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

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №4

Операционные системы

Архитектура памяти Windows

Выполнил: ст. группы ВТ-31  
Ковалёв И. Д.

Проверил: Михелев В. М.

**Белгород 2019**

**Типы памяти**

Физическая память – реальные микросхемы RAM, установленные в компьютере. Каждый байт такой памяти имеет физический адрес. Является исполняемой, то есть из нее можно читать и в нее процессор может записывать данные.

Виртуальная память – адреса автоматически преобразуются с помощью оборудования в физические адреса. Только базовые части ядра ОС пропускают это преобразования и напрямую используют физические адреса. Применяется всегда, даже если объем памяти, требуемый всем запущенным процессам, не превышает объем ОЗУ, установленным в системе.

Страничный блок памяти – самый маленький блок памяти. Объем – 4 кб.

Файл подкачки – страничный файл, располагающийся на жестком диске. Используется для хранения данных и программ так же, как и физическая память, но его объем превышает объем физической памяти. Используется, когда RAM недостаточно для работы некоторых программ.

Если диапазон виртуальных адресов согласуется с адресами в файле подкачки более, чем с адресами физический памяти, то говорят, что виртуальные адреса с проецированы на файл подкачки.

Набор виртуальных адресов может проецироваться на физическую память, файл подкачки или любой другой файл.

Физическая память используется совместно, если она отображается на виртуальное адресное пространство нескольких процессов, хотя виртуальные адреса в каждом процессе могут отличаться.

Каждый процесс получает виртуальное адресное пространство, равное 4 гб (займет 0 – 2^32).

Каждая страница виртуального адресного пространства может находится в таких состояниях:

Reserved – страница зарезервирована для использования

Commited – для данной виртуальной страницы выделена физическая память в файле подкачки или в файле, отображаемом в память.

Free – данная страница не зарезервирована и не передана, и поэтому в данный момент она недоступна для процесса.

Так же, страница имеет параметр защиты, который используется для защиты от доступа, соответствующего вновь выделенной виртуальной памяти.

Гранулярность

Если адрес не является кратным 64 кб, то система округляет указанный адрес в меньшую сторону до ближайшего числа, кратного 64 кб. Windows всегда выравнивает начальный адрес виртуальной памяти на границу гранулярности распределения, которая является числом, кратным 64 кб.

Так же, выделяемый объем памяти всегда кратен объему системной страницы, то есть 4 кб.

Структура ВАП:

В пользовательском ВАП располагаются исполняемый образ процесса, динамически подключаемые библиотеки (DLL, dynamic-link library), куча процесса и стеки потоков.

При запуске программы создается процесс, при этом в память загружаются код и данные программы (исполняемый образ, executable image), а также необходимые программе динамически подключаемые библиотеки (DLL). Формируется куча (heap) – область, в которой процесс может выделять память динамическим структурам данных (т. е. структурам, размер которых заранее неизвестен, а определяется в ходе выполнения программы). По умолчанию размер кучи составляет 1 МБ, но при компиляции приложения или в ходе выполнения процесса может быть изменен. Кроме того, каждому потоку предоставляется стек (stack) для хранения локальных переменных и параметров функций, также по умолчанию размером 1 МБ.



Существуют такие алгоритмы распределения памяти:

1. Без использования внешней памяти

* Фиксированными разделами
* Динамическими разделами
* Перемещаемыми разделами

1. С использованием внешней памяти

* Страничное распределение
* Сегментное распределение
* Сегментно-страничное распределение

**Фиксированные разделы**: Память разбивается на несколько разделов фиксированной величины. Очередной новый процесс, поступивший на выполнение, помещается либо в общую очередь, либо в очередь к некоторому разделу. Подсистема управления памятью в этом случае выполняет такие задачи:

* Сравнивает объем памяти, требуемый для вновь поступившего процесса, с размерами свободных разделов и выбирает подходящий раздел.
* Осуществляет загрузку программы в один из разделов и настройку адресов.

Преимущества: Простота реализации.

Недостатки: Жесткость, поскольку уровень мультипрограммирования заранее ограничен числом разделов.

**Динамические разделы** В этом случае память машины не делится на разделы заранее. Каждому вновь поступающему приложению на этапе создания процесса выделяется вся необходимая ему память (если не хватает памяти – процесс не создается). После выполнения память освобождается.

Для реализации такого подхода необходима ОС, которая будет выполнять такие действия:

* Ведение таблиц свободных и занятых разделов, в которых указываются начальные адреса и размеры участков памяти.
* При создании нового процесса анализировать оставшуюся память путем просмотра таблицы свободных областей и выбор раздела, размер которого достаточен для размещения кодов и данных нового процесса.
* Загрузка программы в выделенный ей раздел и корректировка таблиц свободных и занятых областей.

**Перемещаемые разделы:**

Один из методов борьбы с фрагментацией – перемещение всех занятых участков в сторону старших или младших адресов, так, чтобы вся свободная память образовала единую свободную область. ОС должна сжимать содержимое памяти.

Виртуализация оперативной памяти осуществляется средствами ОС и включает решение следующих задач:

* Размещение данных в запоминающих устройствах разного типа, например часть кодов программы – на диске, часть – в ОЗУ.
* Выбор образов процессов или их частей для перемещения из оперативной памяти на диск и обратно.
* Перемещение по мере необходимости данных между памятью и диском.
* Преобразование виртуальных адресов в физические.

Существуют такие подходы:

* Свопинг – образы процессов выгружаются на диск и возвращаются в оперативную память целиком. Почти не используется в современных ОС.
* Виртуальная память – между оперативной памятью и диском перемещаются части образов процессов.

Для временного хранения сегментов и страниц на диске отводятся специальная область, которая называется файлом свопинга или страничным файлом.

Виртуальная память:

* Страничная – организует перемещение данных между памятью и диском страницами – небольшими частями виртуального адресного пространства, фиксированного и сравнительно небольшого размера.
* Сегментная – предусматривает перемещение данных сегментами – частями виртуального адресного пространства произвольного размера, полученными с учетом смыслового значения данных.
* Сегментно-страничная – использует двухуровневое деление – виртуальное адресное пространство делится на сегменты, а они, в свою очередь, на страницы. Единицей перемещения данных здесь является страница.

Дескриптор страницы – таблица страниц, включающая в себя информацию о номере физической страницы, в которую загружена данная виртуальная страница, признак присутствия, устанавливаемый в 1, если данная страница находится в оперативной памяти, признак модификации страницы, который устанавливается в 1 когда производится запись по адресу, относящемуся к данной странице, признак обращения к странице, который устанавливается в 1 при каждом обращении по адресу, относящемуся к данной странице.

Базисные свойства страничной организации:

* Объем страницы – степень двойки.
* В пределах страницы непрерывная последовательность виртуальных адресов однозначно отображается в непрерывную последовательность физических адресов.

Для двухуровневой структуризации виртуального адресного пространства существует такая схема преобразования:

1. Путем отбрасывания k + n младших разрядов в виртуальном адресном пространстве определяется номер раздела, к которому принадлежит данный виртуальный адрес.
2. По этому номеру из таблицы разделов извлекается дескриптор соответствующей таблицы страниц. Проверяется, находится ли данная таблица страниц в памяти. Если нет, происходит страничное прерывание и система загружает нужную страницу с диска.
3. Далее из этой таблицы страниц извлекается дескриптор виртуальной страницы, номер которой содержится в средних n разделах преобразуемого виртуального адреса. Снова выполняется проверка наличия данной страницы в памяти и при необходимости ее загрузка.
4. Из дескриптора определяется номер физической страницы, в которую загружена данная виртуальная страница. К номеру физической страницы пристыковывается смещение, взятое из k младших разрядов виртуального адреса. В результате получается искомый физический адрес.

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

**Задания к работе:**

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.

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QMap>

#include <QTimer>

#include <QTime>

#include <QtCharts/QChart>

#include <QtCharts/QChartView>

#include <QtCharts/QAreaSeries>

#include <QtMath>

#include <windows.h>

#include <psapi.h>

#include "customchart.h"

QT\_BEGIN\_NAMESPACE

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

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*public*:

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

~***MainWindow***();

void **showSystemInfo**();

void **showMemoryStatus**();

void **showMemoryMap**(QString processName);

void **getProcesses**();

void **updateProcesses**();

*private* slots:

void **on\_chooseProcess\_currentTextChanged**(*const* QString &arg1);

void **updateCharts**();

*private*:

Ui::MainWindow \*ui;

QMap <QString, HANDLE> processes;

QString prevName;

QTimer \* updateTimer;

QTime \* time;

CustomChart \* memoryUsedInPercents, \* memoryUsed, \* virtualUsed, \* virtualAvaliable, \* swapFileUsed;

};

#endif *//* *MAINWINDOW\_H*

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QDebug>

MainWindow::**MainWindow**(QWidget \*parent):QMainWindow(parent), ui(*new* Ui::MainWindow){

ui->setupUi(*this*);

ui->tabWidget->setTabText(0, "Системная информация");

showSystemInfo();

getProcesses();

ui->tabWidget->setTabText(1, "Карта виртуальной памяти");

ui->virtualMemoryMap->clear();

ui->virtualMemoryMap->clearContents();

ui->virtualMemoryMap->setRowCount(0);

ui->virtualMemoryMap->*reset*();

ui->virtualMemoryMap->setColumnCount(7);

ui->virtualMemoryMap->setHorizontalHeaderLabels({"Базовый адрес", "Конечный адрес","Количество страниц",

"Размер области","Состояние памяти", "Тип защиты", "Тип области"});

ui->virtualMemoryMap->horizontalHeader()->setSectionResizeMode (QHeaderView::*Stretch*);

updateProcesses();

time = *new* QTime; time->start();

updateTimer = *new* QTimer; updateTimer->start();

ui->charts->setLayout(*new* QVBoxLayout);

ui->vcharts->setLayout(*new* QVBoxLayout);

showMemoryStatus();

connect(updateTimer, &QTimer::timeout, *this*, &MainWindow::updateCharts);

updateTimer->setInterval(1000);

}

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

*delete* ui;

}

void MainWindow::**showSystemInfo**(){

SYSTEM\_INFO info;

GetSystemInfo(&info);

*switch* (info.wProcessorArchitecture) {

*case* PROCESSOR\_ARCHITECTURE\_AMD64:

ui->processorArch->setText("AMD x64");

*break*;

*case* PROCESSOR\_ARCHITECTURE\_ARM:

ui->processorArch->setText("ARM");

*break*;

*case* PROCESSOR\_ARCHITECTURE\_IA64:

ui->processorArch->setText("IA-64");

*break*;

*case* PROCESSOR\_ARCHITECTURE\_INTEL:

ui->processorArch->setText("Intel x86");

*break*;

*case* PROCESSOR\_ARCHITECTURE\_UNKNOWN:

ui->processorArch->setText("Не удалось получить информацию об архитектуре процессора");

*break*;

}

*switch* (info.dwProcessorType) {

*case* PROCESSOR\_INTEL\_386:

ui->processorType->setText("i386");

*break*;

*case* PROCESSOR\_INTEL\_486:

ui->processorType->setText("i486");

*break*;

*case* PROCESSOR\_INTEL\_PENTIUM:

ui->processorType->setText("Pentium (i586)");

*break*;

*case* PROCESSOR\_INTEL\_IA64:

ui->processorType->setText("Itanium");

*break*;

*case* PROCESSOR\_AMD\_X8664:

ui->processorType->setText("x86-64");

*break*;

}

ui->processorCores->setText(QString::number(info.dwNumberOfProcessors));

ui->processorMask->setText(QString::number(info.dwActiveProcessorMask, 2));

ui->processorLevel->setText(QString::number(info.wProcessorLevel));

ui->processorRevision->setText("Family: " + QString::number(HIBYTE(info.wProcessorRevision)) + ", Stepping: " + QString::number(LOBYTE(info.wProcessorRevision)));

ui->pageSize->setText(QString::number(info.dwPageSize / 1024) + " кб");

ui->lowerBorder->setText("0x" + QString::number((long long)info.lpMinimumApplicationAddress, 16).toUpper().rightJustified(16, '0'));

ui->upperBorder->setText("0x" + QString::number((long long)info.lpMaximumApplicationAddress, 16).toUpper().rightJustified(16, '0'));

ui->gr->setText(QString::number(info.dwAllocationGranularity / 1024) + " кб");

}

void MainWindow::**updateProcesses**(){

getProcesses();

ui->chooseProcess->clear();

*for* (*auto* name : processes.keys())

ui->chooseProcess->addItem(name);

}

void MainWindow::**getProcesses**(){

HANDLE h;

*for* (*auto* name : processes.keys()){

HANDLE h = processes.take(name);

CloseHandle(h);

}

DWORD cbNeeded, p[2048];

WCHAR name[256];

QString qname;

WINBOOL f = *false*;

*if* (!EnumProcesses(p, 2048 \* *sizeof*(DWORD), &cbNeeded)) *return*;

*for* (DWORD i = 0; i < cbNeeded; i++){

h = OpenProcess(PROCESS\_QUERY\_INFORMATION | PROCESS\_VM\_READ, f, p[i]);

*if*(GetModuleBaseNameW(h, NULL, name, 256)){

processes.insert(QString::fromWCharArray(name), h);

qDebug() << QString::fromWCharArray(name);

}

}

}

void MainWindow::**showMemoryMap**(QString processName){

HANDLE processHandle;

*auto* h = processes.find(processName);

*if* (h != processes.end()){

processHandle = \*h;

prevName = processName;

}*else*{

*return*;

}

HMODULE modules[1024];

DWORD sz;

EnumProcessModules(processHandle, modules, 1024, &sz);

QVector <LPVOID> addresses;

QVector <QString> names;

*for* (DWORD i = 0; i < sz; i++){

MODULEINFO moduleInfo;

GetModuleInformation(processHandle, modules[i], &moduleInfo, *sizeof*(moduleInfo));

addresses.append(moduleInfo.lpBaseOfDll);

WCHAR buffer[MAX\_PATH];

GetModuleFileNameExW(processHandle, modules[i], buffer, MAX\_PATH);

names.append(QString::fromWCharArray(buffer));

}

SYSTEM\_INFO systemInfo;

GetSystemInfo(&systemInfo);

long long address = 0;

int row = 0;

MEMORY\_BASIC\_INFORMATION memoryInfo;

*while* (VirtualQueryEx(processHandle, (LPVOID)address, &memoryInfo, *sizeof*(memoryInfo)) != 0) {

ui->virtualMemoryMap->insertRow(row);

*//* *Если* *участок* *памяти* *принадлежит* *модулю,* *надо* *добавить* *в* *таблицу* *строку* *с* *именем*

*if* (memoryInfo.BaseAddress != NULL && addresses.contains(memoryInfo.BaseAddress)){

int index = addresses.indexOf(memoryInfo.BaseAddress);

ui->virtualMemoryMap->setItem(row, 0, *new* QTableWidgetItem(names[index]));

ui->virtualMemoryMap->setSpan(row, 0, 1, 7);

row++;

ui->virtualMemoryMap->insertRow(row);

}

*//* *Начальный* *адрес*

QTableWidgetItem\* baseAddressItem = *new* QTableWidgetItem("0x" + QString("%1").arg((long long)memoryInfo.BaseAddress, 16, 16, QChar('0')).toUpper());

ui->virtualMemoryMap->setItem(row, 0, baseAddressItem);

*//* *Конечный* *адрес*

address += memoryInfo.RegionSize;

QTableWidgetItem\* finalAddressItem = *new* QTableWidgetItem("0x" + QString("%1").arg(address - 1, 16, 16, QChar('0')).toUpper());

ui->virtualMemoryMap->setItem(row, 1, finalAddressItem);

*//* *Количество* *страниц*

ui->virtualMemoryMap->setItem(row, 2, *new* QTableWidgetItem(QString::number(memoryInfo.RegionSize/systemInfo.dwPageSize)));

*//* *Размер* *области*

ui->virtualMemoryMap->setItem(row, 3, *new* QTableWidgetItem(QString::number(memoryInfo.RegionSize / 1024) + " Кб"));

*//* *Состояние* *памяти*

QString memoryState= "Не удалось прочитать значение memoryState";

*switch* (memoryInfo.State){

*case* MEM\_COMMIT: memoryState = "MEM\_COMMIT - зафиксировано"; *break*;

*case* MEM\_RESERVE: memoryState = "MEM\_RESERVE - зарезервировано)"; *break*;

*case* MEM\_FREE: memoryState = "MEM\_FREE - Свободно"; *break*;

}

ui->virtualMemoryMap->setItem(row, 4, *new* QTableWidgetItem(memoryState));

*//* *Если* *не* *свободно,* *то* *режим* *и* *тип* *области*

*if* (memoryInfo.State != MEM\_FREE) {

QString protectMode = "0 - нет доступа";

*switch* (memoryInfo.Protect) {

*case* PAGE\_EXECUTE\_READWRITE: protectMode = "PAGE\_EXECUTE\_READWRITE - Исполнение, чтение и запись"; *break*;

*case* PAGE\_READONLY: protectMode = "PAGE\_READONLY - Только чтение"; *break*;

*case* PAGE\_READWRITE: protectMode = "PAGE\_READWRITE - Чтение и запись"; *break*;

*case* PAGE\_EXECUTE\_WRITECOPY: protectMode = "PAGE\_EXECUTE\_WRITECOPY - Исполнение и запись с копированием"; *break*;

*case* PAGE\_EXECUTE\_READ: protectMode = "PAGE\_EXECUTE\_READ - Исполнение и чтение"; *break*;

*case* PAGE\_EXECUTE: protectMode = "PAGE\_EXECUTE - Исполнение"; *break*;

*case* PAGE\_NOACCESS: protectMode = "PAGE\_NOACCESS - Нет доступа"; *break*;

*case* PAGE\_WRITECOPY: protectMode = "PAGE\_WRITECOPY - Запись копированием"; *break*;

*case* PAGE\_GUARD: protectMode = "PAGE\_GUARD - Защищено"; *break*;

*case* PAGE\_NOCACHE: protectMode = "PAGE\_NOCACHE - Нельзя кэшировать"; *break*;

}

ui->virtualMemoryMap->setItem(row, 5, *new* QTableWidgetItem(protectMode));

QString type = "Неизвестно";

*switch* (memoryInfo.Type) {

*case* MEM\_IMAGE: type = "MEM\_IMAGE - загрузочный";

*case* MEM\_MAPPED: type = "MEM\_MAPPED - файл данных";

*case* MEM\_PRIVATE: type = "MEM\_PRIVATE - приватный";

}

ui->virtualMemoryMap->setItem(row, 6, *new* QTableWidgetItem(type));

}

row++;

}

}

void MainWindow::**on\_chooseProcess\_currentTextChanged**(*const* QString &arg1){

showMemoryMap(arg1);

}

void MainWindow::**showMemoryStatus**(){

MEMORYSTATUSEX memoryStatus;

memoryStatus.dwLength = *sizeof*(MEMORYSTATUSEX);

GlobalMemoryStatusEx(&memoryStatus);

QtCharts::QChartView \* chart = *new* QtCharts::QChartView(*this*);

memoryUsedInPercents = *new* CustomChart(100, 1, Qt::*green*, QString("Использование физ. памяти:"),

QString("Время (сек)"), QString("Занято (%)"));

chart->setChart(memoryUsedInPercents);

ui->charts->layout()->addWidget(chart);

memoryUsed = *new* CustomChart (memoryStatus.ullTotalPhys, pow(2, 30), Qt::*green*, QString("Доступно физической памяти:"),

QString("Время (сек)"), QString("Доступно (Гб)"));

chart = *new* QtCharts::QChartView(*this*);

chart->setChart(memoryUsed);

ui->charts->layout()->addWidget(chart);

virtualUsed = *new* CustomChart(memoryStatus.ullTotalVirtual, pow(2, 30), Qt::*green*, QString("Использование вирт. памяти:"),

QString("Время (сек)"), QString("Доступно (Гб)"));

chart = *new* QtCharts::QChartView(*this*);

chart->setChart(virtualUsed);

ui->vcharts->layout()->addWidget(chart);

virtualAvaliable = *new* CustomChart(memoryStatus.ullTotalVirtual, pow(2, 30), Qt::*green*, QString("Доступно виртуальной памяти"),

QString("Время (сек)"), QString("Доступно (Гб)"));

chart = *new* QtCharts::QChartView(*this*);

chart->setChart(virtualAvaliable);

ui->vcharts->layout()->addWidget(chart);

swapFileUsed = *new* CustomChart(memoryStatus.ullTotalPageFile, pow(2, 30), Qt::*green*, QString("Доступный объём файла подкачки"),

QString("Время (сек)"), QString("Доступно (Гб)"));

chart = *new* QtCharts::QChartView(*this*);

chart->setChart(swapFileUsed);

ui->vcharts->layout()->addWidget(chart);

}

void MainWindow::**updateCharts**(){

MEMORYSTATUSEX memoryStatus;

memoryStatus.dwLength = *sizeof*(MEMORYSTATUSEX);

GlobalMemoryStatusEx(&memoryStatus);

qreal elapsed = time->elapsed() / 1000;

memoryUsedInPercents->update(elapsed, memoryStatus.dwMemoryLoad);

memoryUsed->update(elapsed, memoryStatus.ullAvailPhys);

virtualUsed->update(elapsed, memoryStatus.ullAvailVirtual);

virtualAvaliable->update(elapsed, memoryStatus.ullTotalVirtual - memoryStatus.ullAvailVirtual);

swapFileUsed->update(elapsed, memoryStatus.ullAvailPageFile);

}