DLL CREATION IN MINGW

I've always considered DLLs to be esoterically cool stuff - somehow the idea of one program running another makes my imagination run wild. I've now discovered that in many cases DLLs are a bad idea(TM), but for a fledgling programmer, learning how to create a DLL, especially in a free development environment such as MingW would mean an instant familiarity with a lot of development tools, and a lot of the seemingly hidden options of the incredible gcc compiler.

As usual, we're diving in. You should have a copy of mingw handy. Get the MingW installer from the Current branch in the Mingw download page, that sets you off easy. You should also be familiar with IDE-less coding, or should know enough to set up an IDE to work with Mingw, I'm not going to cover that here.

What are DLLs?

First off, what are DLLs? DLLs are dynamically linked libraries. How are they different from static libraries? In static libraries, the linking is done at compile time: all the library functions are combined with the main fragment of program code to create the executable. When the linking is done at runtime, it is called dynamic linking. Since the linking is done at runtime, it is obvious that the operating system will have something to do with it. That is why most DLL implementations are non-portable.

When a compiled executable that references a DLL is loaded, the OS looks into the file and sees that the executable references a set of "imports" from a DLL file. This is simply a situation equivalent to finding that "This program uses the following functions which are contained in this dll". The OS then looks into the particular DLL. The DLL has a corresponding and matching set of export functions that the OS then maps from the functions referenced in the main executable to the functions in the DLL. Thus, when the executable calls a referenced function, the code in the DLL is executed. Viola! Dynamic linking!

Hello DLL!

I'm now going to describe a standard "Hello world" implementation. The code is in three files: hello.c, dll.h and dll.c. The code is listed and explained below:

hello.c

```
#include <stdio.h>
#include "dll.h"

int main () {
        hello();
        return 0;
}
```

Hello.c is a standard hello world C program except that the hello() function is going to be dynamically linked. The only special thing here is the inclusion of dll.h.

dll.h

```
#ifdef BUILD_DLL
/* DLL export */
#define EXPORT __declspec(dllexport)
#else
/* EXE import */
#define EXPORT __declspec(dllimport)
#endif

EXPORT void hello(void);
```

DLL.h is where most of the magic happens. It begins with checking the BUILD_DLL macro. We manually set this macro when building so that the macro EXPORT is set to *__declspec(dllexport)*, so that gcc can build the dll. When the OS calls up the dll from the executable, BUILD_DLL is not set and therefore EXPORT is set to *__declspec(dllimport)* which is a nice set of macro routines to expose our function to the calling scope.

Note that *__declspec(dllexport)* and *__declspec(dllimport)* are mingw macros to faciliate DLL creation. They are mapped to their equivalent WinAPI headers.

dll.c

TOPICS

» Algorithms
» BSD
» Computing
» Culture
» Humor
» Linux
» Programming
» Science
» Software
» Web

RECENT BLOG POSTS

Algo Puzzle - Product of Elements of an Array
Unlimited Concurrent Remote Desktop Connections
Subversion: Branching and Committing
A Nice C++ Linked List
Super Cavitation - The New Generation Speed Research
Inventor: W. Daniel Hillis
The Two Schools of Cryptography
Threads Cannot Be Implemented as a Library
UNIX Swapping
Introduction to the Xen Virtual Machine

more

```
#include
#include "dll.h"

EXPORT void hello(void) {
        printf ("Hello\n");
}
```

This is the actual code of the hello world routine. There should be nothing special here.

Compiling and Linking the files

DLL creation used to be a tiresome process. Recent advances in gcc and mingw engines have meant that it takes only four steps now to create a dll. They are:

1. Creating the object code for hello.c

   ```
   gcc -c hello.c
   ```

2. Creating the object code for the dll

   ```
   gcc -c -DBUILD_DLL dll.c
   ```

   Notice the use of the -D param by which we set the macro BUILD_DLL. It is used for setting export to *__declspec(dllexport)* so that compilation can take place.

3. Creating the dll

   ```
   gcc -shared -o message.dll dll.o -Wl,--out-implib,libmessage.a
   ```

   The third step requires more explanation.
   The -shared parameter is used for creating a shared libaray; in Win platform it is a dll.
   -Wl means wait for next message to the linker.
   --out-implib is a param to the linker ld which tells it to create an import library which is used by programs which want to link to your dll.
   We are ignoring the definitions file which every well-behaved dll should export. If you follow the step above, GCC will automatically create the definition. For most cases, it will work, but if you want to optimize stuff this is the place to look.

4. Creating the executable

   ```
   gcc -o hello.exe hello.o message.dll
   ```

   The fourth step is actually a bit of gcc magick. The actual step is something like this:

   ```
   gcc -o hello.exe hello.o -L./ -lmessage
   ```

   The -L param means that the linker checks the ./ path for imports
   -lmessage (or -l message) means to search for the message linker name. It extracts this from message.dll

Once you've finished these steps, run the program!

```
C:\>hello
Hello!
```

Wait, that's not dynamic!

You're right, that's not *really* dynamic. Of course, you're loading the function at run-time, but what is the purpose of that if you need to reference the dll at compile-time? One of the reasons why a dll is used is to enable a plugin architecture for your program. Other people can write code that your program can use, and you can't really anticipate other people's dlls, and if you are going to recompile your program every time a new dll comes along, then why use dynamic linking at all?

Solve this by using two functions from the Windows API: LoadLibrary() and GetProcAddress(). We modify the codes for the main module like this:

hello.c

```
#include <windows.h>
#include <stdio.h>

int main () {

        /*Typedef the hello function*/
        typedef void (*pfunc)();

        /*Windows handle*/
        HANDLE hdll;

        /*A pointer to a function*/
        pfunc hello;

        /*LoadLibrary*/
        hdll = LoadLibrary("message.dll");

        /*GetProcAddress*/
        hello = (pfunc)GetProcAddress(hdll, "hello");

        /*Call the function*/
        hello();
        return 0;
}
```

The code should be self explanatory, there are just some things that you should remember:

1. Don't forget to include windows.h
2. The syntax for LoadLibrary is:

   ```
   handle = LoadLibrary("path to dll file");
   ```

   handle is <= HINSTANCE_ERROR if there is an error loading the dll.
3. The syntax for GetProcAddress is:

   ```
   <pointer to function> = (<pointer to function type>)GetProcAddress(<dll handle>, "<function name inside dll>");
   ```

   We can also use global variables inside the dll that are prefixed with export as a valid resource name, it needn't be a function. That is,

   ```
   <variable> = (<type>)GetProcAddress(<dll handle>, "<global variable name inside dll>");
   ```

   is also valid.
4. For further error info at any step in the process, call GetLastError()

To compile it, don't change a thing for the dlls. For compiling hello.c, a simple...

```
gcc -o hello.exe hello.c
```

...would do.

More information

This doesn't solve the problem of implementing a plugin arch, but we can work towards it. One of the ways in which this is done is to reference a function with a unique name from every dll called. For example, a program searches for all dll files beginning with a particular header in the file name and loads it via LoadLibrary. For example, a music program called foobar might look at all dll files which begin with foo. Thus foo_looks.dll will be loaded, while gym.dll won't be. Then inside the dll, it searches for a particular resource say "play", and loads and runs it.

An implementation of the above method, branched off from sortalg, can be found here.

By vishnu on 2004-08-12 Login to post comments

ohkqxsqgnb by Anonymous (not verified)
vhgSbfyxniHwqJvPCP by swzepzqelw (not verified)
umHOoEoIQrMBx by fqrvzitputi (not verified)
xhnFQogLkeBuDG by mayoutdd (not verified)
sex for free free sex video by Drugan888
free swinger exotic by Drugan888
free full length porn by Drugan888
all free porn with email by Drugan888
free porn pics free teen by Drugan888

Comment viewing options

Threaded list - collapsed · Date - oldest first · 300 comments per page · Save settings
Select your preferred way to display the comments and click "Save settings" to activate your changes.