Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

Тульский государственный университет

КАФЕДРА АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ

ФАЙЛЫ, ОТОБРАЖЕННЫЕ В ПАМЯТЬ

Лабораторная работа № 3
по курсу «Операционные системы»

Вариант № 3

Выполнил: студент группы 220601 _____ Белым А.А.
(подпись)
Проверил: _____ Попов А.И.
(подпись)

Тула 2012

Цель работы

Целью работы является ознакомление с методами организации отображения файлов в память и организации с их помощью межпроцессного взаимодействия.

Задание

Написать программу, которая позволит часть символов текстового файла переводить в прописные буквы, а другую часть – в строчные, с условием, что результат требуется записать в отдельный файл.

Теоретическая справка

Функции отображения файлов в память имеют три области применения:

- Система Win32 загружает и выполняет EXE- и DLL-файлы с помощью механизма отображения файлов в память. Это позволяет уменьшить объем страничного обмена и, следовательно, сократить время загрузки данных файлов;
- Имеется возможность организовать доступ к содержимому отображенных в память файлов с помощью обычного указателя на адрес памяти в области отображения. Это не только упрощает доступ к данным файла, но и освобождает разработчика от необходимости применять различные схемы буферизации файлов;
- Отображенные в память файлы позволяют организовать разделение их данных между различными процессами, выполняющимися на одном и том же компьютере.

Отображаемые в памяти файлы (*memory-mapped files*) позволяют обращаться к данным дисковых файлов так же, как к динамически выделенной памяти. Это осуществляется путем отображения всего файла или его части в диапазон адресов процесса. После этого доступ к данным в файле может быть получен с помощью обычного указателя.

Первый этап состоит в получении дескриптора файла, подлежащего отображению в область памяти процесса. Он реализуется посредством вызова функции `FileOpen()`. Этой функции передается режим доступа к файлу, равный значению `fmOpenReadWrite`, что позволяет получить возможность чтения содержимого файла и записи в него.

Затем определяется размер файла и его последний символ устанавливается равным значению нуль-терминатора. На самом деле это будет маркер конца файла, байтовое значение которого совпадает с ограничивающим нулем. Все это делается ради ясности и простоты. Поскольку обращение с данными файла осуществляется как со строками, ограниченными завершающими нулями, наличие такого завершающего нуль-терминатора просто необходимо.

Следующий этап — получение объекта файла отображения в память путем вызова функции `CreateFileMapping()`. При неуспешном выполнении этой функции генерируется исключительная ситуация. В противном случае программа переходит к очередному этапу — представлению объекта отображения файла в окне просмотра. И теперь в случае сбоя в работе этой функции генерируется исключительная ситуация.

Далее выполняется смена регистра представления данных (перевод символов в прописные буквы). Если после выполнения этой процедуры вы просмотрели бы файл в текстовом редакторе, то увидели бы, что все символы этого файла преобразованы в прописные буквы. И, наконец, с помощью вызова функции `UnmapViewOfFile()` выполняется удаление объекта окна просмотра отображенного в память файла.

Можно организовать передачу данных между процессами, работающими в разных адресных пространствах, с использованием файлов, отображенных на память. Методика использования файлов, отображенных на память, для передачи данных между процессами заключается в следующем.

Один из процессов создает такой файл, задавая при этом имя отображения. Это имя является глобальным и доступно для всех процессов, запущенных в системе. Другие процессы могут воспользоваться именем отображения, открыв созданный ранее файл. В результате оба процесса могут получить указатели на область памяти, для которой выполнено отображение, и эти указатели будут ссылаться на одни и те же страницы виртуальной памяти. Обмениваясь данными через эту область, процессы должны обеспечить синхронизацию своей работы, например, с помощью критических секций, событий, объектов Mutex или семафоров (в зависимости от логики процесса обмена данными).

Этот способ обладает высоким быстродействием, так как данные передаются между процессами непосредственно через виртуальную память.

Динамически компонуемые библиотеки представляют собой программные модули, содержащие код, данные или ресурсы, которые могут совместно использоваться несколькими приложениями Windows. Одно из основных назначений библиотек DLL – позволить приложениям загружать код, который отрабатывается во время выполнения, вместо того чтобы компоновать его в само приложение в процессе компиляции. А это значит, что несколько приложений могут одновременно использовать один и тот же код, содержащийся в библиотеке DLL.

Библиотеки DLL могут использовать свой код совместно с другими приложениями благодаря процессу, называемому *динамической компоновкой*. Как правило, когда какое-либо приложение использует библиотеку DLL, система Win32 гарантирует, что в памяти будет размещена только одна копия этой библиотеки. Это достигается с помощью механизма *отображения файла в память* (memory-mapped files).

В среде 16-разрядной Windows управление памятью библиотек DLL происходило совсем не так, как в среде 32-разрядной Win32. Одна из самых характерных особенностей работы 16-разрядных библиотек DLL заключалась в совместном использовании глобальной памяти различными приложениями. Иными словами, если в функции 16-разрядной библиотеки DLL объявляется глобальная переменная, то к ней получит доступ любое приложение, использующее эту DLL. Изменения, внесенные в эту переменную одним приложением, будут видны всем остальным приложениям.

В определенных случаях такое поведение могло быть опасным, поскольку одно приложение способно перезаписать данные, от которых зависит работа другого приложения. В других случаях разработчики специально использовали эту особенность системы.

В среде Win32 подобного разделения глобальных данных библиотек DLL больше не существует. Поскольку каждый процесс приложения отображает

библиотеку DLL на свое собственное адресное пространство, данные библиотеки также отображаются на это адресное пространство. В результате каждое приложение получает собственный экземпляр данных библиотеки DLL. Поэтому изменения, внесенные в глобальные данные библиотеки DLL одним приложением, не будут видны другому.

Инструкция пользователю

Программа позволяет изменять регистр символов входного файла, и записывать результат в другой файл.

Для работы необходимо открыть файл. Сделать это можно с помощью меню. После этого можно выделять символы мышкой в текстовом поле, и изменять их регистр с помощью кнопок на панели инструментов. Для сохранения результата необходимо выбрать файл с помощью меню.

Инструкция программисту

Типы данных:

```
typedef int (*casefunc)(const int);
```

Описывает указатель на функцию, преобразующую каждый символ исходного текста. Описывает стандартные функции `tolower`, `toupper`, а также пользовательскую функцию `swapcase`.

```
typedef struct{
    HANDLE file,mapping;
    char* view;
} MAPPING;
```

Структура, представляющее отображение файла. Содержит дескрипторы файла и отображения, и указатель представления отображения.

Функции:

```
int swapcase(int c)
```

Функция переводит исходный символ к верхнему регистру, если он находится в нижнем, и наоборот, с помощью стандартных функций `tolower`, `toupper`, `isupper`.

```
MAPPING OpenFileMap(QString name);
```

Создает Copy-on-Write отображение файла с именем `name`.

```
void CloseFileMap(MAPPING m);
```

Сбрасывает буфер представления отображения, отключает представление, закрывает дескрипторы файла и отображения.

```
void SaveFileMap(QString name,MAPPING m);
```

Сохраняет отображение `m` в файл с именем `name`. Для этого создается файл, изменяется его размер до размера файла исходного отображения, после чего область памяти исходного отображения копируется в область памяти нового отображения.

```
void MainWindow::ChangeCase(MAPPING m, casefunc f);
```

Выбранные пользователем символы в отображении `m` преобразуются с помощью функции `f`.

Текст программы

Ниже представлен текст программы, преобразующей регистр символов исходного текста и написанной на языке C++, в среде Qt Creator 2.5.2 + MinGW-GCC 4.6 с использованием библиотеки Qt.

mainwindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#define WINVER 0x0620
extern "C"{
    #include <windows.h>
}

typedef struct{
    HANDLE file,mapping;
    char* view;
} MAPPING;

namespace Ui {
class MainWindow;
}

typedef int (*casefunc)(const int);

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
```

```
    MAPPING m;
private slots:
    void ChangeCase(MAPPING m ,casefunc f);

    void on_action_triggered();

    void on_action_2_triggered();

    void on_action_3_triggered();

    void on_action_4_triggered();

    void on_action_5_triggered();

    void on_action_7_triggered();

    void on_action_8_triggered();

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

mainwindow.cpp:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QFileDialog>
#include <QDebug>
#include <cctype>



MAPPING OpenFileMap(QString name);
void CloseFileMap(MAPPING m);
void SaveFileMap(QString name,MAPPING m);

int swapcase(int c){
    if (isupper(c))
        return tolower(c);
    else
        return toupper(c);
}

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_action_triggered()
{
    QString name=QFileDialog::getOpenFileName(this,"Открытие файла","","Текстовые
файлы (*.txt)");
    if(name!=""){
        m=OpenFileMap(name);
        ui->plainTextEdit->setPlainText(QString(m.view));
    };
```

```
}

void MainWindow::on_action_2_triggered()
{
    QString name=QFileDialog::getSaveFileName(this,"Сохранение файла","","Тексто-
вые файлы (*.txt)");
    if(name!="")
        SaveFileMap(name,m);
}

void CloseFileMap(MAPPING m){
    FlushViewOfFile(m.view,0);

    UnmapViewOfFile(m.view);
    CloseHandle(m.mapping);
    CloseHandle(m.file);
}

MAPPING OpenFileMap(QString name){
    MAPPING res;
    LARGE_INTEGER size;
    int namesize=name.length();
    wchar_t *buf=new wchar_t[namesize+1];
    buf[namesize]='\0';
    name.toWCharArray(buf);
    if((res.file=CreateFile(buf,GENERIC_READ,0,NULL,OPEN_ALWAYS,FILE_ATTRIB-
UTE_NORMAL,NULL))!=INVALID_HANDLE_VALUE){
        GetFileSizeEx(res.file,&size);
        qDebug()<<size.HighPart;
        qDebug()<<size.LowPart;
        //size.QuadPart++;
        if((res.mapping=Create-
FileMapping(res.file,NULL,PAGE_WRITECOPY,0,0,NULL))!=NULL){
            res.view=(char*)MapViewOfFile(res.mapping,FILE_MAP_COPY,0,0,0);
            qDebug()<<int(res.view[size.LowPart]);
            //res.view[0]='\0';
        } else {
            qDebug()<<GetLastError();
        }
    }
    return res;
}

void SaveFileMap(QString name,MAPPING m){
    MAPPING res;
    LARGE_INTEGER size;
    int namesize=name.length();
    wchar_t *buf=new wchar_t[namesize+1];
    buf[namesize]='\0';
    name.toWCharArray(buf);
    if((res.file=CreateFile(buf,GENERIC_READ|GENERIC_WRITE,0,NULL,OPEN_AL-
WAYS,FILE_ATTRIBUTE_NORMAL,NULL))!=INVALID_HANDLE_VALUE){
        GetFileSizeEx(m.file,&size);
        qDebug()<<size.QuadPart;
        SetFilePointerEx(res.file,size,NULL,FILE_BEGIN);
        SetEndOfFile(res.file);
        if((res.mapping=CreateFileMapping(res.file,NULL,PAGE_READ-
WRITE,0,0,NULL))!=NULL){
            res.view=(char*)MapViewOfFile(res.mapping,FILE_MAP_ALL_ACCESS,0,0,0);

            strcpy(res.view,m.view);

            CloseFileMap(res);
        } else {
            qDebug()<<GetLastError();
        }
```

```
    }
}

void MainWindow::ChangeCase(MAPPING m ,casefunc f){
    QTextCursor curs=ui->plainTextEdit->textCursor();
    if (curs.hasSelection()){
        int start=curs.anchor()>curs.position()?curs.position():curs.anchor();
        int stop=curs.anchor()<curs.position()?curs.position():curs.anchor();
        for (int i=start;i<stop;++i){
            m.view[i]=f(m.view[i]);
        }
        ui->plainTextEdit->setPlainText(QString(m.view));

    }
}

void MainWindow::on_action_3_triggered()
{
    ChangeCase(m ,&toupper);
}

void MainWindow::on_action_4_triggered()
{
    ChangeCase(m ,&tolower);
}

void MainWindow::on_action_5_triggered()
{
    ChangeCase(m ,&swapcase);
}

void MainWindow::on_action_7_triggered()
{
    CloseFileMap(m);
    ui->plainTextEdit->clear();
}

void MainWindow::on_action_8_triggered()
{
    this->close();
}
```

Тестовый пример

На рисунке 1 представлен пример входных данных программы, преобразующей регистр символов исходного текста.

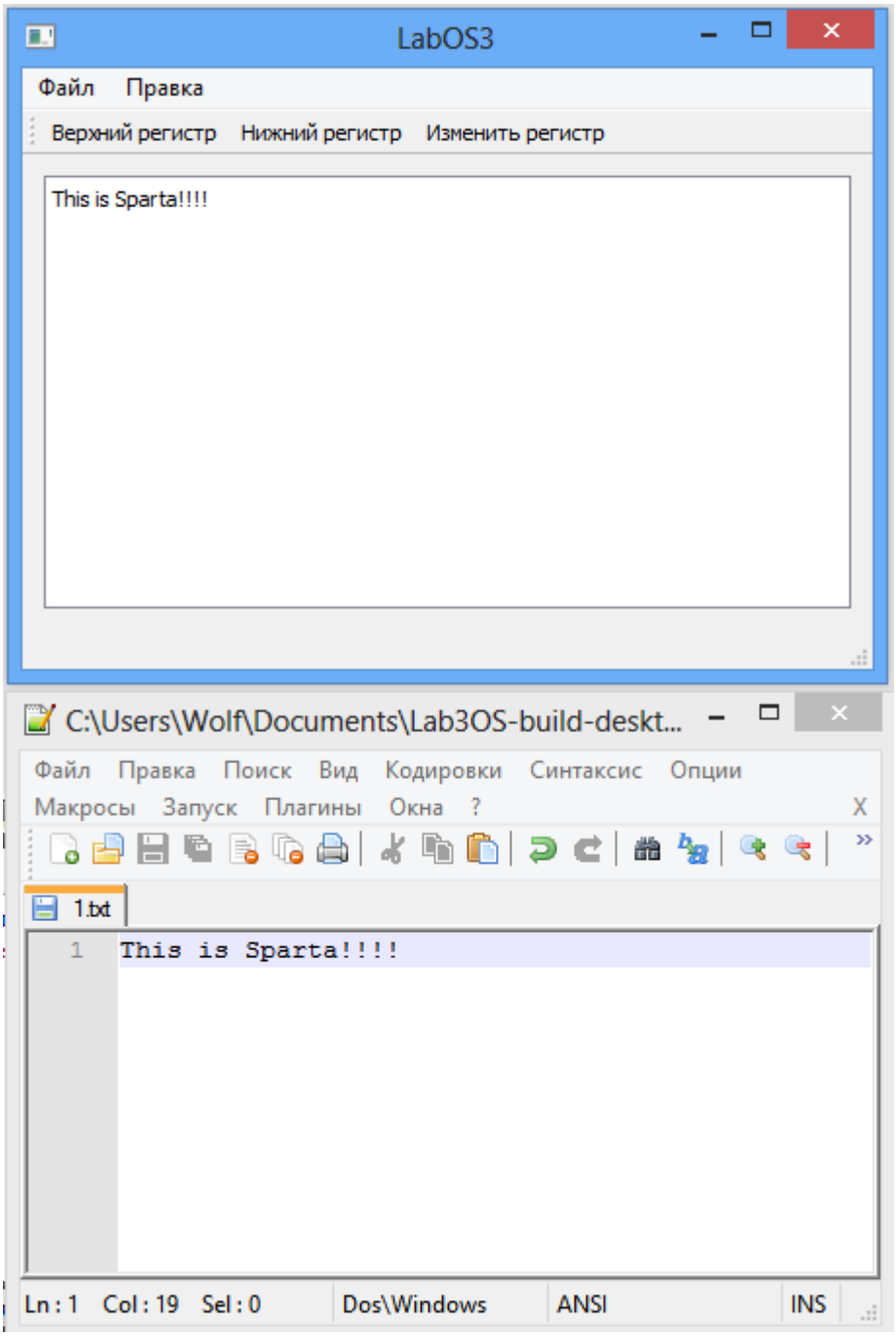

Рисунок 1— Входные данные программы

На рисунке 2 представлен результата работы программы, преобразующей регистр символов исходного текста.

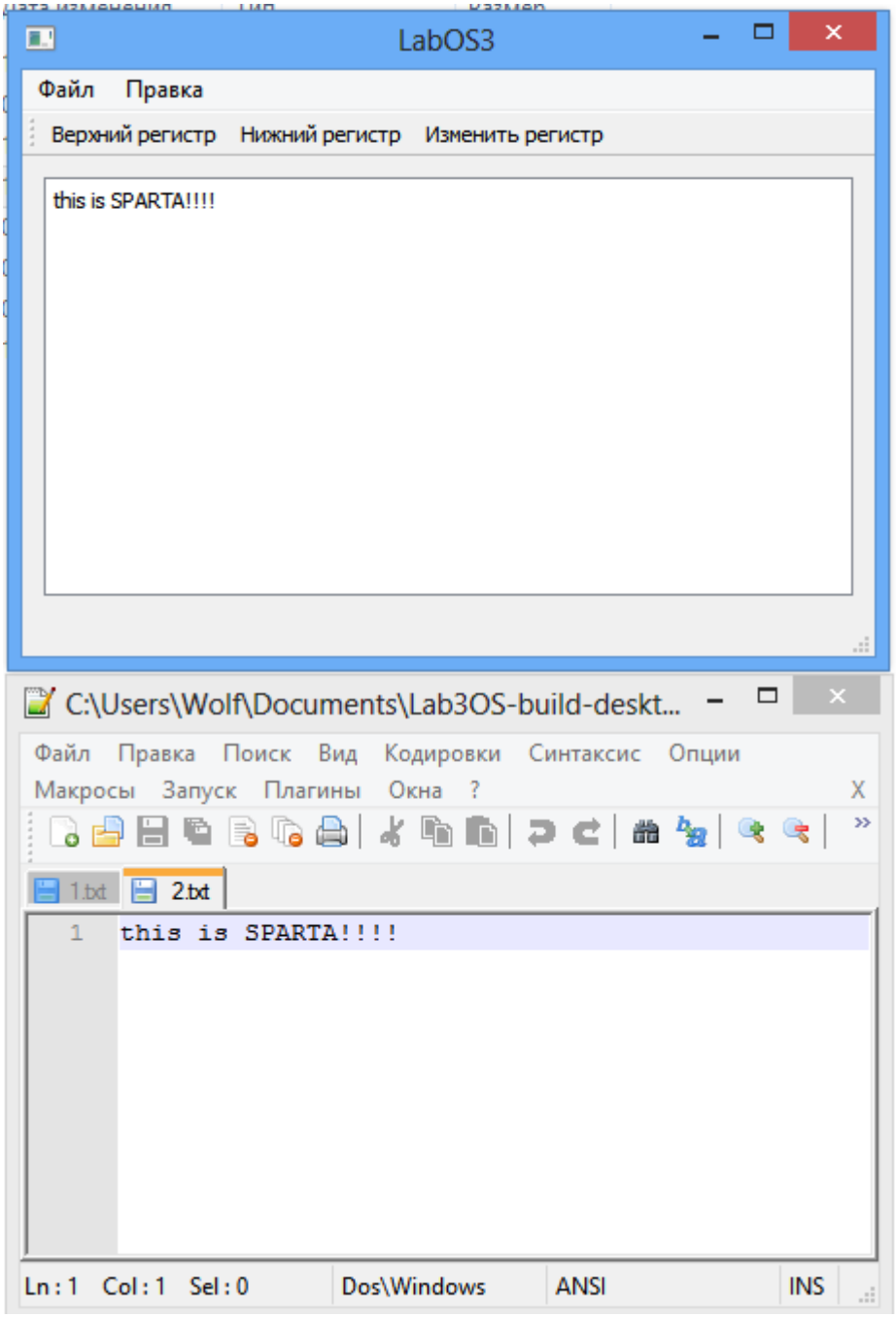

Рисунок 2— Пример работы программы

Вывод

Отображаемые в памяти файлы позволяют обращаться к данным дисковых файлов так же, как к динамически выделенной памяти. Это осуществляется путем отображения всего файла или его части в диапазон адресов процесса. После этого доступ к данным в файле может быть получен с помощью обычного указателя, что очень удобно при необходимости произвольного доступа к большим объемам данных. Также отображаемые на память файлы кэшируются операционной системой, что позволяет не реализовывать пр необходимости эту функцию самостоятельно.