



МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Новосибирский государственный технический университет»



**НГТУ
НЭТИ** | **Факультет прикладной
математики и информатики**

Кафедра теоретической и прикладной информатики
Индивидуальная работа
по дисциплине «Управление ресурсами в вычислительных системах»

УПРАВЛЕНИЕ РЕСУРСАМИ В ОС WINDOWS

Вариант 23 ГЛУШКО ВЛАДИСЛАВ

Группа ПМ-92

Преподаватели СТАСЫШИН ВЛАДИМИР МИХАЙЛОВИЧ
 СИВАК МАРИЯ АЛЕКСЕЕВНА

Новосибирск, 2022

Цель работы

Изучение основных особенностей ОС Windows: работа с потоками, организация графического интерфейса, механизма обработки сообщений, использование динамически подключаемых библиотек (dll), низкоуровневое взаимодействие с процессором.

Содержание работы

В рамках первого уровня необходимо реализовать консольное Windows-приложение, в котором создаётся один дочерний поток, реализующий требования варианта задания и выводящий результат на консоль.

В рамках второго уровня необходимо реализовать минимальное графическое Windows-приложение, в котором при создании окна, но до момента его отображения на экране, создаётся дочерний поток, который реализует задачу в соответствии с выбранным вариантом, после чего полученный результат отображается в графическом окне.

При выполнении задания на третьем уровне необходимо реализовать все требования второго уровня с той разницей, что функция, выполняющая задание варианта, должна быть реализована в виде динамической библиотеки

Вариант задания

№	ЗАДАНИЕ ДЛЯ ВЫПОЛНЕНИЯ 1, 2, 3 УРОВНЯ СЛОЖНОСТИ	ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ ДЛЯ ВЫПОЛНЕНИЯ 4 УРОВНЯ СЛОЖНОСТИ
23	Определить текущую минуту	Размер КЭШа данных третьего уровня

Описание программных средств

Для разработки приложения была использована библиотека Windows API.

В проекте был использован CMake – кроссплатформенное средство с открытым кодом, которое позволяет определить процессы сборки, выполняемые на множестве платформ.

Для удобной сборки и компилирования проекта был написан скрипт (PowerShell):

compile.ps1

```
1 $build_path = 'build'
2 if (Test-Path -Path $build_path) {
3     Remove-Item -Path $build_path -Recurse }
4 New-Item -Path $build_path -ItemType Directory
5 cmake -S . -B $build_path
6 cmake --build $build_path
```

Используемые функции

RegisterClass – регистрация класса окна для последующего использования при вызове функции **CreateWindow**.

CreateWindow – создание перекрывающего, выскакивающего или дочернего окна. Функция определяет класс, заголовок, стиль окна, начальную позицию и размер окна.

GetMessage – извлечение сообщения из очереди сообщений вызывающего потока и помещение его в заданную структуру. Эта функция регулирует поступление отправленных сообщений до тех пор, пока помещенное в очередь сообщение доступно для извлечения.

TranslateMessage – перевод сообщений виртуальных клавиш в символьные сообщения. Символьные сообщения помещаются в очереди сообщений вызывающего потока для прочтения в следующий раз.

DispatchMessage – распределение сообщения оконной процедуре. Обычно используется, чтобы доставить сообщение, извлеченное функцией **GetMessage**.

CreateThread – создание потока, который выполняется в пределах виртуального адресного пространства вызывающего процесса. Возвращает дескриптор потока.

DefWindowProc – вызывается оконной процедурой по умолчанию, чтобы обеспечить обработку по умолчанию любого сообщения окна, которые приложение не обрабатывает. Эта функция гарантирует то, что обрабатывается каждое сообщение.

LoadLibrary – отображение заданного исполняемого модуля в адресное пространство вызывающего процесса.

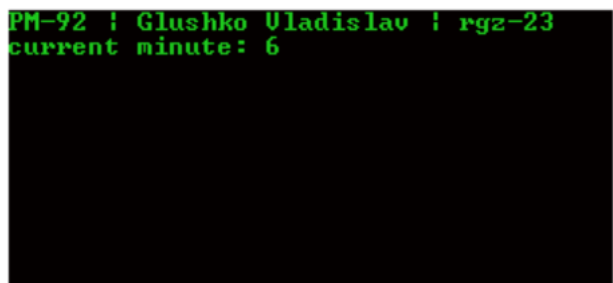
GetProcAddress – обработка явного связывания с вызовом DLL для получения адреса экспортированной функции в DLL. Для вызова функции DLL используется указатель возвращаемой функции.

GetSystemTime – получение текущей даты и времени операционной системы.

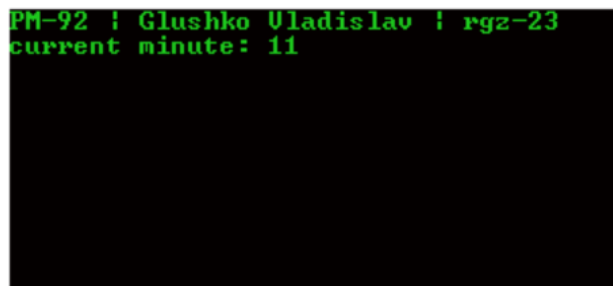
CloseHandle – закрывает дескриптор открытого объекта.

Результат работы программы

При успешном подключении динамической библиотеки:



```
PM-92 ! Glushko Vladislav ! rgz-23
current minute: 6
```



```
PM-92 ! Glushko Vladislav ! rgz-23
current minute: 11
```

При неудачном подключении динамической библиотеки:

```
failed to load time_info.dll
```

При подключении к другой динамической библиотеки:

```
get_time_minute() - not found in  
time_info.dll
```

Исходный текст

CMakeLists.txt

```
1 cmake_minimum_required(VERSION 3.14 FATAL_ERROR)
2
3 set(META_PROJECT_NAME      "urvs_rg23")
4 set(META_VERSION_MAJOR     "1")
5 set(META_VERSION_MINOR     "0")
6 set(META_VERSION_PATCH     "0")
7 set(META_VERSION
  "${META_VERSION_MAJOR}.${META_VERSION_MINOR}.${META_VERSION_PATCH}")
8
9 # Преобразование каждого небуквенно-цифрового символа на входе
10 # в символ подчеркивания
11 string(MAKE_C_IDENTIFIER ${META_PROJECT_NAME} META_PROJECT_ID)
12 # Изменение регистра на (ПРОПИСНЫЕ)
13 string(TOUPPER           ${META_PROJECT_ID}   META_PROJECT_ID)
14
15 set(${META_PROJECT_NAME}_CXX_STANDARD 14)      # Стандарт CXX
16
17 # -----
18 project(
19     ${META_PROJECT_NAME}                # <PROJECT-NAME>
20     VERSION ${META_VERSION}             # [VERSION <major>[.<minor>[.<patch>[.<tweak>]]]]
21     DESCRIPTION "URVS RG23"             # [DESCRIPTION <project-description-string>]
22     LANGUAGES CXX                       # [LANGUAGES <language-name> ... ]
23 )
24 add_subdirectory(src)
25
26 # -----
27 # Подведение итогов настроек, напечатанных при сборке
28 message(STATUS " = Окончательный обзор для ${PROJECT_NAME} = ")
29 message(STATUS "Version:      ${META_VERSION}")
30 message(STATUS "Compiler:    ${CMAKE_CXX_COMPILER}")
31 message(STATUS "")
32 # -----
```

```

1
2 configure_file(cfg/config.h.in ${PROJECT_SOURCE_DIR}/include/config/config.h @ONLY)
3 # ----- LIBRARY ----- #
4 set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/bin)
5 set(CMAKE_SHARED_LIBRARY_PREFIX "")
6 set(SHARED_NAME_LIBRARY time_info) # Имя динамической библиотеки
7 # Исходные файлы динамической библиотеки
8 set(LIB_SOURCES ${CMAKE_CURRENT_SOURCE_DIR}/lib/time_info.cpp)
9 add_library(${SHARED_NAME_LIBRARY} SHARED ${LIB_SOURCES})
10 # ----- LIBRARY ----- #
11 set(SOURCES ${PROJECT_SOURCE_DIR}/src/main.cpp) # Исходные файлы
12 set(HEADERS) # Заголовочные файлы
13 # Компилируем исполняемый файл с заданным именем
14 add_executable(${PROJECT_NAME} WIN32 ${SOURCES} ${HEADERS})
15 # Линкуем динамическую библиотеку к исполняемому файлу
16 target_link_libraries(${PROJECT_NAME} PRIVATE ${SHARED_NAME_LIBRARY})
17 # Переопределение пути до заголовочных файлов
18 target_include_directories(
19     ${PROJECT_NAME} PRIVATE
20     ${PROJECT_SOURCE_DIR}/include)
21 # ----- #
22 set_target_properties(${PROJECT_NAME} PROPERTIES
23     CXX_STANDARD ${${META_PROJECT_NAME}_CXX_STANDARD} # CXX - стандарт
24     CXX_STANDARD_REQUIRED ON # Обязательная поддержка стандарта
25     # Задает рабочий каталог локального отладчика
26     # для целевых объектов Visual Studio C++
27     VS_DEBUGGER_WORKING_DIRECTORY ${PROJECT_SOURCE_DIR})

```

```

1  #include <windows.h>    ///< the best library
2  #include <tchar.h>      ///< для макроса _T и _TEXT
3
4  /// — CXX library
5  #include <iostream>
6  #include <sstream>
7  #include <string>
8  /// — CXX library
9
10 /// Конфигурационный файл
11 #include "config/config.h"
12
13 #define IDC_LABEL1 1025    ///< ID label
14 HWND MSLabel;
15
16
17 LRESULT CALLBACK main_win_proc (HWND, UINT, WPARAM, LPARAM);
18 DWORD WINAPI thread_func ();
19
20 /**
21  * @brief Основная функция
22  * @param hInstance дескриптор текущего экземпляра приложения
23  * @param nWinMode управляет отображением окна
24  * @return 0
25  */
26 int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE, LPSTR, int nWinMode) {
27     FreeConsole();          ///< Избавляемся от консоли
28     WNDCLASS wcl;
29
30     /// #define _T(x)      __T(x)
31     /// #define _TEXT(x)   __T(x)
32     /// _T и _TEXT — аналогичны
33     wcl.lpszClassName = _T(URVS_RGZ23_PROJECT_NAME);    ///< имя класса окна
34     wcl.lpfnWndProc = main_win_proc;                  ///< указатель на процедуру окна
35     wcl.hInstance = hInstance;                        ///< дескриптор экземпляра, содержащего процедуру окна для класса
36
37     wcl.lpszMenuName = NULL;                          ///< имя ресурса меню класса
38     wcl.hIcon = LoadIcon(NULL, IDI_APPLICATION);     ///< TODO: Добавить иконку
39     wcl.hCursor = LoadCursor(NULL, IDC_ARROW);       ///< маркер курсора класса
40     wcl.cbClsExtra = 0;
41     wcl.cbWndExtra = 0;
42
43     wcl.hbrBackground = reinterpret_cast<HBRUSH>(COLOR_WINDOW + 1); ///< Маркер фоновой кисти класса
44     wcl.style = CS_HREDRAW | CS_VREDRAW;              ///< стиль класса
45     /// CS_VREDRAW | CS_HREDRAW :
46     /// осуществляет перерисовку окна при перемещении или изменении высоты и ширины окна
47
48     RegisterClass(&wcl);
49
50     HDC hDCScreen = GetDC(NULL);                      ///< Извлекает дескриптор контекста дисплея
51     int _app_width = 320;                             ///< Ширина application
52     int _app_height = 180;                            ///< Высота application
53
54     /// Создаем главное окно и отображение его
55     HWND hwnd = CreateWindow(
56         _T(URVS_RGZ23_PROJECT_NAME),                ///< указатель на зарегистрированное имя класса
57         _T("RGZ23: current minute"),                 ///< указатель на имя окна
58         WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX, ///< стиль окна
59         (GetDeviceCaps(hDCScreen, HORZRES) - _app_width) / 2,    ///< горизонтальная позиция окна
60         (GetDeviceCaps(hDCScreen, VERTRES) - _app_height) / 2,    ///< вертикальная позиция окна
61         _app_width,                                             ///< ширина окна
62         _app_height,                                           ///< высота окна
63         NULL,                                                  ///< дескриптор родительского или окна владельца
64         NULL,                                                  ///< дескриптор меню или ID дочернего окна
65         hInstance,                                             ///< дескриптор экземпляра приложения
66         NULL);                                                 ///< указатель на данные создания окна
67
68     ShowWindow(hwnd, nWinMode);
69     UpdateWindow(hwnd);
70
71     MSG msg;
72     while (GetMessage(&msg, NULL, 0, 0)) {           ///< Сообщение
73         TranslateMessage(&msg);                     ///< Цикл обработки событий
74         DispatchMessage(&msg);
75     }
76     return static_cast<int>(msg.wParam);
77 }

```



```

78 /**
79 * @brief      Функция обработки сообщений для главного окна
80 * @param _hw  Дескриптор процедуры окна, которая получила сообщение
81 * @param _msg Сообщение
82 * @param _wp  Дополнительная информация о сообщении
83 * @param _lp  Дополнительная информация о сообщении
84 * @return     Результат обработки сообщения и зависит от сообщения
85 */
86 LRESULT CALLBACK main_win_proc (HWND _hw, UINT _msg, WPARAM _wp, LPARAM _lp) {
87     switch (_msg) {
88         case WM_CREATE:          /// Отправляется, когда приложение запрашивает создание окна
89         {
90             RECT rt;
91             GetClientRect(_hw, &rt);
92             int static_width = rt.right;
93             int static_height = rt.bottom;
94             int _border_textbox = 2;
95
96             HINSTANCE hIns = reinterpret_cast<LPCREATESTRUCT>(_lp)→hInstance;
97             /// STATIC - обозначает простое текстовое поле, окно или прямоугольник, используемый для надписей
98             /// BUTTON - обозначает маленькое прямоугольное дочернее окно, которое представляет собой кнопку
99             /// EDIT - обозначает прямоугольное дочернее окно, внутри которого пользователь может напечатать текст с клавиатуры
100
101             MSLabel = CreateWindow(
102                 _T("static"),          ///< указатель на зарегистрированное имя класса
103                 _T("label1"),         ///< указатель на имя окна
104                 WS_CHILD | WS_VISIBLE, ///< стиль окна
105                 _border_textbox,       ///< горизонтальная позиция окна
106                 _border_textbox,       ///< вертикальная позиция окна
107                 static_width - 2 * _border_textbox, ///< ширина окна
108                 static_height - 2 * _border_textbox, ///< высота окна
109                 _hw,                  ///< дескриптор родительского окна
110                 reinterpret_cast<HMENU>(IDC_LABEL1), ///< ID дочернего окна
111                 hIns,                 ///< дескриптор экземпляра приложения
112                 NULL);                ///< указатель на данные создания окна
113
114             HFONT font_mono = reinterpret_cast<HFONT>(GetStockObject(OEM_FIXED_FONT)); /// Меняем шрифт на моноширинный
115             SendDlgItemMessage(_hw, IDC_LABEL1, WM_SETFONT, reinterpret_cast<WPARAM>(font_mono), TRUE);
116
117             DWORD IDThread;
118             /// Поток с вычисление текущей минуты
119             HANDLE hThread = CreateThread(
120                 NULL,                  ///< дескриптор защиты
121                 0,                    ///< начальный размер стека
122                 reinterpret_cast<LPTHREAD_START_ROUTINE>(thread_func), ///< функция потока
123                 NULL,                ///< параметр потока
124                 0,                    ///< опции создания
125                 &IDThread);          ///< идентификатор потока
126
127             /// Закрываем дескриптор открытого объекта
128             CloseHandle(hThread);
129             return 0;
130         }
131         case WM_CTLCOLORSTATIC:      /// Статический элемент управления
132         {
133             DWORD CtrlID = GetDlgCtrlID(reinterpret_cast<HWND>(_lp));
134             HDC hdcStatic = reinterpret_cast<HDC>(_wp);
135             if (CtrlID == IDC_LABEL1) {
136                 SetTextColor(hdcStatic, RGB(30, 200, 30));
137                 SetBkColor(hdcStatic, BLACK_BRUSH);
138                 return reinterpret_cast<LRESULT>(CreateSolidBrush(BLACK_BRUSH));
139             }
140             return 0;
141         }
142         case WM_DESTROY:             /// Отправляется при разрушении окна
143         {
144             PostQuitMessage(0);      ///< Пользователь закрыл окно
145             return 0;                ///< Программа может завершаться
146         }
147         default: return DefWindowProc(_hw, _msg, _wp, _lp);
148     }
149 }

```

```

151 /**
152  * @brief Функция, запускаемая в рамках создаваемого потока
153  *
154  */
155 DWORD WINAPI thread_func () {
156     /// Дескриптор экземпляра динамической библиотеки
157     HINSTANCE hinstLib = LoadLibrary(_T("time_info.dll"));
158
159     /// Если мы получим дескриптор
160     if (hinstLib ≠ NULL) {
161         /// Поток для входного текста
162         std::ostringstream _ostream;
163         _ostream << "PM-92 | Glushko Vladislav | rgz-23" << '\n';
164         typedef size_t (*get_time_minute_>();
165         get_time_minute_ ifconnected =
166             reinterpret_cast<get_time_minute_>(GetProcAddress(hinstLib, "get_time_minute"));
167
168         if(ifconnected ≠ NULL) {
169             std::size_t _m = ifconnected();
170             _ostream << "current minute: " << _m << '\n';
171         } else {
172             std::ostringstream _ostream_err;
173             _ostream_err << "get_time_minute() - not found in time_info.dll" << '\n';
174
175             /// Устанавливаем текст
176             SetWindowTextA(MSLabel, _ostream_err.str().c_str());
177             constexpr unsigned long _err_not_func = { 3 };
178             return _err_not_func;
179         }
180         SetWindowTextA(MSLabel, _ostream.str().c_str());
181
182         /// Освобождаем загруженный модуль
183         FreeLibrary(hinstLib);
184     } else {
185         std::ostringstream _ostream_err;
186         _ostream_err << "failed to load time_info.dll" << '\n';
187
188         SetWindowTextA(MSLabel, _ostream_err.str().c_str());
189         constexpr unsigned long _err_import_dll = { 2 };
190         return _err_import_dll;
191     }
192     return 0;
193 }

```