# 不同模块中分别进行malloc和free

一旦DLL的文件映像被映射到调用进程的地址空间中，DLL的函数就可以供进程中运行的所有线程使用。实际上，DLL几乎将失去它作为DLL的全部特征。对于进程中的线程来说，DLL的代码和数据看上去就像恰巧是在进程的地址空间中的额外代码和数据一样。当一个线程调用DLL函数时，该DLL函数要查看线程的堆栈，以便检索它传递的参数，并将线程的堆栈用于它需要的任何局部变量。此外， <b> DLL中函数的代码创建的任何对象均由调用线程所拥有，而DLL本身从来不拥有任何东西 </b> 。

　　如你所知，可执行文件的全局变量和静态变量不能被同一个可执行文件的多个运行实例共享。Windows98能够确保这一点，方法是在可执行文件被映射到进程的地址空间时为可执行文件的全局变量和静态变量分配相应的存储器。Windows2000确保这一点的方法是使用第13章介绍的写入时拷贝（copy-on-write）机制。DLL中的全局变量和静态变量的处理方法是完全相同的。当一个进程将DLL的映像文件映射到它的地址空间中去时，系统将同时创建全局数据变量和静态数据变量的实例。

　　注意   必须注意的是，单个地址空间是由一个可执行模块和若干个DLL模块组成的。这些模块中，有些可以链接到静态版本的C/C++运行期库，有些可以链接到一个DLL版本的C/C++运行期库，而有些模块（如果不是用C/C++编写   
的话）则根本不需要C/C++运行期库。许多开发人员经常会犯一个常见的错误，因为他们忘记了若干个C/C++运行期库可以存在于单个地址空间中。请看下面的代码：

VOID EXEFunc(){ PVOID pv = DLLFunc(); free(pv);}

PVOID DLLFunc(){return(malloc(100)); }

　　那么你是怎么看待这个问题的呢？上面这个代码能够正确运行吗？DLL函数分配的内存块是由EXE的函数释放的吗？答案是可能的。上面显示的代码并没有为你提供足够的信息。如果EXE和DLL都链接到DLL的C/C++运行期库，那么上面的代码将能够很好地运行。 <b> 但是，如果两个模块中的一个或者两个都链接到静态C/C++运行期库，那么对free函数的调用就会失败。 </b> 我经常看到编程人员编写这样的代码，结果都失败了。     
　　有一个很方便的方法可以解决这个问题。当一个模块提供一个用于分配内存块的函数时，该模块也必须提供释放内存的函数。让我们将上面的代码改写成下面的样子：

VOID EXEFunc(){ PVOID pv = DLLFunc(); DLLFreeFunc(pv);}

PVOID DLLFunc(){ PVOID pv = malloc(100); return(pv);}

BOOL DLLFreeFunc(PVOID pv){ return(free(pv));}

　　这个代码是正确的，它始终都能正确地运行。当你编写一个模块时，不要忘记其他模块中的函数也许没有使用C/C++来编写，因此可能无法使用malloc和free函数进行内存的分配。应该注意不要在代码中使用这些假设条件。

另外，在内部调用malloc和free函数时，这个原则对于C++的new和delete操作符也是适用的。

我的问题：   
1.“如果两个模块中的一个或者两个都链接到静态C/C++运行期库，那么对free函数的调用就会失败”，为什么？   静态CRT   library和动态CRT   library有什么区别？   
2.   “DLL中函数的代码创建的任何对象均由调用线程所拥有，而DLL本身从来不拥有任何东西”，   DLL中的全局变量怎么解释？由多个应用程序使用该DLL时，DLL中的全局变量需要加上同步机制吗？

1.因为   malloc/free,   new/delete   都是调用   HeapAlloc/HeapFree   来实现来实现内存分配是释放的。查看   Windows   的   API   可以看到，这两个函数都需要一个   Heap   的   HANDLE   做为参数。CRT   库采用了全局变量来保存这个   HANDLE。如果是静态链接，   CRT   库的代码会链接到各个   dll   中去，也包括这个全局变量。也就是说，每个静态链接的   dll   都有一个自己的全局堆句柄，他们自己都在这个句柄上使用内存。当一个   dll   释放另外一个   dll   分配的内存时由于使用的堆句柄不一致于是出错。当使用动态链接时，有于每个   dll   都是去调用   CRT   库的   dll   函数来分配和释放内存的，使用的是同一个句柄，所以就没有这个问题。

2.   不同进程之间对   dll   全局变量的访问操作系统可以区分的，平时大家共用一份，当某个进程改了某个全局变量时，操作系统会自动为这个进程生成一份这个变量的拷贝，（copy- on-write）让这个进程访问拷贝的内容而不会影响其它进程中这个变量的值，所以不需要手动做同步。

四.malloc/free   
   这两个函数是使用频率最高的两个函数，由于他们是标准C库中的一部分，所以具有极高的移植性。这里的"移植性"指的是使用他们的代码可以在不同的平台下编译通过，而不同的平台下的C Run-Time Library的具体实现是平台相关的，在Windows平台的C Run-Time Library中的malloc()和free()是通过调用Heap Memory API来实现的。值得注意的是C Run-Time Library拥有独立的Heap对象，我们知道，当一个应用程序初始化的时候，首先被初始化的是C Run-Time Library，然后才是应用程序的入口函数，而Heap对象就是在C Run-Time Library被初始化的时候被创建的。对于动态链接的C Run-Time Library,运行库只被初始化一次，而对于静态连接的运行库，每链接一次就初始化一次，所以对于每个静态链接的运行库都拥有彼此不同的Heap 对象。这样在某种情况下就会出问题，导致程序崩溃,例如一个应用程序调用了多个DLL,除了一个DLL外，其他的DLL，包括应用程序本身动态连接运行库，这样他们就使用同一个Heap对象。而有一个DLL使用静态连接的运行库，它就拥有一个和其他DLL不同的Heap 对象，当在其他DLL中分配的内存在这个DLL中释放时，问题就出现了。