x64内核：系统回调

有些版本的操作系统没有`PsSetXXXNotifyRoutineEx`版本的函数，但是又需要拦截API

思路：拿到相关的结构，改结构里面的字段

注意：在驱动卸载时，需要调用相关的API移除注册的回调

相关API：

- 创建进程类：`PsSetCreateProcessNotifyRoutine`等
- 创建线程类：`PsSetCreateThreadNotifyRoutine`、`PsRemoveCreateThreadNotifyRoutine`等
- 加载模块类：`PsSetLoadImageNotifyRoutine`、`PsRemoveLoadImageNotifyRoutine`等
- 注册表类：`CmRegisterCallbackEx`等
- 错误检查类：`KeRegisterBugCheckCallback`等
- 关闭系统类：`IoRegisterShutdownNotification`等
- 对象句柄操作类：`ObRegisterCallbacks`等
- ...

其中对象句柄操作类（又称对象钩子）可对进程线程桌面句柄进行控制，使得实现保护进程、线程不被恶意操作

对象钩子

ObRegisterCallbacks 注册句柄操作类回调

```
NTSTATUS ObRegisterCallbacks(
  POB_CALLBACK_REGISTRATION CallbackRegistration,
  PVOID                     *RegistrationHandle
);
```

参数

- `CallbackRegistration`
  - 指向指定回调例程列表和其他注册信息的
- `RegistrationHandle`
  - 指向变量的指针，该变量接收一个标识已注册的回调例程集合的值，调用者将此值传递给`ObUnRegisterCallbacks`例程以注销该回调集

返回值

- 成功，则返回`STATUS_SUCCESS`，否则，返回其它`NTSTATUS`错误码

OB_CALLBACK_REGISTRATION 结构体

```
typedef struct _OB_CALLBACK_REGISTRATION {
  USHORT                    Version;
  USHORT                    OperationRegistrationCount;
  UNICODE_STRING            Altitude;
  PVOID                     RegistrationContext;
  OB_OPERATION_REGISTRATION *OperationRegistration;
} OB_CALLBACK_REGISTRATION, *POB_CALLBACK_REGISTRATION;
```

- `Version`
  - 请求的对象回调注册版本，驱动需要填为`OB_FLT_REGISTRATION_VERSION`
- `OperationRegistrationCount`
  - 第五个成员`OperationRegistration`数组的元素个数
- `Altitude`
  - 指定驱动的`Altitude`字符串（Unicode）
- `RegistrationContext`
  - 当运行回调时，系统将`RegistrationContext`值传递给回调。这个值的意义是驱动定义。
- `OperationRegistration`
  - `OB_OPERATION_REGISTRATION`结构的数组，每个结构指定`ObjectPreCallback`（在操作之前调用）和`ObjectPostCallback`（在操作之后调用）回调，以及调用的操作类型

OB_OPERATION_REGISTRATION 结构体

```
typedef struct _OB_OPERATION_REGISTRATION {
  POBJECT_TYPE                *ObjectType;
  OB_OPERATION                Operations;
  POB_PRE_OPERATION_CALLBACK  PreOperation;
  POB_POST_OPERATION_CALLBACK PostOperation;
} OB_OPERATION_REGISTRATION, *POB_OPERATION_REGISTRATION;
```

- `ObjectType`
  - 指向触发回调例程的对象类型的指针，可以为以下值：
    - **PsProcessType** 用于进程操作
    - **PsThreadType** 用于线程操作
    - **ExDesktopObjectType** 用于桌面句柄操作（Win10上支持）
- `Operations`
  - 指定一个或多个标志：
    - **OB_OPERATION_HANDLE_CREATE** 打开一个新进程、线程或桌面句柄
    - **OB_OPERATION_HANDLE_DUPLICATE** 进程、线程或桌面句柄已经（或即将）被复制
- `PreOperation`
  - 回调指针，系统在执行请求的操作之前调用这个回调
- `PostOperation`
  - 回调指针，系统在执行请求的操作之后调用这个回调

回调函数

ObjectPreCallback

`ObjectPreCallback` 回调，系统在执行请求的操作之前调用这个回调

```
POB_PRE_OPERATION_CALLBACK PobPreOperationCallback;

OB_PREOP_CALLBACK_STATUS PobPreOperationCallback(
  PVOID RegistrationContext,
  POB_PRE_OPERATION_INFORMATION OperationInformation
)
{...}
```

参数

- `RegistrationContext`
  - 驱动指定的上下文，由 `OB_CALLBACK_REGISTRATION` 结构体的 `RegistrationContext` 提供
- `OperationInformation`
  - 一个指向 `OB_PRE_OPERATION_INFORMATION` 结构的指针，该结构指定句柄操作的参数

返回值

返回一个 `OB_PREOP_CALLBACK_STATUS` 值。驱动程序必须返回 `B_PREOP_SUCCESS`

ObjectPostCallback

`ObjectPostCallback` 回调，系统在执行请求的操作之后调用这个回调

```
POB_POST_OPERATION_CALLBACK PobPostOperationCallback;

void PobPostOperationCallback(
  PVOID RegistrationContext,
  POB_POST_OPERATION_INFORMATION OperationInformation
)
{...}
```

参数

- `RegistrationContext`
  - 驱动指定的上下文，由 `OB_CALLBACK_REGISTRATION` 结构体的 `RegistrationContext` 提供
- `OperationInformation`
  - 一个指向 `OB_PRE_OPERATION_INFORMATION` 结构的指针，该结构指定句柄操作的参数

OB_PRE_OPERATION_INFORMATION 结构体

```
typedef struct _OB_PRE_OPERATION_INFORMATION {
  OB_OPERATION                 Operation;
  union {
    ULONG Flags;    // 保留
    struct {
```

```
      ULONG KernelHandle : 1;
      ULONG Reserved : 31;      // 保留
    };
  };
  PVOID                         Object;
  POBJECT_TYPE                  ObjectType;
  PVOID                         CallContext;
  POB_PRE_OPERATION_PARAMETERS Parameters;
} OB_PRE_OPERATION_INFORMATION, *POB_PRE_OPERATION_INFORMATION;
```

- `Operation`
  - 句柄操作类型
    - **OB_OPERATION_HANDLE_CREATE** 进程或线程的新句柄将被打开。使用 `Parameters-> CreateHandleInformation` 获取特定于创建的信息
    - **OB_OPERATION_HANDLE_DUPLICATE** 进程或线程句柄将被复制。使用 `Parameters-> DuplicateHandleInformation` 获取特定于重复项的信息
- `KernelHandle`
  - 指定该句柄是否为内核句柄的位。如果此成员为**TRUE**，则该句柄是内核句柄。否则，此句柄不是内核句柄
- `Object`
  - 指向作为句柄操作**目标**的进程或线程对象的指针
- `ObjectType`
  - 指向对象的对象类型的指针
    - `*PsProcessType` 用于进程
    - `*PsThreadType` 用于线程
- `CallContext`
  - 指向该操作的特定于驱动程序的上下文信息的指针
- `Parameters`
  - 指向 `OB_PRE_OPERATION_PARAMETERS` 结构的指针

OB_POST_OPERATION_INFORMATION 结构体

```
typedef struct _OB_POST_OPERATION_INFORMATION {
  OB_OPERATION                  Operation;
  union {
    ULONG Flags;
    struct {
      ULONG KernelHandle : 1;
      ULONG Reserved : 31;
    };
  };
  PVOID                         Object;
  POBJECT_TYPE                  ObjectType;
  PVOID                         CallContext;
  NTSTATUS                      ReturnStatus;    // 句柄操作的NTSTATUS返回值
  POB_POST_OPERATION_PARAMETERS Parameters;
} OB_POST_OPERATION_INFORMATION, *POB_POST_OPERATION_INFORMATION;
```

其他同 `OB_PRE_OPERATION_INFORMATION` 一致

示例

禁止对计算器的操作

驱动入口函数`DriverEntry`：

```
{
    ...
    // 定义结构体
    OB_CALLBACK_REGISTRATION obReg = { 0 };
    OB_OPERATION_REGISTRATION obPer[2] = { 0 };

    // 操作进程、创建与复制句柄，在执行请求操作之前
    obPer[0].ObjectType = PsProcessType;
    obPer[0].Operations = OB_OPERATION_HANDLE_CREATE |
OB_OPERATION_HANDLE_DUPLICATE;
    obPer[0].PreOperation = ObjectPreCallback;

    // 操作线程、创建与复制句柄，在执行请求操作之前
    obPer[1].ObjectType = PsThreadType;
    obPer[1].Operations = OB_OPERATION_HANDLE_CREATE |
OB_OPERATION_HANDLE_DUPLICATE;
    obPer[1].PreOperation = ObjectPreCallback;

    obReg.Version = OB_FLT_REGISTRATION_VERSION;                    // 版本
    obReg.OperationRegistrationCount = 2;                           // 数组有两个成员
    RtlInitUnicodeString(&obReg.Altitude, L"69999");                // 驱动的Altitude字符
串
    obReg.RegistrationContext = NULL;                               // 驱动上下文
    obReg.OperationRegistration = obPer;
    // 注册回调
    status = ObRegisterCallbacks(&obReg, &g_RegistrationHandle);
    ...
}
```

回调函数`ObjectPreCallback`

```
OB_PREOP_CALLBACK_STATUS ObjectPreCallback(
    __in PVOID  RegistrationContext,
    __in POB_PRE_OPERATION_INFORMATION  OperationInformation)
{
    PEPROCESS   SrcProcess = NULL;
    PUCHAR      SrcImageName = NULL;
    PEPROCESS   DestProcess = NULL;
    PUCHAR      DestImageName = NULL;
    PETHREAD Thread;
    SrcProcess = PsGetCurrentProcess();
    SrcImageName = PsGetProcessImageFileName(SrcProcess);

    // 判断操作类型
    if (OperationInformation->ObjectType == *PsProcessType) {
        DestProcess = (PEPROCESS)OperationInformation->Object;  // 如果是进程操作，
拿到目标进程对象
```

```
    } else {
        return OB_PREOP_SUCCESS;
    }
    // 查找目标进程映像名
    DestImageName = PsGetProcessImageFileName(DestProcess);

    // 比较目标进程是否是计算器，不是则放行
    if (strstr((CHAR*)DestImageName, "calc") == NULL)
        return OB_PREOP_SUCCESS;

    DbgPrint("[51asm] SrcImageName:%s DestImageName:%s\n",
        SrcImageName, DestImageName);

    // 比较源进程是否是计算器，是则放行
    if (strstr((CHAR*)SrcImageName, "calc") != NULL)
        return OB_PREOP_SUCCESS;

    // 源与目标进程都不是计算器，则继续执行
    if (OperationInformation->ObjectType == *PsProcessType) {
        DbgPrint("[51asm] SrcImageName:%s DestImageName:%s Operator Process\n",
            SrcImageName, DestImageName);
    } else if (OperationInformation->ObjectType == *PsThreadType) {
        DbgPrint("[51asm] SrcImageName:%s DestImageName:%s Operator Thread\n",
            SrcImageName, DestImageName);
    }

    // 将其他进程线程的创建、复制访问权限清0
    if (OperationInformation->Operation == OB_OPERATION_HANDLE_CREATE) {
        OperationInformation->Parameters->CreateHandleInformation.DesiredAccess
= 0;
    } else if (OperationInformation->Operation == OB_OPERATION_HANDLE_DUPLICATE)
{
        OperationInformation->Parameters-
>DuplicateHandleInformation.DesiredAccess = 0;
    }
    return OB_PREOP_SUCCESS;
}
```

一些未公开的函数

PsLookupProcessByProcessId

```
NTSTATUS
PsLookupProcessByProcessId(
    __in HANDLE ProcessId,
    __deref_out PEPROCESS *Process
);
```

通过进程ID查找EPROCESS

PsLookupThreadByThreadId

```
NTSTATUS
PsLookupThreadByThreadId(
    __in HANDLE ThreadId,
    __deref_out PETHREAD *Thread
);
```

通过线程ID查找PETHREAD

PsGetProcessImageFileName

```
UCHAR *
PsGetProcessImageFileName(
    __in PEPROCESS Process
);
```

通过EPROCESS查找映像名称

ZwQuerySystemInformation

```
NTSTATUS NTAPI ZwQuerySystemInformation(
    __in SYSTEM_INFORMATION_CLASS SystemInformationClass,
    __out_bcount_opt(SystemInformationLength) PVOID SystemInformation,
    __in ULONG SystemInformationLength,
    __out_opt PULONG ReturnLength
);
```

查询系统信息

`SYSTEM_INFORMATION_CLASS`枚举类型

```
typedef enum _SYSTEM_INFORMATION_CLASS {
    SystemBasicInformation,
    SystemProcessorInformation,             // obsolete...delete
    SystemPerformanceInformation,
    SystemTimeOfDayInformation,
    SystemPathInformation,
    SystemProcessInformation,
    SystemCallCountInformation,
    SystemDeviceInformation,
    SystemProcessorPerformanceInformation,
    SystemFlagsInformation,
    SystemCallTimeInformation,
    SystemModuleInformation,
    SystemLocksInformation,
    SystemStackTraceInformation,
    SystemPagedPoolInformation,
    SystemNonPagedPoolInformation,
    SystemHandleInformation,
    SystemObjectInformation,
```

```
    SystemPageFileInformation,
    SystemVdmInstemulInformation,
    SystemVdmBopInformation,
    SystemFileCacheInformation,
    SystemPoolTagInformation,
    SystemInterruptInformation,
    SystemDpcBehaviorInformation,
    SystemFullMemoryInformation,
    SystemLoadGdiDriverInformation,
    SystemUnloadGdiDriverInformation,
    SystemTimeAdjustmentInformation,
    SystemSummaryMemoryInformation,
    SystemMirrorMemoryInformation,
    SystemPerformanceTraceInformation,
    SystemObsolete0,
    SystemExceptionInformation,
    SystemCrashDumpStateInformation,
    SystemKernelDebuggerInformation,
    SystemContextSwitchInformation,
    SystemRegistryQuotaInformation,
    SystemExtendServiceTableInformation,
    SystemPrioritySeperation,
    SystemVerifierAddDriverInformation,
    SystemVerifierRemoveDriverInformation,
    SystemProcessorIdleInformation,
    SystemLegacyDriverInformation,
    SystemCurrentTimeZoneInformation,
    SystemLookasideInformation,
    SystemTimeSlipNotification,
    SystemSessionCreate,
    SystemSessionDetach,
    SystemSessionInformation,
    SystemRangeStartInformation,
    SystemVerifierInformation,
    SystemVerifierThunkExtend,
    SystemSessionProcessInformation,
    SystemLoadGdiDriverInSystemSpace,
    SystemNumaProcessorMap,
    SystemPrefetcherInformation,
    SystemExtendedProcessInformation,
    SystemRecommendedSharedDataAlignment,
    SystemComPlusPackage,
    SystemNumaAvailableMemory,
    SystemProcessorPowerInformation,
    SystemEmulationBasicInformation,
    SystemEmulationProcessorInformation,
    SystemExtendedHandleInformation,
    SystemLostDelayedWriteInformation,
    SystemBigPoolInformation,
    SystemSessionPoolTagInformation,
    SystemSessionMappedViewInformation,
    SystemHotpatchInformation,
    SystemObjectSecurityMode,
    SystemWatchdogTimerHandler,
    SystemWatchdogTimerInformation,
    SystemLogicalProcessorInformation,
    SystemWow64SharedInformation,
    SystemRegisterFirmwareTableInformationHandler,
```

```
    SystemFirmwareTableInformation,
    SystemModuleInformationEx,
    SystemVerifierTriageInformation,
    SystemSuperfetchInformation,
    SystemMemoryListInformation,
    SystemFileCacheInformationEx,
    MaxSystemInfoClass  // MaxSystemInfoClass should always be the last enum
} SYSTEM_INFORMATION_CLASS;
```