Написание экстра-маленьких Win32 приложений на C++ от 1 КБ

Натыкаясь в Интернете на довольно интересные программы, я часто не решался их закачивать после того, как узнавал их размер. Какую ни возьми - все огромные. Да и ресурсов системных потребляют немало. В этой статье я расскажу о том, как сделать программу в среднем в 10 - 100 раз меньше размером, чем попадаются аналогичные.

Цель

Написать очень быструю и маленькую программу, скрывающую по CTRL+F12 заданные окна. При нажатии комбинации CTRL+F10 она должна показать спрятанные окна. Входные данные:

TXT Файл вида
```
------------
Internet Explorer
The Bat!
Visual C++
911
------------
```

Если будут найдены окна, содержащие в своем заголовке указанные строки, они будут спрятаны.
В вышеуказанном примере будут спрятаны все окна IE, окно Microsoft Visual C++, окно почтовой программы "The Bat!" и все окна, в заголовках которых содержится комбинация символов "911".

Итак, писать будем на чистом Win32 API. Создадим окно, привяжем к нему горячие клавиши. По требованию будем осуществлять перебор видимых окон в системе и в заголовке каждого будем искать заданные комбинации символов.

Опции линкера

Если ничего не предпринимать, то нам не удастся получить в итоге файл менее 32 КБ(примерно). Поэтому пишем:

```
#pragma comment(linker,"/MERGE:.rdata=.text")
#pragma comment(linker,"/FILEALIGN:512  /SECTION:.text,EWRX
 /IGNORE:4078")
#pragma comment(linker,"/ENTRY:New_WinMain")
#pragma comment(linker,"/NODEFAULTLIB")
```

На что теперь стоит обратить особое внимание? Обычно точка входа в

программу выглядит так:

int WINAPI WinMain(HINSTANCE hInst,HINSTANCE hPrevInst,LPSTR szCmdLine,int nCmdShow)

(кстати, для Win32 приложений второй параметр всегда NULL)

Но(!)... Так как мы отключили "Runtime library", нам теперь передается в этих параметрах разный мусор. Поэтому называем точку входа не WinMain а New_WinMain, которую объявим, как void New_WinMain(void), чтобы не забыть о том, что нам ничего не передается. А параметр HINSTANCE получаем функцией GetModuleHandle(NULL). Ах да, и выходить из программы будем функцией ExitProcess.

Теперь если собрать нашу пустую программку, которая ничего делать не будет, размер ее будет 1 Кб. Но нам нужно еще дописать 3 Кб кода. Продолжим.

Чтобы все дальнейшее было понятно даже новичку в программировании под Windows, я прокомментирую все.

Объявим кое-какие константы

Это понадобится для регистрации "горячих" клавиш функцией RegisterHotKey.

```
#define HOTKEYHIDE 1
#define HOTKEYSHOW 2
```

Размер буффера, куда будет считываться заголовок окна функцией GetWindowText.

```
#define SSZZ 256
```

Размер буфера, куда будет считываться файл со стоками фильтрации (используется в объявлении char FilterStrings[MAXFIL];)

```
#define MAXFIL 1024
```

(Примечание: При желании можно сделать и выделение памяти динамически - найти файл, узнать его размер и выделить блок. Приблизительный пример:

```
// ..................
WIN32_FIND_DATA FindData;
HANDLE hFind=FindFirstFile(szFilterStringsFile,&FindData);
if (hFind!=INVALID_HANDLE_VALUE)
```

```
{
 i=(FindData.nFileSizeHigh  * MAXDWORD) + FindData.nFileSizeLow;
 HGLOBAL hGA=GlobalAlloc(GMEM_ZEROINIT|GMEM_MOVEABLE,i+1);
 // (+ end-ZERO)
 if (hGA!=NULL)
 {
 LPVOID lpStrings=GlobalLock(hGA);
 DWORD dw;
 if (lpStrings!=NULL)  ReadFile(hFile,lpStrings,i,&dw,NULL);
 }

 }
 FindClose(hFind);
 CloseHandle(hFile);
// ..........................
// Но так как вряд ли файл настроек у нас будет больше одного
// килобайта, я оставил статичный массив.
)
```

Массив хендлов окон (вряд ли будет у нас более 300 окон)

```
HWND aHwnd[300];
```

Кол-во инициализированных элементов в этом массиве

```
unsigned int cHwnd=0;
```

Дескрипторы окон - главное и два дочерних - кнопка "Hide" и кнопка "Edit filter strings"

```
HWND hwndMain,  hwndButtonHide,  hwndButtonEditFilter;
```

Тут будет что-то типа "c:\programs\winhider\winhider.settings.txt"

```
char szFilterStringsFile[MAX_PATH]="(c)2002  KMiNT21";
```

Соответственно, хендл файла с именем "что-то типа"

```
HANDLE  hFile;
```

А это место, куда будем считывать все из этого файла

```
char FilterStrings[MAXFIL];
```

Функции

Обработка сообщений главного окна

LRESULT CALLBACK MainWndProc(HWND hwnd,UINT msg,WPARAM wParam,LPARAM lParam);

Функция, которая будет вызываться для каждого окна при переборе всех окон

static BOOL FAR PASCAL my_EnumWindowsProc(HWND hwnd, DWORD lParam);

Проверка наличия строки str2 в str1

BOOL Contain(char* str1, char* str2);

Скрывание с экрана очередного окна

inline void HideNext(HWND hwnd){
ShowWindow(aHwnd[cHwnd++]=hwnd,SW_HIDE); }

Возврат всех спрятанных окон на экран

inline void ShowAll(void) { while(cHwnd) ShowWindow(aHwnd[--cHwnd],SW_SHOW);}

Пройдемся по главным строкам функции NewWinMain

* Получим INSTANCE модуля. Это нам нужно для регистрации оконного класса

HINSTANCE hInst=GetModuleHandle(NULL);

* Зарегистрируем оконный класс

```
WNDCLASS wc;
wc.style = CS_HREDRAW|CS_VREDRAW ;
wc.lpfnWndProc = (WNDPROC)MainWndProc;
wc.hInstance = hInst;
wc.hbrBackground = (HBRUSH)(COLOR_WINDOW);
wc.lpszClassName = "CKMINT21WINDOWSHIDERPRO";
wc.hCursor = LoadCursor(NULL,IDC_ARROW);
wc.hIcon = LoadIcon(NULL,IDI_APPLICATION);
wc.lpszMenuName=NULL;
wc.cbClsExtra=0;
wc.cbWndExtra=0;
```

```
if (!RegisterClass(&wc)) MessageBox(0,"I can't register window
class.","Error:",0), ExitProcess(0);
```

* Создаем главное окно приложения

```
hwndMain=CreateWindow(wc.lpszClassName,"Small windows hider!",
WS_BORDER|WS_CAPTION|WS_SYSMENU|WS_MINIMIZEBOX,
CW_USEDEFAULT,0,291,180, NULL, NULL, hInst, NULL);
```

И помещаем на него две кнопки. Как видим, кнопки имеют класс "BUTTON". Они являются дочерними окну hwndMain.

```
hwndButtonHide=CreateWindow("BUTTON","Hide!", WS_VISIBLE |
WS_CHILD ,
10,10,261,90, hwndMain, NULL, hInst, NULL);
ShowWindow(hwndButtonHide,SW_SHOW), UpdateWindow(hwndButtonHide);
hwndButtonEditFilter=CreateWindow("BUTTON","Edit filters",
WS_VISIBLE|WS_CHILD|WS_BORDER|WS_TABSTOP ,
10,110,261,30, hwndMain, NULL, hInst, NULL);
ShowWindow(hwndButtonEditFilter,SW_SHOW),
UpdateWindow(hwndButtonEditFilter);
```

Наконец, показываем главное окно

```
ShowWindow(hwndMain,SW_SHOW), UpdateWindow(hwndMain);
```

Примечание: Так как кто-то этого может не знать, хочу отметить, что в языке C++ есть "операция следования" - запятая. Т.е. просто последовательно выполнятся обе функции ShowWindow и UpdateWindow (как отдельный блок). В вышеуказанной строке можно было бы и просто поставить ";", а вообще иногда это помогает избавиться от огромного количества фигурных скобок {}, в тексте программы.

* Затем регистрируем в системе HotKeys. Они будут привязаны к главному окну, которому будут передавается сообщения WM_HOTKEY.

```
RegisterHotKey(hwndMain,HOTKEYHIDE,MOD_CONTROL,VK_F12)
RegisterHotKey(hwndMain,HOTKEYSHOW,MOD_CONTROL,VK_F10)
```

* Затем считываем настройки из файла и запускаем главный цикл обработки оконных сообщений для текущего процесса.

```
MSG msg;
while(GetMessage(&msg,NULL,0,0)) TranslateMessage(&msg),
DispatchMessage(&msg);
```

Оконная процедура

```
// Тут все довольно стандартно. Делаем switch (msg).
// ...
case WM_HOTKEY:
 if (HOTKEYSHOW == (int)wParam)
 // показываем все, что мы до этого прятали, а так же главное
 // окно программы
 ShowAll(), ShowWindow(hwnd,SW_SHOW);

 if (HOTKEYHIDE == (int)wParam)
 // Скрываем наше главное окно и запускаем перебор всех окон в
 // системе - EnumWindows. Теперь будет вызываться функция
 // my_EnumWindowsProc для каждого обнаруженного в системе окна.
 ShowWindow(hwnd,SW_HIDE), EnumWindows((int (__stdcall *)(struct
 HWND__ *,long))my_EnumWindowsProc, 0);
 break;
// ...

// Если программу пытаются минимизировать, просто скрываем ее
// .....
 case WM_SYSCOMMAND:
 if(SC_MINIMIZE == wParam) { ShowWindow(hwnd,SW_HIDE); return 0; }
 break;
 // Внимание, после ShowWindow(hwnd,SW_HIDE) мы пишем return 0,
 // вместо break. Почему? Да потому что не хотим, чтобы это
 // сообщение пошло дальше в систему. Мы его уже обработали
 // по-своему.
// ...
// А затем обрабатываем нажатия на кнопки.
 case BN_CLICKED:
 if (hwndButtonHide==(HWND)lParam)ShowWindow(hwndMain,SW_HIDE);
 if (hwndButtonEditFilter==(HWND)lParam)ShellExecute(NULL,"open",
 szFilterStringsFile,NULL,NULL,SW_SHOWMAXIMIZED);
 break;
```

Рассмотрим функцию my_EnumWindowsProc

Пропустим все невидимые окна

```
if (!IsWindowVisible(hwnd)) return TRUE;
```

Получим TITLE очередного окна

```
GetWindowText(hwnd, szWindowsTitle, SSZZ)
```

Затем перебираем все стоки из файла настроек

```
for(i=0;i<MAXFIL;i++)
if (FilterStrings[i])  // если это начало строки, то
{
if (Contain(szWindowsTitle, FilterStrings+i))  HideNext(hwnd);
// скроем окно, если эта строка содержится в szWindowsTitle
while(FilterStrings[i])  i++;
// сместим указатель на следующий 0
}
```

Продолжаем дальнейший перебор окон

```
return TRUE;
```

(Если бы было return FALSE, перебор бы закончился.)
В остальных функциях особо описывать нечего.

FAQ, возникший в результате множества заданных мне вопросов.

Q: Почему программа не линкуется?

A: Попробуйте собрать не debug, а release версию. А если вам нужна возможность отладки, воспользуйтесь обычными #define. И все-таки есть еще одни вариант. В отладочной версии линкер не может собрать файл потому, что не находит "__chkesp", которая содержится в "CHKESP.OBJ". Что мы можем сделать? Да взять и заменить тот obj на свой, который будет меньше размером и не будет содержать ненужный нам код.

Q: Как теперь получить переданную командную строку?

A: Ну тут все просто. Пользуйтесь стандартными API. То же самое и для Instance приложения. Вот они - GetCommandLine, GetModuleHandle.

Q: А какой минимальный align возможен?

A: Для того, чтобы ваша программа запускалась нормально в любой версии Windows, используйте 512 байт.

Q: А можно ли делать такими маленькими DLL?

A: Да. Назначьте свою точку входа вместо _DllMainCRTStartup.

Q: А почему пропали функции strcmp, strlen и т.п.?

A: Так как они были реализованы в RTL, теперь вы не можете их использовать. Но это не беда. В модуле kernel есть отличная замена этим

функциям. Названия те же, но с буквой "l" вначале. Например - lstrlen, lstrcmp, lstrcat.

Q: А теперь стали недоступны функции работы с памятью - memset, CopyMemory?

A: RTL сам предоставляет интефейс для работы с памятью. Во-первых, чтобы соблюдать снандарт, во-вторых, чтобы упростить работу с памятью в среде Win32. Вот посмотрите на функцию CopyMemory - она на самом деле не является настоящей API функцией. Попробуйте слинковать проект без RTL, в котором используется эта функция. Результат - неудачная попытка линковки - ссылка на _memcpy. Еще один пример - функция new. В среде Win32 вы должы воспользоваться функциями GlobalAlloc, GlobalLoc и т.п. Однако вы можете просто заменить RTL функции своими. В файле add.txt вы можете взять уже готовые функции, если не хотите писать их сами.