Windows 和 Unix 下动态链接库的区别

最近慢慢将开发环境转向了 freebsd ，渐渐的发现了许多东西跟 Windows 下不太一样。我指的是，跟我以前想当然的理解不太一样。比如，unix 下对动态链接库 so 的处理，和 Windows 下的 DLL 就不太相同。

之前，我对 Windows 下更为了解一些。早几年做 Windows 开发，老是为动态链接，静态链接这些问题纠缠不清；慢慢的才有了比较清晰的理解，现在基本上不会为这类问题困绕了。前段时间，碰到一些朋友写 lua 的扩展库(windows 版本)时遇到的几个 bug ，就是因为链接问题导致的。公司内部的一个项目，在服务器程序上也碰到了错误链接 lua 引起的 bug ，在内部 maillist 上争论了好久。当时，作为 windows 程序员，跟同事中的 unix 程序员争论看似相同的问题时，却老说不到一起去；今天才发现，原来是相互之间对动态链接库的理解不同导致的。

今天，写下本文，或许可以让以后跨平台开发的朋友少走点弯路。

动态链接库在 unix 下，习惯以 .so 为文件名结尾（通常还以 lib 开头）。而 Windows 下是以 .DLL 为文件后缀。Windows 在处理 dll 上还有一些细节容易被人忽略，我曾经为这个[写过一篇 Blog](http://blog.codingnow.com/2005/10/windows_dll.html)。

如果需要运行时主动加载一个动态链接库，windows 下可以使用 LoadLibrary 这个 kernel API (在 kernel32.dll 中)；unix 下是用 dlopen 。Windows 下找到 dll 中导出符号的地址，可以用 GetProcAddress ，而 unix 也有对应的 api ...

这些相互对应的 api ，似乎预示着对等的功能，但事实上是有区别的。

DLL 事实上和 EXE 文件一样，同属 PE 格式的执行文件。对于隐式的引用外部符号，需要把外部符号所在的位置写在 PE 头上。PE 加载器将从 PE 头上找到依赖的符号表，并加载依赖的其它 DLL 文件。

但是，unix 上并非如此，so 文件大多为 elf 执行文件格式。当它们需要的外部符号，可以不写明这些符号所在的位置。也就是说，通常 so 文件本身并不知道它依赖的那些符号在哪些 so 里面。这些符号是由调用 dlopen 的进程运行时提供的。而 unix 下的执行文件本身会暴露自己静态链接的符号，（可以是自己本身实现的，或者是从静态库 .a 文件里链入的）。dlopen 将把这些符号通报给 dlopen 加载的 .so 文件，最终完成动态链接。（事实上 dlopen 还可以指定 mode ，完成更复杂的操作）

因为这个区别，unix 下的 lua 解释器可以完全静态链接所有的 lua api ；我们为 lua 扩展的库，以 so 的形式存在被运行时加载不会有任何隐患。而 Windows 下，必须生成一个 luacore 的 DLL 文件，由 lua 解释器于扩展库共享 lua api (还包括 crt 的实现) 才不会出问题。

也因为这个区别，VC 下才会有让 windows 开发新手困惑不解的动态链接 CRT ，静态链接 CRT ，多线程库，单线程库，等等的选项。没点点 windows 开发功力的人，多少都要在上面栽几个跟头。

从动态链接库的这个设计上来看，我个人感觉，Windows 弄的真是糟糕透顶。尤其是对开发者来说是这样。至少我们在 windows 下做一个 dll 文件给大家使用还需要携带一个 .lib 文件；而 unix 下一般只需要有相应的头文件就够了。对于编写新的 .so ，找不到的符号可以就让它在那里，直到最终执行文件来把所有需要的符号联合到一起。windows 可以存在一个 dll 对另一个 dll 的隐式依赖；而 unix 下一般不需要让 so 和 so 有隐式依赖关系。这让我们全局替换类似 CRT 的东西变的困难许多；而以我自己的编程经验来看，好处却没有多少。

顺便一提的是 unix 下需要用 ldconfig 来管理动态库，这比 windows 下 copy DLL 到当前目录下就可以用的方式，无疑提高了系统的安全性。