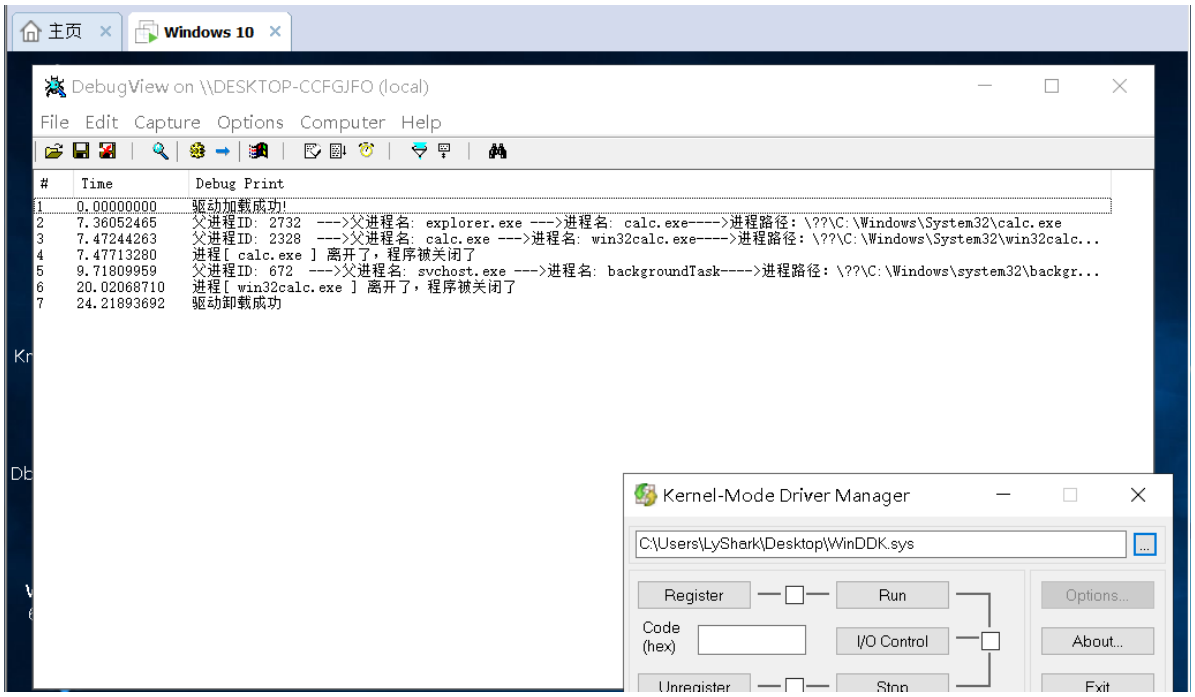
监控进程的启动与退出可以使用 `PsSetCreateProcessNotifyRoutineEx` 来创建回调，当新进程产生时，回调函数会被率先执行，然后执行我们自己的`MyCreateProcessNotifyEx`函数，并在内部进行打印输出。

```
#include <ntddk.h>

NTKERNELAPI PCHAR PsGetProcessImageFileName(PEPROCESS Process);
NTKERNELAPI NTSTATUS PsLookupProcessByProcessId(HANDLE ProcessId, PEPROCESS
*Process);

PCHAR GetProcessNameByProcessId(HANDLE ProcessId)
{
    NTSTATUS st = STATUS_UNSUCCESSFUL;
    PEPROCESS ProcessObj = NULL;
    PCHAR string = NULL;
    st = PsLookupProcessByProcessId(ProcessId, &ProcessObj);
    if (NT_SUCCESS(st))
    {
        string = PsGetProcessImageFileName(ProcessObj);
        ObfDereferenceObject(ProcessObj);
    }
    return string;
}

VOID MyCreateProcessNotifyEx(PEPROCESS Process, HANDLE ProcessId,
PPS_CREATE_NOTIFY_INFO CreateInfo)
{
    char ProcName[16] = { 0 };
    if (CreateInfo != NULL)
    {
        strcpy(ProcName, PsGetProcessImageFileName(Process));
        DbgPrint("父进程ID: %ld  --->父进程名: %s --->进程名: %s---->进程路径: %wZ",
CreateInfo->ParentProcessId,
            GetProcessNameByProcessId(CreateInfo->ParentProcessId),
            PsGetProcessImageFileName(Process),CreateInfo->ImageFileName);
    }
    else
    {
        strcpy(ProcName, PsGetProcessImageFileName(Process));
        DbgPrint("进程[ %s ] 离开了，程序被关闭了",ProcName);
    }
}

VOID UnDriver(PDRIVER_OBJECT driver)
{

PsSetCreateProcessNotifyRoutineEx((PCREATE_PROCESS_NOTIFY_ROUTINE_EX)MyCreatePro
cessNotifyEx, TRUE);
}
NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    NTSTATUS status;
    status =
PsSetCreateProcessNotifyRoutineEx((PCREATE_PROCESS_NOTIFY_ROUTINE_EX)MyCreatePro
cessNotifyEx, FALSE);
    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

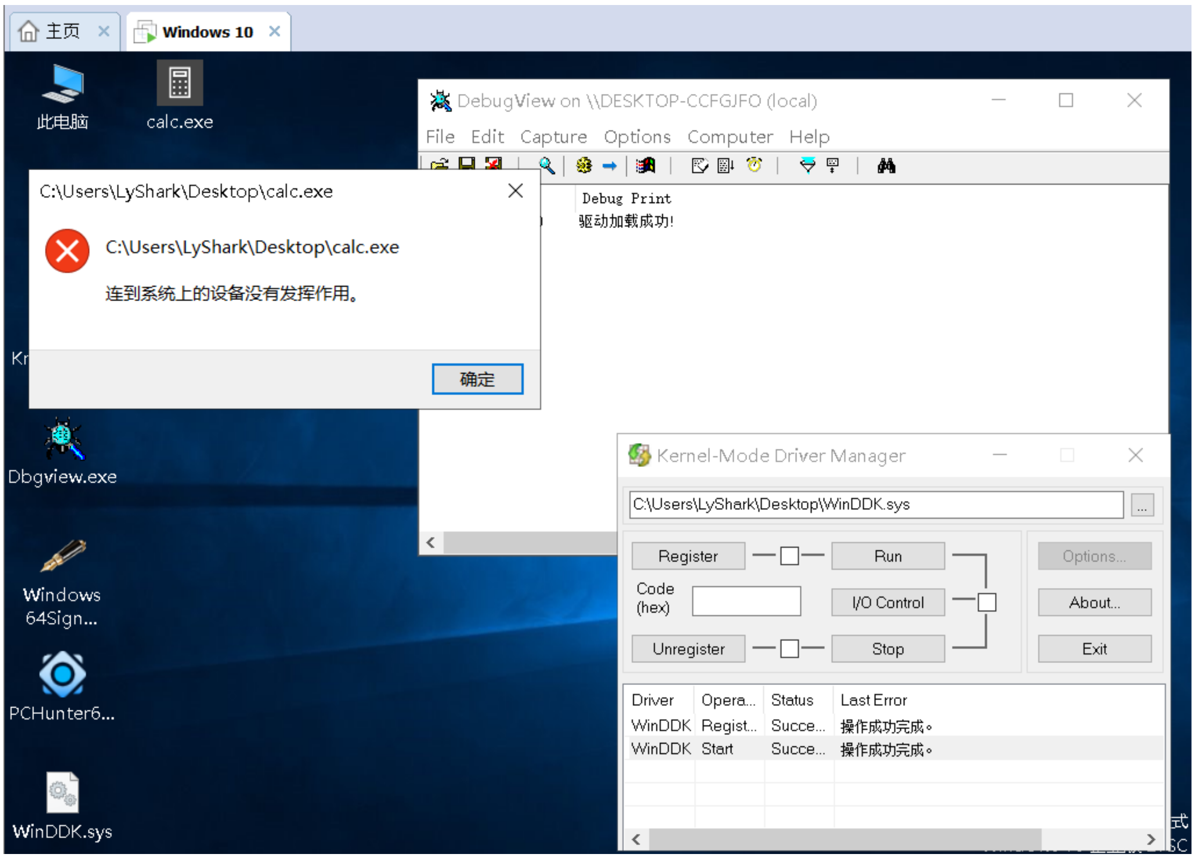

主页
Windows 10
DebugView on \\DESKTOP-CCFGJFO (local)
File Edit Capture Options Computer Help
Kernel-Mode Driver Manager
C:\Users\LyShark\Desktop\WinDDK.sys
Register
Run
Options...
Code (hex)
I/O Control
About...
Unregister
Stop
Exit

```
#   Time          Debug Print
1   0.00000000    驱动加载成功!
2   7.36052465    父进程ID: 2732  --->父进程名: explorer.exe --->进程名: calc.exe---->进程路径: \??\C:\Windows\System32\calc.exe
3   7.47244263    父进程ID: 2328  --->父进程名: calc.exe --->进程名: win32calc.exe---->进程路径: \??\C:\Windows\System32\win32calc...
4   7.47713280    进程[ calc.exe ] 离开了，程序被关闭了
5   9.71809959    父进程ID: 672  --->父进程名: svchost.exe --->进程名: backgroundTask---->进程路径: \??\C:\Windows\system32\backgr...
6   20.02068710   进程[ win32calc.exe ] 离开了，程序被关闭了
7   24.21893692   驱动卸载成功
```

在上方代码基础上进行一定的改进，思路：通过`PsGetProcessImageFileName`即将PID转换为进程名，然后通过`_stricmp`对比，如果发现是`calc.exe`进程则拒绝执行，禁止特定服务的运行，实现代码如下：

```
#include <ntddk.h>

NTKERNELAPI PCHAR PsGetProcessImageFileName(PEPROCESS Process);
NTKERNELAPI NTSTATUS PsLookupProcessByProcessId(HANDLE ProcessId, PEPROCESS
*Process);

PCHAR GetProcessNameByProcessId(HANDLE ProcessId)
{
    NTSTATUS st = STATUS_UNSUCCESSFUL;
    PEPROCESS ProcessObj = NULL;
    PCHAR string = NULL;
    st = PsLookupProcessByProcessId(ProcessId, &ProcessObj);
    if (NT_SUCCESS(st))
    {
        string = PsGetProcessImageFileName(ProcessObj);
        ObfDereferenceObject(ProcessObj);
    }
    return string;
}

VOID MyCreateProcessNotifyEx(PEPROCESS Process, HANDLE ProcessId,
PPS_CREATE_NOTIFY_INFO CreateInfo)
{
    char ProcName[16] = { 0 };
    if (CreateInfo != NULL)
    {
        strcpy(ProcName, PsGetProcessImageFileName(Process));
        if (!_stricmp(ProcName, "calc.exe"))
        {
            CreateInfo->CreationStatus = STATUS_UNSUCCESSFUL;
        }
    }
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    PsSetCreateProcessNotifyRoutineEx((PCREATE_PROCESS_NOTIFY_ROUTINE_EX)MyCreateProcessNotifyEx, TRUE);
    DbgPrint(("驱动卸载成功"));
}
NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    NTSTATUS status;
    status = PsSetCreateProcessNotifyRoutineEx((PCREATE_PROCESS_NOTIFY_ROUTINE_EX)MyCreateProcessNotifyEx, FALSE);
    Driver->DriverUnload = UnDriver;
    DbgPrint("驱动加载成功!");
    return STATUS_SUCCESS;
}
```

将上方代码编译，当我们加载驱动程序以后，再次打开`C:\Windows\System32\calc.exe`计算器进程则提示无法打开，我们的驱动已经成功的拦截了本次的请求。



C:\Users\LyShark\Desktop\calc.exe

C:\Users\LyShark\Desktop\calc.exe

连到系统上的设备没有发挥作用。

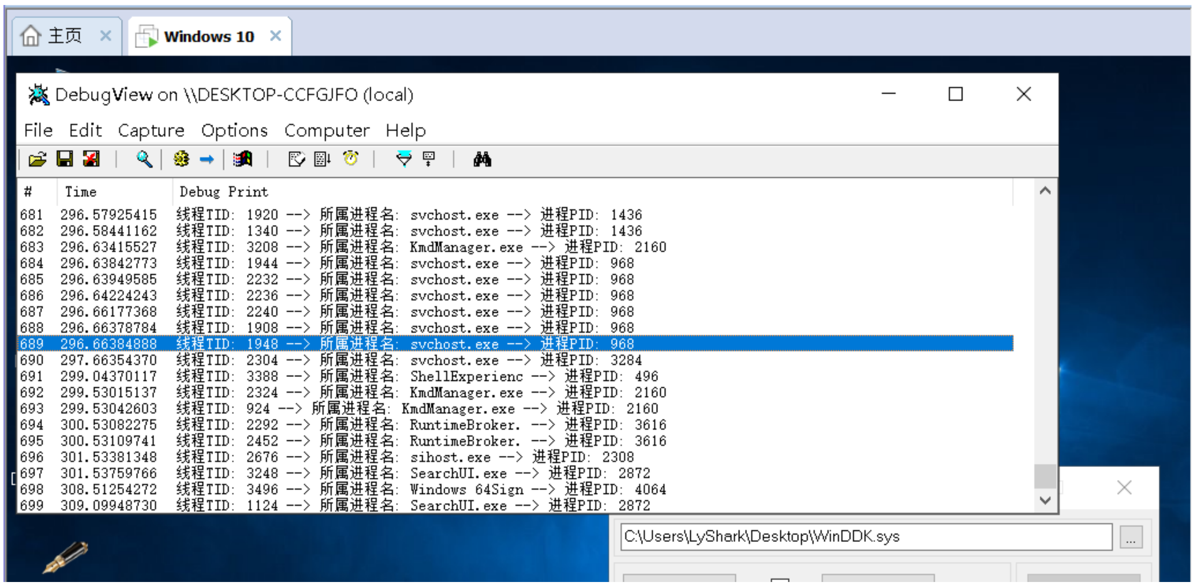
```
Debug Print
驱动加载成功!
```

C:\Users\LyShark\Desktop\WinDDK.sys

Driver	Opera...	Status	Last Error
WinDDK	Regist...	Succe...	操作成功完成。
WinDDK	Start	Succe...	操作成功完成。

而检测线程操作与检测进程差不多，检测线程需要调用`PsSetCreateThreadNotifyRoutine`创建回调函数，然后就可以检测线程的创建了，具体代码如下：

```c
#include <ntddk.h>

NTKERNELAPI PCHAR PsGetProcessImageFileName(PEPROCESS Process);
NTKERNELAPI NTSTATUS PsLookupProcessByProcessId(HANDLE ProcessId, PEPROCESS
*Process);

VOID MyCreateThreadNotify(HANDLE  ProcessId, HANDLE  ThreadId, BOOLEAN  Create)
{
    PEPROCESS eprocess = NULL;
    PsLookupProcessByProcessId(ProcessId, &eprocess);              // 通过此函数
拿到程序的EPROCESS结构
    if (Create)
        DbgPrint("线程TID: %1d --> 所属进程名: %s --> 进程PID: %1d \n", ThreadId,
PsGetProcessImageFileName(eprocess), PsGetProcessId(eprocess));
    else
        DbgPrint("%s 线程已退出...", ThreadId);
}
VOID UnDriver(PDRIVER_OBJECT driver)
{
    PsRemoveCreateThreadNotifyRoutine(MyCreateThreadNotify);
    DbgPrint(("驱动卸载成功"));
}
NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    NTSTATUS status;
    status = PsSetCreateThreadNotifyRoutine(MyCreateThreadNotify);
    DbgPrint("PsSetCreateThreadNotifyRoutine: %x", status);
    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```


主页
Windows 10
DebugView on \\DESKTOP-CCFGJFO (local)
File Edit Capture Options Computer Help
C:\Users\LyShark\Desktop\WinDDK.sys

```
#    Time          Debug Print
681  296.57925415  线程TID: 1920 --> 所属进程名: svchost.exe --> 进程PID: 1436
682  296.58441162  线程TID: 1340 --> 所属进程名: svchost.exe --> 进程PID: 1436
683  296.63415527  线程TID: 3208 --> 所属进程名: KmdManager.exe --> 进程PID: 2160
684  296.63842773  线程TID: 1944 --> 所属进程名: svchost.exe --> 进程PID: 968
685  296.63949585  线程TID: 2232 --> 所属进程名: svchost.exe --> 进程PID: 968
686  296.64224243  线程TID: 2236 --> 所属进程名: svchost.exe --> 进程PID: 968
687  296.66177368  线程TID: 2240 --> 所属进程名: svchost.exe --> 进程PID: 968
688  296.66378784  线程TID: 1908 --> 所属进程名: svchost.exe --> 进程PID: 968
689  296.66384888  线程TID: 1948 --> 所属进程名: svchost.exe --> 进程PID: 968
690  297.66354370  线程TID: 2304 --> 所属进程名: svchost.exe --> 进程PID: 3284
691  299.04370117  线程TID: 3388 --> 所属进程名: ShellExperienc --> 进程PID: 496
692  299.53015137  线程TID: 2324 --> 所属进程名: KmdManager.exe --> 进程PID: 2160
693  299.53042603  线程TID: 924 --> 所属进程名: KmdManager.exe --> 进程PID: 2160
694  300.53082275  线程TID: 2292 --> 所属进程名: RuntimeBroker. --> 进程PID: 3616
695  300.53109741  线程TID: 2452 --> 所属进程名: RuntimeBroker. --> 进程PID: 3616
696  301.53381348  线程TID: 2676 --> 所属进程名: sihost.exe --> 进程PID: 2308
697  301.53759766  线程TID: 3248 --> 所属进程名: SearchUI.exe --> 进程PID: 2872
698  308.51254272  线程TID: 3496 --> 所属进程名: Windows 64Sign --> 进程PID: 4064
699  309.09948730  线程TID: 1124 --> 所属进程名: SearchUI.exe --> 进程PID: 2872
```

本书作者: 王瑞 (LyShark)
作者邮箱: me@lyshark.com
作者博客: https://lyshark.cnblogs.com
团队首页: www.lyshark.com