


```
int __usercall sub_7C936587<eax>(int a1<ebp>, int a2<esi>)\n{\n    LdrpTopLevelDllBeingLoaded = a2;\n    return LdrUnlockLoaderLock(1, *(_DWORD *) (a1 - 572));\n}
```

我们看到在LdrpLoadDll是在临界区中执行的。其实在LdrpLoadDll中也会进入该临界区，但是我们不必关注了。因为只要一次没出临界区就可以满足死锁的条件了。

我们再看下卸载DLL时发生的进入临界区场景，请看堆栈

Line	Language
line21.0: #pragma omp parallel	
line21.1: n12.0: #pragma omp for	
line21.2: n12.1: #pragma omp for	
line21.3: n12.2: #pragma omp for	
line21.4: n12.3: #pragma omp for	
line21.5: n12.4: #pragma omp for	
line21.6: n12.5: #pragma omp for	
line21.7: n12.6: #pragma omp for	
line21.8: n12.7: #pragma omp for	
line21.9: n12.8: #pragma omp for	
line21.10: n12.9: #pragma omp for	
line21.11: n12.10: #pragma omp for	
line21.12: n12.11: #pragma omp for	
line21.13: n12.12: #pragma omp for	
line21.14: n12.13: #pragma omp for	
line21.15: n12.14: #pragma omp for	
line21.16: n12.15: #pragma omp for	
line21.17: n12.16: #pragma omp for	
line21.18: n12.17: #pragma omp for	
line21.19: n12.18: #pragma omp for	
line21.20: n12.19: #pragma omp for	
line21.21: n12.20: #pragma omp for	
line21.22: n12.21: #pragma omp for	
line21.23: n12.22: #pragma omp for	
line21.24: n12.23: #pragma omp for	
line21.25: n12.24: #pragma omp for	
line21.26: n12.25: #pragma omp for	
line21.27: n12.26: #pragma omp for	
line21.28: n12.27: #pragma omp for	
line21.29: n12.28: #pragma omp for	
line21.30: n12.29: #pragma omp for	
line21.31: n12.30: #pragma omp for	
line21.32: n12.31: #pragma omp for	
line21.33: n12.32: #pragma omp for	
line21.34: n12.33: #pragma omp for	
line21.35: n12.34: #pragma omp for	
line21.36: n12.35: #pragma omp for	
line21.37: n12.36: #pragma omp for	
line21.38: n12.37: #pragma omp for	
line21.39: n12.38: #pragma omp for	
line21.40: n12.39: #pragma omp for	
line21.41: n12.40: #pragma omp for	
line21.42: n12.41: #pragma omp for	
line21.43: n12.42: #pragma omp for	
line21.44: n12.43: #pragma omp for	
line21.45: n12.44: #pragma omp for	
line21.46: n12.45: #pragma omp for	
line21.47: n12.46: #pragma omp for	
line21.48: n12.47: #pragma omp for	
line21.49: n12.48: #pragma omp for	
line21.50: n12.49: #pragma omp for	
line21.51: n12.50: #pragma omp for	
line21.52: n12.51: #pragma omp for	
line21.53: n12.52: #pragma omp for	
line21.54: n12.53: #pragma omp for	
line21.55: n12.54: #pragma omp for	
line21.56: n12.55: #pragma omp for	
line21.57: n12.56: #pragma omp for	
line21.58: n12.57: #pragma omp for	
line21.59: n12.58: #pragma omp for	
line21.60: n12.59: #pragma omp for	
line21.61: n12.60: #pragma omp for	
line21.62: n12.61: #pragma omp for	
line21.63: n12.62: #pragma omp for	
line21.64: n12.63: #pragma omp for	
line21.65: n12.64: #pragma omp for	
line21.66: n12.65: #pragma omp for	
line21.67: n12.66: #pragma omp for	
line21.68: n12.67: #pragma omp for	
line21.69: n12.68: #pragma omp for	
line21.70: n12.69: #pragma omp for	
line21.71: n12.70: #pragma omp for	
line21.72: n12.71: #pragma omp for	
line21.73: n12.72: #pragma omp for	
line21.74: n12.73: #pragma omp for	
line21.75: n12.74: #pragma omp for	
line21.76: n12.75: #pragma omp for	
line21.77: n12.76: #pragma omp for	
line21.78: n12.77: #pragma omp for	
line21.79: n12.78: #pragma omp for	
line21.80: n12.79: #pragma omp for	
line21.81: n12.80: #pragma omp for	
line21.82: n12.81: #pragma omp for	
line21.83: n12.82: #pragma omp for	
line21.84: n12.83: #pragma omp for	
line21.85: n12.84: #pragma omp for	
line21.86: n12.85: #pragma omp for	
line21.87: n12.86: #pragma omp for	
line21.88: n12.87: #pragma omp for	
line21.89: n12.88: #pragma omp for	
line21.90: n12.89: #pragma omp for	
line21.91: n12.90: #pragma omp for	
line21.92: n12.91: #pragma omp for	
line21.93: n12.92: #pragma omp for	
line21.94: n12.93: #pragma omp for	
line21.95: n12.94: #pragma omp for	
line21.96: n12.95: #pragma omp for	
line21.97: n12.96: #pragma omp for	
line21.98: n12.97: #pragma omp for	
line21.99: n12.98: #pragma omp for	
line21.100: n12.99: #pragma omp for	
line21.101: n12.100: #pragma omp for	
line21.102: n12.101: #pragma omp for	
line21.103: n12.102: #pragma omp for	
line21.104: n12.103: #pragma omp for	
line21.105: n12.104: #pragma omp for	
line21.106: n12.105: #pragma omp for	
line21.107: n12.106: #pragma omp for	
line21.108: n12.107: #pragma omp for	
line21.109: n12.108: #pragma omp for	
line21.110: n12.109: #pragma omp for	
line21.111: n12.110: #pragma omp for	
line21.112: n12.111: #pragma omp for	</

我们将关注FreeLibrary和LdrpCallInitRoutine之间的代码逻辑。我们用IDA查看LdrUnLoadDll

```

int __stdcall LdrUnloadDll(int a1)
{
    .....
    v73 = 0;
    v70 = *(_DWORD *) (*MK_FP(__FS__, 24) + 48);
    v71 = 0;
    ms_exc.disabled = 0;
    if ( !LdrpInLdrInit )
        RtlEnterCriticalSection(&LdrpLoaderLock);
    ++LdrpActiveUnloadCount;
    if ( !LdrpShutdownInProgress )
    {
        if ( LdrpCheckForLoadedDllHandle(a1, (int)&v78) )
        {
            if ( *(_WORD *) (v78 + 56) != -1 )
            {
                .....
                if ( (unsigned __int8)LdrpActiveUnloadCount <= 1u )
                {
                    .....
                    v15 = (int *)LdrpUnloadHead;
                    v77 = (int *)LdrpUnloadHead;
                    while ( v15 != &LdrpUnloadHead )
                    {
                        .....
                        LdrpCallInitRoutine((int ( __stdcall *) (_DWORD, _DWORD, _DWORD))v20, *(_DWORD *) (v78 + 24), 0, 0);
                    }
                }
            }
        }
    }
}

```

```

        .....
        v15 = (int *)LdrpUnloadHead;
        v77 = (int *)LdrpUnloadHead;
        ms_exc.disabled = 0;
        v3 = 0;
    }
    .....
}
}
else
{
    v71 = 0xC0000135u;
}
}
ms_exc.disabled = -1;
sub_7C937424();
.....
return v71;
}

int __cdecl sub_7C937424()
{
    int result; // eax@3

    --LdrpActiveUnloadCount;
    if ( !LdrpInLdrInit )
        result = RtlLeaveCriticalSection(&LdrpLoaderLock);
    return result;
}

```

我们看到**LdrUnloadDll**几乎所有操作都是在临界区执行的。

以上两段从源码级证明了加载和卸载**DLL**导致的**DllMain**的调用（以及不调用）都是在临界区中完成的。

[上一篇](#)
[下一篇](#)

发表评论

提交

查看评论


2楼 [Breaksoftware](#) 2014-06-04 13:34

[reply]qweqweqwexdd3[/reply] 谢谢，多多交流。

1楼 [qweqweqwexdd3](#) 2014-06-02 20:09

lz的这几篇文章写得很好啊，对调试的理解很深。应该是花了很多时间研究的吧。看完整个系列再翻翻还有木有好文章

更多评论（2）

 回顶部