МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

Факультет радіоелектроніки, комп'ютерних систем та інфокомунікацій

Кафедра комп'ютерних систем, мереж і кібербезпеки

Лабораторна робота

з Системного програмування
(назва дисципліни)

на тему: «Вивчення системних викликів Win32 API роботи з реєстром»

Виконала: студентка 3-го курсу групи №525ст2
напряму підготовки (спеціальності)
123-«Комп'ютерна інженерія»
(шифр і назва напряму підготовки (спеціальності))
Коваленко Я.О.
(прізвище й ініціали студента)

Прийняв: асистент каф.503
Мозговий М.В.
(посада, науковий ступінь, прізвище й ініціали)

Національна шкала: _____
Кількість балів: _____
Оцінка: ECTS _____

Харків – 2020

Цель работы:

Изучение системных вызовов Win32 API работы с реестром.

Постановка задачи:

Требуется разроботать программу работы с реестром, которая бы реализовывала такие функции:

1. По имени ключа выводит перечень подключей

2. По имени уключа выводит перечень параметров ключа с их значением и типами

3. Выполняет поиск по реестру заданной строки в названиях ключей, названиях параметров и их значениях. Ключ относительно которого выполнять поиск задается пользователем.

4. Выполняет выгрузку заданного пользователем ключа в виде файла.

Ход работы:

Код программы:

```
#define _CRT_SECURE_NO_WARNINGS
#include "Main.h"

int main()
{
char choice = 0;

for (;;)
{
	PrintMenu();
	cin >> choice;
	switch (choice)
	{
		case '1':
		{
			HKEY hKey = { 0 };
			PHKEY phKey = &hKey;
			if (OpenKey(&phKey, KEY_READ, NULL) == true)
			{
				PrintListSubkeysByKey(hKey);
			}
		}break;
		case '2':
		{
			HKEY hKey = { 0 };
			PHKEY phKey = &hKey;
			if (OpenKey(&phKey, KEY_QUERY_VALUE, NULL) == true)
			{
```

```
                                PrintListParamsByKey(hKey);
                        }
                } break;
                case '3':
                {
                        HKEY hKey = { 0 };
                        PHKEY phKey = &hKey;
                        CHAR fullPath[MAX_PATH];
                        if (OpenKey(&phKey, KEY_ALL_ACCESS, fullPath) == true)
                        {
                                CHAR reqString[MAX_PATH] = {'\0'};
                                cout << "Input string for searching:";
                                ReadStringWithWhitespaces(reqString, MAX_PATH, false);
                                FindStringInReg(hKey, reqString, fullPath);
                        }
                } break;
                case '4':
                {
                        HANDLE hToken;

                        if                              (!OpenProcessToken(GetCurrentProcess(),
TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, &hToken))
                        {
                                cout << "Cant get access rights (SE_BACKUP_NAME)\n Error
code:" << GetLastError() << endl;
                        }
                        if (SetPrivilege(hToken, SE_BACKUP_NAME, true))
                        {
                                HKEY hKey = { 0 };
                                PHKEY phKey = &hKey;
                                if (OpenKey(&phKey, KEY_ALL_ACCESS, NULL) == true)
                                {
                                        SaveKeyIntoFile(hKey);
                                }
                        }
                } break;
                case '5':
                {
                        return 0;
                } break;
        default:
                cout << "Error choice, try again\n";
                break;
        }
}

return 0;
}

bool FindStringInReg(HKEY hKey, LPCSTR reqStr, LPSTR fullPath)
{
KEY_INFO keyInfo = { 0 };
```

```
	DWORD retCode = ERROR_SUCCESS;
	LPSTR newSubkeyPath;
	if (!GetKeyInfo(hKey, &keyInfo))
	{
		return false;
	}
	if (keyInfo.cSubKeys)
	{
		for (int i = 0; i < keyInfo.cSubKeys; i++)
		{
			keyInfo.cbName = MAX_KEY_LENGTH;
			retCode = RegEnumKeyEx(hKey,
				i,
				keyInfo.achKey,
				&keyInfo.cbName,
				NULL,
				NULL,
				NULL,
				NULL);
			if (retCode == ERROR_SUCCESS)
			{
				if (_strcmpi(keyInfo.achKey, reqStr) == 0)
				{
					cout << " * Found in subkey name: " << fullPath << "\\" << keyInfo.achKey << endl;
				}
				newSubkeyPath = (LPSTR)malloc(MAX_VALUE_NAME * sizeof(TCHAR));
				strcpy(newSubkeyPath, fullPath);
				strcat(newSubkeyPath, "\\");
				strcat(newSubkeyPath, keyInfo.achKey);
				HKEY newKey = { 0 };
				if (RegOpenKeyEx(hKey, keyInfo.achKey, 0, KEY_ALL_ACCESS, &newKey) == ERROR_SUCCESS)
				{
					FindStringInReg(newKey, reqStr, newSubkeyPath);
				}
				free(newSubkeyPath);
			}
		}
	}
	if (keyInfo.cValues)
	{
		LPSTR lpValue = NULL;
		DWORD dwValue = keyInfo.cchMaxValue + 1;

		DWORD dwType = 0;

		LPBYTE lpData = NULL;
		DWORD dwData = 0;

		lpValue = (LPSTR)malloc((keyInfo.cchMaxValue + 1) * sizeof(BYTE));
```

```
		for (int i = 0; i < keyInfo.cValues; i++)
		{
			retCode = RegEnumValueA(hKey, i, lpValue, &dwValue, NULL, NULL, NULL, &dwData);
			lpData = (LPBYTE)malloc((dwData + 1) * sizeof(BYTE));

			dwValue = keyInfo.cchMaxValue + 1;

			retCode = RegEnumValueA(hKey,
				i,
				lpValue,
				&dwValue,
				NULL,
				&dwType,
				lpData,
				&dwData);

			if (retCode == ERROR_SUCCESS)
			{
				if (_strcmpi(lpValue, reqStr) == 0)
				{
					cout << " * Found in value name: " << fullPath << "; " << lpValue << endl;
				}
				if (((dwType & REG_EXPAND_SZ) == REG_EXPAND_SZ) || ((dwType & REG_SZ) == REG_SZ))
				{
					if (_strcmpi((LPSTR)lpData, reqStr) == 0)
					{
						cout << " * Found in data of value " << fullPath << "; " << lpValue << ";\n   data:" << lpData << endl;
					}
				}
			}
		}
	}
	RegCloseKey(hKey);
	}

	BOOL SetPrivilege(
	HANDLE hToken,          // access token handle
	LPCTSTR lpszPrivilege,  // name of privilege to enable/disable
	BOOL bEnablePrivilege   // to enable or disable privilege
	)
	{
	TOKEN_PRIVILEGES tp;
	LUID luid;

	if (!LookupPrivilegeValue(
		NULL,           // lookup privilege on local system
		lpszPrivilege,   // privilege to lookup
```

```
        &luid))        // receives LUID of privilege
{
        printf("LookupPrivilegeValue error: %u\n", GetLastError());
        return FALSE;
}

tp.PrivilegeCount = 1;
tp.Privileges[0].Luid = luid;
if (bEnablePrivilege)
        tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
else
        tp.Privileges[0].Attributes = 0;


if (!AdjustTokenPrivileges(
        hToken,
        FALSE,
        &tp,
        sizeof(TOKEN_PRIVILEGES),
        (PTOKEN_PRIVILEGES)NULL,
        (PDWORD)NULL))
{
        printf("AdjustTokenPrivileges error: %u\n", GetLastError());
        return FALSE;
}

if (GetLastError() == ERROR_NOT_ALL_ASSIGNED)
{
        printf("You account doesnt have SE_BACKUP_NAME privilege \n");
        return FALSE;
}
return TRUE;
}

bool SaveKeyIntoFile(HKEY hKey)
{
CHAR filePath[MAX_PATH];
DWORD retCode = ERROR_SUCCESS;
cout << "Input path to new file:\n";
ReadStringWithWhitespaces(filePath, MAX_PATH, false);

retCode = RegSaveKey(hKey, filePath, NULL);
switch (retCode)
{
case ERROR_SUCCESS:
{
        cout << "Key saved in the file:\n" << filePath << endl;
        RegCloseKey(hKey);
        return true;
} break;
case ERROR_ALREADY_EXISTS:
{
```

```
		cout << "Error! File already exists!\n Entered file path:\n" << filePath << endl;
	} break;
	default:
		cout << "Error! Cant save key into the file\n Error code:" << retCode << endl;
	}
	RegCloseKey(hKey);
	return false;
	}

	void PrintListParamsByKey(HKEY key)
	{
	DWORD i, retCode = ERROR_SUCCESS;
	KEY_INFO keyInfo = { 0 };

	DWORD dwType = 0;

	LPBYTE lpData = NULL;
	DWORD dwData = 0;

	LPSTR lpValue = NULL;
	DWORD dwValue = 0;

	GetKeyInfo(key, &keyInfo);

	if (keyInfo.cValues)
	{
		cout << "\t Values count:" << keyInfo.cValues << endl;
		lpValue = (LPSTR)malloc((keyInfo.cchMaxValue + 1) * sizeof(BYTE));
		dwValue = keyInfo.cchMaxValue + 1;

		for (int i = 0; i < keyInfo.cValues; i++)
		{
			retCode = RegEnumValueA(key, i, lpValue, &dwValue, NULL, NULL, NULL,
&dwData);
			lpData = (LPBYTE)malloc((dwData + 1) * sizeof(BYTE));

			dwValue = keyInfo.cchMaxValue + 1;

			retCode = RegEnumValueA(key,
				i,
				lpValue,
				&dwValue,
				NULL,
				&dwType,
				lpData,
				&dwData);

			if (retCode == ERROR_SUCCESS)
			{
				if (strcmp(lpValue, "") == 0)
				{
					printf("\n(%d) Value name: %s\n", i + 1, "Default value");
```

```
                        }
                        else
                        {
                                printf("\n(%d) Value name: %s\n", i + 1, lpValue);
                        }

                        switch (dwType)
                        {
                                case REG_BINARY:
                                {
                                        printf("     Value type: REG_BINARY\n     Value data:
binary\n");
                                } break;
                                case REG_DWORD:
                                {
                                        DWORD data = *(DWORD*)(lpData);
                                        printf("     Value type: REG_DWORD\n     Value data:
%#x|%u\n", data, data);
                                } break;
                                case REG_EXPAND_SZ:
                                {
                                        printf("     Value type: REG_EXPAND_SZ\n     Value
data: %s\n", lpData);
                                } break;
                                case REG_LINK:
                                {
                                        wprintf(L"     Value type: REG_LINK\n     Value data:
%ws\n", lpData);
                                } break;
                                case REG_SZ:
                                {
                                        printf("     Value type: REG_SZ\n     Value data: %s\n",
lpData);
                                } break;
                                case REG_NONE:
                                {
                                        printf("     Value type: REG_NONE\n     Value data:
%x\n", *(DWORD*)(lpData));
                                } break;
                                default:
                                        printf("     Value type: unknown\n     Value data: %x\n",
*(DWORD*)(lpData));
                                        break;
                        }
                }
                free(lpData);
        }
        free(lpValue);
    }
    RegCloseKey(key);
    }
```

```
void PrintListSubkeysByKey(HKEY key)
{
DWORD i, retCode;
KEY_INFO keyInfo = {0};

GetKeyInfo(key, &keyInfo);

if (keyInfo.cSubKeys)
{
	cout << "\t Subkeys count:" << keyInfo.cSubKeys << endl;
	for (int i = 0; i < keyInfo.cSubKeys; i++)
	{
		keyInfo.cbName = MAX_KEY_LENGTH;
		retCode = RegEnumKeyEx(key,
			i,
			keyInfo.achKey,
			&keyInfo.cbName,
			NULL,
			NULL,
			NULL,
			NULL);
		if (retCode == ERROR_SUCCESS)
		{
			printf("(%d) %s\n", i + 1, keyInfo.achKey);
		}
	}
}
RegCloseKey(key);
}

void ReadStringWithWhitespaces(CHAR sBuffNewPath[], DWORD maxBuffSize, BOOL isUsedBeforeInputChar)
{
memset(sBuffNewPath, '\0', sizeof(sBuffNewPath));
if (isUsedBeforeInputChar)
fgets(sBuffNewPath, maxBuffSize, stdin);
if ((strlen(sBuffNewPath) > 0) && (sBuffNewPath[strlen(sBuffNewPath) - 1] == '\n'))
	sBuffNewPath[strlen(sBuffNewPath) - 1] = '\0';
}

bool GetKeyInfo(HKEY key, KEY_INFO * keyInfo)
{
	DWORD retCode = RegQueryInfoKey(key,
					(*keyInfo).achClass,
					&(*keyInfo).cchClassName,
					NULL,
					&(*keyInfo).cSubKeys,
					&(*keyInfo).cbMaxSubKey,
					&(*keyInfo).cchMaxClass,
					&(*keyInfo).cValues,
					&(*keyInfo).cchMaxValue,
					&(*keyInfo).cbMaxValueData,
```

```
                                                    &(*keyInfo).cbSecurityDescriptor,
                                                    &(*keyInfo).ftLastWriteTime);
if (retCode == ERROR_SUCCESS) return true;
else return false;
}

bool OpenKey(HKEY** hKey, DWORD dwOpenAccess, LPSTR fullPath)
{
HKEY predKey;
if(fullPath != NULL)  memset(fullPath, '\0', sizeof(fullPath));
int choice = 0;
cout << "Predefined keys:\n";
cout << "1 - HKEY_CLASSES_ROOT\n";
cout << "2 - HKEY_CURRENT_USER\n";
cout << "3 - HKEY_LOCAL_MACHINE\n";
cout << "4 - HKEY_USERS\n";
cout << "5 - HKEY_CURRENT_CONFIG\n";
cout << "6 - HKEY_PERFORMANCE_DATA\n";
cout << "Choose predefined key:";
scanf("%d", &choice);
switch (choice)
{
        case 1:
        {
                predKey = HKEY_CLASSES_ROOT;
                if(fullPath != NULL) strcpy(fullPath, "HKEY_CLASSES_ROOT\\");
        } break;
        case 2:
        {
                predKey = HKEY_CURRENT_USER;
                if(fullPath != NULL) strcpy(fullPath, "HKEY_CURRENT_USER\\");
        } break;
        case 3:
        {
                predKey = HKEY_LOCAL_MACHINE;
                if(fullPath != NULL) strcpy(fullPath, "HKEY_LOCAL_MACHINE\\");
        } break;
        case 4:
        {
                predKey = HKEY_USERS;
                if(fullPath != NULL) strcpy(fullPath, "HKEY_USERS\\");
        } break;
        case 5:
        {
                predKey = HKEY_CURRENT_CONFIG;
                if(fullPath != NULL) strcpy(fullPath, "HKEY_CURRENT_CONFIG\\");
        } break;
        case 6:
        {
                predKey = HKEY_PERFORMANCE_DATA;
                if(fullPath != NULL) strcpy(fullPath, "HKEY_PERFORMANCE_DATA\\");
        } break;
```

```
		default:
			return false;
	}

	CHAR keyArr[MAX_KEY_LENGTH] = { '\0' };
	LPSTR key = keyArr;
	cout << "Input subkey(path to subkey) in the given predefined key:\n";
	ReadStringWithWhitespaces(key, MAX_KEY_LENGTH, true);
	if (RegOpenKeyEx(predKey, (LPCSTR)key, 0, dwOpenAccess, *hKey) == ERROR_SUCCESS)
	{
		if (fullPath != NULL) strcat(fullPath, key);
		return true;
	}
	return false;
	}

	void PrintMenu()
	{
	cout << "Menu\n";
	cout << "1 Print a list of subkeys by key name\n";
	cout << "2 Print a list of keys parameters with their value and type\n";
	cout << "3 Searches the registry for a given string in the key names, key values and their types.\n\t\t Base key set user\n";
	cout << "4 Save key as a file\n";
	cout << "5 Exit\n";
	}
	#include <stdio.h>
	#include "windows.h"
	#include "iostream"
	#include "tchar.h"
	#include "processthreadsapi.h"


	#define MAX_KEY_LENGTH 255
	#define MAX_VALUE_NAME 16383

	using namespace std;

	// struct for key information (mostly use in RegQueryInfoKey)
	typedef struct {
	TCHAR    achKey[MAX_KEY_LENGTH];   // buffer for subkey name
	DWORD    cbName;                   // size of name string
	TCHAR    achClass[MAX_PATH] = TEXT("");  // buffer for class name
	DWORD    cchClassName = MAX_PATH;  // size of class string
	DWORD    cSubKeys = 0;             // number of subkeys
	DWORD    cbMaxSubKey;              // longest subkey size
	DWORD    cchMaxClass;              // longest class string
	DWORD    cValues;            // number of values for key
	DWORD    cchMaxValue;          // longest value name
	DWORD    cbMaxValueData;       // longest value data
	DWORD    cbSecurityDescriptor; // size of security descriptor
```

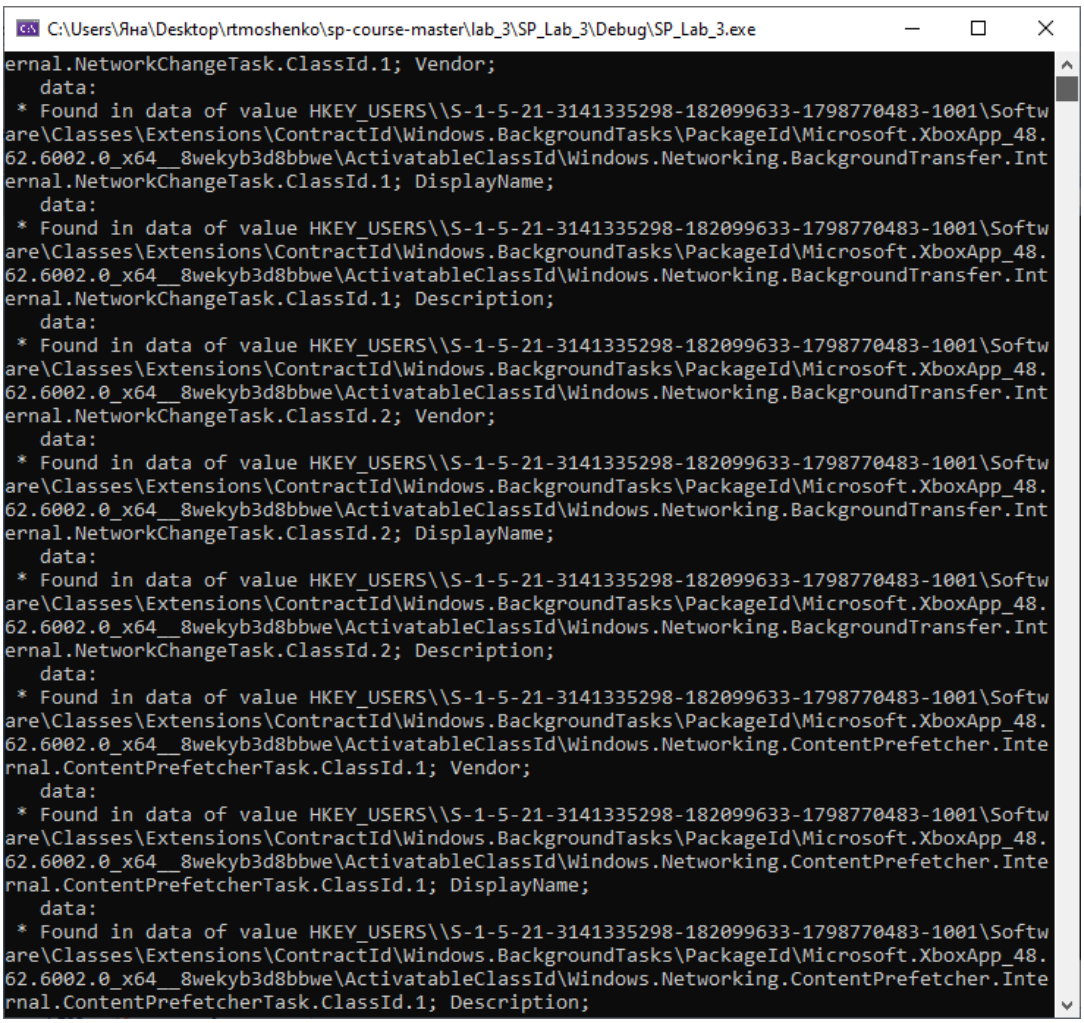
```
FILETIME ftLastWriteTime;      // last write time
} KEY_INFO, * pKEY_INFO;

// common functions
// print menu to console
void PrintMenu();

// open key in registry, fullPath can be NULL
bool OpenKey(HKEY** hKey, DWORD dwOpenAccess, LPSTR fullPath);

// Read string form sdtin
void ReadStringWithWhitespaces(CHAR sBuffNewPath[], DWORD maxBuffSize, BOOL isUsedBeforeInputChar);

// Get key information (KEY_INFO struct)
bool GetKeyInfo(HKEY key, KEY_INFO* keyInfo);

//-------------------------------------------//
// function for print list subkeys by key name
void PrintListSubkeysByKey(HKEY key);

//-------------------------------------------//
// function for print all key parameters and their types
void PrintListParamsByKey(HKEY key);

//-------------------------------------------//
// function for search string in reg (recursive function), output in
// stdout all hits
bool FindStringInReg(HKEY hKey, LPCSTR reqStr, LPSTR fullPath);

//-------------------------------------------//
// function for save key in file
// save key into the file
bool SaveKeyIntoFile(HKEY hKey);
// set requierd privilege (SE_BACKUP_NAME) for current process
BOOL SetPrivilege(
HANDLE hToken,          // access token handle
LPCTSTR lpszPrivilege,  // name of privilege to enable/disable
BOOL bEnablePrivilege   // to enable or disable privilege
);
```

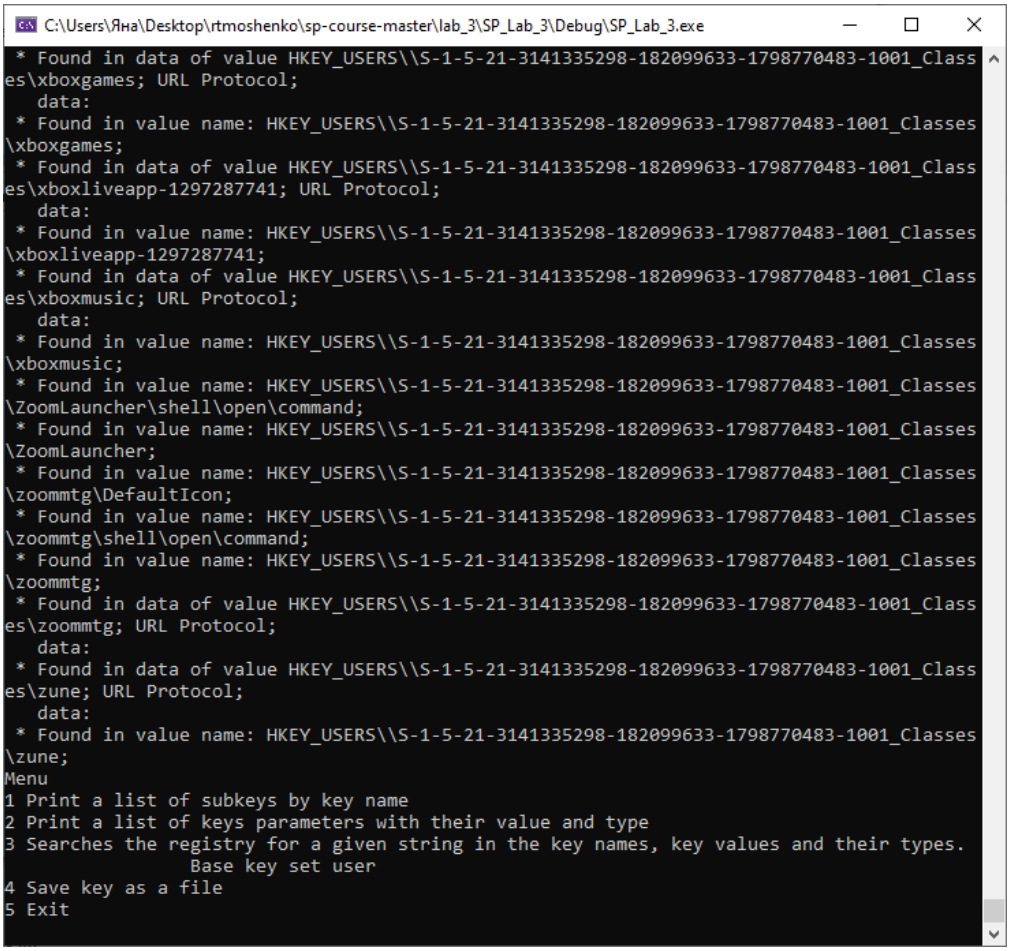
Результат работы:

```
Menu
1 Print a list of subkeys by key name
2 Print a list of keys parameters with their value and type
3 Searches the registry for a given string in the key names, key values and their types.
                Base key set user
4 Save key as a file
5 Exit
1
Predefined keys:
1 - HKEY_CLASSES_ROOT
2 - HKEY_CURRENT_USER
3 - HKEY_LOCAL_MACHINE
4 - HKEY_USERS
5 - HKEY_CURRENT_CONFIG
6 - HKEY_PERFORMANCE_DATA
Choose predefined key:2
Input subkey(path to subkey) in the given predefined key:
         Subkeys count:12
(1) AppEvents
(2) Console
(3) Control Panel
(4) Environment
(5) EUDC
(6) Keyboard Layout
(7) Network
(8) Printers
(9) Software
(10) System
(11) Uninstall
(12) Volatile Environment
```

C:\Users\Яна\Desktop\rtmoshenko\sp-course-master\lab_3\SP_Lab_3\Debug\SP_Lab_3.exe

```
4
You account doesnt have SE_BACKUP_NAME privilege
Menu
1 Print a list of subkeys by key name
2 Print a list of keys parameters with their value and type
3 Searches the registry for a given string in the key names, key values and their types.
                Base key set user
4 Save key as a file
5 Exit
2
Predefined keys:
1 - HKEY_CLASSES_ROOT
2 - HKEY_CURRENT_USER
3 - HKEY_LOCAL_MACHINE
4 - HKEY_USERS
5 - HKEY_CURRENT_CONFIG
6 - HKEY_PERFORMANCE_DATA
Choose predefined key:4
Input subkey(path to subkey) in the given predefined key:
Menu
1 Print a list of subkeys by key name
2 Print a list of keys parameters with their value and type
3 Searches the registry for a given string in the key names, key values and their types.
                Base key set user
4 Save key as a file
5 Exit
4
You account doesnt have SE_BACKUP_NAME privilege
Menu
1 Print a list of subkeys by key name
2 Print a list of keys parameters with their value and type
3 Searches the registry for a given string in the key names, key values and their types.
                Base key set user
4 Save key as a file
5 Exit
3
Predefined keys:
1 - HKEY_CLASSES_ROOT
2 - HKEY_CURRENT_USER
3 - HKEY_LOCAL_MACHINE
4 - HKEY_USERS
```


C:\Users\Яна\Desktop\rtmoshenko\sp-course-master\lab_3\SP_Lab_3\Debug\SP_Lab_3.exe
ernal.NetworkChangeTask.ClassId.1; Vendor;
data:
* Found in data of value HKEY_USERS\\S-1-5-21-3141335298-182099633-1798770483-1001\Software\Classes\Extensions\ContractId\Windows.BackgroundTasks\PackageId\Microsoft.XboxApp_48.62.6002.0_x64__8wekyb3d8bbwe\ActivatableClassId\Windows.Networking.BackgroundTransfer.Internal.NetworkChangeTask.ClassId.1; DisplayName;
data:
* Found in data of value HKEY_USERS\\S-1-5-21-3141335298-182099633-1798770483-1001\Software\Classes\Extensions\ContractId\Windows.BackgroundTasks\PackageId\Microsoft.XboxApp_48.62.6002.0_x64__8wekyb3d8bbwe\ActivatableClassId\Windows.Networking.BackgroundTransfer.Internal.NetworkChangeTask.ClassId.1; Description;
data:
* Found in data of value HKEY_USERS\\S-1-5-21-3141335298-182099633-1798770483-1001\Software\Classes\Extensions\ContractId\Windows.BackgroundTasks\PackageId\Microsoft.XboxApp_48.62.6002.0_x64__8wekyb3d8bbwe\ActivatableClassId\Windows.Networking.BackgroundTransfer.Internal.NetworkChangeTask.ClassId.2; Vendor;
data:
* Found in data of value HKEY_USERS\\S-1-5-21-3141335298-182099633-1798770483-1001\Software\Classes\Extensions\ContractId\Windows.BackgroundTasks\PackageId\Microsoft.XboxApp_48.62.6002.0_x64__8wekyb3d8bbwe\ActivatableClassId\Windows.Networking.BackgroundTransfer.Internal.NetworkChangeTask.ClassId.2; DisplayName;
data:
* Found in data of value HKEY_USERS\\S-1-5-21-3141335298-182099633-1798770483-1001\Software\Classes\Extensions\ContractId\Windows.BackgroundTasks\PackageId\Microsoft.XboxApp_48.62.6002.0_x64__8wekyb3d8bbwe\ActivatableClassId\Windows.Networking.BackgroundTransfer.Internal.NetworkChangeTask.ClassId.2; Description;
data:
* Found in data of value HKEY_USERS\\S-1-5-21-3141335298-182099633-1798770483-1001\Software\Classes\Extensions\ContractId\Windows.BackgroundTasks\PackageId\Microsoft.XboxApp_48.62.6002.0_x64__8wekyb3d8bbwe\ActivatableClassId\Windows.Networking.ContentPrefetcher.Internal.ContentPrefetcherTask.ClassId.1; Vendor;
data:
* Found in data of value HKEY_USERS\\S-1-5-21-3141335298-182099633-1798770483-1001\Software\Classes\Extensions\ContractId\Windows.BackgroundTasks\PackageId\Microsoft.XboxApp_48.62.6002.0_x64__8wekyb3d8bbwe\ActivatableClassId\Windows.Networking.ContentPrefetcher.Internal.ContentPrefetcherTask.ClassId.1; DisplayName;
data:
* Found in data of value HKEY_USERS\\S-1-5-21-3141335298-182099633-1798770483-1001\Software\Classes\Extensions\ContractId\Windows.BackgroundTasks\PackageId\Microsoft.XboxApp_48.62.6002.0_x64__8wekyb3d8bbwe\ActivatableClassId\Windows.Networking.ContentPrefetcher.Internal.ContentPrefetcherTask.ClassId.1; Description;



Выводы:

В результате выполнения данной лабораторной работы были изучены системных вызовов Win32 API работы с реестром.