

#endif // MAINWINDOW\_H

mainwoindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QMessageBox>

#include <QDebug>

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow){

ui->setupUi(this);

showProcesesNT();

showDrivers();

showProcesses();

tmr = new QTimer(this);

tmr->setInterval(1000);

connect(tmr, SIGNAL(timeout()), this, SLOT(updateTables()));

tmr->start();

}

MainWindow::~***MainWindow***(){

delete ui;

}

void MainWindow::**showProcesesNT**(){

// установка привелегий

HANDLE curProcess = GetCurrentProcess();

TOKEN\_PRIVILEGES tp;

LUID sdv;

HANDLE hToken;

OpenProcessToken(curProcess, TOKEN\_ADJUST\_PRIVILEGES | TOKEN\_READ | TOKEN\_QUERY, &hToken );

LookupPrivilegeValue(NULL, SE\_DEBUG\_NAME, &sdv );

tp.PrivilegeCount = 1;

tp.Privileges[0].Luid = sdv;

tp.Privileges[0].Attributes = SE\_PRIVILEGE\_ENABLED;

AdjustTokenPrivileges( hToken, FALSE, &tp, sizeof(tp), (PTOKEN\_PRIVILEGES)NULL, 0 );

ui->pr\_table\_NT->horizontalHeader()->setSectionResizeMode(QHeaderView::ResizeToContents);

QTableWidgetItem \*item;

DWORD len = 1000;

DWORD size\_needed, mod\_size\_needed;

DWORD ID\_arr[len];

HANDLE h;

HMODULE h\_m;

WCHAR w\_name[255];

QString name;

EnumProcesses(ID\_arr, len\*sizeof(DWORD), &size\_needed);

ui->pr\_table\_NT->setRowCount(size\_needed/sizeof(DWORD));

for (uint i = 0; i < size\_needed/sizeof(DWORD); i++){

item = new QTableWidgetItem(QString::number(ID\_arr[i]));

ui->pr\_table\_NT->setItem(i, 0, item);

h = OpenProcess(PROCESS\_QUERY\_INFORMATION | PROCESS\_VM\_READ, false, ID\_arr[i]);

if (!GetModuleBaseNameW(h, NULL, w\_name, 255)) name = "<нет доступа>";

else name = QString::fromWCharArray(w\_name);

item = new QTableWidgetItem(name);

ui->pr\_table\_NT->setItem(i, 1, item);

CloseHandle(h);

}

}

void MainWindow::**showThreadsNT**(){

}

void MainWindow::**showDrivers**(){

ui->drivers\_table\_NT->horizontalHeader()->setSectionResizeMode(QHeaderView::ResizeToContents);

QTableWidgetItem \*item;

DWORD len = 1000;

DWORD size\_needed, name\_size\_needed;

WCHAR w\_name[255];

QString name;

LPVOID dr\_arr[len];

EnumDeviceDrivers(dr\_arr, sizeof(dr\_arr), &size\_needed);

ui->drivers\_table\_NT->setRowCount(size\_needed/sizeof(dr\_arr[0]));

for (uint i = 0; i < size\_needed/sizeof(dr\_arr[0]); i++){

item = new QTableWidgetItem("0x" + QString::number((long long)(dr\_arr[i]), 16));

ui->drivers\_table\_NT->setItem(i, 0, item);

name\_size\_needed = GetDeviceDriverBaseName(dr\_arr[i], w\_name, 255);

w\_name[name\_size\_needed] = '\0';

if (name\_size\_needed == 0) name = "<нет доступа>";

else name = QString::fromWCharArray(w\_name);

item = new QTableWidgetItem(name);

ui->drivers\_table\_NT->setItem(i, 1, item);

}

}

void MainWindow::**updateTables**(){

ui->pr\_table\_NT->clearContents();

ui->pr\_table\_NT->setRowCount(0);

showProcesesNT();

ui->drivers\_table\_NT->clearContents();

ui->drivers\_table\_NT->setRowCount(0);

showDrivers();

//Win 9x

ui->proc\_table->clearContents();

ui->proc\_table->setRowCount(0);

showProcesses();

}

void MainWindow::**showModulesNT**(DWORD proc\_ID, QString proc\_name){

window = new Form();

window->setWindowTitle(proc\_name);

QTableWidget \*table = window->getTable();

table->setColumnCount(5);

QTableWidgetItem \*item = new QTableWidgetItem(QString("Имя модуля"));

table->setHorizontalHeaderItem(0, item);

item = new QTableWidgetItem(QString("Путь к модулю"));

table->setHorizontalHeaderItem(1, item);

item = new QTableWidgetItem(QString("Адрес загрузки модуля"));

table->setHorizontalHeaderItem(2, item);

item = new QTableWidgetItem(QString("Размер модуля"));

table->setHorizontalHeaderItem(3, item);

item = new QTableWidgetItem(QString("Точка входа модуля"));

table->setHorizontalHeaderItem(4, item);

table->horizontalHeader()->setSectionResizeMode(QHeaderView::ResizeToContents);

HANDLE h;

HMODULE h\_m[100];

WCHAR w\_name[255];

QString name;

DWORD h\_m\_needed;

MODULEINFO m\_inf;

h = OpenProcess(PROCESS\_QUERY\_INFORMATION | PROCESS\_VM\_READ, false, proc\_ID);

EnumProcessModules(h, h\_m, 100\*sizeof(DWORD), &h\_m\_needed);

table->setRowCount(h\_m\_needed/sizeof(HMODULE));

for (int i = 0; i < h\_m\_needed/sizeof(HMODULE); i++){

if (!GetModuleBaseNameW(h, h\_m[i], w\_name, 255)) name = "<нет доступа>";

else name = QString::fromWCharArray(w\_name);

item = new QTableWidgetItem(name);

table->setItem(i, 0, item);

if (!GetModuleFileName(h\_m[i], w\_name, 255)) name = "<нет доступа>";

else name = QString::fromWCharArray(w\_name);

item = new QTableWidgetItem(name);

table->setItem(i, 1, item);

GetModuleInformation(h, h\_m[i], &m\_inf, sizeof(MODULEINFO));

item = new QTableWidgetItem("0x" + QString::number((long long)m\_inf.lpBaseOfDll, 16));

table->setItem(i, 2, item);

item = new QTableWidgetItem(QString::number((long long)m\_inf.SizeOfImage));

table->setItem(i, 3, item);

item = new QTableWidgetItem("0x" + QString::number((long long)m\_inf.EntryPoint, 16));

table->setItem(i, 4, item);

}

window->setModal(false);

window->*exec*();

}

void MainWindow::**on\_pr\_table\_NT\_cellClicked**(int row, int column){

DWORD ID = ui->pr\_table\_NT->item(row, 0)->text().toULong();

showModulesNT(ID, ui->pr\_table\_NT->item(row, 1)->text());

}

void MainWindow::**showProcesses**(){

ui->proc\_table->horizontalHeader()->setSectionResizeMode(QHeaderView::ResizeToContents);

HANDLE snap = CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, 0);

if ((long long)snap == -1){

QMessageBox::warning(this, "Ошибка!", "Не удалось сделать снимок процессов системы!");

ui->proc\_table->clearContents();

}

PROCESSENTRY32W proc;

proc.dwSize = sizeof(PROCESSENTRY32W);

bool f = Process32First(snap, &proc);

while (f){

int count = ui->proc\_table->rowCount();

ui->proc\_table->insertRow(count);

ui->proc\_table->setItem(count, 0, new QTableWidgetItem(QString::number(proc.th32ProcessID)));

ui->proc\_table->setItem(count, 1, new QTableWidgetItem(QString::fromWCharArray(proc.szExeFile)));

ui->proc\_table->setItem(count, 2, new QTableWidgetItem(QString::number(proc.cntThreads)));

ui->proc\_table->setItem(count, 3, new QTableWidgetItem(QString::number(proc.th32ParentProcessID)));

ui->proc\_table->setItem(count, 4, new QTableWidgetItem(QString::number(proc.pcPriClassBase)));

f = Process32Next(snap, &proc);

}

CloseHandle(snap);

}

void MainWindow::**showThreadsAndModules**(DWORD proc\_ID, QString name){

window1 = new Form1();

window1->setWindowTitle(name);

QTableWidget \*table = window1->getThTable();

HANDLE snap = CreateToolhelp32Snapshot(TH32CS\_SNAPTHREAD, 0);

if ((long long)snap == -1)

QMessageBox::warning(this, "Ошибка!", "Не удалось сделать снимок потоков системы!");

THREADENTRY32 thr;

thr.dwSize = sizeof(THREADENTRY32);

bool f = Thread32First(snap, &thr);

while (f){

if (thr.th32OwnerProcessID == proc\_ID){

int count = table->rowCount();

table->insertRow(count);

table->setItem(count, 0, new QTableWidgetItem(QString::number(thr.th32ThreadID)));

table->setItem(count, 1, new QTableWidgetItem(QString::number(thr.th32OwnerProcessID)));

table->setItem(count, 2, new QTableWidgetItem(QString::number(thr.tpBasePri)));

}

f = Thread32Next(snap, &thr);

}

CloseHandle(snap);

table = window1->getModTable();

snap = CreateToolhelp32Snapshot(TH32CS\_SNAPMODULE, proc\_ID);

if ((long long)snap == -1)

QMessageBox::warning(this, "Ошибка!", "Не удалось сделать снимок модулей процесса!");

MODULEENTRY32 mod;

mod.dwSize = sizeof(MODULEENTRY32);

f = Module32FirstW(snap, &mod);

while (f){

int count = table->rowCount();

table->insertRow(count);

table->setItem(count, 0, new QTableWidgetItem(QString::number(mod.th32ProcessID)));

table->setItem(count, 1, new QTableWidgetItem(QString::fromWCharArray(mod.szModule)));

table->setItem(count, 2, new QTableWidgetItem(QString::fromWCharArray(mod.szExePath)));

table->setItem(count, 3, new QTableWidgetItem("0x"+QString::number((long long)mod.modBaseAddr, 16)));

table->setItem(count, 4, new QTableWidgetItem(QString::number(mod.modBaseSize)));

table->setItem(count, 5, new QTableWidgetItem("0x"+QString::number((long long)mod.hModule, 16)));

f = Module32NextW(snap, &mod);

}

CloseHandle(snap);

window1->setModal(false);

window1->*exec*();

}

void MainWindow::**on\_proc\_table\_cellClicked**(int row, int column){

DWORD ID = ui->proc\_table->item(row, 0)->text().toULong();

showThreadsAndModules(ID, ui->proc\_table->item(row, 1)->text());

}

main,cpp

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(*argc*, argv);

MainWindow w;

w.show();

return a.exec();

}