在笔者上一篇文章《内核注册并监控对象回调》介绍了如何运用`ObRegisterCallbacks`注册`进程与线程`回调，并通过该回调实现了`拦截`指定进行运行的效果，本章`LyShark`将带大家继续探索一个新的回调注册函数，`PsSetLoadImageNotifyRoutine`常用于注册`LoadImage`映像监视，当有模块被系统加载时则可以第一时间获取到加载模块信息，需要注意的是该回调函数内无法进行拦截，如需要拦截则需写入返回指令这部分内容将在下一章进行讲解，本章将主要实现对模块的监视功能。

PsSetLoadImageNotifyRoutine和PsRemoveLoadImageNotifyRoutine是Windows操作系统提供的两个内核API函数，用于注册和取消注册LoadImage映像的回调函数。

LoadImage映像回调函数是一种内核回调函数，它可以用于监视和拦截系统中的模块加载事件，例如进程启动时加载的DLL、驱动程序等。当有新的模块被加载时，操作系统会调用注册的LoadImage映像回调函数，并将加载模块的相关信息传递给回调函数。

PsSetLoadImageNotifyRoutine函数用于注册LoadImage映像的回调函数，而PsRemoveLoadImageNotifyRoutine函数则用于取消注册已经注册的回调函数。开发者可以在LoadImage映像回调函数中执行自定义的逻辑，例如记录日志、过滤敏感数据、或者阻止某些操作。

需要注意的是，LoadImage映像回调函数的注册和取消注册必须在内核模式下进行，并且需要开发者有一定的内核开发经验。同时，LoadImage映像回调函数也需要遵守一些约束条件，例如不能阻塞或挂起进程或线程的创建或访问，不能调用一些内核API函数等。

内核监视LoadImage映像回调在安全软件、系统监控和调试工具等领域有着广泛的应用。开发者可以利用这个机制来监视和拦截系统中的模块加载事件，以保护系统安全。

监视模块加载与卸载需要费别使用两个函数，这两个函数的参数传递都是自己的回调地址。

- PsSetLoadImageNotifyRoutine 设置回调
- PsRemoveLoadImageNotifyRoutine 移除回调

此处`MyLySharkLoadImageNotifyRoutine`回调地址必须有三个参数传递组成，其中`FullImageName`代表完整路径，`ModuleStyle`代表模块类型，一般来说`ModuleStyle=0`表示加载SYS驱动，如果`ModuleStyle=1`则表示加载的是`DLL`，最后一个参数`ImageInfo`则是映像的详细参数结构体。

```
VOID MyLySharkLoadImageNotifyRoutine(PUNICODE_STRING FullImageName, HANDLE ModuleStyle,
PIMAGE_INFO ImageInfo)
```

那么如何实现监视映像加载呢，来看如下完整代码片段，首先`PsSetLoadImageNotifyRoutine`注册回调，当有模块被加载则自动执行`MyLySharkLoadImageNotifyRoutine`回调函数，其内部首先判断`ModuleStyle`得出是什么类型的模块，然后再通过`GetDriverEntryByImageBase`拿到当前进程详细参数并打印输出。

```
#include <ntddk.h>
#include <ntimage.h>

// 未导出函数声明
PUCHAR PsGetProcessImageFileName(PEPROCESS pEProcess);

// 获取到镜像装载基地址
PVOID GetDriverEntryByImageBase(PVOID ImageBase)
{
    PIMAGE_DOS_HEADER pDOSHeader;
    PIMAGE_NT_HEADERS64 pNTHeader;
    PVOID pEntryPoint;
```

```
    pDOSHeader = (PIMAGE_DOS_HEADER)ImageBase;
    pNTHeader = (PIMAGE_NT_HEADERS64)((ULONG64)ImageBase + pDOSHeader->e_lfanew);
    pEntryPoint = (PVOID)((ULONG64)ImageBase + pNTHeader->OptionalHeader.AddressOfEntryPoint);
    return pEntryPoint;
}

// 获取当前进程名
UCHAR* GetCurrentProcessName()
{
    PEPROCESS pEProcess = PsGetCurrentProcess();
    if (NULL != pEProcess)
    {
        UCHAR *lpszProcessName = PsGetProcessImageFileName(pEProcess);
        if (NULL != lpszProcessName)
        {
            return lpszProcessName;
        }
    }
    return NULL;
}

// 设置自己的回调函数
VOID MyLySharkLoadImageNotifyRoutine(PUNICODE_STRING FullImageName, HANDLE ModuleStyle, PIMAGE_INFO ImageInfo)
{
    PVOID pDrvEntry;

    // MmIsAddress 验证地址可用性
    if (FullImageName != NULL && MmIsAddressValid(FullImageName))
    {
        // ModuleStyle为零表示加载sys
        if (ModuleStyle == 0)
        {
            // 得到装载主进程名
            UCHAR *load_name = GetCurrentProcessName();
            pDrvEntry = GetDriverEntryByImageBase(ImageInfo->ImageBase);
            DbgPrint("[LyShark SYS加载] 模块名称:%wZ --> 装载基址:%p --> 镜像长度: %d --> 装载主进程: %s \n", FullImageName, pDrvEntry, ImageInfo->ImageSize, load_name);
        }
        // ModuleStyle非零表示加载DLL
        else
        {
            // 得到装载主进程名
            UCHAR *load_name = GetCurrentProcessName();
            pDrvEntry = GetDriverEntryByImageBase(ImageInfo->ImageBase);
            DbgPrint("[LyShark DLL加载] 模块名称:%wZ --> 装载基址:%p --> 镜像长度: %d --> 装载主进程: %s \n", FullImageName, pDrvEntry, ImageInfo->ImageSize, load_name);
        }
    }
}

VOID UnDriver(PDRIVER_OBJECT driver)
```

```
{

PsRemoveLoadImageNotifyRoutine((PLOAD_IMAGE_NOTIFY_ROUTINE)MyLySharkLoadImageNotifyRoutine);
    DbgPrint("[LyShark.com] 驱动卸载完成...");
}

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("hello lyshark.com \n");


PsSetLoadImageNotifyRoutine((PLOAD_IMAGE_NOTIFY_ROUTINE)MyLySharkLoadImageNotifyRoutine);
    DbgPrint("[LyShark.com] 驱动加载完成...");
    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

运行这个驱动程序，则会输出被加载的驱动详细参数。