方亮的专栏

# [原]DllMain中不当操作导致死锁问题的分析——线程中调用GetModuleFileName、GetModuleHandle等导致死锁

2012-11-9 阅读2079 评论0

之前的几篇文章已经讲解了在DllMain中创建并等待线程导致的死锁的原因。是否还记得，我们分析了半天汇编才知道在线程中的死锁位置。如果对于缺乏调试经验的同学来说，可能发现这个位置有点麻烦。那么本文就介绍几个例子，它们会在线程明显的位置死锁掉。（转载请指明出于breaksoftware的csdn博客）

DLL中的代码依旧简单。它获取叫EVENT的命名事件，然后等待这个事件被激活。激活的操作自然放在线程中。这次我们不用在DLL中创建线程，而是在Exe中创建。

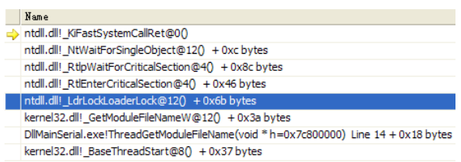
```
switch (ul_reason_for_call)
{
case DLL_PROCESS_ATTACH: {
    printf("DLL DllGetModuleHandle:\tProcess attach (tid = %d)\n", tid);
    HANDLE hEvent = CreateEvent( NULL, FALSE, FALSE, L"EVENT" );
    if ( NULL != hEvent ) {
        WaitForSingleObject(hEvent, INFINITE);
    }
    }break;
```

## 1 线程中调用GetModuleFileName死锁

线程函数是

```
static DWORD WINAPI ThreadGetModuleFileName(LPVOID h) {
    HMODULE hDll = (HMODULE)h;
    WCHAR wszFileName[MAX_PATH] = {0};
    GetModuleFileName( hDll, wszFileName, MAX_PATH );
    HANDLE hEvent = CreateEvent( NULL, FALSE, FALSE, L"EVENT" );
    SetEvent( hEvent );
    return 0;
}
```

死锁后，DLL中的死锁位置和前几篇文章中一样，本文之后均不再说明。我们关注线程的堆栈，它是



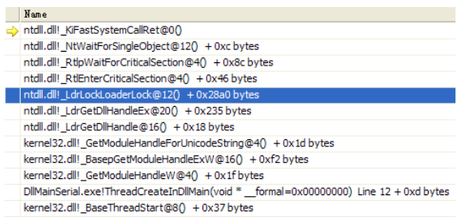
我们看到GetModuleFileName在内部要调用LdrLockLoderLock，以进入PEB的LoaderLock临界区。可是该临界区被主线程占用着（在调用DllMain前进入临界区），主线程还要等待工作线程调用GetModuleFileName后激活事件才退出，于是就死锁了。

### 2 线程中调用**GetModuleHandle**死锁

线程函数是

```
static DWORD WINAPI ThreadGetModuleHandle(LPVOID) {
    Sleep(1000);
    GetModuleHandle( L"DllWithoutDisableThreadLibraryCalls_A.dll" );
    HANDLE hEvent = CreateEvent( NULL, FALSE, FALSE, L"EVENT" );
    SetEvent( hEvent );
    return 0;
}
```

内容我就不说明了，我们直接看线程堆栈。



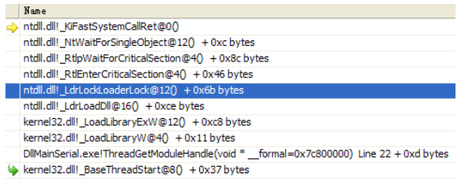

我们看到GetModuleHandleW底层还是进入了加载器函数中。并在加载器函数中进入了LdrLockLoderLock，该函数内部要进入PEB的LoaderLock临界区。可是该临界区被主线程占用着（在调用DllMain前进入临界区），主线程还要等待工作线程调用GetModuleHandle后激活事件才退出，于是就死锁了。

### 3 线程中调用**LoadLibrary**死锁

线程函数

```
static DWORD WINAPI ThreadLoadLibrary(LPVOID) {
    Sleep(1000);
    LoadLibraryW( L"DllWithoutDisableThreadLibraryCalls_A.dll" );
    HANDLE hEvent = CreateEvent( NULL, FALSE, FALSE, L"EVENT" );
    SetEvent( hEvent );
    return 0;
}
```

死锁后线程堆栈



上一篇 下一篇

发表评论

提 交

查看评论

更多评论（0）

回顶部