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 № 4
дисциплина «Операционные системы»
по теме «Архитектура памяти Windows»

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

Белгород 2019

Лабораторная работа № 4

«Архитектура памяти Windows»

Цель работы: получение практических навыков по использованию Win32 API для исследования памяти Windows.

Вариант 9

Содержание отчета:

1. Наименование лабораторной работы, ее цель.
2. Краткое изложение теоретических основ архитектуры памяти ОС Windows.
3. Разработать программное обеспечение для приложения, которое:
   - выдает информацию, получаемую при использовании API GetSystemInfo.
   - выдает информацию, получаемую при использовании API GlobalMemoryStatus. При выводе информации использовать диаграммы.
   - составляет карту виртуальной памяти для любого процесса.
4. Примеры разработанных приложений (результаты и тексты программ).

Ход работы

Краткие теоритические сведения

Физическая память - оперативная память (RAM), установленная в компьютер, каждый байт этой памяти имеет свой адрес (физический адрес) он равен $n - 1$, где n - номер байта в массиве памяти.

Внешняя память - долговременная память представленная внешними носителями.

Файл подкачки (swap) - файл находящийся на внешнем носителе (локальном диске или flash носителе(*ReadyBoost*)), используемый точно так же как и ram память, имеет больший обьем но гораздо меньшую скорость чтения записи по сравнению с ram.

Виртуальная память - набор чисел представленный в виде набора виртуальных адресов, эти адреса нельзя использовать для непосредственного обращения к физической памяти, однако после некоторых преобразований из вирутальных адресов получаются физические адреса с требуемыми данными, за это ответственен диспетчер виртуальной памяти (*VMM*). Минимальным оперируемым блоком памяти в *VMM* является страница(page), размером 4 байта. Виртуальные адреса проецируются как на физическую память, так и на файлы подкачки.

Каждый процесс в Windows получает свое виртуальное адресное пространство, объемом 4 Гб:

- 0-64 Кб : зарезервированно для NULL указателей(неинициализированных).
- 128 Кб - 2 Гб : находятся модули программы,dll и другие файлы отображаемые в память, доступно в пользовательском режиме.
- 0-64 Кб : зарезервированно для некорректных указателей.
- 2 Гб : находятся драйвера устройств и другие системные объекты Windows, не доступно в пользовательском режиме.

Один и тот же участок памяти в физической памяти может быть отображен в разных виртуальных адресных пространствах по разным адресам, это позволяет совместно использовать этот участок памяти несколькими процессами.

Windows фиксирует состояние каждой физической страницы памяти в структуре данных называемой Page Frame Database. Каждая физическая страница может находиться в одном из восьми стояний: * **активная** (*active*), страница в текущий момент отображается на виртуальную память

- **переходная** (*transition*), в состоянии переходном к активному состоянию
- **ждущая** (*standby*), страница вышла из состояния active, но осталась неизменной
- **измененная** (*modified*), страница вышла из состояния active. ее содержимое изменено но еще не записано на диск
- **измененная** незаписанная (*modified no write*), страница находится в состоянии «измененная», но особо помечена как страница, содержимое которой не записано на диск. Используется драйверами файловой системы Windows
- **свободная** (*free*), страница свободна, но содержит произвольные записи, не может использоваться процессом
- **очищенная** (*zeroed*), страница свободна и инициализирована нулями потоком нулевой страницы. Может быть использована процессом
- **страница с ошибками** (*bad*), странице были обнаружены ошибки или другие аппаратные ошибки,в следствии чего не может быть использована

Примеры работы программы

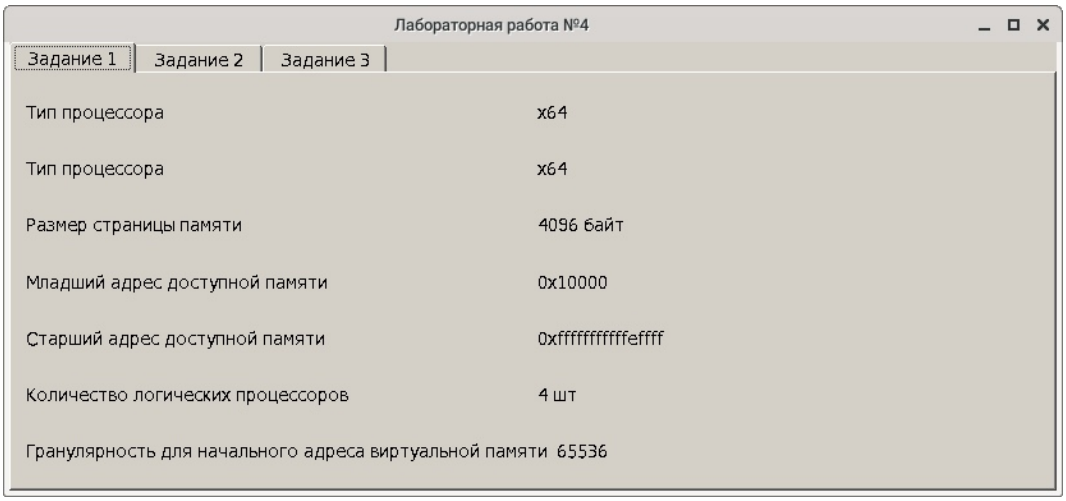

Рис. 1: Задание 1. Информация полученная при помощи API GetSystemInfo

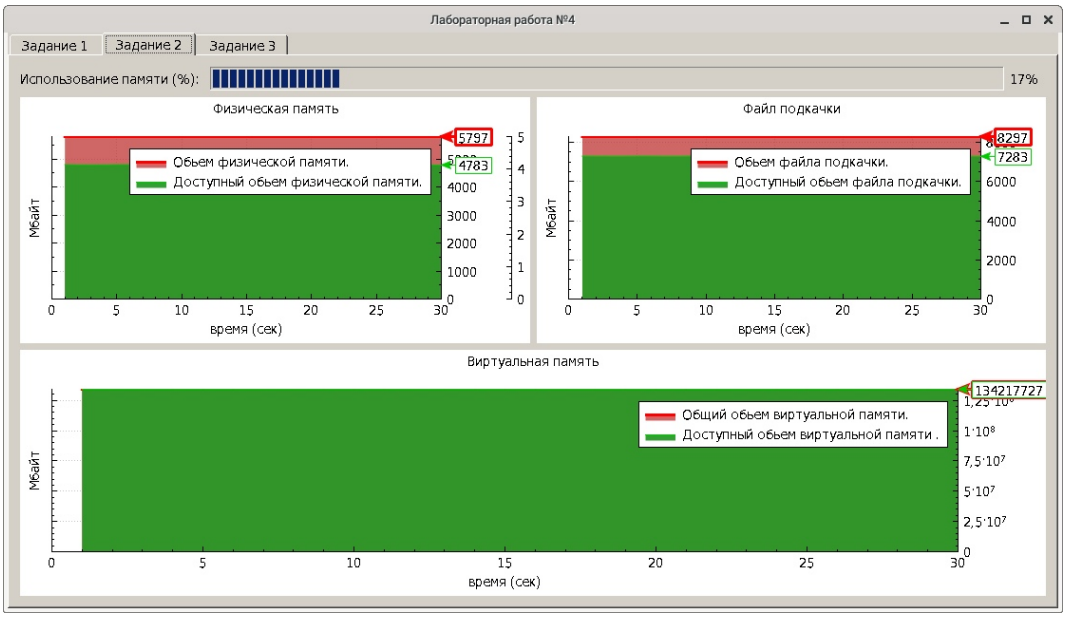

Рис. 2: Задание 2. Диаграммы использования памяти.

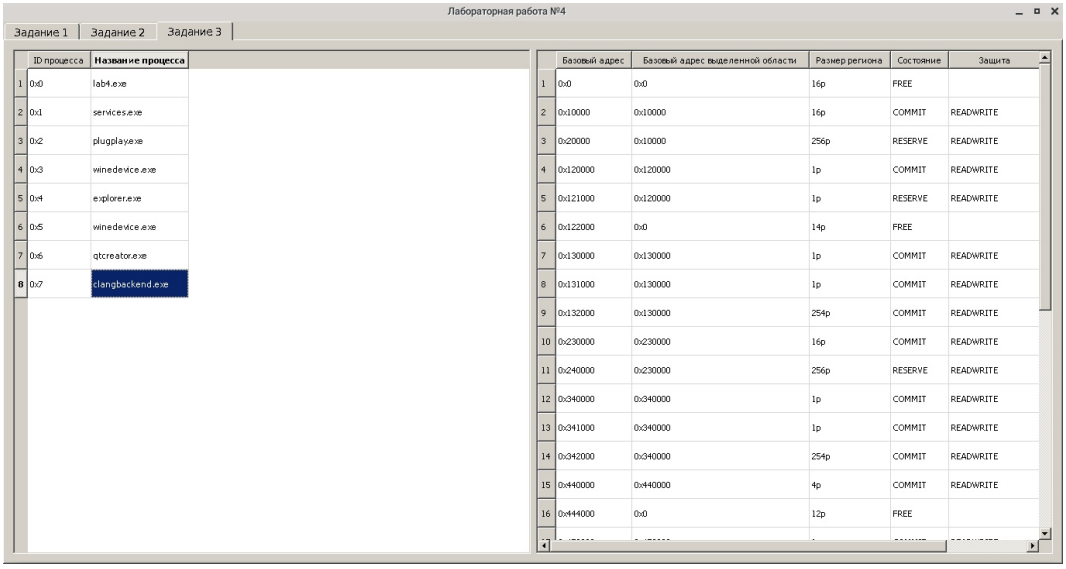

Рис. 3: Задание 3. Карта виртуальной памяти.

Приложение

Содержимое файла main.cpp

```
#include <QtWidgets>
#include <windows.h>
#include <wchar.h>
#include "task1.hpp"
#include "task2.hpp"
#include "task3.hpp"

int main (int argc,char* argv[]){
    QApplication* app = new QApplication (argc,argv);
    QTabWidget *lab4 = new QTabWidget;
    Task1 *t1 = new Task1;
    Task2 *t2 = new Task2;
    Task3 *t3 = new Task3;
    lab4->setWindowTitle("Лабораторная работа №4");
    lab4->addTab(t1,QString("Задание 1"));
    lab4->addTab(t2,QString("Задание 2"));
    lab4->addTab(t3,QString("Задание 3"));
    lab4->show();
    return app->exec();
}
```

Содержимое файла task1.cpp

```
#include "task1.hpp"

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

    SYSTEM_INFO info;
    GetSystemInfo(&info);

    //Архитектура процессора
    QHBoxLayout *lCpuArch = new QHBoxLayout;
    QLabel *nCpuArch = new QLabel(QString("Тип процессора"));
    QLabel *iCpuArch = new QLabel;
    QString strCpuArch;
    switch(info.wProcessorArchitecture){
        case PROCESSOR_ARCHITECTURE_AMD64:
            strCpuArch = "x64";
            break;
        case PROCESSOR_ARCHITECTURE_IA64 :
            strCpuArch = "Itanium";
            break;
        case PROCESSOR_ARCHITECTURE_INTEL :
            strCpuArch = "x86";
            break;
        default:
            strCpuArch = "Неизвестный тип";
            break;
    };
    iCpuArch->setText(strCpuArch);
    lCpuArch->addWidget(nCpuArch);
    lCpuArch->addWidget(iCpuArch);


    //Тип процессора
```

```
QHBoxLayout *lCpuType = new QHBoxLayout;
QLabel *nCpuType = new QLabel(QString("Тип процессора"));
QLabel *iCpuType = new QLabel;
QString strCpuType;
switch(info.dwProcessorType){
    case PROCESSOR_INTEL_386:
        strCpuType = "i386";
        break;
    case PROCESSOR_INTEL_486:
        strCpuType = "i486";
        break;
    case PROCESSOR_INTEL_PENTIUM:
        strCpuType = "Intel Pentium";
        break;
    case PROCESSOR_INTEL_IA64:
        strCpuType = "Intel x64";
        break;
    case PROCESSOR_AMD_X8664 :
        strCpuType = "x64";
        break;
    default:
        strCpuType = "Неопознанно";
        break;
};
iCpuType->setText(strCpuType);
lCpuType->addWidget(nCpuType);
lCpuType->addWidget(iCpuType);

//Размер страницы памяти
QHBoxLayout *lPageSize = new QHBoxLayout;
QLabel *nPageSize = new QLabel(QString("Размер страницы памяти"));
QLabel *iPageSize = new QLabel;
QString strPageSize = QString::number(info.dwPageSize) + QString(" байт");
iPageSize->setText(strPageSize);
lPageSize->addWidget(nPageSize);
lPageSize->addWidget(iPageSize);

//Указатель на младший адрес доступной памяти
QHBoxLayout *lMinimumApplicationAddress = new QHBoxLayout;
QLabel *nMinimumApplicationAddress = new QLabel(QString("Младший адрес доступной
↪  памяти"));
QLabel *iMinimumApplicationAddress = new QLabel;
QString strMinimumApplicationAddress = QString("0x") +
↪  QString::number((int)info.lpMinimumApplicationAddress,16);
iMinimumApplicationAddress->setText(strMinimumApplicationAddress);
lMinimumApplicationAddress->addWidget(nMinimumApplicationAddress);
lMinimumApplicationAddress->addWidget(iMinimumApplicationAddress);

//Указатель на старший адрес доступной памяти
QHBoxLayout *lMaximumApplicationAddress = new QHBoxLayout;
QLabel *nMaximumApplicationAddress = new QLabel(QString("Старший адрес доступной
↪  памяти"));
QLabel *iMaximumApplicationAddress = new QLabel;
QString strMaximumApplicationAddress = QString("0x") +
↪  QString::number((int)info.lpMaximumApplicationAddress,16);
iMaximumApplicationAddress->setText(strMaximumApplicationAddress);
lMaximumApplicationAddress->addWidget(nMaximumApplicationAddress);
lMaximumApplicationAddress->addWidget(iMaximumApplicationAddress);

//Количество процессоров
```

```
    QHBoxLayout *lCpuCount = new QHBoxLayout;
    QLabel *nCpuCount = new QLabel(QString("Количество логических процессоров"));
    QLabel *iCpuCount = new QLabel;
    QString strCpuCount = QString::number(info.dwNumberOfProcessors) + QString(" шт");
    iCpuCount->setText(strCpuCount);
    lCpuCount->addWidget(nCpuCount);
    lCpuCount->addWidget(iCpuCount);


    //Количество процессоров
    QHBoxLayout *lAllocationGranularity = new QHBoxLayout;
    QLabel *nAllocationGranularity = new QLabel(QString("Гранулярность для начального адреса
    ↪  виртуальной памяти"));
    QLabel *iAllocationGranularity = new QLabel;
    QString strAllocationGranularity = QString::number(info.dwAllocationGranularity);
    iAllocationGranularity->setText(strAllocationGranularity);
    lAllocationGranularity->addWidget(nAllocationGranularity);
    lAllocationGranularity->addWidget(iAllocationGranularity);

    //установка layout-ов
    layout->addLayout(lCpuArch);
    layout->addLayout(lCpuType);
    layout->addLayout(lPageSize);
    layout->addLayout(lMinimumApplicationAddress);
    layout->addLayout(lMaximumApplicationAddress);
    layout->addLayout(lCpuCount);
    layout->addLayout(lAllocationGranularity);
    this->setLayout(layout);
};
```

Содержимое файла task1.hpp

```
#include <QtWidgets>
#include <windows.h>

class Task1: public QWidget{
    Q_OBJECT
    private:
    public:
        Task1();

};
```

Содержимое файла task2.cpp

```
#include "task2.hpp"

Task2::Task2(){
    QVBoxLayout* layout = new QVBoxLayout;
    QGridLayout* plotLayout = new QGridLayout;


    customPlot1 = new QCustomPlot;
    customPlot2 = new QCustomPlot;
    customPlot3 = new QCustomPlot;

    QCPTextElement *title1 = new QCPTextElement(customPlot1);
    title1->setText("Физическая память");
    customPlot1->plotLayout()->insertRow(0);
    customPlot1->plotLayout()->addElement(0, 0, title1);
```

```
QCPTextElement *title2 = new QCPTextElement(customPlot2);
title2->setText("Файл подкачки");
customPlot2->plotLayout()->insertRow(0);
customPlot2->plotLayout()->addElement(0, 0, title2);

QCPTextElement *title3 = new QCPTextElement(customPlot3);
title3->setText("Виртуальная память");
customPlot3->plotLayout()->insertRow(0);
customPlot3->plotLayout()->addElement(0, 0, title3);

customPlot1->xAxis->setLabel("время (сек)");
customPlot1->yAxis->setLabel("Мбайт");
customPlot2->xAxis->setLabel("время (сек)");
customPlot2->yAxis->setLabel("Мбайт");
customPlot3->xAxis->setLabel("время (сек)");
customPlot3->yAxis->setLabel("Мбайт");

memStatus.dwLength = sizeof(MEMORYSTATUSEX);

dataTimer = new QTimer;
dataTimer->setInterval(1000);

customPlot1->yAxis->setTickLabels(false);
connect(customPlot1->yAxis, SIGNAL(rangeChanged(QCPRange)), customPlot1->yAxis2,
↪  SLOT(setRange(QCPRange)));
customPlot1->yAxis2->setVisible(true);
customPlot1->axisRect()->addAxis(QCPAxis::atRight);
customPlot1->axisRect()->axis(QCPAxis::atRight, 0)->setPadding(30);
customPlot1->legend->setVisible(true);

customPlot2->yAxis->setTickLabels(false);
connect(customPlot2->yAxis, SIGNAL(rangeChanged(QCPRange)), customPlot2->yAxis2,
↪  SLOT(setRange(QCPRange)));
customPlot2->yAxis2->setVisible(true);
customPlot2->axisRect()->axis(QCPAxis::atRight, 0)->setPadding(30);
customPlot2->legend->setVisible(true);

customPlot3->yAxis->setTickLabels(false);
connect(customPlot3->yAxis, SIGNAL(rangeChanged(QCPRange)), customPlot3->yAxis2,
↪  SLOT(setRange(QCPRange)));
customPlot3->yAxis2->setVisible(true);
customPlot3->axisRect()->axis(QCPAxis::atRight, 0)->setPadding(30);
customPlot3->legend->setVisible(true);

//Общий объем физической памяти в мегабайтах
customPlot1->addGraph()->setName("Обьем физической памяти.");
customPlot1->graph(0)->setPen(QPen(QColor("#FF0000"),Qt::DotLine));
customPlot1->graph(0)->setBrush(QBrush(QColor("#BBBF3030")));

//Объем доступной физической памяти в байтах
customPlot1->addGraph()->setName("Доступный обьем физической памяти.");
customPlot1->graph(1)->setPen(QPen(QColor("#00CC00")));
customPlot1->graph(1)->setBrush(QBrush(QColor("#EF269926")));

//Размер файла подкачки в байтах
customPlot2->addGraph()->setName("Обьем файла подкачки.");
customPlot2->graph(0)->setPen(QPen(QColor("#FF0000"),Qt::DotLine));
customPlot2->graph(0)->setBrush(QBrush(QColor("#BBBF3030")));
```

```
    //Доступный объем байтов в файле подкачки
    customPlot2->addGraph()->setName("Доступный обьем файла подкачки.");
    customPlot2->graph(1)->setPen(QPen(QColor("#00CC00")));
    customPlot2->graph(1)->setBrush(QBrush(QColor("#EF269926")));

    //Общий объем виртуальной памяти в байтах
    customPlot3->addGraph()->setName("Общий обьем виртуальной памяти.");
    customPlot3->graph(0)->setPen(QPen(QColor("#FF0000"),Qt::DotLine));
    customPlot3->graph(0)->setBrush(QBrush(QColor("#BBBF3030")));

    //Объем доступной виртуальной памяти
    customPlot3->addGraph()->setName("Доступный обьем виртуальной памяти .");
    customPlot3->graph(1)->setPen(QPen(QColor("#00CC00")));
    customPlot3->graph(1)->setBrush(QBrush(QColor("#EF269926")));

    mTag1_0 = new AxisTag(customPlot1->graph(0)->valueAxis());
    mTag1_1 = new AxisTag(customPlot1->graph(1)->valueAxis());
    mTag2_0 = new AxisTag(customPlot2->graph(0)->valueAxis());
    mTag2_1 = new AxisTag(customPlot2->graph(1)->valueAxis());
    mTag3_0 = new AxisTag(customPlot3->graph(0)->valueAxis());
    mTag3_1 = new AxisTag(customPlot3->graph(1)->valueAxis());
    mTag1_0->setPen(customPlot1->graph(0)->pen());
    mTag1_1->setPen(customPlot1->graph(1)->pen());
    mTag2_0->setPen(customPlot2->graph(0)->pen());
    mTag2_1->setPen(customPlot2->graph(1)->pen());
    mTag3_0->setPen(customPlot3->graph(0)->pen());
    mTag3_1->setPen(customPlot3->graph(1)->pen());

    connect(dataTimer, SIGNAL(timeout()), this, SLOT(realtimeDataSlot()));
    dataTimer->start(1000);

    QHBoxLayout* lMemUsage = new QHBoxLayout;
    QLabel* nMemUsage = new QLabel("Использование памяти (%): ");
    memUsage = new QProgressBar;
    memUsage->setMaximum(100);
    memUsage->setMinimum(0);
    lMemUsage->addWidget(nMemUsage);
    lMemUsage->addWidget(memUsage);

    layout->addLayout(lMemUsage);
    plotLayout->addWidget(customPlot1,0,0);

    plotLayout->addWidget(customPlot2,0,1);

    plotLayout->addWidget(customPlot3,1,0,1,2);

    layout->addLayout(plotLayout);
    this->setLayout(layout);
};

void Task2::realtimeDataSlot(){
    GlobalMemoryStatusEx(&memStatus);
    static double key = 0;
    key += 1;
    if(key == 1000){
        key=0;
        customPlot1->graph(0)->data()->clear();
        customPlot1->graph(1)->data()->clear();
        customPlot2->graph(0)->data()->clear();
        customPlot2->graph(1)->data()->clear();
```

```
        customPlot3->graph(0)->data()->clear();
        customPlot3->graph(1)->data()->clear();
    };
    memUsage->setValue((int)memStatus.dwMemoryLoad);
    customPlot1->graph(0)->addData(key,memStatus.ullTotalPhys/1024/1024);
    customPlot1->graph(1)->addData(key,memStatus.ullAvailPhys/1024/1024);
    customPlot2->graph(0)->addData(key,memStatus.ullTotalPageFile/1024/1024);
    customPlot2->graph(1)->addData(key,memStatus.ullAvailPageFile/1024/1024);
    customPlot3->graph(0)->addData(key,memStatus.ullTotalVirtual/1024/1024);
    customPlot3->graph(1)->addData(key,memStatus.ullAvailVirtual/1024/1024);

    customPlot1->graph(0)->rescaleAxes(true);
    customPlot1->graph(1)->rescaleAxes(true);
    customPlot2->graph(0)->rescaleAxes(true);
    customPlot2->graph(1)->rescaleAxes(true);
    customPlot3->graph(0)->rescaleAxes(true);
    customPlot3->graph(1)->rescaleAxes(true);

    mTag1_0->updatePosition(memStatus.ullTotalPhys/1024/1024);
    mTag1_1->updatePosition(memStatus.ullAvailPhys/1024/1024);
    mTag2_0->updatePosition(memStatus.ullTotalPageFile/1024/1024);
    mTag2_1->updatePosition(memStatus.ullAvailPageFile/1024/1024);
    mTag3_0->updatePosition(memStatus.ullTotalVirtual/1024/1024);
    mTag3_1->updatePosition(memStatus.ullAvailVirtual/1024/1024);

    mTag1_0->setText(QString::number(memStatus.ullTotalPhys / 1024 / 1024));
    mTag1_1->setText(QString::number(memStatus.ullAvailPhys / 1024 / 1024));
    mTag2_0->setText(QString::number(memStatus.ullTotalPageFile / 1024 / 1024));
    mTag2_1->setText(QString::number(memStatus.ullAvailPageFile / 1024 / 1024));
    mTag3_0->setText(QString::number(memStatus.ullTotalVirtual / 1024 / 1024));
    mTag3_1->setText(QString::number(memStatus.ullAvailVirtual / 1024 / 1024));

    customPlot1->replot();
    customPlot2->replot();
    customPlot3->replot();
};


AxisTag::AxisTag(QCPAxis *parentAxis) :
  QObject(parentAxis),
  mAxis(parentAxis)
{
  mDummyTracer = new QCPItemTracer(mAxis->parentPlot());
  mDummyTracer->setVisible(false);
  mDummyTracer->position->setTypeX(QCPItemPosition::ptAxisRectRatio);
  mDummyTracer->position->setTypeY(QCPItemPosition::ptPlotCoords);
  mDummyTracer->position->setAxisRect(mAxis->axisRect());
  mDummyTracer->position->setAxes(0, mAxis);
  mDummyTracer->position->setCoords(1, 0);
  mArrow = new QCPItemLine(mAxis->parentPlot());
  mArrow->setLayer("overlay");
  mArrow->setClipToAxisRect(false);
  mArrow->setHead(QCPLineEnding::esSpikeArrow);
  mArrow->end->setParentAnchor(mDummyTracer->position);
  mArrow->start->setParentAnchor(mArrow->end);
  mArrow->start->setCoords(15, 0);
  mLabel = new QCPItemText(mAxis->parentPlot());
  mLabel->setLayer("overlay");
  mLabel->setClipToAxisRect(false);
  mLabel->setPadding(QMargins(3, 0, 3, 0));
```

```
  mLabel->setBrush(QBrush(Qt::white));
  mLabel->setPen(QPen(Qt::blue));
  mLabel->setPositionAlignment(Qt::AlignLeft|Qt::AlignVCenter);
  mLabel->position->setParentAnchor(mArrow->start);
}

AxisTag::~AxisTag()
{
  if (mDummyTracer)
    mDummyTracer->parentPlot()->removeItem(mDummyTracer);
  if (mArrow)
    mArrow->parentPlot()->removeItem(mArrow);
  if (mLabel)
    mLabel->parentPlot()->removeItem(mLabel);
}

void AxisTag::setPen(const QPen &pen)
{
  mArrow->setPen(pen);
  mLabel->setPen(pen);
}

void AxisTag::setBrush(const QBrush &brush)
{
  mLabel->setBrush(brush);
}

void AxisTag::setText(const QString &text)
{
  mLabel->setText(text);
}

void AxisTag::updatePosition(double value)
{
  mDummyTracer->position->setCoords(1, value);
  mArrow->end->setCoords(mAxis->offset(), 0);
}
```

Содержимое файла task2.hpp

```
#include <QtWidgets>
#include <windows.h>
#include "qcustomplot.h"


class AxisTag : public QObject
{
  Q_OBJECT
public:
  explicit AxisTag(QCPAxis *parentAxis);
  virtual ~AxisTag();

  // setters:
  void setPen(const QPen &pen);
  void setBrush(const QBrush &brush);
  void setText(const QString &text);

  // getters:
  QPen pen() const { return mLabel->pen(); }
  QBrush brush() const { return mLabel->brush(); }
```

```
  QString text() const { return mLabel->text(); }

  // other methods:
  void updatePosition(double value);

protected:
  QCPAxis *mAxis;
  QPointer<QCPItemTracer> mDummyTracer;
  QPointer<QCPItemLine> mArrow;
  QPointer<QCPItemText> mLabel;
};


class Task2: public QWidget{
    Q_OBJECT
    private:
        QCustomPlot *customPlot1, *customPlot2, *customPlot3;
        AxisTag *mTag1_0, *mTag1_1, *mTag2_0, *mTag2_1, *mTag3_0, *mTag3_1;
        MEMORYSTATUSEX memStatus;
        QTimer *dataTimer;
        QProgressBar *memUsage;
    public:
        Task2();
    public slots:
        void realtimeDataSlot();
};
```

Содержимое файла task3.cpp

```
#include "task3.hpp"

Task3::Task3(){
    QHBoxLayout *layout = new QHBoxLayout;
    processList = new QTableWidget;
    memMap = new QTableWidget;

    th32SnapToQTable(processList);

    connect(processList,SIGNAL(cellDoubleClicked(int,int)),
            this,SLOT(memoryMapBuild(int,int)));
    connect(this,SIGNAL(clearMapTable()),
            memMap,SLOT(clear()));
    layout->addWidget(processList);
    layout->addWidget(memMap);
    this->setLayout(layout);
};

void Task3::memoryMapBuild(int row, int column){
    QString rowProcessName = this->processList->item(row,1)->text();
    emit clearMapTable();
    QMessageBox *msg = new QMessageBox;
    msg->setText(rowProcessName);
    msg->show();
    memoryMapToQTable(memMap,rowProcessName);
};

void th32SnapToQTable(QTableWidget* table){
    // Снимки из ToolHelp32
    table->setRowCount(0);
    HANDLE hProcessSnap;
```

```
HANDLE hProcess;
PROCESSENTRY32 pe32;
DWORD dwPriorityClass;

table->setColumnCount(2);
table->setHorizontalHeaderLabels({"ID процесса","Название процесса"});
table->setEditTriggers(QAbstractItemView::NoEditTriggers);
int i=0;
// Получаем снимок
hProcessSnap = CreateToolhelp32Snapshot(
            TH32CS_SNAPPROCESS, // Включить процессы
            0 // текущий процесс
            );
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 );
}
// Перебор процессов для определения количества
do{
    i++;
}while(Process32Next(hProcessSnap, &pe32));
table->setRowCount(i);
Process32First(hProcessSnap, &pe32);
i=0;
// Заносим информацию о процессах в таблицу
do{
    i++;
    // Имя
    QTableWidgetItem* nameItem = new
    ↪  QTableWidgetItem(QString::fromWCharArray(pe32.szExeFile));
    table->setItem(i,1,nameItem);

    QTableWidgetItem* idItem;

    // ID
    idItem = new QTableWidgetItem("0x"+QString::number(i,16));
    table->setItem(i,0,idItem);
    table->resizeColumnsToContents();
} while(Process32Next(hProcessSnap, &pe32));

// В первую строку допишем текущий процесс
i = 0;
// Имя
QTableWidgetItem* nameItem = new
↪  QTableWidgetItem(QString::fromWCharArray(pe32.szExeFile));
table->setItem(i,1,nameItem);
```

```
    QTableWidgetItem* idItem;

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

void memoryMapToQTable(QTableWidget* table,QString processName){
    table->setColumnCount(6);
    table->setHorizontalHeaderLabels({"Базовый адрес","Базовый адрес выделенной
    ↪  области","Размер региона","Состояние","Защита","Тип"});
    table->setEditTriggers(QAbstractItemView::NoEditTriggers);

    int i = 0;
    table->setRowCount(1);
    MEMORY_BASIC_INFORMATION mbi;
    SYSTEM_INFO info;
    GetSystemInfo(&info);
    int hMod = 0;
    int pageSize= 4096;
    DWORD lpList = 0;
    HANDLE hProcess = getProcessHandleByName(processName);
    if(hProcess == NULL){
        int last_error=GetLastError();
        QMessageBox *errHandle = new QMessageBox;
        errHandle->setText(QString("Дескриптор не был получен. Error
        ↪  code:")+QString::number(last_error));
        errHandle->show();
    }
    else{
        int flag = 1;
        while (flag) {
                table->setRowCount(i+1);
                flag = VirtualQueryEx(hProcess, (LPCVOID)hMod, &mbi,
                ↪  sizeof(MEMORY_BASIC_INFORMATION));
                hMod = hMod + mbi.RegionSize;
                //QMessageBox *msg = new QMessageBox;
                //msg->setText(QString::number(i)+":
                ↪  "+"0x"+QString::number((DWORD)mbi.BaseAddress,16)+" flag:
                ↪  "+QString::number(flag));
                //msg->show();

                QTableWidgetItem* baseAdress = new
                ↪  QTableWidgetItem("0x"+QString::number((long)mbi.BaseAddress,16));
                QTableWidgetItem* allocationBase = new
                ↪  QTableWidgetItem("0x"+QString::number((long)mbi.AllocationBase,16));
                QTableWidgetItem* regionSize = new
                ↪  QTableWidgetItem(QString::number(mbi.RegionSize/pageSize)+QString("p"));
                QString state;
                switch(mbi.State){
                  case MEM_COMMIT:
                     state = "COMMIT";
                     break;
                  case MEM_FREE:
                     state = "FREE";
                     break;
                  case MEM_RESERVE:
                     state = "RESERVE";
                     break;
```

```
                };
                QTableWidgetItem* stateTable = new QTableWidgetItem(state);
                QString type;
                switch(mbi.Type){
                    case MEM_IMAGE:
                        state = "IMAGE";
                        break;
                    case MEM_MAPPED:
                        state = "MAPPED";
                        break;
                    case MEM_PRIVATE:
                        state = "PRIVATE";
                        break;
                };
                QTableWidgetItem* typeTable = new QTableWidgetItem(type);
                QString protect;
                switch(mbi.AllocationProtect){
                    case PAGE_EXECUTE:
                        protect = "EXECUTE";
                        break;
                    case PAGE_EXECUTE_READ:
                        protect = "EXECUTE_READ:";
                        break;
                    case PAGE_EXECUTE_READWRITE:
                        protect = "EXECUTE_READWRITE";
                        break;
                    case PAGE_EXECUTE_WRITECOPY:
                        protect = "EXECUTE_WRITECOPY";
                        break;
                    case PAGE_READONLY:
                        protect = "READONLY";
                        break;
                    case PAGE_READWRITE:
                        protect = "READWRITE";
                        break;
                    case PAGE_WRITECOPY:
                        protect = "WRITECOPY";
                        break;
                }
                QTableWidgetItem* protectTable = new QTableWidgetItem(protect);
                table->setItem(i,0,baseAdress);
                table->setItem(i,1,allocationBase);
                table->setItem(i,2,regionSize);
                table->setItem(i,3,stateTable);
                table->setItem(i,4,protectTable);
                table->setItem(i,5,typeTable);
                table->resizeColumnsToContents();
                i++;
                if(i==30){
                    flag=0;
                }
            }
        }
        CloseHandle(hProcess);
};

HANDLE getProcessHandleByName(QString &pName){
    char fname[255];
    int str_size = 255;
    DWORD* processesId = new DWORD[255];
```

```
    DWORD bytes, size = 255 * sizeof(DWORD);
    //id процессов, /кол-во эл типа dword, передаваемых с hepl первого параметра/, кол-во байт
    if (EnumProcesses(processesId, size, &bytes)) {
        uint count = bytes / sizeof(DWORD);

        for (uint i = 0; i < count; i++) {
            HANDLE h = OpenProcess(PROCESS_QUERY_INFORMATION | PROCESS_VM_READ | SYNCHRONIZE,
            ↪  false, (int) processesId[i]);
            if (GetModuleFileNameExA(h, NULL, fname, size)) {
                QString name = fname;
                name.remove(0, name.lastIndexOf("\\") + 1);
                if (name == pName){
                    QMessageBox *msg = new QMessageBox;
                    msg->setText(QString("Дескриптор найден.")+name+" - "+pName );
                    msg->show();
                    return h;
                }
            }
            CloseHandle(h);
        }
    }
    return NULL;
}
```

Содержимое файла task3.hpp

```
#include <QtWidgets>
#include <windows.h>
#include <tlhelp32.h>
#include <psapi.h>

void th32SnapToQTable(QTableWidget* table);
void memoryMapToQTable(QTableWidget* table,QString processName);
HANDLE getProcessHandleByName(QString &pName);

class Task3: public QWidget{
    Q_OBJECT
    private:
        QTableWidget *processList;
        QTableWidget *memMap;
    public:
        Task3();
        void updateMap(int processId);
    public slots:
        void memoryMapBuild(int row, int column);
    signals:
        void clearMapTable();

};
```