通常使用Windows系统自带的任务管理器可以正常地结束掉一般进程，而某些特殊的进程在应用层很难被结束掉，例如某些系统核心进程其权限是在0环内核态，但有时我们不得不想办法结束掉这些特殊的进程，当然某些正常进程在特殊状态下也会无法被正常结束，此时使用驱动前行在内核态将其结束掉就变得很有用了，驱动结束进程有多种方法。

- 1.标准方法就是使用ZwOpenProcess打开进程获得句柄，然后使用ZwTerminateProcess这个内核API实现结束进程，最后使用ZwClose关闭句柄。
- 2.第二种方法，通过动态定位的方式找到PspTerminateThreadByPointer这个内核函数地址，然后调用该函数结束掉进程中所有的线程，当线程为空则进程也就消亡了。
- 3.第三种方法，我将其称作是内存清零法，其核心原理是通过打开进程，得到进程的基址，通过内存填充的方式将对端内存全部置0实现类似于结束的效果。

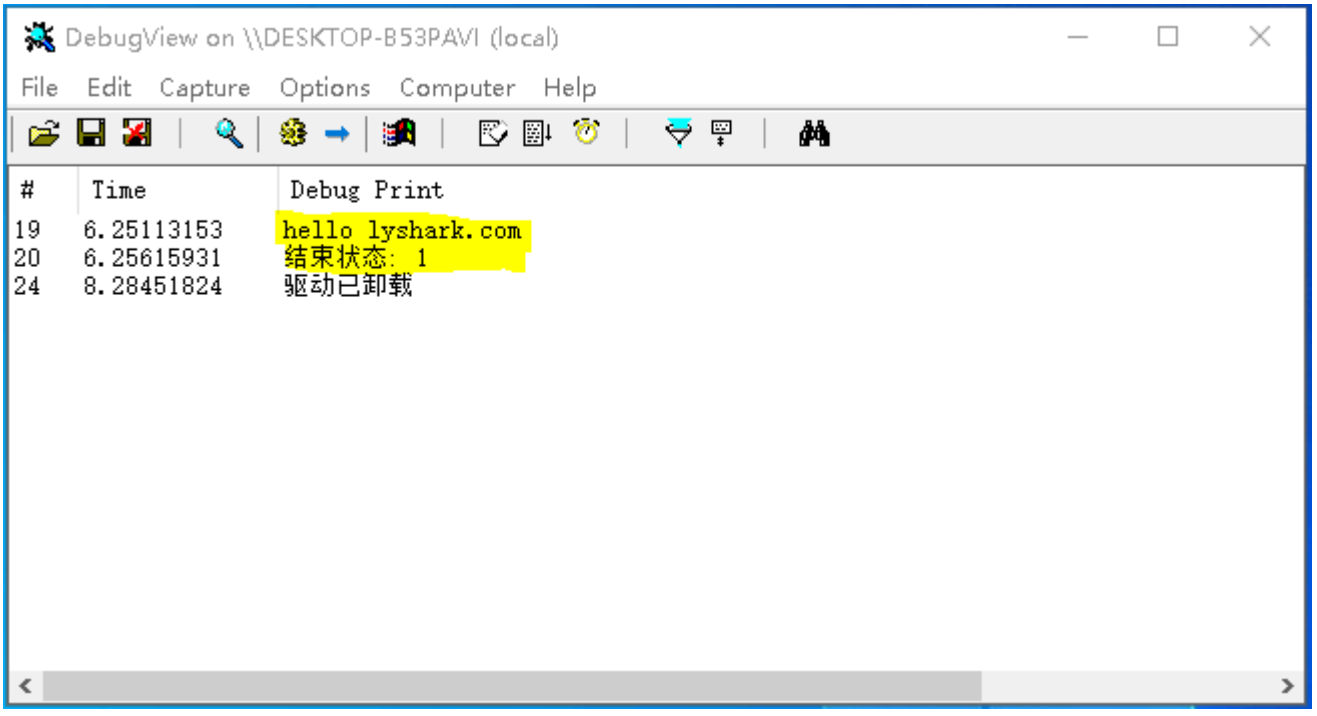
首先是第一种方法结束进程，封装实现KillProcess函数，用户传入lyshark.exe进程名，进程内执行PsGetProcessImageFileName判断是否是我们要结束的如果是则，调用ZwOpenProcess打开进程，并发送ZwTerminateProcess终止信号从而正常结束，其核心代码如下所示。

```
#include <ntifs.h>

NTKERNELAPI UCHAR* PsGetProcessImageFileName(IN PEPROCESS Process);

// 根据进程ID返回进程EPROCESS结构体,失败返回NULL
PEPROCESS GetProcessNameByProcessId(HANDLE pid)
{
    PEPROCESS ProcessObj = NULL;
    NTSTATUS Status = STATUS_UNSUCCESSFUL;
    Status = PsLookupProcessByProcessId(pid, &ProcessObj);
    if (NT_SUCCESS(Status))
        return ProcessObj;
    return NULL;
}

// 根据ProcessName获取到进程的PID号
HANDLE GetPidByProcessName(char *ProcessName)
{
    PEPROCESS pCurrentEprocess = NULL;
    HANDLE pid = 0;
    for (int i = 0; i < 1000000000; i += 4)
    {
        pCurrentEprocess = GetProcessNameByProcessId((HANDLE)i);
        if (pCurrentEprocess != NULL)
        {
            pid = PsGetProcessId(pCurrentEprocess);
            if (strstr(PsGetProcessImageFileName(pCurrentEprocess), ProcessName) != NULL)
            {
                ObDereferenceObject(pCurrentEprocess);
                return pid;
            }
            ObDereferenceObject(pCurrentEprocess);
        }
    }
    return (HANDLE)-1;
```

```
}

// 传入进程名称,终止掉该进程
BOOLEAN KillProcess(PCHAR ProcessName)
{
    PEPROCESS pCurrentEprocess = NULL;
    HANDLE pid = 0;
    HANDLE Handle = NULL;
    OBJECT_ATTRIBUTES obj;
    CLIENT_ID cid = { 0 };
    NTSTATUS Status = STATUS_UNSUCCESSFUL;

    for (int i = 0; i < 10000000; i += 4)
    {
        pCurrentEprocess = GetProcessNameByProcessId((HANDLE)i);
        if (pCurrentEprocess != NULL)
        {
            pid = PsGetProcessId(pCurrentEprocess);

            // 判断当前镜像名称是否是需要结束的进程
            if (strstr(PsGetProcessImageFileName(pCurrentEprocess), ProcessName) != NULL)
            {
                ObDereferenceObject(pCurrentEprocess);

                // 找到后开始结束
                InitializeObjectAttributes(&obj, NULL, OBJ_KERNEL_HANDLE |
OBJ_CASE_INSENSITIVE, NULL, NULL);
                cid.UniqueProcess = (HANDLE)pid;
                cid.UniqueThread = 0;

                // 打开进程
                Status = ZwOpenProcess(&Handle, GENERIC_ALL, &obj, &cid);
                if (NT_SUCCESS(Status))
                {
                    // 发送终止信号
                    ZwTerminateProcess(Handle, 0);
                    ZwClose(Handle);
                }
                ZwClose(Handle);
                return TRUE;
            }
            ObDereferenceObject(pCurrentEprocess);
        }
    }
    return FALSE;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint("驱动已卸载 \n");
}

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
```

```
{
    DbgPrint("hello lyshark.com \n");

    BOOLEAN Retn;
    Retn = KillProcess("lyshark.exe");
    DbgPrint("结束状态: %d \n", Retn);

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

我们运行这个驱动，当进程 `lyshark.exe` 存在时则可以看到结束效果，当然这种方式只是在内核层面调用了结束进程函数，其本质上还是正常结束，只是这种方式权限要大一些仅此而已。



第二种方法，其原理就是将进程内的线程全部结束掉从而让进程自动结束，由于 `PspTerminateThreadByPointer` 没有被导出，所以我们需要动态的这个内存地址，然后动态调用即可，这个寻找方法可以总结为以下步骤。

- 1.寻找 `PsTerminateSystemThread` 函数地址，这个地址可以直接通过 `MmGetSystemRoutineAddress` 函数得到。
- 2.在 `PsTerminateSystemThread` 函数地址内向下扫描特征 `e80cb6f6ff` 得到 `call nt!PspTerminateThreadByPointer` 地址。

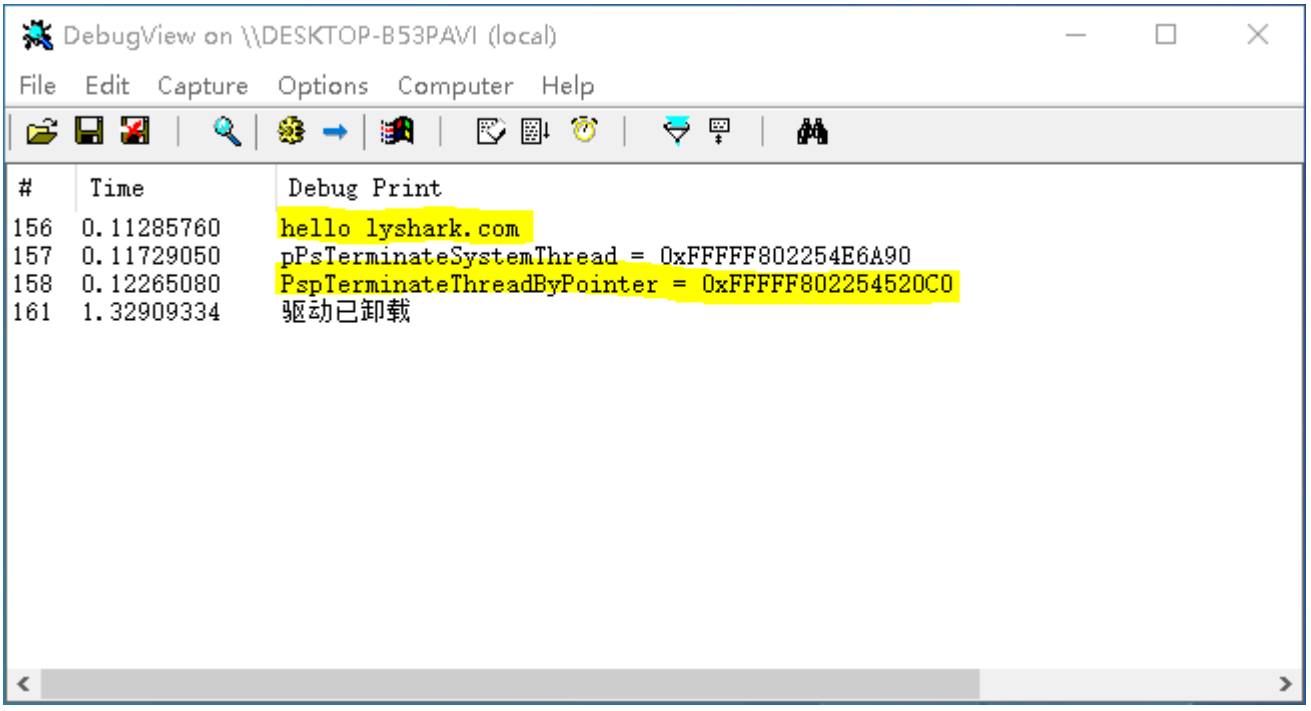
根据 `《内核枚举LoadImage映像回调》` 中使用的 `SearchMemory` 函数实现搜索 `PspTerminateThreadByPointer` 内存地址。

```
#include <ntifs.h>

// 得到PspTerminateThreadByPointer内存地址
PVOID PspTerminateThreadByPointer()
{
    UNICODE_STRING ustrFuncName;
    PVOID pAddress = NULL;
    LONG lOffset = 0;
```

```
    PVOID pPsTerminateSystemThread = NULL;
    PVOID pPspTerminateThreadByPointer = NULL;

    // 获取 PsTerminateSystemThread 函数地址
    RtlInitUnicodeString(&ustrFuncName, L"PsTerminateSystemThread");
    pPsTerminateSystemThread = MmGetSystemRoutineAddress(&ustrFuncName);

    DbgPrint("pPsTerminateSystemThread = 0x%p \n", pPsTerminateSystemThread);
    if (NULL == pPsTerminateSystemThread)
    {
        return 0;
    }

    // 查找 PspTerminateThreadByPointer 函数地址

    /*
    1: kd> uf PsTerminateSystemThread
            nt!PsTerminateSystemThread:
            fffff802`254e6a90 4883ec28        sub     rsp,28h
            fffff802`254e6a94 8bd1            mov     edx,ecx
            fffff802`254e6a96 65488b0c2588010000 mov   rcx,qword ptr gs:[188h]
            fffff802`254e6a9f f7417400040000  test    dword ptr [rcx+74h],400h
            fffff802`254e6aa6 0f8444081100    je      nt!PsTerminateSystemThread+0x110860
(fffff802`255f72f0)  Branch

            nt!PsTerminateSystemThread+0x1c:
            fffff802`254e6aac 41b001          mov     r8b,1
            fffff802`254e6aaf e80cb6f6ff      call    nt!PspTerminateThreadByPointer
(fffff802`254520c0)

            nt!PsTerminateSystemThread+0x24:
            fffff802`254e6ab4 4883c428        add     rsp,28h
            fffff802`254e6ab8 c3              ret

            nt!PsTerminateSystemThread+0x110860:
            fffff802`255f72f0 b80d0000c0      mov     eax,0C000000Dh
            fffff802`255f72f5 e9baf7eeff      jmp     nt!PsTerminateSystemThread+0x24
(fffff802`254e6ab4)  Branch
    */

    UCHAR pSpecialData[50] = { 0 };
    ULONG ulSpecialDataSize = 0;

    // fffff802`254e6aaf e80cb6f6ff      call    nt!PspTerminateThreadByPointer
(fffff802`254520c0)
    pSpecialData[0] = 0xE8;
    ulSpecialDataSize = 1;

    // 搜索地址 PsTerminateSystemThread --> PsTerminateSystemThread + 0xff 查找 e80cb6f6ff
    pAddress = SearchMemory(pPsTerminateSystemThread, (PVOID)
((PUCHAR)pPsTerminateSystemThread + 0xFF), pSpecialData, ulSpecialDataSize);
    if (NULL == pAddress)
    {
```

```
        return 0;
    }

    // 先获取偏移,再计算地址
    lOffset = *(PLONG)pAddress;
    pPspTerminateThreadByPointer = (PVOID)((PUCHAR)pAddress + sizeof(LONG) + lOffset);
    if (NULL == pPspTerminateThreadByPointer)
    {
        return 0;
    }

    return pPspTerminateThreadByPointer;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint("驱动已卸载 \n");
}

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

    PVOID address = PspTerminateThreadByPointer();

    DbgPrint("PspTerminateThreadByPointer = 0x%p \n", address);

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

运行驱动程序，首先得到`PspTerminateThreadByPointer`的内存地址，效果如下。



得到内存地址以后直接将地址`typedef`转为指针函数，调用并批量结束进程内的线程即可。

```
// 署名权
// right to sign one's name on a piece of work
// PowerBy: LyShark
// Email: me@lyshark.com

#include <ntifs.h>

typedef NTSTATUS(__fastcall *PSPTERMINATETHREADBYPOINTER) (PETHREAD pEThread, NTSTATUS
ntExitCode, BOOLEAN bDirectTerminate);

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint("驱动已卸载 \n");
}

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

    PVOID pPspTerminateThreadByPointerAddress = 0xFFFFF802254520C0;
    HANDLE hProcessId = 6956;

    PEPROCESS pEProcess = NULL;
    PETHREAD pEThread = NULL;
    PEPROCESS pThreadEProcess = NULL;
    NTSTATUS status = STATUS_SUCCESS;
    ULONG i = 0;

    // 获取结束进程的进程结构对象EPROCESS
    status = PsLookupProcessByProcessId(hProcessId, &pEProcess);
    if (!NT_SUCCESS(status))
    {
        return status;
    }
    // 遍历所有线程，并结束所有指定进程的线程
    for (i = 4; i < 0x80000; i = i + 4)
    {
        status = PsLookupThreadByThreadId((HANDLE)i, &pEThread);
        if (NT_SUCCESS(status))
        {
            // 获取线程对应的进程结构对象
            pThreadEProcess = PsGetThreadProcess(pEThread);

            // 结束进程中的线程
            if (pEProcess == pThreadEProcess)
            {
                ((PSPTERMINATETHREADBYPOINTER)pPspTerminateThreadByPointerAddress)(pEThread,
0, 1);
                DbgPrint("结束线程: %d \n", i);
            }
            ObDereferenceObject(pEThread);
        }
    }
```

```
    ObDereferenceObject(pEProcess);

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

循环结束进程 `6956` 内的所有线程信息，效果如下；