当你需要在Windows操作系统中监控进程的启动和退出时，可以使用`PsSetCreateProcessNotifyRoutineEx`函数来创建一个`MyCreateProcessNotifyEx`回调函数，该回调函数将在每个进程的创建和退出时被调用。

PsSetCreateProcessNotifyRoutineEx 用于在系统启动后向内核注册一个回调函数，以监视新进程的创建和退出，其函数原型如下：

```
NTSTATUS PsSetCreateProcessNotifyRoutineEx(
  PCREATE_PROCESS_NOTIFY_ROUTINE_EX NotifyRoutine,
  BOOLEAN                           Remove
);
```

其中，参数`NotifyRoutine`是一个指向回调函数的指针，该函数将在新进程创建或退出时被调用。参数`Remove`是一个布尔值，用于指定是否从内核中删除之前注册的回调函数。如果要删除之前注册的回调函数，则将此参数设置为TRUE。如果要注册一个新的回调函数，则将此参数设置为FALSE。

如上`PCREATE_PROCESS_NOTIFY_ROUTINE_EX`用于接收一个自定义回调，该回调函数的参数传递需要定义为如下所示的样子；

```
VOID CreateProcessNotifyRoutineEx(
  PEPROCESS           Process,
  HANDLE              ProcessId,
  PPS_CREATE_NOTIFY_INFO CreateInfo
);
```

其中，参数`Process`是一个指向新创建进程的`EPROCESS`结构的指针。参数`ProcessId`是新进程的PID（进程ID）。参数`CreateInfo`是一个指向一个`PS_CREATE_NOTIFY_INFO`结构的指针，该结构包含了有关新进程的详细信息。如果新进程是由系统启动的，CreateInfo将为空。

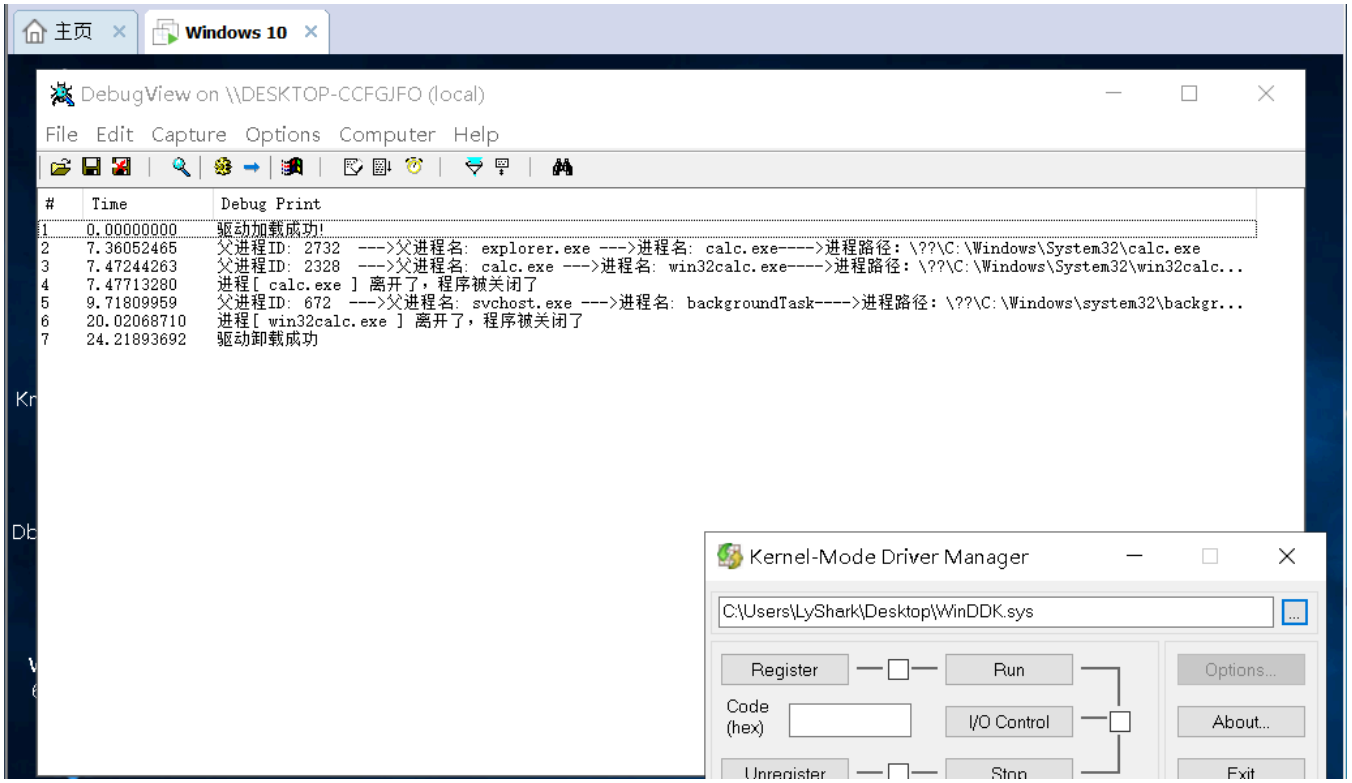
回调函数应该在执行完后尽快返回，以避免对系统性能的影响。同时，回调函数不应该调用任何可能导致死锁或系统崩溃的函数。

对于进程的创建与退出，则可通过`MyCreateProcessNotifyEx`自定义函数的`PPS_CREATE_NOTIFY_INFO`字段进行判断，如果`PPS_CREATE_NOTIFY_INFO`不等于NULL则说明该进程是被创建了，反之则说明进程是退出了，有了这些基础知识那么实现监视进程加载将变得很容易，如下案例所示；

```
#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)MyCreateProcessNotifyEx
, TRUE);
}

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

输出效果图如下所示:



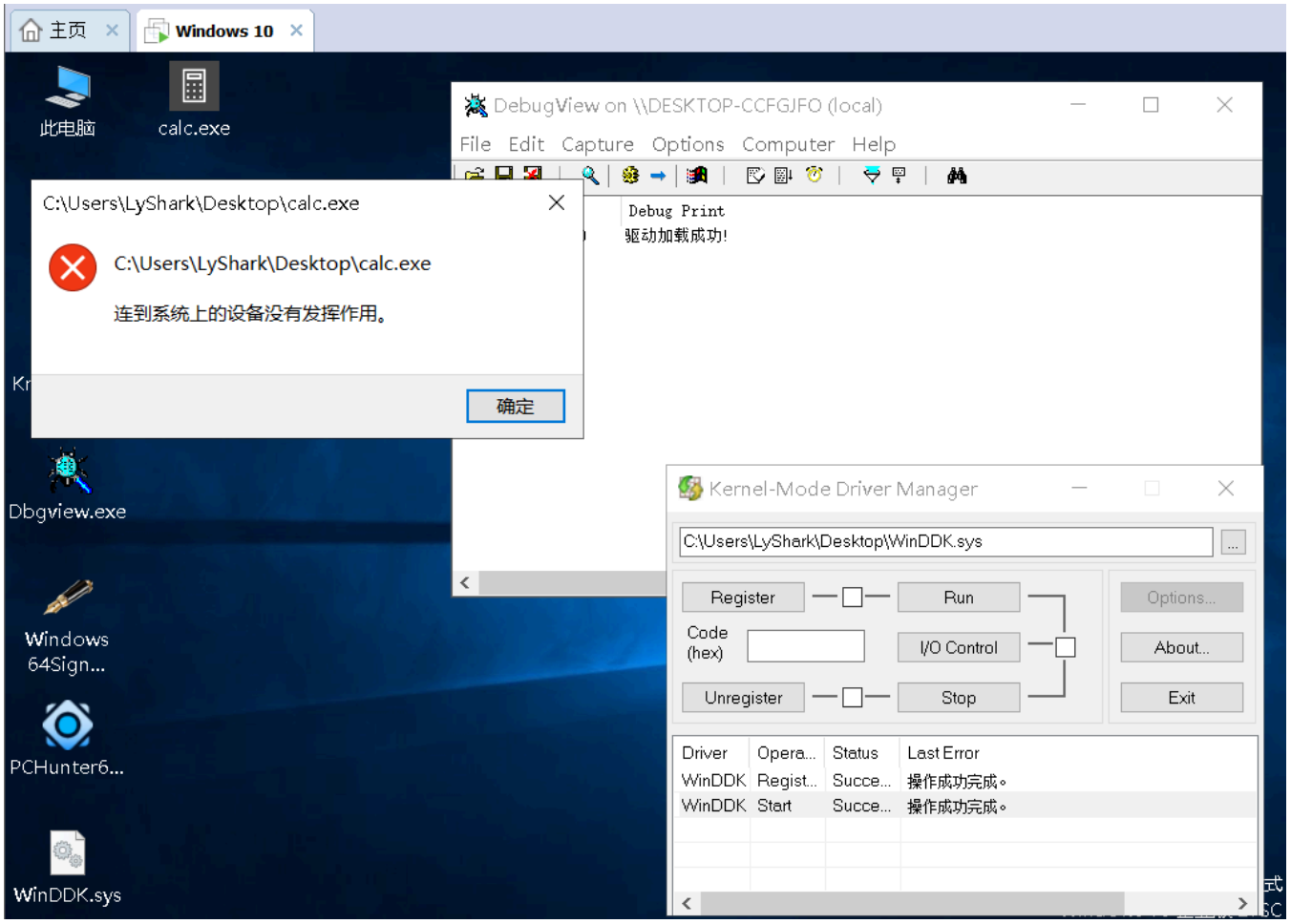
那么如何进行进程监控呢?

进程监控的思路很简单，只需要在如上`MyCreateProcessNotifyEx()`这个自定义回调函数中进行改进即可，首先通过`PsGetProcessImageFileName`即将进程的PID转换为进程名，然后就可以通过`_stricmp`对比该进程是否为我们所需要的，如果发现是`calc.exe`等特定进程名，则可以将`CreateInfo->CreationStatus`中的参数修改为`STATUS_UNSUCCESSFUL`这意味着对象的创建过程未成功完成，从而实现拒绝进行执行的目的;

```
#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。



与进程检测类似，如果要检测线程创建则只需要通过`PsSetCreateThreadNotifyRoutine`创建线程回调即可，PsSetCreateThreadNotifyRoutine 函数的原型如下：

```
NTKERNELAPI
VOID
PsSetCreateThreadNotifyRoutine(
  _Inout_ PCREATE_THREAD_NOTIFY_ROUTINE NotifyRoutine,
  _In_ BOOLEAN                          Remove
);
```

PsSetCreateThreadNotifyRoutine 它允许注册一个回调函数，以便在新的线程被创建时被调用。该函数有两个参数：

- 第一个参数：是一个指向回调函数的指针，这个回调函数将在新的线程被创建时被调用。
- 第二个参数：是一个布尔值，表示是否将此回调函数添加到一个已有的回调列表中。如果此参数为TRUE，则将该回调函数添加到列表中，如果为FALSE，则将替换掉已有的回调函数。

当一个新的线程被创建时，操作系统会调用所有已注册的回调函数，并将新线程的`ThreadID`和进程ID作为参数传递给回调函数。这些参数可以用来识别新线程所属的进程以及新线程本身的标识符。

对于`PCREATE_THREAD_NOTIFY_ROUTINE`来说，它指向一个回调函数，用于通知进程中新线程的创建。该函数指针的定义如下：

```
typedef VOID (*PCREATE_THREAD_NOTIFY_ROUTINE) (
  _In_ HANDLE ProcessId,
  _In_ HANDLE ThreadId,
  _In_ BOOLEAN Create
);
```

回调函数的参数说明如下：

- ProcessId：新线程所属进程的进程ID。
- ThreadId：新线程的线程ID。
- Create：布尔值，指示新线程是创建还是销毁。如果为TRUE，则表示新线程已创建；如果为FALSE，则表示新线程已销毁。

在 PsSetCreateThreadNotifyRoutine 函数中注册的回调函数应该符合这个函数指针的定义，以便在新线程被创建或销毁时被调用。

而当调用结束后，用户需要通过 `PsRemoveCreateThreadNotifyRoutine` 来删除已注册的回调函数，目前该函数只需要一个参数，只需要传入注册时的函数指针即可；

```
NTKERNELAPI
VOID
PsRemoveCreateThreadNotifyRoutine(
  _Inout_ PCREATE_THREAD_NOTIFY_ROUTINE NotifyRoutine
);
```

通过上述知识点的总结，相信你也可以很容易的编写处线程检测相关代码片段，具体代码如下：

```
#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;
  // 通过此函数拿到程序的EPROCESS结构
    PsLookupProcessByProcessId(ProcessId, &eprocess);
    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;
}
```

输出效果图如下所示: