**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»**
**(БГТУ им. В.Г.Шухова)**

Лабораторная работа № 2
дисциплина «Операционные системы»
по теме «Архитектура Windows»

Выполнил: студент группы ВТ-31 Макаров Д.С.
Проверил: Михелев В.М.

Белгород 2019

# Лабораторная работа № 2

## «Архитектура Windows»

**Цель работы:** изучение архитектуры операционной системы Windows.

### Вариант 9

Провести исследование ОС с использованием Системного монитора (*Monitor из пункта Administrative Tools*):

- определить количество процессов, потоков, дескрипторов в ОС, изменить их число запуская на выполнение новые приложения;
- определить процент работы в пользовательском режиме (*% User Time*), процент работы в привилегированном режиме (*% Privileged Time*) и процент времени бездействия при выполнении, связанными с интенсивными графическими операциями ( *например, откройте Windows Paint*);
- включить в отчет полученные графики и привести их объяснение.

Составить три программы, которые :

- принимая дескриптор, имя или полное имя модуля, возвращает другие два элемента в своих выходных параметрах. Возможны четыре варианта работы этой функции.
- будет выполнять последовательно по шагам следующее:
  - Используя функцию **GetCurrentProcessId** определить идентификатор текущего процесса.
  - Используя функцию **GetCurrentProcess** определить псевдодескриптор текущего процесса.
  - Используя функцию **DuplicateHandle** и значение псевдодескриптора определить дескриптора текущего процесса. Используя функцию **OpenProcess** определит копию дескриптора текущего процесса.
  - Закроет дескриптор, полученный функцией **DuplicateHandle**.
  - Закроет дескриптор, полученный функцией **OpenProcess**.
- выдает списка перечисления всех процессов, потоков, модулей и их свойства в системе:
  - для Windows NT ( *и список загруженных драйверов устройств*).
  - для Windows 9x.

# Ход работы

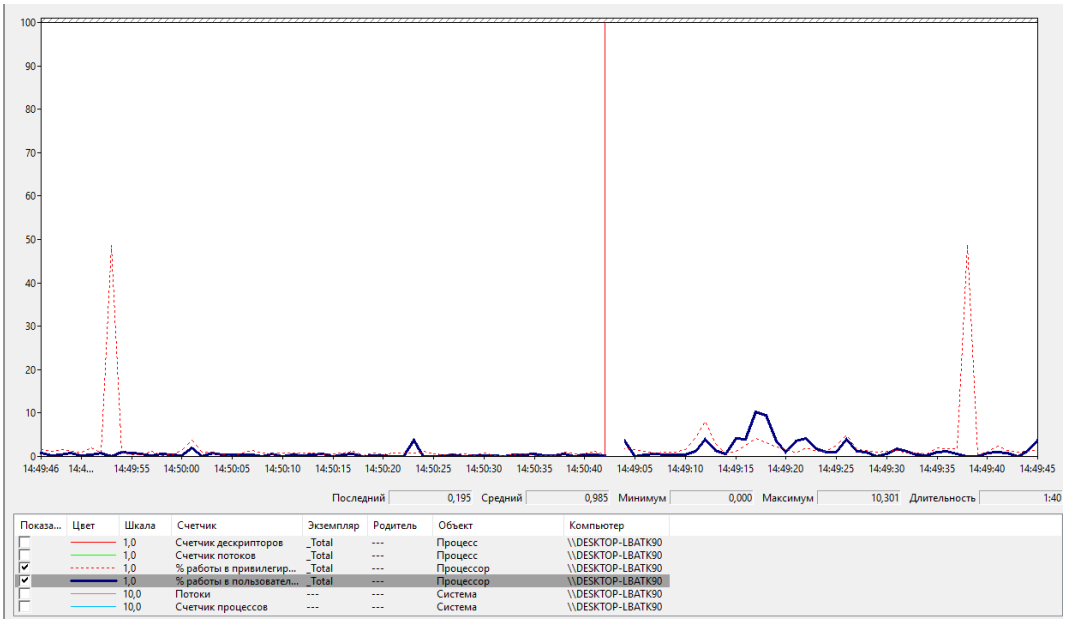



Рис. 1: Графики работы процессора при бездействии

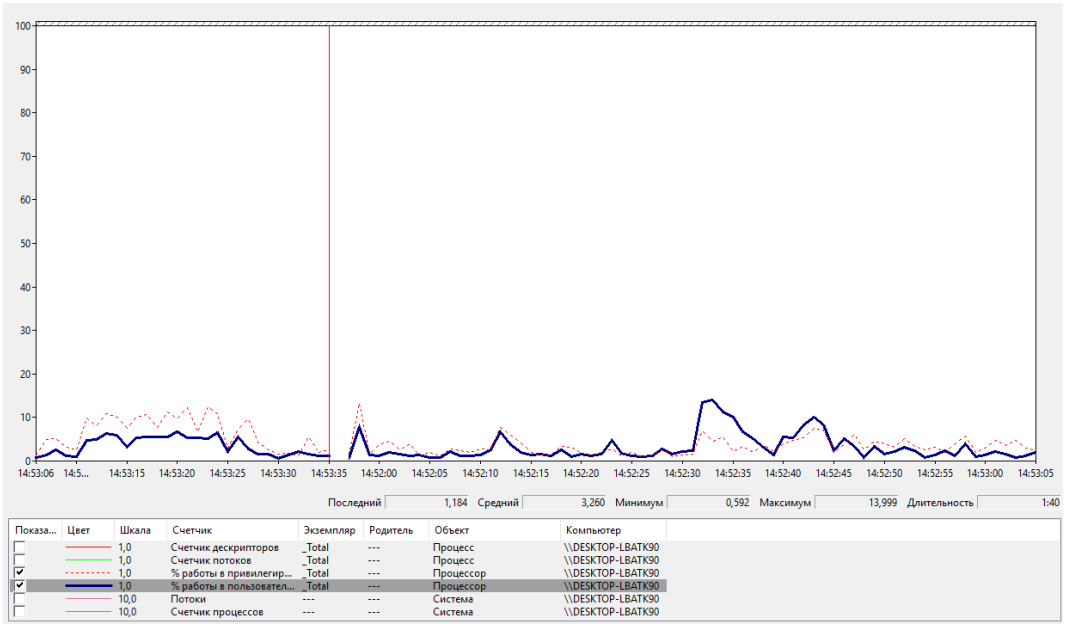



Рис. 2: Графики работы процессора при работе с Paint 3D

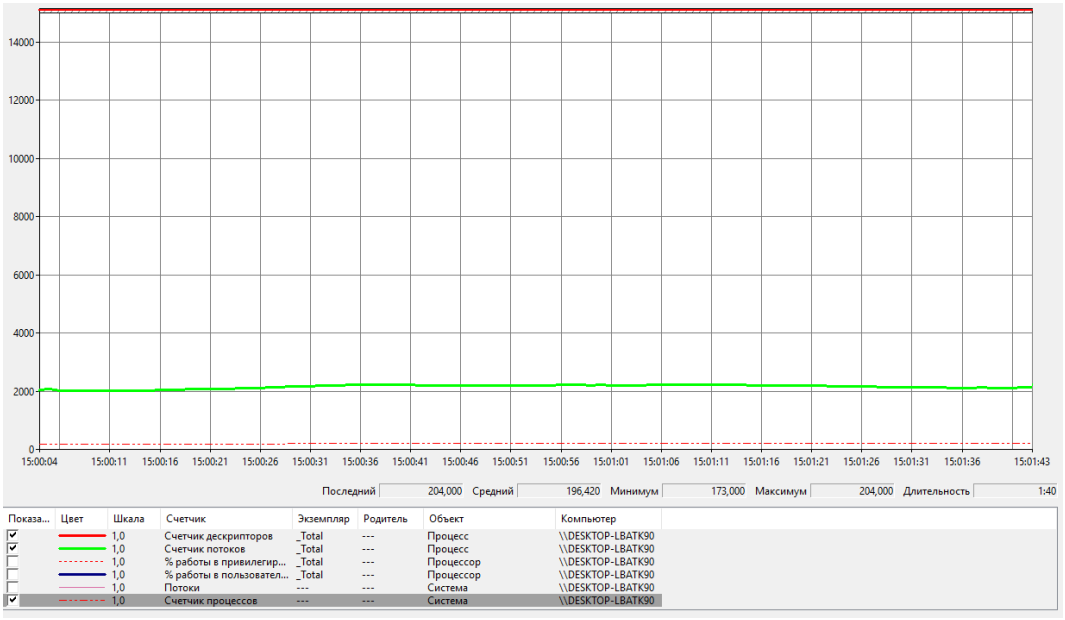



Рис. 3: Графики счетчиков процессов, потоков, дескрипторов

### Бездействие

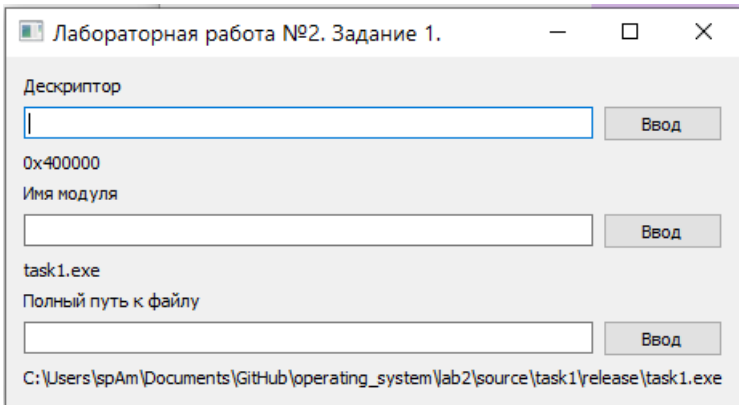
- Дескрипторы - 119 952 шт.
- Потоки - 2061 шт.
- Процессы - 171 шт.

### Открытие 30 экземпляров приложения “Блокнот”

- Дескрипторы - 128 655 шт.
- Потоки - 2170 шт.
- Процессы - 202 шт.

## Примеры работы программы

1. Задание 1.




2. Задание 3.

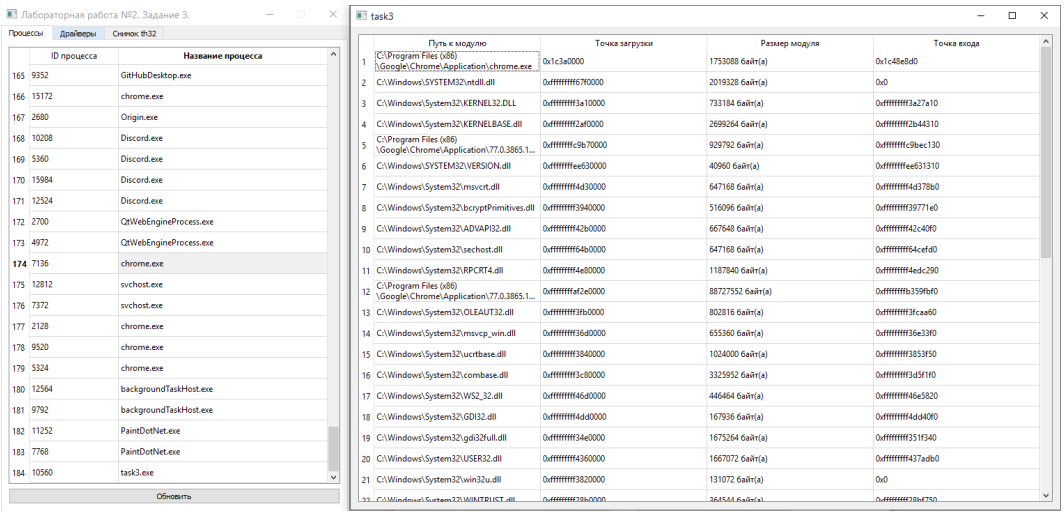



Рис. 4: Вкладка Process

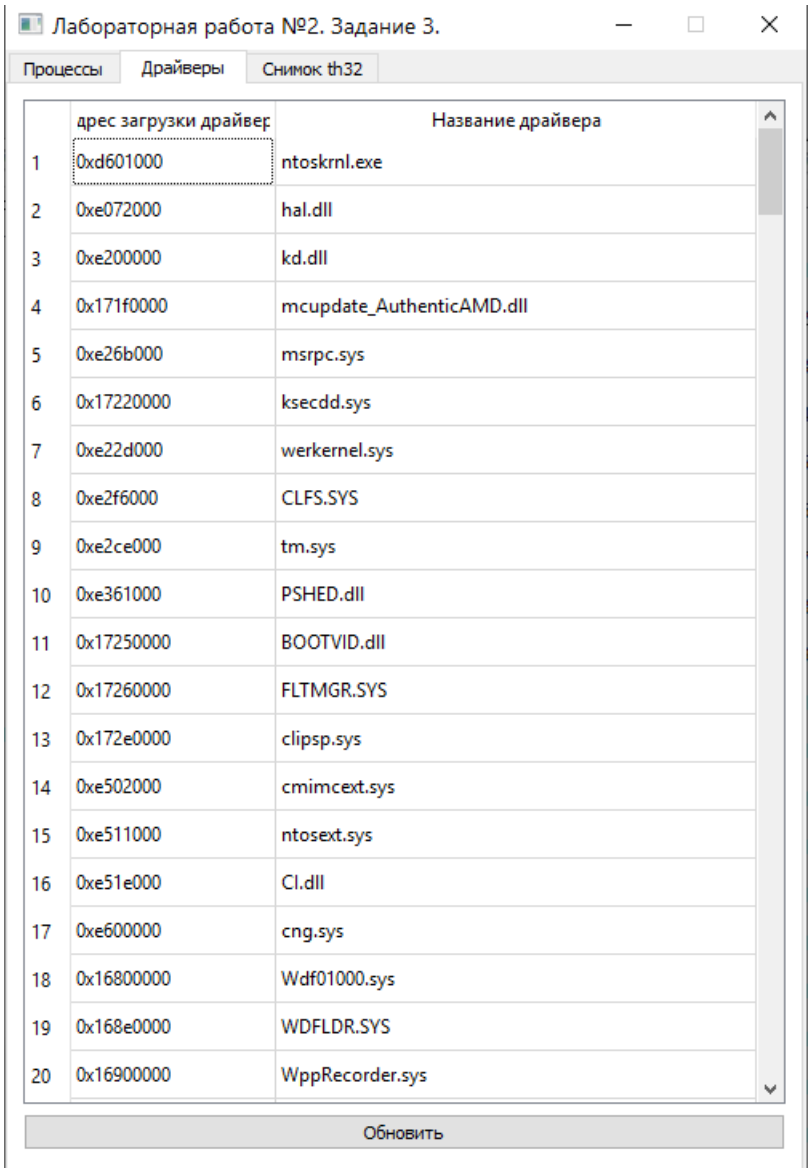



Рис. 5: Вкладка Drivers

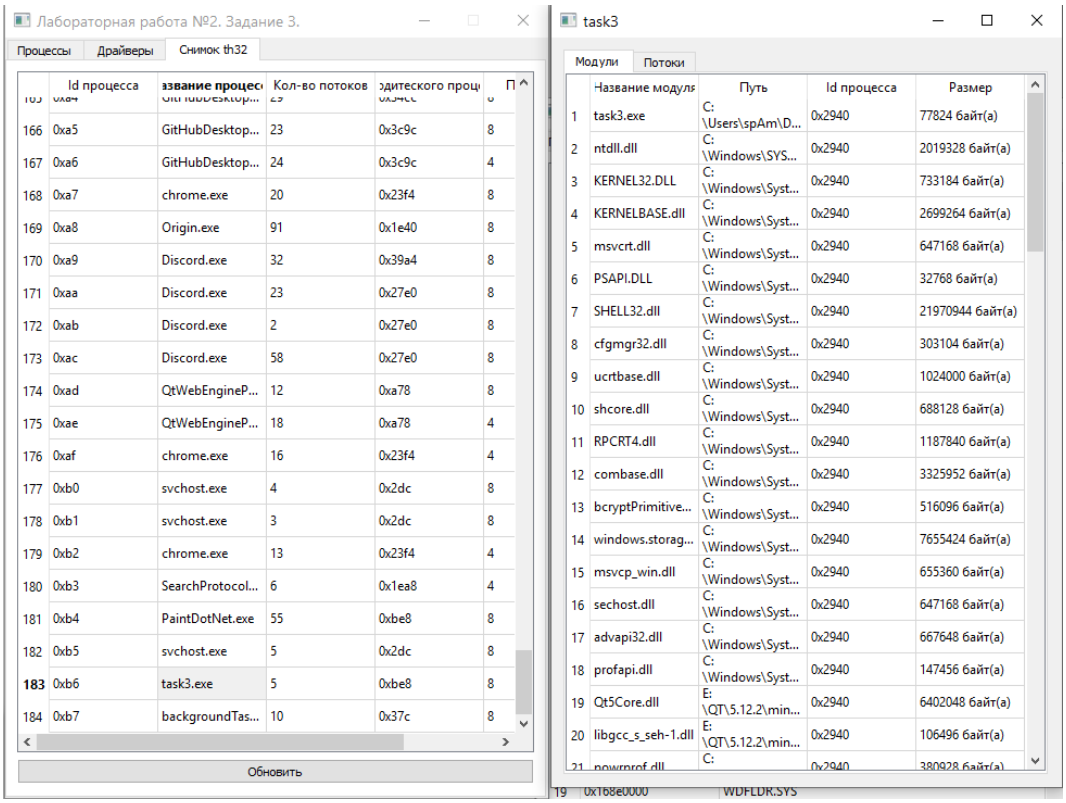



Рис. 6: Вкладка ToolHelper32

# Приложение

## Содержимое файла mainTask1.cpp

```
#include <QtWidgets>
#include <windows.h>
#include <wchar.h>
#include "task1.hpp"

int main (int argc,char* argv[]){
    QApplication* app = new QApplication (argc,argv);
    Task1 *t1 =new Task1;
    t1->setWindowTitle("Лабораторная работа №2. Задание 1.");
    t1->show();
    return app->exec();
}
```

## Содержимое файла mainTask2.cpp

```
#include <Windows.h>
#include <stdio.h>
#include <stdlib.h>

int main(){
    DWORD processId;
    HANDLE fakeProcessHandle;
    HANDLE trueDuplicateProcessHandle;
    HANDLE trueOpenProcessHandle;

    processId = GetCurrentProcessId();
    fakeProcessHandle = GetCurrentProcess();

    if(DuplicateHandle(fakeProcessHandle,
                       fakeProcessHandle,
                       fakeProcessHandle,
                       &trueDuplicateProcessHandle,
                       0,
                       TRUE,
                       DUPLICATE_SAME_ACCESS))
    {
        printf("DuplicateHandle succesfuly create copy of Handle.\n");


    }else{
        printf("DuplicateHandle ERROR.\n");
    };

    if(CloseHandle(trueDuplicateProcessHandle)){
        printf("DuplicateHandle copy of Handle closed.\n");
    }else{
        printf("DuplicateHandle close ERROR.\n");
    };

    trueOpenProcessHandle = OpenProcess(PROCESS_ALL_ACCESS,TRUE,processId);
    if(trueOpenProcessHandle){
        printf("OpenProcess succesfuly create copy of Handle.\n");
    }else{
        printf("OpenProcess ERROR.\n");
    };
```

```
    if(CloseHandle(trueOpenProcessHandle)){
        printf("OpenProcess copy of Handle closed.\n");
    }else{
        printf("OpenProcess close ERROR.\n");
    };
    return 0;
};
```

## Содержимое файла mainTask3.cpp

```
#include <QtWidgets>
#include <windows.h>
#include <wchar.h>
#include "task3.hpp"
int main (int argc,char* argv[]){
    QApplication* app = new QApplication (argc,argv);
    QTabWidget* tabs = new QTabWidget;
    ProcessPage* p1 = new ProcessPage;
    DriversPage* p2 = new DriversPage;
    Th32Page* p3 = new Th32Page;

    tabs->addTab(p1,"Процессы");
    tabs->addTab(p2,"Драйверы");
    tabs->addTab(p3,"Снимок th32");
    tabs->setFixedSize(500,700);
    tabs->setWindowTitle("Лабораторная работа №2. Задание 3.");
    tabs->show();
    return app->exec();
}
```

## Содержимое файла task1.cpp

```
#include "task1.hpp"


Task1::Task1(){
        layout = new QVBoxLayout;

        handle = new QLabel("<адрес дескриптора>");
        name = new QLabel("<имя модуля>");
        path = new QLabel("<путь к файлу>");

        handleLabel = new QLabel("Дескриптор");
        nameLabel = new QLabel("Имя модуля");
        pathLabel = new QLabel("Полный путь к файлу");

        handleLayout = new QHBoxLayout;
        pathLayout = new QHBoxLayout;
        nameLayout = new QHBoxLayout;

        handleEdit = new QLineEdit;
        pathEdit = new QLineEdit;
        nameEdit = new QLineEdit;

        handleButton = new QPushButton("Ввод");
        nameButton = new QPushButton("Ввод");
        pathButton = new QPushButton("Ввод");

        pathLayout->addWidget(pathEdit);
        pathLayout->addWidget(pathButton);
```

```
        handleLayout->addWidget(handleEdit);
        handleLayout->addWidget(handleButton);

        nameLayout->addWidget(nameEdit);
        nameLayout->addWidget(nameButton);

        layout->addWidget(handleLabel);
        layout->addLayout(handleLayout);
        layout->addWidget(handle);

        layout->addWidget(nameLabel);
        layout->addLayout(nameLayout);
        layout->addWidget(name);

        layout->addWidget(pathLabel);
        layout->addLayout(pathLayout);
        layout->addWidget(path);

        this->setLayout(layout);

        int length,moduleLength;
        TCHAR filePath[MAX_PATH];
        TCHAR moduleName[MAX_PATH];
        processID = GetCurrentProcessId();
        handleProcess = OpenProcess(PROCESS_QUERY_INFORMATION,TRUE,processID);
        length = GetModuleFileNameEx(handleProcess,NULL,filePath,MAX_PATH);
        handleModule = GetModuleHandle(filePath);
        moduleLength = GetModuleFileName(handleModule, moduleName, MAX_PATH);
        filename  = QString::fromWCharArray(filePath);
        processName  = QString::fromWCharArray(moduleName);
        processName.remove(0, processName.lastIndexOf("\\") + 1);
        handle->setText("0x"+QString::number((int)handleModule,16));
        path->setText(filename);
        name->setText(processName);
        CloseHandle(handleProcess);

        connect(handleButton,SIGNAL(pressed()),this,SLOT(setInfoByHandle()));
        connect(nameButton,SIGNAL(pressed()),this,SLOT(setInfoByName()));
        connect(pathButton,SIGNAL(pressed()),this,SLOT(setInfoByPath()));
};

void Task1::setInfoByHandle(){
    name->setText("");
    nameEdit->setText("");
    path->setText("");
    pathEdit->setText("");
    HMODULE moduleHandle = (HMODULE)(this->handleEdit->text().toInt(0,0));
    int pathLength;
    TCHAR filePath[MAX_PATH];
    pathLength = GetModuleFileName(moduleHandle, filePath, MAX_PATH);
    if(pathLength){
        QString pathStr = QString::fromWCharArray(filePath);
        path->setText(pathStr);
        name->setText(pathStr.remove(0, pathStr.lastIndexOf("\\") + 1));
    }else{
        QMessageBox *err = new QMessageBox;
        err->setText("Некорректный дескриптор.");
        err->show();
        name->setText("ОШИБКА");
```

```
        path->setText("ОШИБКА");

    }
};
void Task1::setInfoByName(){
    path->setText("");
    pathEdit->setText("");
    handle->setText("");
    handleEdit->setText("");
    HMODULE moduleHandle;
    TCHAR moduleName[MAX_PATH];


    int moduleLength = nameEdit->text().toWCharArray(moduleName);
    moduleName[moduleLength] = 0;

    moduleHandle = GetModuleHandle(moduleName);
    if(moduleHandle==NULL){
        QMessageBox *err = new QMessageBox;
        err->setText("Некорректное имя модуля.");
        err->show();
        path->setText("ОШИБКА");
        handle->setText("ОШИБКА");
    }else{
        TCHAR pathName[MAX_PATH];
        int pathLength = GetModuleFileName(moduleHandle, pathName, MAX_PATH);;

        path->setText(QString::fromWCharArray(pathName));
        handle->setText("0x"+QString::number((int)moduleHandle,16));
    };
};
void Task1::setInfoByPath(){
    name->setText("");
    nameEdit->setText("");
    handle->setText("");
    handleEdit->setText("");
    HMODULE moduleHandle;
    TCHAR moduleName[MAX_PATH];
    QString nameStr = pathEdit->text();
    nameStr = nameStr.remove(0, nameStr.lastIndexOf("\\") + 1);


    int moduleLength = nameStr.toWCharArray(moduleName);
    moduleName[moduleLength] = 0;

    moduleHandle = GetModuleHandle(moduleName);
    if(moduleHandle==NULL){
        QMessageBox *err = new QMessageBox;
        err->setText("Некорректный путь.");
        err->show();
        name->setText("ОШИБКА");
        handle->setText("ОШИБКА");
    }else{
        name->setText(nameStr);
        handle->setText("0x"+QString::number((int)moduleHandle,16));
    };
};
```

## Содержимое файла task1.hpp

```
#include <QtWidgets>
```

```
#include <windows.h>
#include <tchar.h>
#include <psapi.h>
#pragma once

class Task1 : public QWidget{
    Q_OBJECT
    private:
        QString filename;
        QString processName;

        DWORD processID;
        HMODULE handleModule;
        HANDLE handleProcess;

        QVBoxLayout *layout;
        QLabel *handle,*name,*path;
        QLabel *handleLabel,*nameLabel,*pathLabel;
        QHBoxLayout *handleLayout,*nameLayout,*pathLayout;
        QLineEdit *handleEdit,*nameEdit,*pathEdit;
        QPushButton *handleButton,*nameButton,*pathButton;
    public slots:
        void setInfoByHandle();
        void setInfoByName();
        void setInfoByPath();
    public:
        Task1();
};
```

## Содержимое файла task3.cpp

```
#include "task3.hpp"

QString processNameFromId(DWORD processID){
    QString result;
    DWORD length = MAX_PATH+1;
    wchar_t processName[1024];
    //Создаем HANDLE процесса по id с правами чтения памяти процесса и информации о нем.
    HANDLE handleProcess = OpenProcess((PROCESS_QUERY_INFORMATION|PROCESS_VM_READ),FALSE,
    ↪  processID );
    //Получаем имя процесса
    if (NULL != handleProcess ){
        HMODULE handleModule;
        DWORD moduleArrSize;
        //Получаем массив дескрипторов модулей процесса, и по первому модулю получаем его имя.

        ↪  if(EnumProcessModules(handleProcess,&handleModule,sizeof(handleModule),&moduleArrSize)){
            length = GetModuleBaseNameW(handleProcess, handleModule,
            ↪  processName,sizeof(processName)/sizeof(TCHAR) );
        }
    }
    //Закрываем созданный дескриптор процесса.
    CloseHandle(handleProcess);
    //Конвертируем wChar в QString
    if(length){
        processName[length+1] = 0;
        result = QString::fromWCharArray(processName);
        return result;
    }else{
        return QString("Ошибка чтения имени процесса");
```

```
    }
}

int processToQTable(QTableWidget *table){
    table->setRowCount(0);
    table->setColumnCount(2);
    table->setColumnWidth(0,128);
    table->setColumnWidth(1,300);
    table->setHorizontalHeaderLabels({"ID процесса","Название процесса"});
    table->setEditTriggers(QAbstractItemView::NoEditTriggers);

    DWORD processArr[1024], processArrSize, processCount,realProcessCount;
    //Получаем массив всех ID процессов processArr.

    QString systemProcessName = processNameFromId(4);

    if (!EnumProcesses( processArr, sizeof(processArr), &processArrSize)){
        return 1;
    }
    //Вычисляем кол-во полученных процессов.
    processCount = processArrSize / sizeof(DWORD);
    table->setRowCount(processCount);
    realProcessCount = processCount;
    //Заполняем таблицу
    int j = 0;
    for (int i = 0; i < processCount; i++ ){
        if(processArr[i] == 0 ){
            realProcessCount--;
            table->setRowCount(realProcessCount);
        }
        else{
            QString processName = processNameFromId(processArr[i]);
            if(!processName.isEmpty()){
                QTableWidgetItem* idItem = new
                ↪  QTableWidgetItem(QString::number(processArr[i]));
                table->setItem(j,0,idItem);
                QTableWidgetItem* nameItem = new QTableWidgetItem(processName);
                table->setItem(j,1,nameItem);
                j++;
            };
        };
    };
    return 0;
};


ProcessPage::ProcessPage(){
    layout = new QVBoxLayout;
    table = new QTableWidget;
    refreshButton = new QPushButton("Обновить");
    processToQTable(table);
    layout->addWidget(table);
    connect(table,SIGNAL(cellDoubleClicked(int,int)),
            this,SLOT(showModules(int,int)));
     connect(refreshButton,SIGNAL(pressed()),
            this,SLOT(refreshTable()));
    layout->addWidget(refreshButton);
    this->setLayout(layout);
};
```

```
void ProcessPage::showModules(int row, int column){
    unsigned processId = this->table->item(row,0)->text().toInt();
    ModulesTable *modules = new ModulesTable(processId);
    modules->show();
};

ModulesTable::ModulesTable(int processId){
    QVBoxLayout *layout = new QVBoxLayout;
    QTableWidget* table = new QTableWidget;
    modulesToQTable(processId,table);
    layout->addWidget(table);
    this->setLayout(layout);
};

void modulesToQTable(DWORD processID,QTableWidget* table){
    table->setRowCount(0);
    HMODULE hMods[1024];
    HANDLE processHandle;
    HANDLE hProcess;
    DWORD cbNeeded;

    table->setColumnCount(4);
    table->setHorizontalHeaderLabels({"Путь к модулю","Точка загрузки","Размер модуля","Точка
    ↪  входа"});
    table->setEditTriggers(QAbstractItemView::NoEditTriggers);
    table->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

    //Открытие дескриптора потока
    hProcess = OpenProcess( PROCESS_QUERY_INFORMATION |
                            PROCESS_VM_READ,
                            FALSE, processID );
    if (NULL == hProcess){
        QMessageBox msgBox;
        msgBox.setText("Дескриптор потока не был получен");
        msgBox.exec();
        return 0;
    }

    if( EnumProcessModules(hProcess, hMods, sizeof(hMods), &cbNeeded)){
        int modulesCount = cbNeeded / sizeof(HMODULE);
        QMessageBox msgBox1;
        msgBox1.setText("Кол-во модулей: "+QString::number(modulesCount));
        msgBox1.exec();
        table->setRowCount(modulesCount);
        int realModulesCount = modulesCount;
        for (int i = 0,j=0; i < (modulesCount); i++ ){
            wchar_t szModName[MAX_PATH];
            if ( GetModuleFileNameEx( hProcess, hMods[i], szModName,
                                      sizeof(szModName) / sizeof(TCHAR)))
            {
                QTableWidgetItem* nameItem = new
                ↪  QTableWidgetItem(QString::fromWCharArray(szModName));
                table->setItem(j,0,nameItem);

                MODULEINFO moduleInfo;
                QTableWidgetItem *baseItem,*sizeItem,*entryItem;
                if(GetModuleInformation(hProcess,hMods[i],&moduleInfo,sizeof(moduleInfo))){
                    baseItem = new QTableWidgetItem(
                        "0x"+QString::number((int)moduleInfo.lpBaseOfDll,16));
                    sizeItem = new QTableWidgetItem(
```

```
                        QString::number(moduleInfo.SizeOfImage)+" байт(а)");
                    entryItem = new QTableWidgetItem(
                        "0x"+QString::number((int)moduleInfo.EntryPoint,16));
                }else{
                    baseItem = new QTableWidgetItem("Не удалось получить структуру");
                    sizeItem = new QTableWidgetItem("Не удалось получить структуру");
                    entryItem = new QTableWidgetItem("Не удалось получить структуру");
                };

                table->setItem(j,1,baseItem);
                table->setItem(j,2,sizeItem);
                table->setItem(j,3,entryItem);

                j++;

            }else{
                realModulesCount--;
                table->setRowCount(realModulesCount);
            };
        }
    }else{
        QMessageBox msgBox2;
        msgBox2.setText("Не удалось считать список модулей");
        msgBox2.exec();
    };

    //Закрытие дескриптора потока
    CloseHandle(hProcess);
};

void driversToQTable(QTableWidget* table){
    table->setRowCount(0);
    table->setColumnCount(2);
    table->setColumnWidth(0,128);
    table->setColumnWidth(1,300);
    table->setHorizontalHeaderLabels({"Адрес загрузки драйвера","Название драйвера"});
    table->setEditTriggers(QAbstractItemView::NoEditTriggers);

    LPVOID drivers[1024];
    DWORD cbNeeded;
    int cDrivers;
    if( EnumDeviceDrivers(drivers, sizeof(drivers), &cbNeeded) && cbNeeded < sizeof(drivers)){
      char szDriver[1024];
      cDrivers = cbNeeded / sizeof(drivers[0]);
      table->setRowCount(cDrivers);
      for (int i=0; i < cDrivers; i++ ){
        QTableWidgetItem *addressItem = new QTableWidgetItem(
                        "0x"+QString::number((int)drivers[i],16));
        table->setItem(i,0,addressItem);
        if(GetDeviceDriverBaseNameA(drivers[i],
        ↪  szDriver,sizeof(szDriver)/sizeof(szDriver[0]))){
            QTableWidgetItem *nameItem = new QTableWidgetItem(
                        QString(szDriver));
            table->setItem(i,1,nameItem);
        };
      };
    };
};

DriversPage::DriversPage(){
```

```
    layout = new QVBoxLayout;
    table = new QTableWidget;
    refreshButton = new QPushButton("Обновить");
    driversToQTable(table);
    layout->addWidget(table);
    layout->addWidget(refreshButton);
    connect(refreshButton,SIGNAL(pressed()),
            this,SLOT(refreshTable()));
    this->setLayout(layout);
};

Th32Page::Th32Page(){
    layout = new QVBoxLayout;
    tableProcess = new QTableWidget;
    refreshButton = new QPushButton("Обновить");
    th32SnapToQTable(tableProcess);
    layout->addWidget(tableProcess);
    connect(tableProcess,SIGNAL(cellDoubleClicked(int,int)),
            this,SLOT(showSubTables(int,int)));
    connect(refreshButton,SIGNAL(pressed()),
            this,SLOT(refreshTable()));
    layout->addWidget(refreshButton);
    this->setLayout(layout);
};

void th32SnapToQTable(QTableWidget* table){
    table->setRowCount(0);
    HANDLE hProcessSnap;
    HANDLE hProcess;
    PROCESSENTRY32 pe32;
    DWORD dwPriorityClass;

    table->setColumnCount(5);
    table->setHorizontalHeaderLabels({"Id процесса","Название процесса","Кол-во потоков","Id
    ↪  родитеского процесса","Приоритет"});
    table->setEditTriggers(QAbstractItemView::NoEditTriggers);
    int i=0;
    // Получаем снимок
    hProcessSnap = CreateToolhelp32Snapshot( TH32CS_SNAPPROCESS, 0 );
    if( hProcessSnap == INVALID_HANDLE_VALUE ){
        QMessageBox boxError1;
        boxError1.setText("Ошибка получения снимка.");
        boxError1.exec();
    }

    // Устанавливаем размер структуры
    pe32.dwSize = sizeof( PROCESSENTRY32 );

    // Получаем информацию о первом процессе в списке
    // при неудаче завершаем работу.
    if( !Process32First( hProcessSnap, &pe32 ) ){
        QMessageBox boxError2;
        boxError2.setText("Ошибка получения информации о процессе.");
        boxError2.exec();
        CloseHandle( hProcessSnap );
    }
    do{
        i++;
    }
    while(Process32Next( hProcessSnap, &pe32 ));
```

```
    table->setRowCount(i);
    Process32First( hProcessSnap, &pe32);
    i=0;
    // Заносим информацию о процессах в таблицу
    do{
        i++;
        QTableWidgetItem* nameItem = new
        ↪  QTableWidgetItem(QString::fromWCharArray(pe32.szExeFile));
        table->setItem(i,1,nameItem);

        QTableWidgetItem* idItem;
        QTableWidgetItem* threadCountItem;
        QTableWidgetItem* parentIdItem;
        QTableWidgetItem* idPriorityBaseItem;

        idItem = new QTableWidgetItem("0x"+QString::number(i,16));
        table->setItem(i,0,idItem);

        threadCountItem = new QTableWidgetItem(QString::number(pe32.cntThreads));
        table->setItem(i,2,threadCountItem);

        parentIdItem = new
        ↪  QTableWidgetItem("0x"+QString::number(pe32.th32ParentProcessID,16));
        table->setItem(i,3,parentIdItem);

        idPriorityBaseItem = new QTableWidgetItem(QString::number(pe32.pcPriClassBase));
        table->setItem(i,4,idPriorityBaseItem);

    } while( Process32Next( hProcessSnap, &pe32 ) );

    CloseHandle( hProcessSnap );

};

void Th32Page::showSubTables(int row, int column){
    unsigned processId = this->tableProcess->item(row,0)->text().toInt();
    Th32SubTable *subTable = new Th32SubTable(processId);
    subTable->show();
};

Th32SubTable::Th32SubTable(int processId){
    layout = new QVBoxLayout;
    tabs = new QTabWidget;
    tableModules = new QTableWidget;
    th32ModulesToQTable(processId,tableModules);
    tableThreads = new QTableWidget;
    th32ThreadsToQTable(processId,tableThreads);
    tabs->addTab(tableModules,"Модули");
    tabs->addTab(tableThreads,"Потоки");
    layout->addWidget(tabs);
    this->setLayout(layout);
}

void th32ThreadsToQTable(int processId,QTableWidget* table){
    table->setRowCount(0);
    table->setColumnCount(2);
    table->setHorizontalHeaderLabels({"Id потока","Базовый приоритет"});
    table->setEditTriggers(QAbstractItemView::NoEditTriggers);

    HANDLE hThreadSnap = INVALID_HANDLE_VALUE;
```

```
    THREADENTRY32 te32;
    hThreadSnap = CreateToolhelp32Snapshot( TH32CS_SNAPTHREAD, processId );
    if( hThreadSnap == INVALID_HANDLE_VALUE ){
        return;
    };

    te32.dwSize = sizeof(THREADENTRY32 );

    if(!Thread32First( hThreadSnap, &te32 )){
        CloseHandle( hThreadSnap  );
        return;
    };

    int i = 0;
    do i++; while( Thread32Next( hThreadSnap, &te32 ) );
    table->setRowCount(i);
    i=0;

    QTableWidgetItem* idItem;
    QTableWidgetItem* priorityItem;

    Thread32First( hThreadSnap, &te32 );
    do {
        idItem = new QTableWidgetItem(
            QString::number(te32.th32ThreadID )
        );
        priorityItem = new QTableWidgetItem(
            QString::number(te32.tpBasePri )
        );

        table->setItem(i,0,idItem);
        table->setItem(i,1,priorityItem);
        i++;
    } while( Thread32Next( hThreadSnap, &te32 ) );
    CloseHandle( hThreadSnap );
};

void th32ModulesToQTable(int processId,QTableWidget* table){
    table->setRowCount(0);
    table->setColumnCount(4);
    table->setHorizontalHeaderLabels({"Название модуля","Путь","Id процесса","Размер"});
    table->setEditTriggers(QAbstractItemView::NoEditTriggers);

    HANDLE hModuleSnap = INVALID_HANDLE_VALUE;
    MODULEENTRY32 me32;
    hModuleSnap = CreateToolhelp32Snapshot( TH32CS_SNAPMODULE, processId );

    if( hModuleSnap == INVALID_HANDLE_VALUE ){
        return;
    };

    me32.dwSize = sizeof( MODULEENTRY32 );

    if(!Module32First( hModuleSnap, &me32)){
        CloseHandle( hModuleSnap );     // Must clean up the snapshot object!
        return;
    };

    int i = 0;
    do i++; while( Module32Next( hModuleSnap, &me32 ) );
```

```
    table->setRowCount(i);
    i=0;

    QTableWidgetItem* nameItem;
    QTableWidgetItem* pathItem;
    QTableWidgetItem* processIdItem;
    QTableWidgetItem* sizeItem;

    Module32First( hModuleSnap, &me32);
    do {
        nameItem = new QTableWidgetItem(
            QString::fromWCharArray(me32.szModule)
        );
        pathItem = new QTableWidgetItem(
            QString::fromWCharArray(me32.szExePath)
        );
        processIdItem = new QTableWidgetItem(
            "0x"+QString::number(me32.th32ProcessID,16)
        );
        sizeItem = new QTableWidgetItem(
            QString::number(me32.modBaseSize)+" байт(а)"
        );

        table->setItem(i,0,nameItem);
        table->setItem(i,1,pathItem);
        table->setItem(i,2,processIdItem);
        table->setItem(i,3,sizeItem);
        i++;
    } while( Module32Next( hModuleSnap, &me32 ) );
    CloseHandle( hModuleSnap );
};


void ProcessPage::refreshTable(){
    processToQTable(table);
};

void DriversPage::refreshTable(){
    driversToQTable(table);
};

void Th32Page::refreshTable(){
    th32SnapToQTable(tableProcess);
};
```

## Содержимое файла task3.hpp

```
#include <QtWidgets>
#include <windows.h>
#include <psapi.h>
#include <tlhelp32.h>
#include <tchar.h>
#pragma once

QString processNameFromId(DWORD processID);
int processToQTable(QTableWidget *table);
void modulesToQTable(DWORD processId,QTableWidget* table);
void driversToQTable(QTableWidget* table);
void th32SnapToQTable(QTableWidget* table);
void th32ThreadsToQTable(int processId,QTableWidget* table);
```

```
void th32ModulesToQTable(int processId,QTableWidget* table);

class ModulesTable:public QWidget{
    Q_OBJECT
    public:
        ModulesTable(int processId);
};

class ProcessPage : public QWidget{
    Q_OBJECT
    private:
        QVBoxLayout *layout;
        QTableWidget* table;
        QPushButton* refreshButton;
    public slots:
        void refreshTable();
        void showModules(int row, int column);
    public:
        ProcessPage();
};

class DriversPage : public QWidget{
    Q_OBJECT
    private:
        QVBoxLayout *layout;
        QTableWidget* table;
        QPushButton* refreshButton;
    public slots:
        void refreshTable();
    public:
        DriversPage();
};

class Th32Page : public QWidget{
    Q_OBJECT
    private:
        QVBoxLayout *layout;
        QTableWidget* tableProcess;
        QPushButton* refreshButton;
    public slots:
        void showSubTables(int row, int column);
        void refreshTable();
    public:
        Th32Page();
};


class Th32SubTable : public QWidget{
    Q_OBJECT
    private:
        QVBoxLayout *layout;
        QTabWidget* tabs;
        QTableWidget* tableModules;
        QTableWidget* tableThreads;
    public:
        Th32SubTable(int processId);
};
```