Лабораторная работа №1

студента группы ИТ – 32

Курбатовой Софьи Андреевны

Выполнение:_____ Защита _____

ВВЕДЕНИЕ В WINDOWS API

Цель работы: Знакомство с понятием интерфейс прикладного программирования и получение практических навыков программирования для Windows на языке C/C++ с применением Win32 API.

Содержание работы

Вариант 4

1. Разработать в Visual C++ статическую библиотеку, в которой будут реализованы функции вывода в стандартный поток следующей информации:

– имя локального компьютера;

– доменное имя, назначенное локальному компьютеру;

– имя пользователя в текущем сеансе;

– полное имя пользователя в текущем сеансе.

2. Разработать в Visual C++ библиотеку DLL, в которой будут реализованы функции вывода в стандартный поток следующей информации:

– пути к системным каталогам (см. рис. Рис. 1.1);

– версия операционной системы;

– текущая дата и время (см. рис. Рис. 1.2);

Таблица 1.9. Значения CSIDL

Значение	Описание
CSIDL_APPDATA	Каталог, в котором размещены данные приложений пользователя **В Windows XP и выше**: %APPDATA%
CSIDL_COMMON_APPDATA	Каталог, в котором размещены данные приложений всех пользователей **В Windows XP**: %ALLUSERSPROFILE%\Start Menu\Programs\Administrative Tools **В Windows Vista и выше**: %ALLUSERSPROFILE%
CSIDL_COMMON_DOCUMENTS	Каталог «Общие документы» **В Windows XP**: %ALLUSERSPROFILE%\Documents **В Windows Vista и выше**: %PUBLIC%\Documents

Рис. 1.1. Часть идентификаторов системных каталогов

№	Формат даты	Формат времени	Системные метрики	Системные параметры
10,25	dd-MM-yyyy	hh:mm:ss tt	SM_CXEDGE SM_CYEDGE SM_CXMINSPACING SM_CYMINSPACING SM_SHOWSOUNDS	SPI_GETWORKAREA SPI_GETMOUSEWHEELROUTING

Рис. 1.2. Задание для варианта 10.

3. Изучить системные метрики и параметры, указанные в варианте задания. Включить в отчет назначение и описание изученных системных метрик и параметров.

4. Разработать в Visual C++ библиотеку DLL, в которой будут реализованы функции вывода в стандартный поток значений системных метрик, изученных в п.3.

5. Разработать в Visual C++ библиотеку DLL, в которой будут реализованы функции вывода в стандартный поток значений системных параметров, изученных в п.3.

6. Разработать в Visual C++ консольное приложение Win32, в котором будут использоваться функции, реализованные в следующих библиотеках:

– статистическая библиотека из п.1.

– DLL из п.2. (использовать неявное подключение);

– DLL из п.4. (использовать явное подключение);

– DLL из п.5. (использовать отложенную загрузку);

7. Протестировать работу приложения, созданного в п.6. на компьютере под управлением Windows. Результаты тестирования отразить в отчете.

8. Включить в отчет исходный программный код и выводы о проделанной работе.

Ход работы

1. Все функции можно описать непосредственно в исходном файле с расширением .cpp или .c при создании консольного приложения. Такой подход не позволяет достаточно гибко обращаться к описанным функциям. Поэтому, чтобы повторно не описывать функции, рекомендуется создавать библиотеки с этими функции и при необходимости подключать к проекту.

Статическая библиотека, согласно указанному в п.1 заданию, должна реализовывать функции вывода в стандартный поток такой информации, как имя локального компьютера, доменное имя, имя пользователя в текущем сеансе, полное имя пользователя в текущем сеансе. При компоновке эти функции будут скопированы в исполняемый фай приложения при его создании.

В представленных ниже листингах за вывод указанной в п.1. информации отвечают следующие функции:

– имя локального компьютера GetComputerName ;

– доменное имя, назначенное локальному компьютеру PrintDNSName;

– имя пользователя в текущем сеансе PrintUserName;

– полное имя пользователя в текущем сеансе PrintUserNameExtended.

Заголовочный файл STATILIBR.H:

```
#pragma once

#if !defined (_STATI_LIBR_H__)
#define _STATI_LIBR_H_
BOOL PrintCompName();
BOOL PrintDNSName();
BOOL PrintUserName();
BOOL PrintUserNameExtended();
	#endif
```

Файл STATILIBR.cpp:

```
#include <Windows.h>
#include <stdio.h>
#include <tchar.h>
#include <lmcons.h> //для функций GetUserName и GetUserNameEx
//#include <secext.h> //для функции GetUserNameEx

//#include "StatLib.h"

#pragma comment(lib,"Secur32.lib")
#define SECURITY_WIN32
#include <security.h>//для функции GetUserNameEx

BOOL PrintCompName()
{
	TCHAR szbuffer[MAX_COMPUTERNAME_LENGTH + 1];
	DWORD cchar = _countof(szbuffer); //считаем размер выделенного под строку буфера символах
	BOOL ret = GetComputerName(szbuffer, &cchar); //если имя компьютера определено вернет
TRUE?
	if (FALSE != ret)
	{
		wprintf(TEXT("Имя компьютера в локальной сети:%s\n"), szbuffer);
	}
	return ret;
}

BOOL PrintDNSName()
{
	//Для определения размера буфера установим NULL и 0
	LPWSTR szbuffer = NULL;
```

```
	DWORD cchar = 0;
	GetComputerNameEx(ComputerNameDnsHostname, szbuffer, &cchar); //определение размера буфера
в cchar
	szbuffer = new TCHAR[cchar];//выделение вычисленной памяти из ОС
	BOOL ret = GetComputerNameEx(ComputerNameDnsHostname, szbuffer, &cchar); //если имя
компьютера определено вернет TRUE?
	if (FALSE != ret)
	{
		wprintf(TEXT("Доменное имя локального компьютера:%s\n"), szbuffer);
	}
	delete[] szbuffer;
	return ret;
}

BOOL PrintUserName()
{
	TCHAR szbuffer[UNLEN + 1];
	DWORD cchar = _countof(szbuffer); //считаем размер выделенного под строку буфера символах
	BOOL ret = GetUserName(szbuffer, &cchar); //извлечение имени пользователя в текущем сеансе
	if (FALSE != ret)
	{
		wprintf(TEXT("Имя пользователя в текущем сеансе:%s\n"), szbuffer);
	}
	return ret;
}

BOOL PrintUserNameExtended()
{
	//Для определения размера буфера установим NULL и 0
	TCHAR szbuffer[UNLEN + 1];
	ULONG cchar = _countof(szbuffer);
	BOOLEAN ret = GetUserNameEx(NameSamCompatible, szbuffer, &cchar); //если имя пользователя
определено вернет TRUE
	if (FALSE != ret)
	{
		wprintf(TEXT("Доменное имя локального компьютера:%s\n"), szbuffer);
	}
	return ret;}
```

2.Главное отличие динамических библиотек от статических заключается в том, что описанные в них функции загружаются во время запуска исполняемого файла при создании приложения. Поэтому при внесении в нее изменений нет необходимости пересоздавать все зависимые от нее файлы.

Первая создаваемая в данной работе динамически подключаемая библиотека DYNLIB1.dll должна осуществить вывод в стандартный поток информацию о пути к системным папкам, текущую дату и время, версию установленной на ПК операционной системы. Для этого описываются следующие функции:

– пути к системным каталогам DYNLIB1_API void PrintSysDir(const long csidl[], unsigned long nCount);

– версия операционной системы DYNLIB1_API void PrintOSinfo();

– текущая дата DYNLIB1_API void TimeDateInfo(LPCWSTR lplocalname, DWORD dwFlags,const SYSTEMTIME * lpDate, LPCWSTR lpDateFormat, LPCWSTR lpTimeFormat);

– текущее время DYNLIB1_API void TimeDateInfo(LPCWSTR lplocalname, DWORD dwFlags, LPCWSTR lpDateFormat, LPCWSTR lpTimeFormat);

DYNLIB1_API является константой, через которую определяются ключевые слова __declspec(dllimport) и __declspec(dllexport), с помощью которых осуществляется импорт и экспорт данных из динамической библиотеки.

Использование extern “C” {} обусловлено тем, что компилятор может осуществлять декорирование имен функций и из-за этого при компоновке имя функции может быть прочитано неправильно и сама функция не найдена. Макрос __cplusplus нужен для создания заголовков функций, совместимых с C++. DllMain является дополнительной точкой входа. Если функция используется, то она вызывается системой тогда, когда процессы и потоки инициализированы и завершили работу или при вызове функции LoadLibrary и FreeLibrary.

Заголовочный файл DYNLIB1.h:

```
#pragma once
#if !defined(_DYN_LIB_1_H_)
	#define _DYN_LIB_1_H_
	#ifdef DYNLIB1_EXPORTS
		#define DYNLIB1_API __declspec(dllexport)
	#else
		#define DYNLIB1_API __declspec(dllimport)
	#endif //
#endif // !!defined(_DYN_LIB_1_H)
#ifdef __cplusplus
	DYNLIB1_API void PrintSysDir(const long csidl[], unsigned long nCount);
	DYNLIB1_API void PrintOSinfo();
	DYNLIB1_API void TimeDateInfo(LPCWSTR lplocalname,DWORD dwFlags,const SYSTEMTIME *
lpDate,LPCWSTR lpDateFormat,LPCWSTR lpTimeFormat);
	DYNLIB1_API void TimeDateInfo(LPCWSTR lplocalname, DWORD dwFlags, LPCWSTR lpDateFormat,
LPCWSTR lpTimeFormat);

#endif // _cplusplus
```

Файл DYNLIB1.cpp:

```
#include <Windows.h>
#include <stdio.h>
#include <wchar.h>
#include <ShlObj.h>
#include "DYNLIB1.h"

#pragma warning(disable : 4996) //отключает Ошибку deprecate. Возникает, когда используется
устаревшая функция

BOOL WINAPI DllMain(HINSTANCE hinstDll,DWORD idReason, LPVOID lpReserved )
{
	return TRUE;
}
DYNLIB1_API void PrintSysDir(const long csidl[], unsigned long nCount)
{
	TCHAR szPath[MAX_PATH + 1];
	for (unsigned long i = 0; i < nCount; ++i)
	{
		HRESULT hr = SHGetFolderPath(NULL, csidl[i], NULL, SHGFP_TYPE_CURRENT, szPath);
		if (S_OK == hr)
			wprintf(TEXT("%d:%s\n"), i + 1, szPath);
	}
			return;
}
DYNLIB1_API void PrintOSinfo()
{
	OSVERSIONINFO osver;
	osver.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);
	BOOL ret = GetVersionEx(&osver);
	if (ret != FALSE)
	{
		wprintf(L"Version:%d.%d.%d\n", osver.dwMajorVersion, osver.dwMinorVersion,
osver.dwBuildNumber);
		switch (osver.dwMajorVersion)
		{
		case 5:
```

```
		{
			switch (osver.dwMinorVersion)
			{
			case 1:
				wprintf(TEXT("Windows XP"));
				break;
			case 2:
				wprintf(TEXT("Windows 2003"));
				break;
			}
		}
		break;
		case 6:
		{
			switch (osver.dwMinorVersion)
			{
			case 0:
				wprintf(TEXT("Windows Vista"));
				break;
			case 1:
				wprintf(TEXT("Windows 7"));
				break;
			case 2:
				wprintf(TEXT("Windows 8"));
				break;
			case 3:
				wprintf(TEXT("Windows 8.1"));
				break;
			}
		}
			break;
		case 10:
			switch (osver.dwMinorVersion)
			{
			case 0:
				{
				wprintf(TEXT("WINDOWS 10"));
				}
				break;
			}
		}}}
```

Вторая динамическая библиотека DYNLIB2.dll должна осуществить вывод в стандартный поток следующих метрик:

– SM_CXEDGE, SM_CYEDGE: определение ширины трехмерной границы;

– SM_CXMINSPACING, SM_CYMINSPACING: определение размеры ячейки сетки для свернутых окон;

– SM_SHOWSOUNDS: Ненулевое значение в случае вызова приложения с информацией представленной только в звуковом формате.

Заголовочный файл DYNLIB2.h:

```
#pragma once
#if !defined(__DYN_LIB_2_H_)
#define __DYN_LIB_2_H_

#ifdef DYNLIB2_EXPORTS
#define DYNLIB2_API __declspec(dllexport)
#else
#define DYNLIB2_API __declspec(dllimport)
#endif //

typedef void (*PRINT_SYSMETR_PROC)(LPCWSTR LpDisplayNam, int index);
	#ifdef __cplusplus
	extern "C"{
	#endif
```

```
            DYNLIB2_API void PrintSYSmetr(LPCWSTR LpDisplayNam, int index);
      #ifdef __cplusplus
      }
      #endif // __cplusplus
       #endif
```

Файл DYNLIB2.cpp:

```
#include <Windows.h>
#include <stdio.h>
#include <wchar.h>
#include <locale.h>
#include "DYNLIB2.h"
#define DYNLIB2_PROTOTYPES

//точка входа в dll при загрузке в адресное пространство запущенного приложения
BOOL WINAPI DllMain(HINSTANCE hInstDll, DWORD dwReason, LPVOID LpvResearved)
{
      return TRUE;
}

DYNLIB2_API void PrintSYSmetr(LPCWSTR LpDisplayNam, int index)
{
      int value = GetSystemMetrics(index);
      switch (index)
      {
      case SM_CXEDGE:
            wprintf(L"%s:%d px\n", LpDisplayNam, value);
            break;
      case SM_CYEDGE:
            wprintf(L"%s:%d px\n", LpDisplayNam, value);
            break;
      case SM_CXMINSPACING:
            wprintf(L"%s:%d px\n", LpDisplayNam, value);
      case SM_CYMINSPACING:
            wprintf(L"%s:%d px\n", LpDisplayNam, value);
            break;
      case SM_SHOWSOUNDS:
            wprintf(L"%s:%d px\n",LpDisplayNam,value);
                  break;
      }
}
```

Третья динамическая библиотека DYNLIB3.dll должна осуществить вывод в стандартный поток следующих параметров:

– SPI_GETMOUSEWHEELROUTING: получение настройки пути для ввода колесика кнопок. Параметр определяет отправляется ввод с колесика мыши в приложение, ориентированное на передний план или в приложение находящееся под курсором мыши.

– SPI_GETWORKAREA: получение размера рабочей области на главном мониторе. Эту область не закрывает панель задач или панель инструментов. Параметр pvParam должен указывать на структуру RECT, которая получает координаты рабочей области в пикселях.

Заголовочный файл DYNLIB3.h:

```
#pragma once
#if !defined(_DYN_LIB_3_H_)
#define _DYN_LIB_3_H_

#ifdef DYNLIB3_EXPORTS
#define DYNLIB3_API __declspec(dllexport)
#else
#define DYNLIB3_API __declspec(dllimport)
#endif //

#ifndef SPI_GETMOUSEWHEELROUTNG
```

```
#define SPI_GETMOUSEWHEELROUTING 0x201C
#endif // !SPI_GETMOUSEWHEELROUTNG
#ifdef __cplusplus
extern "C"
{
#endif
	DYNLIB3_API BOOL PrintSYSparamInfo(LPCWSTR lpDispalyNam, UINT uiAction);
#ifdef  __cplusplus
}
#endif // __cplusplus
#endif // _DYN_LIB_3_H_
```

Файл DYNLIB3.cpp:

```
#include <Windows.h>
#include <wchar.h>
#include <stdio.h>
#include "DYNLIB3.h"

#ifndef MOUSEWHEEL_ROUTING_FOCUS
#define MOUSEWHEEL_ROUTING_FOCUS         0
#endif // !MOUSEWHEEL_ROUTING_FOCUS

#ifndef MOUSEWHEEL_ROUTING_HYBRID
#define MOUSEWHEEL_ROUTING_HYBRID        1
#endif // !MOUSEWHEEL_ROUTING_HYBRID

#ifndef MOUSEWHEEL_ROUTING_MOUSE_POS
#define MOUSEWHEEL_ROUTING_MOUSE_POS     2
#endif // MOUSEWHEEL_ROUTING_MOUSE_POS

BOOL WINAPI DllMain(HINSTANCE hInstDll, DWORD dwReason, LPVOID LpvResearved)
{
	return TRUE;
}
DYNLIB3_API BOOL PrintSYSparamInfo(LPCWSTR lpDispalyNam, UINT uiAction)
{
	BOOL bReturn = FALSE;
	switch (uiAction)
	{
	case SPI_GETMOUSEWHEELROUTING:
		{
			DWORD value;
			bReturn = SystemParametersInfo(uiAction, 0, &value, 0);
			if (FALSE != bReturn)
			{
				switch (value)
				{
				case MOUSEWHEEL_ROUTING_FOCUS:
					/*информация вводимая колесом мышм отправляется в приложение с
фокусом*/
					wprintf(L"%s:MOUSEWHEEL_ROUTING_FOCUS\n", lpDispalyNam);
					break;
				case MOUSEWHEEL_ROUTING_HYBRID:
					/* информация введенная колёсиком мыши отправляетсяв приложение
					рабочего стола с фокусом или приложения Window Store под
курсором мыши*/
					wprintf(L"%s:MOUSEWHEEL_ROUTING_HYBRID \n", lpDispalyNam);
					break;
				case MOUSEWHEEL_ROUTING_MOUSE_POS:
					wprintf(L"%s:MOUSEWHEEL_ROUTING_MOUSE_POS \n", lpDispalyNam);
					break;
				default:
					wprintf(L"%s:%u \n", lpDispalyNam,value);
					break;
				}
			}
		}
```

```
		break;
	case SPI_GETWORKAREA:
	{
		RECT rcWorkArea;
		bReturn = SystemParametersInfo(uiAction, 0, &rcWorkArea, 0);
		if (FALSE != bReturn)
		{
			wprintf(L"%s:%d x %d \n", lpDispalyNam, (rcWorkArea.right - rcWorkArea.left),
(rcWorkArea.bottom - rcWorkArea.top));
		}
	}
	break;
	}
	return bReturn;}
```

Вызов описанных ранее библиотек осуществляется в файле mainenter.cpp:

```
#include <Windows.h>
#include <wchar.h>
#include <locale.h>
#include <stdio.h>
#include <delayimp.h>
#include <ShlObj.h>
#include <STATILIBR.h>
#include <DYNLIB1.h>
#include <DYNLIB2.h>
#include <DYNLIB3.h>

#pragma comment (lib,"STATILIBR.lib")
#pragma comment (lib,"DYNLIB1.lib") // неявное подключение к динам.библиотеке

int wmain()
{
	_wsetlocale(LC_ALL, TEXT(""));
	wprintf(TEXT("\n1. Вывод данных статической библиотеки\n"));
	PrintCompName();
	PrintDNSName();
	PrintUserName();
	PrintUserNameExtended();
	wprintf(TEXT("\n\n2. Вывод данных динамической библиотеки\n"));
	wprintf(TEXT("\nПути к системным каталогам\n"));
	const long csdir[] =
	{
		CSIDL_APPDATA, CSIDL_COMMON_APPDATA, CSIDL_COMMON_DOCUMENTS,
		CSIDL_HISTORY, CSIDL_INTERNET_CACHE, CSIDL_LOCAL_APPDATA,
		CSIDL_PERSONAL, CSIDL_PROGRAMS, CSIDL_PROGRAM_FILES,
		CSIDL_PROGRAM_FILES_COMMON, CSIDL_SYSTEM, CSIDL_WINDOWS,
		CSIDL_DESKTOP, CSIDL_STARTUP
	};//список части идентификаторов системных папок
	PrintSysDir(csdir, _countof(csdir));
	wprintf(TEXT("\nВерсия операционной системы\n\n"));
	PrintOSinfo();
	wprintf(TEXT("\nТекущая дата и время\n\n"));
	TimeDateInfo(LOCALE_NAME_INVARIANT, TIME_NOTIMEMARKER, L"\tdd-MM-yyyy", L"\thh:mm:ss tt");

	HMODULE Hdll = LoadLibrary(TEXT("DYNLIB2.dll")); //явное подключение к библиотеке

	if (NULL != Hdll)
	{
		PRINT_SYSMETR_PROC PrintSYSmetr = (PRINT_SYSMETR_PROC)GetProcAddress(Hdll,
"PrintSYSmetr");
		if (NULL != PrintSYSmetr)
		{
			PrintSYSmetr(L"SM_CXEDGE", SM_CXEDGE);
			PrintSYSmetr(L"SM_CYEDGE", SM_CYEDGE);
			PrintSYSmetr(L"SM_CXMINSPACING", SM_CXMINSPACING);
			PrintSYSmetr(L"SM_CYMINSPACING", SM_CYMINSPACING);
			PrintSYSmetr(L"SM_SHOWSOUNDS", SM_SHOWSOUNDS);
```

```
        }
        else
        {
                wprintf(TEXT("Функция не найдена %d"),GetLastError());
        }
        FreeLibrary(Hdll);//для выгрузки библиотеки из адресного пространства
}
else
{
        wprintf(TEXT("Функция не найдена %d \n"), GetLastError());
}
__try
{
        PrintSYSparamInfo(L"\nSPI_GETMOUSEWHEELROUTING", SPI_GETMOUSEWHEELROUTING);
        PrintSYSparamInfo(L"\nSPI_GETWORKAREA", SPI_GETWORKAREA);
}
__except (EXCEPTION_EXECUTE_HANDLER)
{
        switch (GetLastError())
        {
        case ERROR_MOD_NOT_FOUND:
                wprintf(TEXT(":%d DYNLIB3.dll"), ERROR_MOD_NOT_FOUND);
                break;
        case ERROR_PROC_NOT_FOUND:
                wprintf(TEXT(":%d PrintSYSparamInfo"), ERROR_PROC_NOT_FOUND);
                break;
        }
}
__FUnloadDelayLoadedDLL2("DYNLIB3.dll");//для выгрузки отложено загруженного DLL
system("pause");}
```

7. При тестировании приложения возникла ошибка: LNK 1104: не удается открыть файл STATILIBR.lib. Решение заключалось в том, чтобы разместить исполняемый файл в том же месте, где хранятся используемые библиотеки. Результат работы отлаженной программы продемонстрирован на рис.

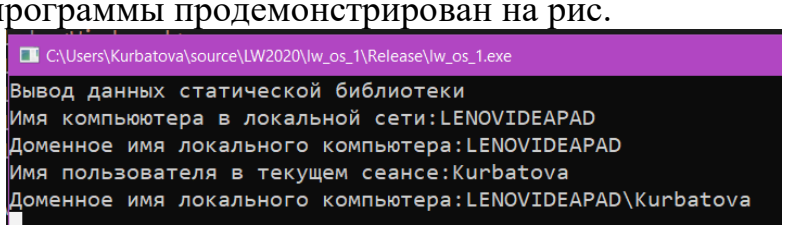

Рис. 1.3. Вывод данных из статистической библиотеки

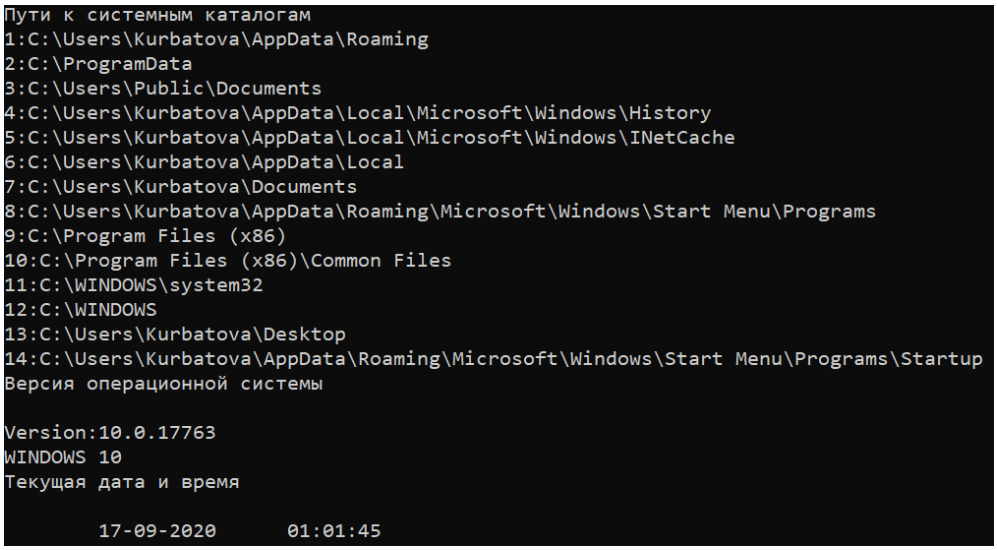

Рис. 1.4. Результат работы динамической библиотеки 1

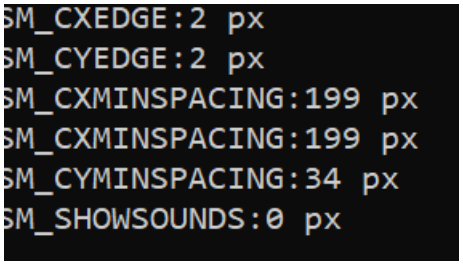

Рис. 1.5. Результат работы динамической библиотеки 2

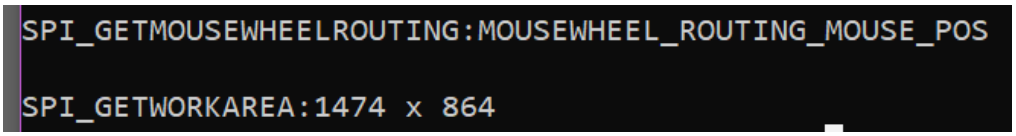

Рис. 1.6. Результат работы динамической библиотеки 3

Вывод: Таким образом, в ходе выполнения лабораторной работы было осуществлено знакомство с интерфейсом прикладного программирования Windows. Его содержимое включает совокупность функций, соглашения об использовании этих функций , которые позволяют создавать приложения работающие под управлением операционной системыMicrosoft Windows.

В работе использовались функции позволяющие узнать имя компьютера, имя текущего пользователя, разнообразные системные метрики и параметры, а также выяснить текущие время и дату. Было замечено также, что в WinAPI определены собственные типы данных, которые внешне похожи на представленные в C/C++ типы, но отличающиеся от них. Например, BOOL (typedef int BOOL, привязан к int, в то время как bool (из C/C++) привязан к char) или CHAR. Они определяются в заголовочных файлах через typedef или #define.

Все описанные в содержании работы задачи, можно было бы реализовать в одном файле, описав все необходимые функции. Но, это не обеспечивает гибкости программы и не позволило бы в дальнейшем повторно использовать описанные функции. Поэтому были созданы статическая и динамическая библиотеки. Это также было обусловлено и важной особенностью WinAPI: связывание библиотечных функций с кодом программы на этапе компоновки может быть отложено. Осуществлено оно будет только после загрузки необходимой библиотеки DLL. То есть библиотека будет использоваться только тогда, когда из нее вызвана функция.

Наиболее часто возникшие в работе ошибки были связаны с подключением библиотек к создаваемому приложению. Было выяснено, что запускаемое приложение и библиотеки должны храниться в одном месте или адрес до них должен быть точно определен для исполняемого файла.

Таким образом, в ходе выполнения лабораторной работы было создано консольное приложение для Windows, отображающее пользователю данные о его компьютере, о текущем времени и дате, об адресах системных папок, представлены некоторые системные параметры и метрики.