Základy DLL

Dynamicky linkované knižnice (DLL) sú základným kameňom Microsoft Windows od prvej verzii systému. Všetky funkcie Windows API sú obsiahnuté v DLL súboroch. Tri najdôležitejšie sú Kernel-32.dll, ktorý obsahuje funkcie pre správu pamäte, procesov a vlákien; User32.dll, ktorý obsahuje funkcie pre úlohy užívateľského rozhrania; a GDI32.dll, ktorý obsahuje funkcie pre kreslenie grafických obrázkov a zobrazovanie textu.

Niekoľko dôvodov, prečo vytvárať a používať DLL súbory:

- Rozširujú vlastnosti aplikácií.
- Môžu byť napísané v rôznych programovacích jazykoch.
- Uľahčujú riadenie projektu.
- Pomáhajú chrániť pamäť.
- Uľahčujú zdieľanie zdrojov.
- Uľahčujú lokalizáciu.
- Pomáhajú riešiť platformové odlišnosti.
- Môžu slúžiť k špeciálnym účelom.

DLL súbory a adresový priestor procesu

Často je jednoduchšie vytvoriť DLL súbor než aplikáciu. DLL súbor je totižto súbor modulov obsahujúcich zdrojový kód a každý z týchto modulov obsahuje množinu funkcií, ktoré môže volať nejaký EXE, prípadne iný DLL súbor. Po skompilovaní súborov so zdrojovým kódom, sú tieto súbory linkované pomocou linkeru tak, ako by bol linkovaný spustiteľný súbor. Pre DLL súbor však musí byť špecifikovaný /DLL prepínač linkeru. Tento prepínač spôsobí, že linker umiestni do výsledného obrazu DLL súboru informácie, ktoré umožnia operačnému systému rozoznať obraz súboru ako DLL a nie EXE súbor.

Skôr než aplikácia (prípadne iný DLL súbor) môže volať funkcie obsiahnuté v DLL, musí byť obraz DLL súboru namapovaný do adresového priestoru volaného procesu. To môže byť uskutočnené buď implicitným *load-time* linkovaním, alebo explicitným *run-time* linkovaním.

Keď už je DLL súbor namapovaný do adresového priestoru volajúceho procesu, jeho funkcie sú dostupné vláknam bežiacim v rámci tohto procesu. DLL súbor v podstate stráca svoju identitu ako DLL: vlákna procesu berú DLL kód a dáta len ako dodatočný kód a dáta obsiahnuté v adresovom priestore procesu. Keď vlákno volá DLL funkciu, DLL funkcia nazrie do zásobníku vlákna, aby získala odovzdané parametre a takisto použije zásobník vlákna pre lokálne premenné, ktoré potrebuje. Dokonca aj objekty vytvorené kódom DLL funkcie sú vlastnené volajúcim vláknom, príp. procesom.

Poznámka:

Ak vaše DLL obsahuje funkciu, ktorá alokuje pamäť, malo by obsahovať aj funkciu, ktorá túto pamäť uvoľní.

Implicitné *load-time* linkovanie EXE a DLL modulov

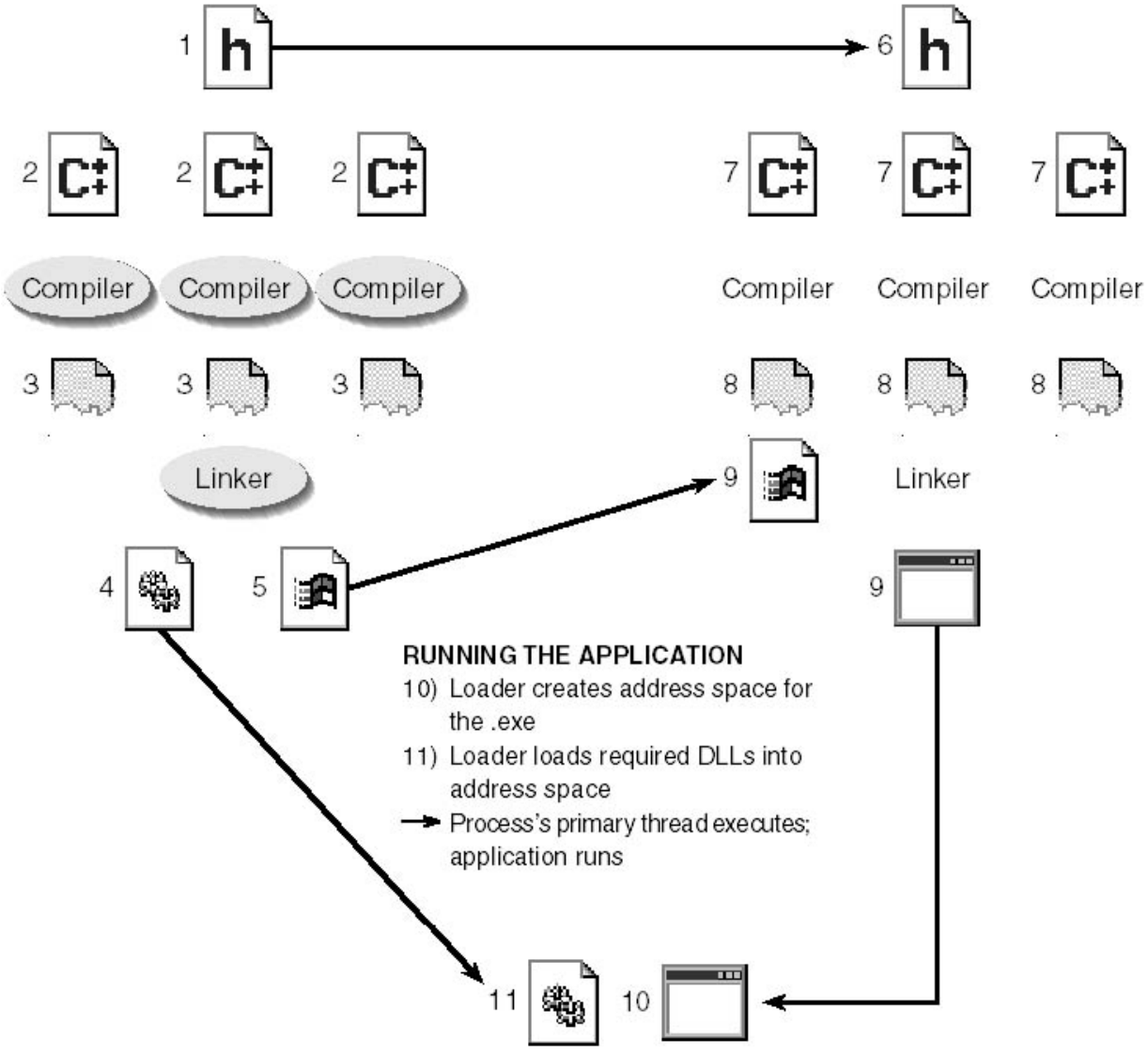
Implicitné *load-time* linkovanie EXE a DLL modulov prebieha nasledovne:

BUILDING THE DLL

1) Header with *exported* prototypes/structures/ symbols
2) C/C++ source files implementing exported functions/variables
3) Compiler produces .obj file for each C/C++ source file
4) Linker combines .obj module producing DLL
5) Linker also produces .lib file if at least one function/variable is exported

BUILDING THE EXE

6) Header with *imported* prototypes/structures/ symbols
7) C/C++ source files referencing imported functions/variables
8) Compiler produces .obj file for each C/C++ source file
9) Linker combines .obj modules resolving references to imported functions/variables using .lib file producing .exe (containing import table-list of required DLLs and imported symbols)



Pre vytvorenie spustiteľného modulu, ktorý importuje funkcie a pramenné z DLL modulu, je nutné vytvoriť najskôr DLL modul. Vytvorenie DLL modulu zahŕňa nasledujúce kroky:

1. Najskôr musíte vytvoriť hlavičkový súbor, ktorý bude obsahovať prototypy funkcií, štruktúry a symboly, ktoré chcete exportovať z DLL súboru. Tento hlavičkový súbor je zahrnutý vo všetkých moduloch obsahujúcich zdrojový kód DLL súboru, aby pomohol pri jeho stavbe. Tento hlavičkový súbor bude potrebný aj pri vytváraní EXE súboru, ktorý bude používať DLL funkcie.

2. Vytvorte modul zdrojového kódu, ktorý bude implementovať funkcie a premenné, ktoré chcete vo vašom DLL module.

3. Pri zostavení DLL modulu kompilátor spracuje každý modul zdrojového kódu zvlášť a ku každému vytvorí .obj modul.

4. Po vytvorení všetkých .obj modulov, linker skombinuje ich obsah a vytvorí obraz jedného DLL súboru. Tento súbor je vyžadovaný pri stustení EXE súboru.

5. Pokiaľ linker zistí, že zdrojový kód DLL súboru exportuje aspoň jednu funkciu alebo premennú, vyprodukuje sa aj .lib súbor. Tento .lib súbor je malý, nakoľko neobsahuje žiadne funkcie alebo premenné. Obsahuje len zoznam exportovaných funkcií a premenných. Tento súbor je vyžadovaný pri spustení EXE súboru.

Po zostavení DLL modulu, môžete vytvoriť EXE modul, a to nasledujúcimi krokmi:

6. Do všetkých modulov zdrojového kódu odkazujúcich sa na funkcie, premenné, dátové štruktúry alebo symboly z DLL modulu musíte zahrnúť DLL hlavičkový súbor.

7. Vytvorte moduly zdrojového kódu spustiteľného súboru. V kóde sa môžete odkazovať na funkcie a premenné definované v DLL hlavičkovom súbore.

8. Pri zostavení EXE modulu kompilátor spracuje každý modul zdrojového kódu zvlášť a ku každému vytvorí .obj modul.

9. Po vytvorení všetkých .obj modulov, linker skombinuje ich obsah a vytvorí obraz jedného EXE súboru. Tento súbor obsahuje aj import sekciu, v ktorej je zoznam názvov DLL modulov vyžadovaných EXE modulom. Okrem toho je pre každé DLL zo zoznamu určené, na ktoré funkcie a premenné sa odkazuje kód EXE súboru.

V okamihu, keď sú zostavené DLL aj EXE moduly, je možné spustiť aplikáciu. Pri pokuse o spustenie loader operačného systému postupuje nasledovne:

10. Loader vytvorí virtuálny adresový priestor pre proces. EXE modul je namapovaný do tohto adresového priestoru. Loader analyzuje import sekciu EXE modulu. Pre každý názov DLL v zozname lokalizuje DLL modul a namapuje ho do daného adresového priestoru. Nakoľko DLL môže importovať funkcie a premenné iných DLL súborov, môže obsahovať vlastnú import sekciu. Pre plnú inicializáciu procesu loader analyzuje každú import sekciu každého modulu a postupne namapuje všetky DLL súbory.

Akonáhle sú EXE a všetky DLL súbory namapované do adresového priestoru procesu, začne sa vykonávať primárne vlákno procesu a aplikácia beží.

Zostavenie DLL modulu

Hlavičkový súbor DLL modulu by mal mať nasledujúci formát:

```
/********************************************************************
Module:  MyLib.h
********************************************************************/


#ifdef MYLIBAPI

// MYLIBAPI should be defined in all of the DLL's source
// code modules before this header file is included.

// All functions/variables are being exported.

#else

// This header file is included by an EXE source code module.
// Indicate that all functions/variables are being imported.
#define MYLIBAPI extern "C" __declspec(dllimport)

#endif

////////////////////////////////////////////////////////////////////////

// Define any data structures and symbols here.

////////////////////////////////////////////////////////////////////////

// Define exported variables here. (NOTE: Avoid exporting variables.)
MYLIBAPI int g_nResult;

////////////////////////////////////////////////////////////////////////

// Define exported function prototypes here.
MYLIBAPI int Add(int nLeft, int nRight);

////////////////////////////// End of File /////////////////////////////////
```

Každý súbor so zdrojovým kódom v DLL by mal zahŕňať daný hlavičkový súbor nasledovným spôsobom:

```
/********************************************************************
Module:  MyLibFile1.cpp
********************************************************************/


// Include the standard Windows and C-Runtime header files here.
#include <windows.h>


// This DLL source code file exports functions and variables.
```

```
#define MYLIBAPI extern "C" __declspec(dllexport)

// Include the exported data structures, symbols, functions, and variables.
#include "MyLib.h"

////////////////////////////////////////////////////////////////////////

// Place the code for this DLL source code file here.
int g_nResult;


int Add(int nLeft, int nRight) {
   g_nResult = nLeft + nRight;
   return(g_nResult);
}

////////////////////////////// End of File ///////////////////////////////
```

Pri kompilovaní zdrojového kódu daného DLL súboru je definované `MYLIBAPI` s použitím *`__declspec(dllexport)`* skôr, než je priložený hlavičkový súbor MyLib.h. Ak kompilátor pred funkciou alebo premennou vidí *`__declspec(dllexport)`*, vie, že daná funkcia (premenná) bude exportovaná z výsledného DLL súboru.

Nakoľko je symbol `MYLIBAPI` v hlavičkovom súbore umiestnený pred exportované funkcie (premenné), nie je už nevyhnutné dávať ho pred funkcie aj v zdrojovom kóde – kompilátor si pamätá, ktoré funkcie sú exportované z hlavičkového súboru.

Použitie hlavičkového súboru v EXE module je odlišné. Pred priložením hlavičkového súboru tu nesmie byť definované `MYLIBAPI`. To spôsobí, že `MYLIBAPI` bude definované až v rámci hlavičkového súboru a to pomocou *`__declspec(dllimport)`*. Kompilátor tak bude vidieť, že zdrojový kód EXE modulu importuje funkcie a premenné z DLL modulu.

Exportovanie

Jediná skutočne zaujímavá vec z predchádzajúcej časti je použitie modifikátora *`__declspec(dllexport)`*. Keď prakladač vidí tento modifikátor pred premennou, prototypom funkcie, alebo C++ triedou, vloží do výsledného .obj súboru dodatočné informácie. Linker analzyuje tieto informácie pri linkovaní všetkých .obj súborov do DLL.

Keď sa zistia tieto dodatočné informácie pri linkovaní DLL súboru, linker vytvorí .lib súbor, ktorý bude obsahovať zoznam symbolov exportovaných DLL súborom. Tento .lib súbor je nutné potom prilinkovať ku každému modulu, ktorý sa odkazuje na dané exportované symboly. Popri vytvorení .lib súboru linker vloží do výsledného DLL súboru aj tabuľku exportovaných symbolov *(export section)* a relatívnu virtuálnu adresu (RVA) určujúcu, kde v DLL module môže byť daný symbol nájdený.

Zostavenie EXE modulu

Nasledujúca časť kódu ukazuje zdrojový kód aplikácie, ktorá importuje exportované symboly z DLL modulu a odkazuje na tieto symboly v kóde.

```
/*************************************************************************
Module:  MyExeFile1.cpp
*************************************************************************/
```

```
// Include the standard Windows and C-Runtime header files here.
#include <windows.h>


// Include the exported data structures, symbols, functions, and variables.
#include "MyLib\MyLib.h"

//////////////////////////////////////////////////////////////////////////

int WINAPI WinMain(HINSTANCE hinstExe, HINSTANCE, LPTSTR pszCmdLine, int) {

   int nLeft = 10, nRight = 25;

   TCHAR sz[100];
   wsprintf(sz, TEXT("%d + %d = %d"), nLeft, nRight, Add(nLeft, nRight));

   wsprintf(sz, TEXT("The result from the last Add is: %d"), g_nResult);
   return(0);
}

////////////////////////////// End of File ///////////////////////////////
```

Pri tvorení zdrojového kódu EXE súboru musíte priložiť hlavičkový súbor DLL modulu. Bez neho by importované symboly neboli definované a prekladač by hlásil chybu.

Zdrojový kód spustiteľného súboru by nemal definovať symbol `MYLIBAPI` skôr než hlavičkový súbor DLL modulu. Pri kompilácií je potom tento symbol definovaný ako *`__declspec(dllimport)`* podľa hlavičkového súboru MyLib.h. Keď prekladač vidí tento modifikátor, vie, že daný symbol má byť importovaný z nejakého DLL súboru – nezáleží mu na tom, z akého. Keď potom linker linkuje .obj moduly do výsledného spustiteľného súboru, musí rozhodnúť, ktoré DLL súbory obsahujú importované symboly. Na to musíte linkeru odovzdať .lib súbor DLL modulu.

Importovanie

Prekladač sa dozvie, ktoré symboly budú importované z nejakého .lib súboru DLL modulu na základe modifikátora *`__declspec(dllimport)`*. Keď potom linker pracuje s importovanými symbolmi, vloží tzv. *imports section* do výsledného EXE modulu. Táto sekcia obsahuje zoznam požadovaných DLL modulov a symbolov, na ktoré sa zdrojový kód odkazuje.

Explicitné *run-time* linkovanie EXE a DLL modulov

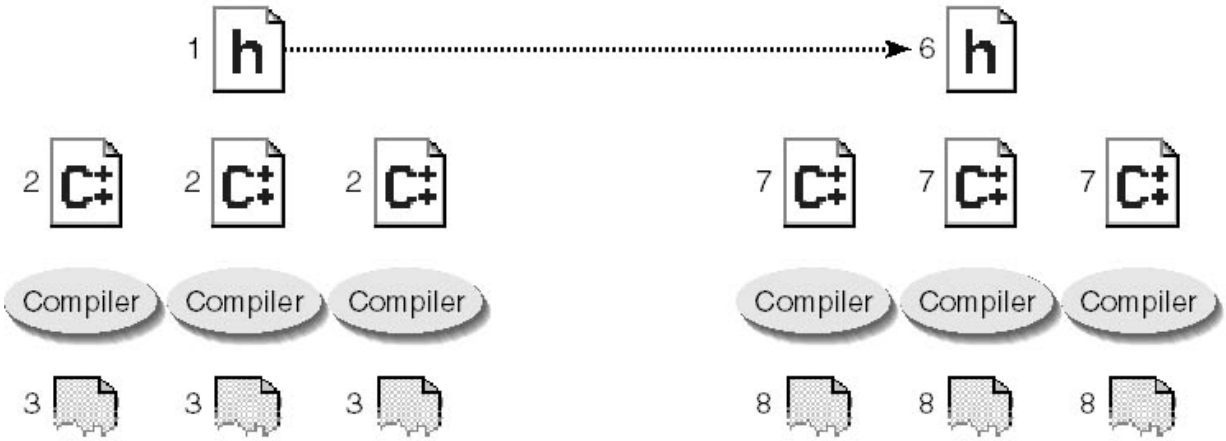
Explicitné linkovanie v podstate znamená, že vlákno aplikácie explicitne načíta požadovaný DLL modul počas behu aplikácie. Po načítaní DLL modulu do adresového priestoru procesu získa virtuálnu adresu funkcie obsiahnutej v DLL súbore a potom pomocou nej zavolá požadovanú funkciu. Priebeh je popísaný v nasledujúcej schéme:

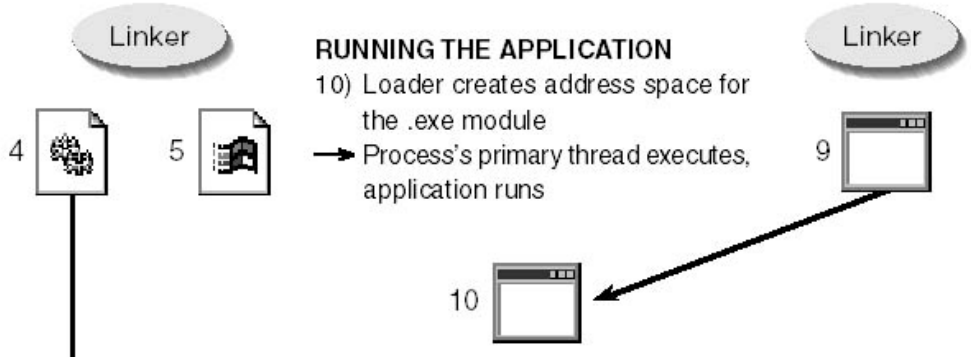
BUILDING THE DLL

1) Header with *exported* prototypes/structures/ symbols
2) C/C++ source files implementing exported functions/variables
3) Compiler produces .obj file for each C/C++ source file
4) Linker combines .obj modules producing DLL
5) Linker also produces .lib file if at least 1 function/variable is exported
   NOTE: This .lib file is not used for explicit llinking

BUILDING THE EXE

6) Header with *imported* prototypes/structures/ symbols (optional)
7) C/C++ source files that DO NOT reference inported functions/variables
8) Compiler produces .obj file for each C/C++ source file
9) Linker combines .obj modules producing .exe module
   NOTE: DLL's .lib file is not needed since there are no direct references to exported symbols. The .exe file does not contain an import table



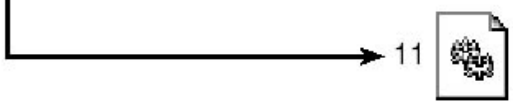


RUNNING THE APPLICATION

10) Loader creates address space for the .exe module
→ Process's primary thread executes, application runs

EXPLICITLY LOADING THE DLL

11) A thread calls *LoadLibrary*(*Ex*) to load the DLL into the process's address space
→ Threads may now call *GetProcAddress* to indirectly reference the DLL's exported symbols



Vlákno procesu sa kedykoľvek môže rozhodnúť namapovať DLL súbor do adresového priestoru procesu a to volaním jednej z nasledujúcich funkcií:

```
HINSTANCE LoadLibrary(PCTSTR dllPathName);

HINSTANCE LoadLibraryEx(
    PCTSTR dllPathName,
    HANDLE hFile,
    DWORD flags
    );
```

Obe tieto funkcie lokalizujú obraz DLL súboru v systéme a pokúsia sa ho namapovať do adresového priestoru procesu.Vrátená hodnota `HINSTANCE` indentifikuje adresu virtuálnej pamäte, kde je súbor namapovaný. Pokiaľ nie je možné DLL súbor namapovať, funkcie vrátia `NULL`.

Pokiaľ už vlákno ďalej nepotrebuje symboly z namapovaného DLL súboru, malo by ho uvoľniť, a to volaním funkcie *`FreeLibrary`*:

```
BOOL FreeLibrary(HINSTANCE hinstDll);
```

Ak sa vlákno v tom istom procese pokúsi dvakrát načítať tú istú DLL knižnicu, neznamená to, žeby sa táto knižnica mapovala do adresového priestoru procesu dvakrát. Namapovaná je len raz, ale zvýši sa jej *usage count*. Knižnica bude odmapovaná z adresového priestoru až ked jej *usage count* číslo klesne na 0 – inými slovami, funkciu *`FreeLibrary`* musíte volať toľkokrát, koľkokrát ste pre daný DLL súbor volali funkciu *`LoadLibrary(Ex)`*.

Vlákno môže zistiť, či už daný DLL modul bol namapovaný do adresového priestoru procesu volaním funkcie *`GetModuleHandle`*:

```
HINSTANCE GetModuleHandle(PCTSTR moduleName);
```

Rovnako je možné aj zistiť plnú cestu k DLL modulu:

```
DWORD GetModuleFileName(
    HINSTANCE hinstModule,
    PTSTR pathName,
    DWORD pathSize
    );
```

V okamihu, keď je explicitne načítaný DLL súbor, vlákno musí získať adresu symbolu, na ktorý sa chce odkázať, volaním nasledujúcej funkcie:

```
FARPROC GetProcAddress(
    HINSTANCE hinstDll,
    PCSTR symbolName
    );
```

Parameter *`hinstDll`* je *handle* DLL súboru, ktorý obsahuje požadovaný symbol. Parameter *`symbol-Name`* môže mať dve rôzne formy. Prvou je adresa nulou ukončeného reťazca obsahujúca meno symbolu, ktorého adresu chcete:

```
FARPROC symbolAddress = GetProcessAddress(hinstDll, “SomeFuncInDll”);
```

Všimnite si, že parameter *symbolName* je typu PCSTR, čo znamená, že funkcia *GetProcAddress* akceptuje výhradne ANSI reťazce – nikdy nepoužívajte Unicode reťazce, pretože prekladač/linker ukladajú mená symbolov do export sekcie DLL modulu ako ANSI reťazce.

Ďalšia forma parametru *symbolName* určuje ordinálne číslo symbolu, ktorého adresu chcete:

```
FARPROC symbolAddress = GetProcAddress(hinstDll, MAKEINTRESOURCE(2));
```

Tento spôsob predpokladá, že viete, že požadovanému symbolu bola pridelená ordinálna hodnota 2. Túto formu zápisu sa však neodporúča používať.

Obe metódy poskytnú adresu žiadaného symbolu z daného DLL modulu. Ak požadovaný symbol v export sekcii DLL modulu neexistuje, funkcia *GetProcAddress* vráti NULL.

Funkcia vstupného bodu DLL súboru

DLL súbor môže obsahovať jednu funkciu vstupného bodu. Volanie tejto funkcie je informatívne a zvyčajne je využívané DLL súborom pre nejakú inicializáciu alebo upratanie procesu/vlákna. Pokiaľ váš DLL súbor túto notifikáciu nevyžaduje, nemusíte funkciu vstupného bodu do svojho kódu vôbec zahrnúť. Pokiaľ notifikáciu prijímať chcete, funkciu vstupného bodu implemtujte:

```
BOOL WINAPI DllMain(HINSTANCE hinstDll, DWORD reason, PVOID impLoad)
{
    switch (reason)
    {
        case DLL_PROCESS_ATTACH:
            // The DLL is being mapped into the process's address space.
            break;

        case DLL_THREAD_ATTACH:
            // A thread is being created.
            break;

        case DLL_THREAD_DETACH:
            // A thread is existing cleanly.
            break;

        case DLL_PROCESS_DETACH:
            // The DLL is being unmapped from the process's address space.
            break;
    }
}
```

Pozor na názov funkcie vstupného bodu – je *case-sensitive*. Parameter *hinstDll* obsahuje *instance handle* DLL súboru, ktorá identifikuje adresu virtuálnej pamäte, kde je namapovaný obraz DLL súboru. Parameter *impLoad* má nenulovú hodnotu, pokiaľ je DLL súbor načítaný implicitne a 0, pokiaľ je načítaný explicitne.

Parameter reason určuje, prečo systém zavolal túto funkciu. Tento parameter nadobúda jednu z týchto štyroch hodnôt: DLL_PROCESS_ATTACH, DLL_PROCESS_DETACH, DLL_THREAD_ATTACH, DLL_THREAD_DETACH.

Poznámka:
Pamätajte, že DLL súbory volajú funkciu DllMain, aby sa inicializovali. Inicializácia by mala byť čo najjednoduchšia – vytvorenie objektov jadra, otvorenie súborov, ... V žiadnom prípade by nemala obsahovať pokus o volanie funkcií LoadLibrary(Ex), príp. FreeLibrary!!

DLL_PROCESS_ATTACH

Keď je DLL súbor prvýkrát mapovaný do adresového priestoru procesu, systém zavolá jeho *DllMain* funkciu s hodnotou DLL_PROCESS_ATTACH ako *reason* parametrom. Toto sa deje iba pri prvom namapovaní. Pokiaľ je DLL súbor už namapovaný, s každým volaním funkcie *LoadLibrary(Ex)* sa už len zvýši jeho *usage count*.

Po obdržaní tejto notifikácie by mala funkcia previesť inicializáciu súvisiacu s procesom, ku ktorému bol DLL modul namapovaný. Po úspešnej inicializácií by funkcia mala vrátiť hodnotu TRUE, inak FALSE. Notifikácia DLL_PROCESS_ATTACH je jedinou zo všetkých 4 notifikácií, pri ktorej systém neignoruje návratovú hodnotu *DllMain* funkcie.

Pri implicitnom načítaní sa o vykonanie *DllMain* funkcie stará primárne vlákno procesu skôr, než začne vykonávať kód samotnej spustiteľnej aplikácie. Pri explicitnom načítaní sa o vykonanie tejto funkcie stará vlákno, ktoré zavolalo funkciu *LoadLibrary(Ex)*.

DLL_PROCESS_DETACH

S notifikáciou DLL_PROCESS_DETACH je funkcia *DllMain* volaná v okamihu, keď je DLL súbor odmapovávaný z adresového priestoru procesu. Funkcia by v tomto okamihu mala upratať proces.

Ak sa jedná o odmapovanie pri ukončení procesu (*ExitProcess*), o *DllMain* funkciu sa stará zvyčajne primárne vlákno procesu po návrate z funkcie vstupného bodu procesu. Pokiaľ sa DLL súbor odmapováva po volaní funkcie *FreeLibrary*, stará sa o *DllMain* funkciu vlákno, ktoré túto funkciu zavolalo.

Pokiaľ je proces ukončený tým, že nejaké vlákno v systéme zavolalo funkciu *TerminateProcess*, systém nevolá funkciu *DllMain*. To znamená, že namapovaný DLL súbor nemá príležitosť upratať a môže dôjsť ku strate dát.

DLL_THREAD_ATTACH

Keď je v procese vytvorené vlákno, systém preskúma všetky DLL súbory namapované do adresového priestoru tohto procesu a zavolá ich *DllMain* funkcie s notifikáciou DLL_THREAD_ATTACH. To DLL súborom indikuje, že majú previesť inicializáciu vlákna. Za vykonanie všetkých *DllMain* funkcií je zodpovedné novovytvorené vlákno. Až po tom, ako sú vykonané všetky *DllMain* funkcie, môže vlákno vykonávať svoj vlastný kód.

DLL_THREAD_DETACH

Preferovaný spôsob ukončenia vlákna je návrat funkcie vlákna. To spôsobí, že systém zavolá funkciu *ExitThread*. Systém však dané vlákno nezabije hneď – najskôr ho donúti zavolať *DllMain* funkcie všetkých namapovaných DLL súborov s notifikáciou DLL_THREAD_DETACH. Tieto funkcie by následne mali vlákno upratať. Vlákno je živé, až kým sa nevrátia všetky volané *DllMain* funkcie.

Poznámka:
Volania funkcie DllMain sú sériové! Pokiaľ sa funkcia práve vykonáva, každé vlákno, ktoré sa pokúsi ju vtedy zavolať je pozastavené, až kým sa funkcia nevráti.

Použitá literatúra:

RICHTER, Jeffrey. *Programming Applications for Microsoft Windows*. 4. vyd. 1999. ISBN 1-57231-996-8