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 № 5
дисциплина «Операционные системы»
по теме «Операции с файлами»

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

Белгород 2019

Лабораторная работа № 5

«Операции с файлами»

Цель работы: получение практических навыков по использованию Win32 API для работы с файлами.

Вариант 9

Содержание отчета:

1. Наименование лабораторной работы, ее цель.
2. Краткое изложение теоретических основ работы с файлами с использованием Win32 API.
3. Реализовать программу экспорта выбранной ветки системного реестра в бинарный файл, а также программу для просмотра созданного файла.
4. Примеры разработанных приложений (программы и результаты).

Ход работы

Краткие теоритические сведения

Реестр представляет собой системную базу данных представляющую собой дерево, в которой приложения и системные компоненты хранят и получают данные конфигурации. Данные, хранящиеся в реестре, различаются в зависимости от версии Microsoft Windows. Приложения используют API реестра для извлечения, изменения или удаления данных реестра.

Реестр состоит из ключей, которые в свою очередь состоят из подключей, и значений ключа.

В Windows NT есть 5 предопределенных корневых веток реестра:

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_USER
- HKEY_LOCAL_MACHINE
- HKEY_USERS
- HKEY_CURRENT_CONFIG

Типы значений реестра:

- REG_BINARY
- REG_DWORD
- REG_DWORD_BIGENDIAN
- REG_DWORD_LITTLE_ENDIAN
- REG_EXPAND_SZ
- REG_LINK

- REG_MULTI_SZ
- REG_NONE
- REG_RESOURCE_LIST
- REG_SZ

Для чтения и записи в файл используются API функции CreateFile,ReadFile,WriteFile

Примеры работы программы

Рис. 1: Задание 1. Прочитанный файл с информацией о ветки реестра

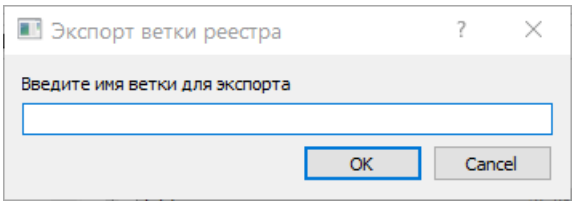

Рис. 2: Задание 2. Окно выбора ветки реестра.

Приложение

Содержимое файла exporter.cpp

```
#include "exporter.h"
#include <QDebug>

exporter::exporter(QWidget *parent) : QWidget(parent){
    this->root = HKEY_CURRENT_USER;
    bool is_valid_branch_name = false;
    while(!is_valid_branch_name){
        this->branch_str = QInputDialog::getText(nullptr,QString("Экспорт ветки
        ↪  реестра"),QString("Введите имя ветки для экспорта"));
        if(branch_str.length()==0){
            is_valid_branch_name = true;
            exit(0);
        }
        is_valid_branch_name = validate_branch_name();
    }


    char subkey_name_buffer[255];
    DWORD subkey_name_buffer_size;
    DWORD subkey_count = 0;

    DWORD longest_subkey_size;
    DWORD longest_class_string;

    char subkey_value_buffer[255];
    DWORD subkey_value_buffer_size;
    DWORD values_count;

    DWORD longest_value_name;
    DWORD longest_value_data;

    DWORD sec_desc;
    FILETIME flwt;

    HKEY test_key;
    DWORD str_size = 255;
    str_size = (DWORD)this->branch_str.length();
    char* test_char_name = (char*) malloc((int)str_size);
    QByteArray temp_arr = this->branch_str.toLocal8Bit();
    strcpy(test_char_name,temp_arr.constData());

    RegOpenKeyExA(this->root,test_char_name,str_size,KEY_READ,&test_key);
    RegQueryInfoKeyA(test_key,
                     test_char_name,
                     &str_size,
                     NULL,
                     &subkey_count,
                     &longest_subkey_size,
                     &longest_class_string,
                     &values_count,
                     &longest_value_name,
                     &longest_value_data,
                     &sec_desc,
                     &flwt);

    char char_subkey_count;
```

```
if(subkey_count>255){
    char_subkey_count = 255;
}else{
    char_subkey_count = subkey_count;
}

char char_values_count;
if(values_count>255){
    char_values_count = 255;
}else{
    char_values_count = values_count;
}

//(1 byte) Заголовок [2d]
//(1 byte) Сепаратор [28d]
//(N byte) Название ключа
//(1 byte) Конец строки [0d]
//(1 byte) Сепаратор [28d]
//(1 byte) Кол-во дочерних ключей [0-255d]
//(1 byte) Сепаратор [28d]
//(1 byte) Кол-во значений [0-255d]
//(1 byte) Начало начало перечисления дочерних ключей [29d]

    //0-255 дочерних ключа
    //(1 byte) Сепаратор [30d]
    //(N byte) Конец строки [0d]
//(1 byte) Начало начало перечисления значений ключа [29d]
    //0-255 дочерних ключа
    //(N byte) значение
    //(1 byte) Сепаратор [30d]
    //(N byte) значение
//(1 byte) Конец передачи [4d]

if(branch_str.length()>255){
    exit(1);
}
result.append((char)2);
result.append((char)28);
result.append(this->branch_str.toLatin1());
result.append((char)0);
result.append((char)28);
result.append(char_subkey_count);
result.append((char)28);
result.append(char_values_count);
result.append((char)29);
for(int index = 0;index<char_subkey_count;index++){
    char subkey_name[255];
    DWORD cbName = 255;

    ↪ if(RegEnumKeyExA(test_key,index,subkey_name,&cbName,NULL,NULL,NULL,NULL)==ERROR_SUCCESS){
        result.append(subkey_name);
        result.append((char)0);
    }
}
result.append((char)29);
for(int index = 0;index<char_values_count;index++){
    char value_name[255];
    QByteArray str = QByteArray(value_name);
    DWORD cbName = 255;
    DWORD valueType;
```

```
unsigned char *valueData = (unsigned char*)malloc(longest_value_data);
unsigned long valueDataSize;

↪ if(RegEnumValueA(test_key,index,value_name,&cbName,NULL,&valueType,valueData,&valueDataSize
    result.append(str);
    result.append((char)0);
    result.append((char)valueType);
    result.append((char)valueDataSize);
    switch (valueType) {
        case REG_BINARY:{
            QByteArray temp;
            for(int pointer = 0;pointer<valueDataSize;pointer++){
                temp.append(valueData[pointer]);
            };
            result.append(temp);
            break;
        }
        case REG_DWORD_LITTLE_ENDIAN:{
            QByteArray temp;
            for(int pointer = 0;pointer<valueDataSize;pointer++){
                temp.append(valueData[pointer]);
            };
            result.append(temp);
            std::reverse(temp.begin(),temp.end());
            break;}
        case REG_DWORD_BIG_ENDIAN:{
            QByteArray temp;
            for(int pointer = 0;pointer<valueDataSize;pointer++){
                temp.append(valueData[pointer]);
            };
            result.append(temp);
            break;}
        case REG_EXPAND_SZ:{
            QByteArray temp;
            for(int pointer = 0;pointer<valueDataSize;pointer++){
                temp.append(valueData[pointer]);
            };
            result.append(temp);
            break;}
        case REG_LINK:{
            QByteArray temp;
            for(int pointer = 0;pointer<valueDataSize;pointer++){
                temp.append(valueData[pointer]);
            };
            result.append(temp);
            break;}
        case REG_MULTI_SZ:{
            QByteArray temp;
            for(int pointer = 0;pointer<valueDataSize;pointer++){
                temp.append(valueData[pointer]);
            };
            result.append(temp);
            break;}
        case REG_NONE:{
            QByteArray temp;
            for(int pointer = 0;pointer<valueDataSize;pointer++){
                temp.append(valueData[pointer]);
            };
            result.append(temp);
            break;}
```

```
                case REG_QWORD:{
                    QByteArray temp;
                    for(int pointer = 0;pointer<valueDataSize;pointer++){
                        temp.append(valueData[pointer]);
                    };
                    result.append(temp);
                    break;}
                case REG_SZ:{
                    QByteArray temp;
                    for(int pointer = 0;pointer<valueDataSize;pointer++){
                        temp.append(valueData[pointer]);
                    };
                    result.append(temp);
                    break;}
                result.append((char)28);
            }
        }
    }
    result.append((char)4);
    qDebug() << result;
}

bool exporter::validate_branch_name(){
    HKEY test_key;
    DWORD str_size = 255;
    str_size = (DWORD)this->branch_str.length();
    char* test_char_name = (char*) malloc((int)str_size);
    QByteArray temp_arr = this->branch_str.toLocal8Bit();
    strcpy(test_char_name,temp_arr.constData());

    LSTATUS return_value;
    return_value = RegOpenKeyExA(this->root,test_char_name,str_size,KEY_READ,&test_key);

    RegCloseKey(test_key);
    if(return_value == ERROR_SUCCESS){
        return true;
    }else{
        return false;
    }
};

QByteArray exporter::get_result_array(){
    return this->result;
};
```

Содержимое файла exporter.h

```
#ifndef EXPORTER_H
#define EXPORTER_H

#include <QWidget>
#include <QString>
#include <windows.h>
#include <QInputDialog>
#include <QMessageBox>
#include <QByteArray>

class exporter : public QWidget
{
    Q_OBJECT
```

```
private:
    QString branch_str;
public:
    QByteArray result;
    explicit exporter(QWidget *parent = nullptr);
    bool validate_branch_name();
    HKEY root;
    QByteArray get_result_array();
};

#endif // EXPORTER_H
```

Содержимое файла fileoperation.cpp

```
#include "fileoperation.h"

#include <windows.h>
#include <QDebug>
#include <QDir>

FileOperation::FileOperation(){
    // Необходимо получить размер сектора для выравнивания
    GetDiskFreeSpace(NULL, NULL, &bytes_per_sector, NULL, NULL);
}

QByteArray FileOperation::read(QString path){
    TCHAR *buffer = new TCHAR[MAX_PATH];
    FillMemory(buffer, MAX_PATH, 0);
    // Преобразовали путь к WCHAR
    int len = path.toWCharArray(buffer);
    buffer[len] = '\0';

    // Открытие файла
    HANDLE file_h = CreateFileW(
                buffer,             // Имя
                GENERIC_READ,       // На чтение
                FILE_SHARE_READ,    // Может быть открыт в другой программе
                NULL,               // Флаги безопасности не установлены, наследование не
                ↪  нужно
                OPEN_EXISTING,      // Открывать только существующие
                FILE_FLAG_OVERLAPPED,  // Асинхронные операции
                NULL                    // Шаблона нет
                );

    if (file_h == INVALID_HANDLE_VALUE){
        qDebug() << GetLastError();
        throw std::runtime_error("Не удалось открыть файл");
    }

    // Подготовка буфера и overlapped
    // Помимо размера файла надо оставить место для 0-символа
    DWORD size_in_bytes = aligned(
                fileSize(file_h)  + sizeof (BYTE)
                );
    BYTE *big_buffer = new BYTE[size_in_bytes];

    FillMemory(big_buffer, size_in_bytes, 0);

    // Создание события для асинхронных операций
    HANDLE event = CreateEvent(
```

```
                NULL,
                FALSE,
                FALSE,
                NULL
                );
    // Структура OVERLAPPED для асинхронных операций
    OVERLAPPED ovl;
    ovl.hEvent = event;
    ovl.Offset = 0;
    ovl.OffsetHigh = 0;

    // Чтение файла
    BOOL bResult = ReadFile(
                file_h,     // Хендл
                big_buffer, // Буфер
                size_in_bytes, // Размер в байтах
                NULL,       // Количество считанных должно быть NULL при асинхр. оп.
                &ovl        // Структура OVERLAPPED
                );
    DWORD err;
    if (!bResult && (err = GetLastError()) != ERROR_IO_PENDING){
        qDebug() << err;
        CloseHandle(file_h);
        CloseHandle(event);
        delete[] big_buffer;
        throw std::runtime_error("Не удалось прочесть файл");
    }

    // Ожидаем завершения операции чтения
    WaitForSingleObject(event, INFINITE);

    // Получим количество считанных байт
    DWORD actually_read;
    GetOverlappedResult(
                file_h, // Файл
                &ovl,   // Структура OVERLAPPED
                &actually_read, // Сколько прочитали
                TRUE    // Ожидать завершения операции
                );

    // Установим терминатор после всех прочитанных
    big_buffer[actually_read / sizeof(BYTE)] = '\0';

    // Кодировка файла Utf-8
    QByteArray result;
    result.append((char *)big_buffer);
    // Закрытие и очистка
    CloseHandle(file_h);
    CloseHandle(event);
    delete[] big_buffer;

    return result;
}

bool FileOperation::write(QByteArray arr, QString path, bool append){
    // Преобразовываем кодировку текста к UTF-8
    BYTE *big_buffer = (BYTE *) arr.data();
    qDebug() <<"запись в файл" <<arr;
    ULONGLONG buffer_size = arr.size();
```

```
ULONGLONG free_space;
// Проверяем наличие свободного места
try {
    free_space = freeSpace(path);
} catch (std::runtime_error e) {
    throw e;
}
if (free_space < buffer_size){
    throw std::runtime_error("Недостаточно места на диске");
}
TCHAR *buffer = new TCHAR[MAX_PATH];
FillMemory(buffer, MAX_PATH, 0);
// Преобразовали путь к WCHAR
int len = path.toWCharArray(buffer);
buffer[len] = '\0';

HANDLE file_h;
ULONGLONG current_size = 0;

// Открытие файла
if (append){
    file_h = CreateFileW(
                buffer,             // Имя
                FILE_APPEND_DATA,        // На дозапись
                0,                  // не разделять
                NULL,               // Флаги безопасности не установлены, наследование не
                ↪  нужно
                OPEN_ALWAYS,      // открывать если существует, создавать если нет
                FILE_FLAG_OVERLAPPED,  // Асинхронные операции
                NULL                    // Шаблона нет
                );
     current_size = fileSize(file_h);
}else{
    file_h = CreateFileW(
                buffer,             // Имя
                GENERIC_WRITE,       // На запись
                0,                  // не разделять
                NULL,               // Флаги безопасности не установлены, наследование не
                ↪  нужно
                CREATE_ALWAYS,      // Только создавать новый
                FILE_FLAG_OVERLAPPED,  // Асинхронные операции
                NULL                    // Шаблона нет
                );
}

if (file_h == INVALID_HANDLE_VALUE){
    qDebug() << GetLastError();
    throw std::runtime_error("Не удалось создать файл");
}

DWORD size_in_bytes = aligned(
            buffer_size
            );

// Создание события для асинхронных операций
HANDLE event = CreateEvent(
            NULL,
            FALSE,
            FALSE,
            NULL
```

```
                );
    // Структура OVERLAPPED для асинхронных операций
    OVERLAPPED ovl;
    ovl.hEvent = event;
    ovl.Offset = 0;
    ovl.OffsetHigh = 0;

    // Запись файла
    BOOL bResult = WriteFile(
                file_h,     // Хендл
                big_buffer, // Буфер
                size_in_bytes, // Размер в байтах
                NULL,
                &ovl        // Структура OVERLAPPED
                );
    DWORD err;
    if (!bResult && (err = GetLastError()) != ERROR_IO_PENDING){
        qDebug() << err;
        CloseHandle(file_h);
        CloseHandle(event);
        delete[] big_buffer;
        throw std::runtime_error("Не удалось записать файл");
    }

    // Ожидаем завершения операции записи
    WaitForSingleObject(event, INFINITE);

    // Урезаем файл по реально хранимым данным
    LONG bytes_to_cut = buffer_size + current_size;
    SetFilePointer(file_h, bytes_to_cut, 0, FILE_BEGIN);
    SetEndOfFile(file_h);
    qDebug() << "Сделано";;
    // Закрытие и очистка
    CloseHandle(file_h);
    CloseHandle(event);
    delete[] big_buffer;

    return true;
}

bool FileOperation::exists(QString path){
    TCHAR *buffer = new TCHAR[MAX_PATH];
    FillMemory(buffer, MAX_PATH, 0);
    path.toWCharArray(buffer);
    bool exists = GetFileAttributes(buffer) != DWORD(-1);
    delete[] buffer;
    return exists;
}

QString FileOperation::currentDir(){
    TCHAR *buffer = new TCHAR[MAX_PATH];
    FillMemory(buffer, MAX_PATH, 0);

    GetCurrentDirectoryW(MAX_PATH, buffer);
    QString path = QString::fromWCharArray(buffer);

    delete[] buffer;
    return path;
}
```

```
DWORD FileOperation::aligned(DWORD size){
    // Получает ближайшее большее или равное, кратное размеру сектора
    return ((bytes_per_sector + ((size + bytes_per_sector)-1)) & ~(bytes_per_sector -1));
}

// Возвращает размер файла в байтах
DWORD FileOperation::fileSize(HANDLE hFile){
    LARGE_INTEGER file_size;
    GetFileSizeEx(hFile, &file_size);
    DWORD filesize = file_size.LowPart;
    return filesize;
}

ULONGLONG FileOperation::freeSpace(QString path){
    QString win_path = QDir::toNativeSeparators(path);
    int first_delim = win_path.indexOf(QDir::separator());
    int last_delim = win_path.lastIndexOf(QDir::separator());
    // Удаляю букву диска
    win_path.remove(0, first_delim);
    // Удаляю файл
    win_path.chop(win_path.length() - last_delim + 2);
    TCHAR *buffer = new TCHAR[MAX_PATH];
    FillMemory(buffer, MAX_PATH, 0);
    win_path.toWCharArray(buffer);
    ULARGE_INTEGER free_size;

    bool result = GetDiskFreeSpaceEx(
                buffer,         // Путь
                &free_size,     // Кол-во байт доступных пользователю
                NULL,
                NULL
                );
    if (!result){
        throw std::runtime_error("Не удалось получить свободное место");
    }
    ULONGLONG byte_size = free_size.QuadPart;
    delete[] buffer;
    return byte_size;
}
```

Содержимое файла fileoperation.h

```
#ifndef FILEOPERATION_H
#define FILEOPERATION_H

#include <QString>

#include <windows.h>

class FileOperation
{
public:
    FileOperation();
    QByteArray read(QString path);
    bool write(QByteArray, QString path, bool append = false);
    bool exists(QString path);
    QString currentDir();
private:
    DWORD aligned(DWORD);
    DWORD fileSize(HANDLE);
```

```
    ULONGLONG freeSpace(QString);
    DWORD bytes_per_sector;

};

#endif // FILEOPERATION_H
```

Содержимое файла view.cpp

```
#include "view.h"
#include <QDebug>
#include <QList>
#include "fileoperation.h"
#include <QFileDialog>
#include <QLayout>
#include <QLabel>
view::view(QWidget *parent) : QWidget(parent){
    this->root = HKEY_CURRENT_USER;
    FileOperation file;
    this->arr = file.read(QFileDialog::getOpenFileName(nullptr, "Открыть ветку
    ↪  реестра","",("Бинарный файл реестра (*.breg)")));
    qDebug()<< this->arr;
    int current_arr_pointer = 0;
      if(arr[current_arr_pointer]!=(char)2){
          qDebug() << "Ошибка\n";
      }else current_arr_pointer+=2;
      qDebug() << "Заголовок прочитан";

      QString branch_name;
      while(arr[current_arr_pointer]!=(char)0){
          branch_name.append(QChar(arr[current_arr_pointer]));
          current_arr_pointer++;
      }
      current_arr_pointer+=2;
      qDebug() << branch_name;

      short count_of_subkey;
      short count_of_values;
      count_of_subkey = (short)arr[current_arr_pointer];
      current_arr_pointer+=2;
      count_of_values = (short)arr[current_arr_pointer];
      current_arr_pointer+=2;
      qDebug() << "Начало перечисления подключей";
      QList<QString> subkeys;
      for(int i=0; i< count_of_subkey;i++){
          QString subkey_name;
          while(arr[current_arr_pointer]!=(char)0){
              subkey_name.append(QChar(arr[current_arr_pointer]));
              current_arr_pointer++;
          }
          qDebug() << subkey_name << "\n";
          subkeys.append(subkey_name);
      }
      current_arr_pointer+=2;
      QString value_str = QString("Значения:\n");
      qDebug() << "Начало перечисления значений";
      for(int i=0;i<count_of_values;i++){
          QString subkey_name;
          while(arr[current_arr_pointer]!=(char)0){
              subkey_name.append(QChar(arr[current_arr_pointer]));
```

```
            current_arr_pointer++;
        }
        current_arr_pointer++;
        int type = arr[current_arr_pointer];
        current_arr_pointer++;
        int size = arr[current_arr_pointer];
        QByteArray temp;
        for(int ptr=0;ptr<size;ptr++){
            temp.append(arr[current_arr_pointer]);
            current_arr_pointer++;
        }
        if(type==REG_DWORD_LITTLE_ENDIAN){
            std::reverse(temp.begin(),temp.end());
        }
        current_arr_pointer++;
        QString type_str;
        switch(type){
        case REG_BINARY:{
            type_str = "REG_BINARY";
            break;
        }
        case REG_DWORD_LITTLE_ENDIAN:{
            type_str = "REG_DWORD_LITTLE_ENDIAN";
            break;}
        case REG_DWORD_BIG_ENDIAN:{
            type_str = "REG_DWORD_BIG_ENDIAN";
            break;}
        case REG_EXPAND_SZ:{
            type_str = "REG_EXPAND_SZ";
            break;}
        case REG_LINK:{
            type_str = "REG_LINK";
            break;}
        case REG_MULTI_SZ:{
            type_str = "REG_MULTI_SZ";
            break;}
        case REG_NONE:{
            type_str = "REG_NONE";
            break;}
        case REG_QWORD:{
            type_str = "REG_QWORD";
            break;}
        case REG_SZ:{
            type_str = "REG_SZ";
            break;}
        }
        value_str.append(" --- "+subkey_name+QString(" ")+type_str+QString("
        ↪  ")+QString::number(size)+QString(" ")+QString(temp)+QString("\n\n"));
        qDebug() <<subkey_name << type << " " << size << " " << temp << "\n";
    };
QVBoxLayout* layout = new QVBoxLayout();
QLabel *lname = new QLabel(QString("Имя ветки реестра: ")+branch_name);
QLabel *lsubkeys = new QLabel(QString("Кол-во дочерних ключей:
↪  ")+QString::number(count_of_subkey));
QLabel *lvalues = new QLabel("Кол-во значений ключа: "+QString::number(count_of_values));
QLabel *keys = new QLabel;
QString key_str = QString("Дочерние ключи:\n");
for(int i = 0;i<subkeys.length();i++){
    key_str.append(subkeys[i]+QString("\n"));
};
```

```
    qDebug() << key_str;
    keys->setText(key_str);
    QLabel *values = new QLabel;
    values->setText(value_str);
    for(int i = 0;i<subkeys.length();i++){
        key_str.append(" --- "+subkeys[i]+QString("\n\n"));
    };
    layout->addWidget(lname);
    layout->addWidget(lsubkeys);
    layout->addWidget(lvalues);
    layout->addWidget(keys);
    layout->addWidget(values);
    this->setLayout(layout);
};
```

Содержимое файла view.h

```
#ifndef VIEW_H
#define VIEW_H

#include <QWidget>
#include <QString>
#include <windows.h>
#include <QInputDialog>
#include <QMessageBox>
#include <QByteArray>

class view : public QWidget
{
    Q_OBJECT
private:
    QByteArray arr;
public:
    explicit view(QWidget *parent = nullptr);
    void test(QByteArray arr);
    HKEY root;
};

#endif // VIEW_H
```