What behind key simulation – Windows Input System Internal

Kelvin Chan

Tencent

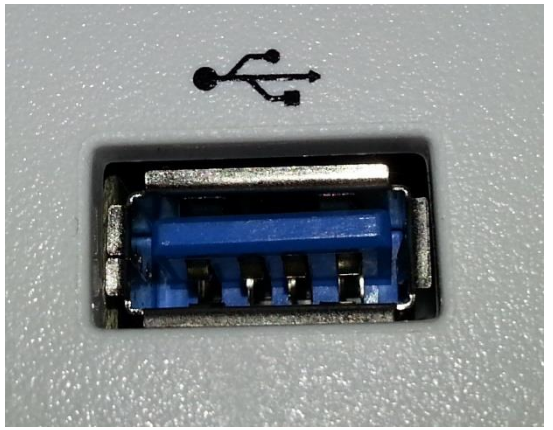
Overview

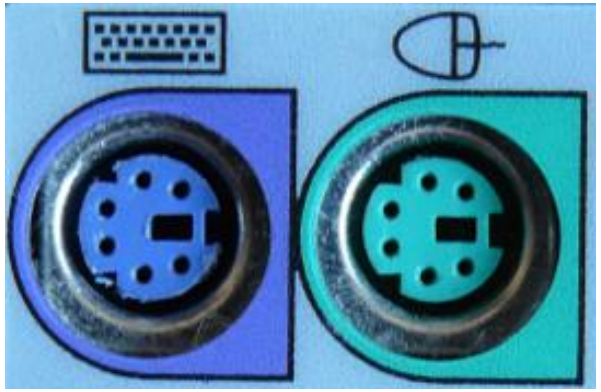
- Windows Support Input Model
- HID device introduction
- HID descriptor
- Windows input system architecture
- Windows HID Internals anaylsis
- Direct Input
- From user mode to kernel mode overview
- cheat
- solution

Windows supported input mode

- PS/2 vs. USB

Function	PS/2	USB
Hot-plug	✗	✓
Multi-device	✗	✓





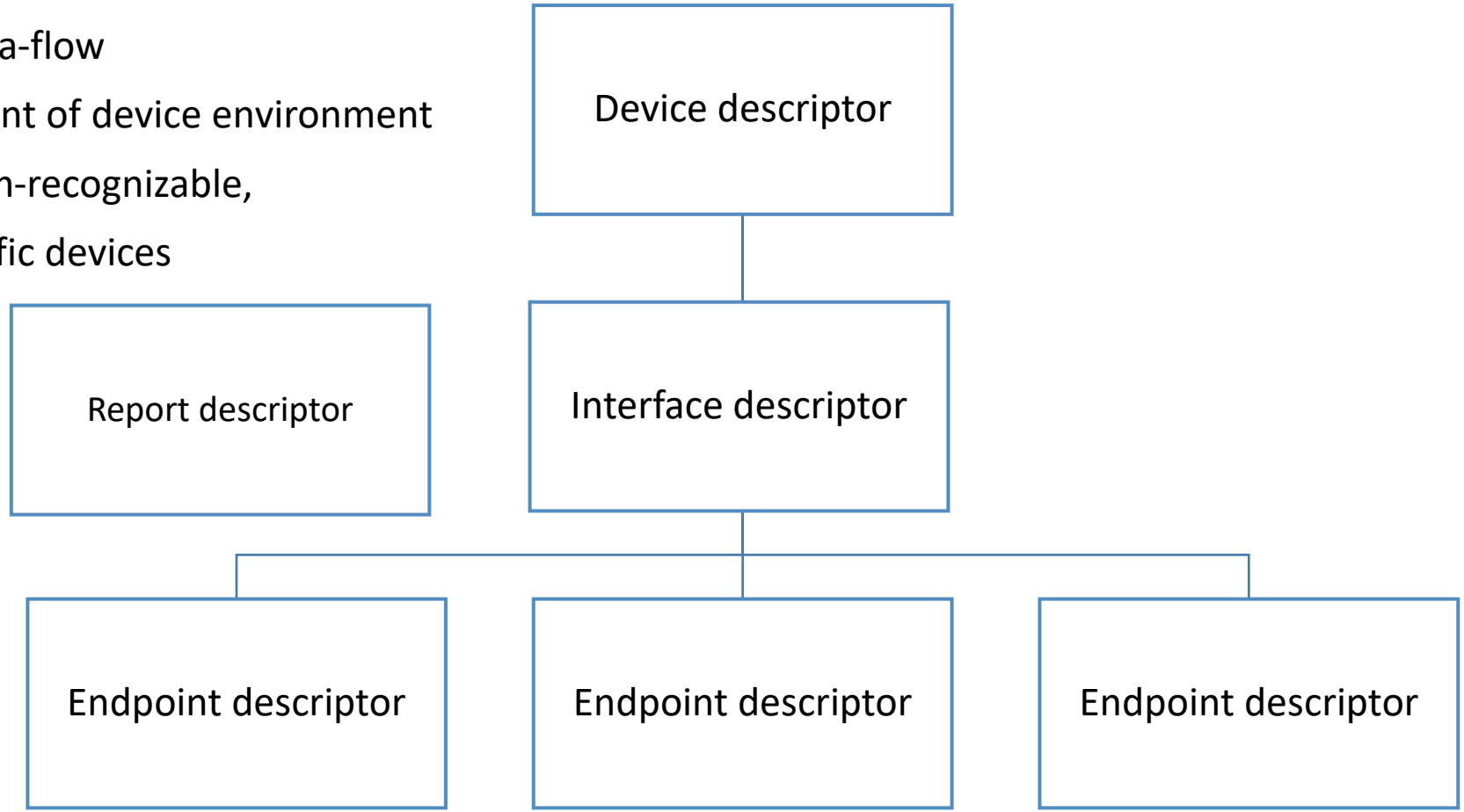
HID device

- HID , Human Interface Device
- All HID device used unified descriptor format (mouse, keyboard, joystick)
- Interface descriptor
- HID report descriptor
- Endpoint descriptor

HID descriptor

- Descriptor is basic of USBHID device
- Describing the device data-flow
- Describing the requirement of device environment
- Provides operating system-recognizable,

resolvable content for specific devices



Configure descriptors

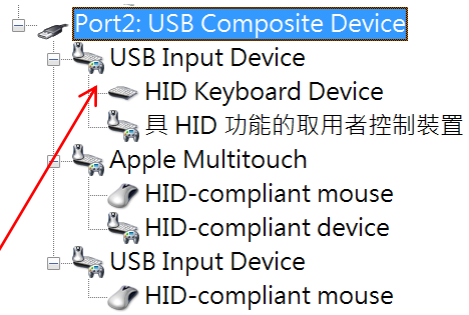
- Primary records properties related to hardware foundation
- The configuration descriptor is the root of a USBHID device
- He recorded how many interfaces the device had.
- How to supply electricity

Configuration Descriptor 1 Bus Powered, 40 mA

Offset	Field	Size	Value	Description
0	bLength	1	09h	
1	bDescriptorType	1	02h	Configuration
2	wTotalLength	2	0054h	
4	bNumInterfaces	1	03h	
5	bConfigurationValue	1	01h	
6	iConfiguration	1	00h	
7	bmAttributes	1	A0h	Bus Powered, Remote Wakeup
	4..0: Reserved		...00000	
	5: Remote Wakeup		..1.....	Yes
	6: Self Powered		.0......	No, Bus Powered
	7: Reserved (set to one) (bus-powered for 1.0)		1.......	
8	bMaxPower	1	14h	40 mA

Interface descriptor

- Features that primarily describe specific endpoints
- A device with multiple interface descriptors, also known as a hybrid device
- Each HID device can have multiple interfaces
- Such as SmartLink also has a keyboard/mouse:
- Interfaces can be designed more abstractly, not necessarily filled with all th
- There's more detailed information on the bottom.



Interface Descriptor 0/0 HID, 1 Endpoint

Offset	Field	Size	Value	Description
0	bLength	1	09h	
1	bDescriptorType	1	04h	Interface
2	bInterfaceNumber	1	00h	
3	bAlternateSetting	1	00h	
4	bNumEndpoints	1	01h	
5	bInterfaceClass	1	03h	HID
6	bInterfaceSubClass	1	01h	Boot Interface
7	bInterfaceProtocol	1	01h	Keyboard
8	iInterface	1	03h	"Apple Internal Keyboard"

Interface Descriptor 1/0 HID, 1 Endpoint

Offset	Field	Size	Value	Description
0	bLength	1	09h	
1	bDescriptorType	1	04h	Interface
2	bInterfaceNumber	1	01h	
3	bAlternateSetting	1	00h	
4	bNumEndpoints	1	01h	
5	bInterfaceClass	1	03h	HID
6	bInterfaceSubClass	1	00h	
7	bInterfaceProtocol	1	00h	
8	iInterface	1	04h	"Touchpad"

Report descriptor

- Key technologies for HIDUSB devices
- There is only one report for one interface
- One report describes multiple actual functions
- When transferring data, the driver receives a piece of binary data
- Drivers rely on the report descriptor provided by the hardware to parse the corresponding data
- For example, the Nth byte represents X/Y coordinates or key state, etc.
-

USB Properties

Item Tag (Value)	Raw Data
Usage Page (Generic Desktop)	05 01
Usage (Mouse)	09 02
Collection (Application)	A1 01
Usage (Pointer)	09 01
Collection (Physical)	A1 00
Report ID (2)	85 02
Usage Page (Button)	05 09
Usage Minimum (Button 1)	19 01
Usage Maximum (Button 2)	29 02
Logical Minimum (0)	15 00
Logical Maximum (1)	25 01
Report Count (2)	95 02
Report Size (1)	75 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Report Count (1)	95 01
Report Size (6)	75 06
Input (Cnst,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 03
Usage Page (Generic Desktop)	05 01
Usage (X)	09 30
Usage (Y)	09 31
Usage (Wheel)	09 38
Logical Minimum (-127)	15 81
Logical Maximum (127)	25 7F
Report Size (8)	75 08
Report Count (3)	95 03
Input (Data,Var,Rel,NWrp,Lin,Pref,NNul,Bit)	81 06
Usage Page (Consumer Devices)	05 0C
Usage (AC Pan)	0A 38 02
Logical Minimum (-127)	15 81
Logical Maximum (127)	25 7F
Report Size (8)	75 08
Report Count (1)	95 01
Input (Data,Var,Rel,NWrp,Lin,Pref,NNul,Bit)	81 06
End Collection	C0
End Collection	C0
Usage Page	05 FF
Usage	09 01
Collection (Physical)	A1 00
Usage	09 02
Logical Minimum (1)	15 01
Logical Maximum (65)	25 41
Physical Minimum (0)	35 00
Physical Maximum (-1)	45 FF
Report ID (68)	85 44
Report Size (8)	75 08
Report Count (63)	95 3F
Input (Data,Ary,Abs)	81 00
Usage	09 04
Logical Minimum (-128)	15 80
Logical Maximum (127)	25 7F
Report Size (8)	75 08
Report Count (16)	95 10
Feature (Data,Var,Abs,NWrp,Lin,Pref,NNul,NVol,Bit)	B1 02
End Collection	C0

Report descriptor

- Usage Page - Apps for Devices
- Usage - The actual functionality of the app
- Report Count - Bit amount for a Report field
- Report Size - How many Report fields are complete this Usage
- Input (...) - Represents a beam description

Interface 0 HID Report Descriptor Keyboard

Item Tag (Value)	Raw Data
Usage Page (Generic Desktop)	05 01
Usage (Keyboard)	09 06
Collection (Application)	A1 01
Usage Page (Keyboard/Keypad)	05 07
Usage Minimum (Keyboard Left Control)	19 E0
Usage Maximum (Keyboard Right GUI)	29 E7
Logical Minimum (0)	15 00
Logical Maximum (1)	25 01
Report Count (8)	95 08
Report Size (1)	75 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Report Count (8)	95 08
Report Size (1)	75 01
Input (Cnst,Ary,Abs)	81 01
Usage Page (LEDs)	05 08
Usage Minimum (Num Lock)	19 01
Usage Maximum (Scroll Lock)	29 03
Report Count (3)	95 03
Report Size (1)	75 01
Output (Data,Var,Abs,NWrp,Lin,Pref,NNul,NVol,Bit)	91 02
Report Count (1)	95 01
Report Size (5)	75 05
Output (Cnst,Ary,Abs,NWrp,Lin,Pref,NNul,NVol,Bit)	91 01
Usage Page (Keyboard/Keypad)	05 07
Usage Minimum (Undefined)	19 00
Usage Maximum	2A FF 00
Logical Minimum (0)	15 00
Logical Maximum (255)	26 FF 00
Report Count (6)	95 06
Report Size (8)	75 08
Input (Data,Ary,Abs)	81 00
End Collection	C0

URB	0044-0041	16:58:23.112	43.625...	583.922 ms	Bulk or Interrupt Transfer	Input Report len:8	
URB	0045	16:58:23.112	43.625...		Bulk or Interrupt Transfer	8 bytes buffer	
URB	0046-0043	16:58:23.136	43.649...	527.949 ms	Bulk or Interrupt Transfer	Input Report len:8	04 00 2B 00 00 00 00 00
URB	0047	16:58:23.136	43.649...		Bulk or Interrupt Transfer	8 bytes buffer	
URB	0048-0045	16:58:23.200	43.713...	87.969 ms	Bulk or Interrupt Transfer	Input Report len:8	00 00 2B 00 00 00 00 00
URB	0049	16:58:23.200	43.713...		Bulk or Interrupt Transfer	8 bytes buffer	
URB	0050-0047	16:58:23.216	43.729...	80.002 ms	Bulk or Interrupt Transfer	Input Report len:8	00 00 00 00 00 00 00 00
URB	0051	16:58:23.216	43.729...		Bulk or Interrupt Transfer	8 bytes buffer	

Endpoint descriptor

- An interface can have multiple endpoints, such as a keyboard
- Endpoints are used to transmit data, and the specified endpoints and interfaces are determined
- Its descriptor also describes the speed at which the packet is transmitted and information such as the IO address
- One EndPoint for each PipeHandle under Windows
-

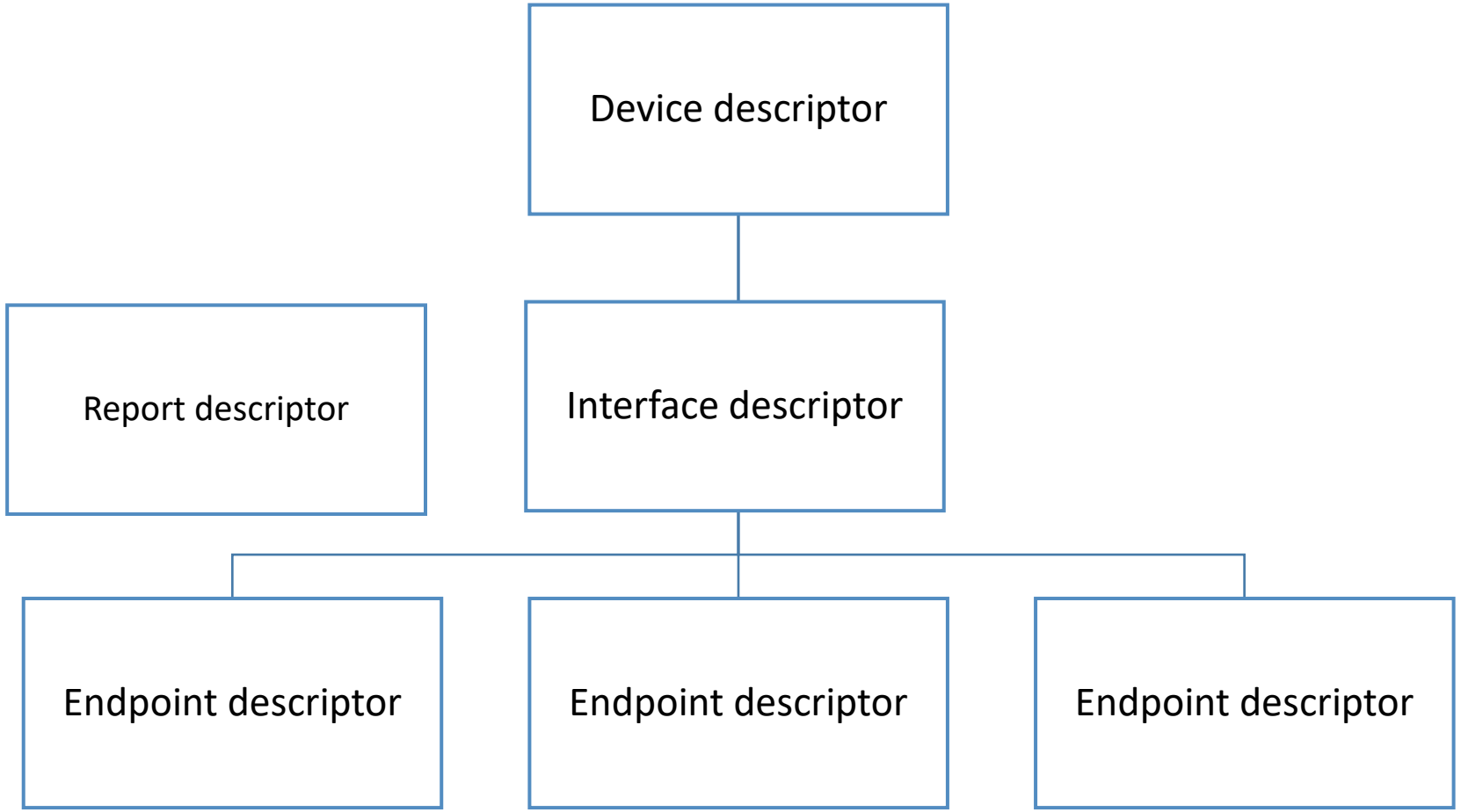
Endpoint Descriptor 81 1 In, Interrupt, 2 ms

Offset	Field	Size	Value	Description
0	bLength	1	07h	
1	bDescriptorType	1	05h	Endpoint
2	bEndpointAddress	1	81h	1 In
3	bmAttributes	1	03h	Interrupt
	1..0: Transfer Type		11	Interrupt
	7..2: Reserved		000000..	
4	wMaxPacketSize	2	0040h	64 bytes
6	bInterval	1	02h	2 ms

Endpoint Descriptor 84 4 In, Interrupt, 8 ms

Offset	Field	Size	Value	Description
0	bLength	1	07h	
1	bDescriptorType	1	05h	Endpoint
2	bEndpointAddress	1	84h	4 In
3	bmAttributes	1	03h	Interrupt
	1..0: Transfer Type		11	Interrupt
	7..2: Reserved		000000..	
4	wMaxPacketSize	2	0008h	8 bytes
6	bInterval	1	08h	8 ms

Basic relationship between USBHID descriptors



Drive basic callback functions

```
NTSTATUS
AddDevice(
_In_ struct _DRIVER_OBJECT *DriverObject,
_In_ struct _DEVICE_OBJECT *PhysicalDeviceObject
);
```

AddDevice

- The PnP manager reads the registry and loads and invokes its driven AddDevice to indicate that a device has been successfully plugged in

- After receiving the AddDevice notification, the driver needs to create the upper device object and attach it to the underlying device object.

I/O Request Packet (IRP)

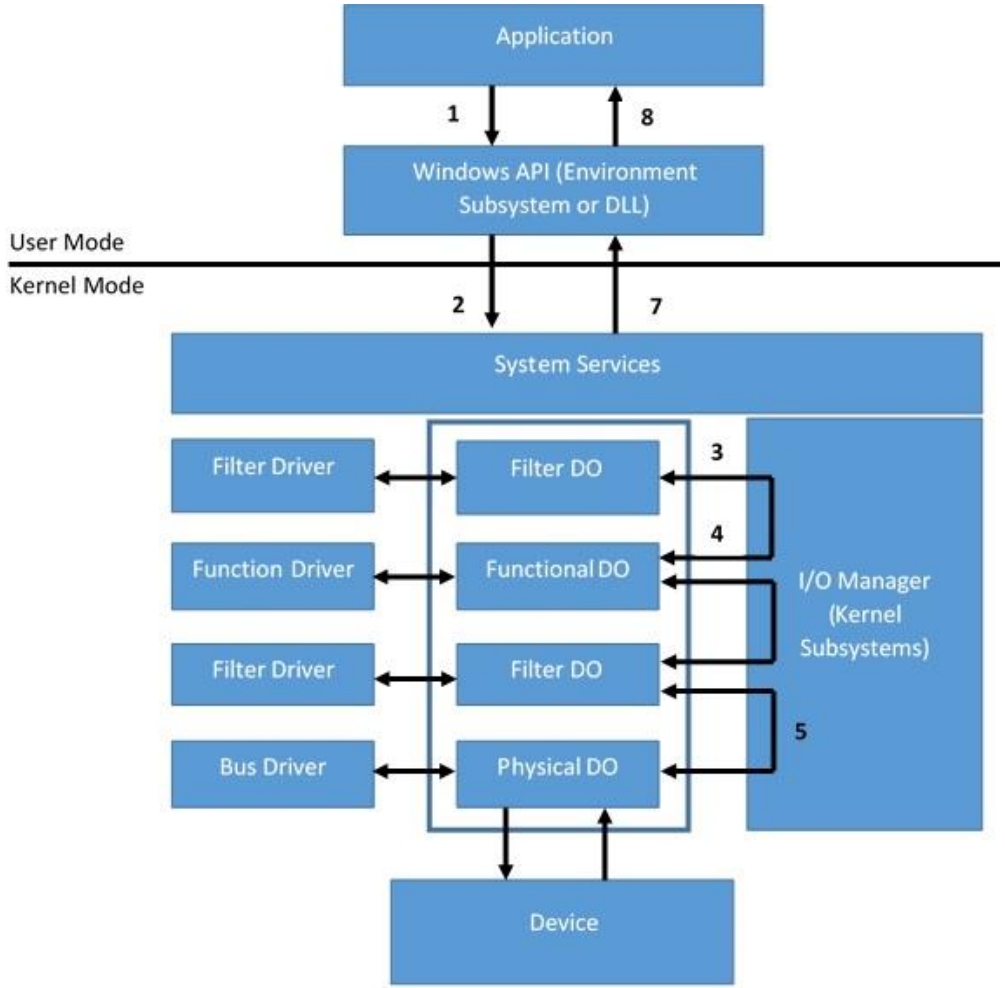
- Responsible for communication between drivers

- IRP is top-down, has target devices, common LYIs such as CreateFile, ReadFile, etc.

- These requests are converted into IRP requests synchronously or abnormally when they enter the kernel

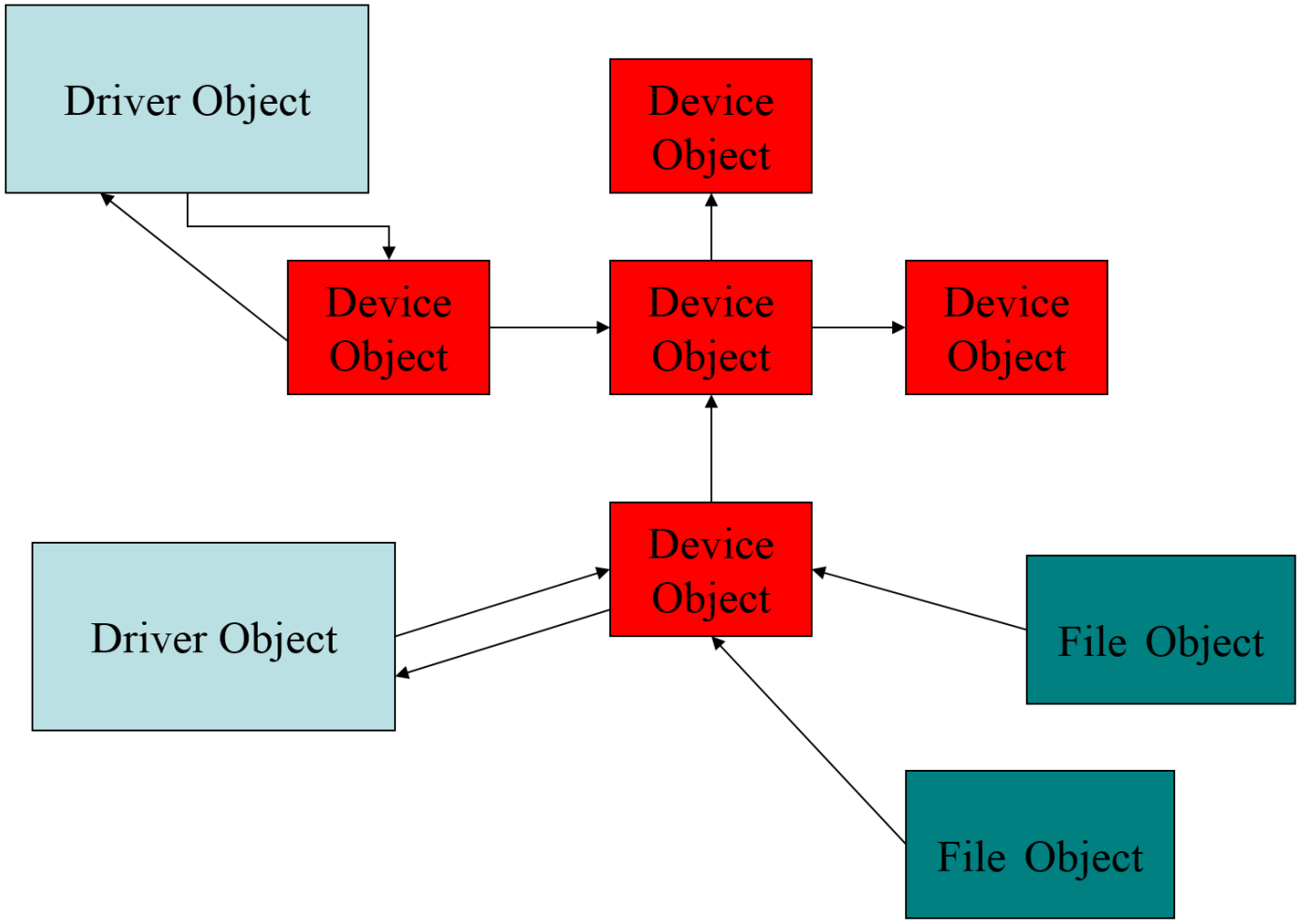
- Drivers can be delivered all the way to the lower drive using IoCallDriver

- Until either driver calls IoCancelIrp or IoCompleteRequest to cancel or complete the request



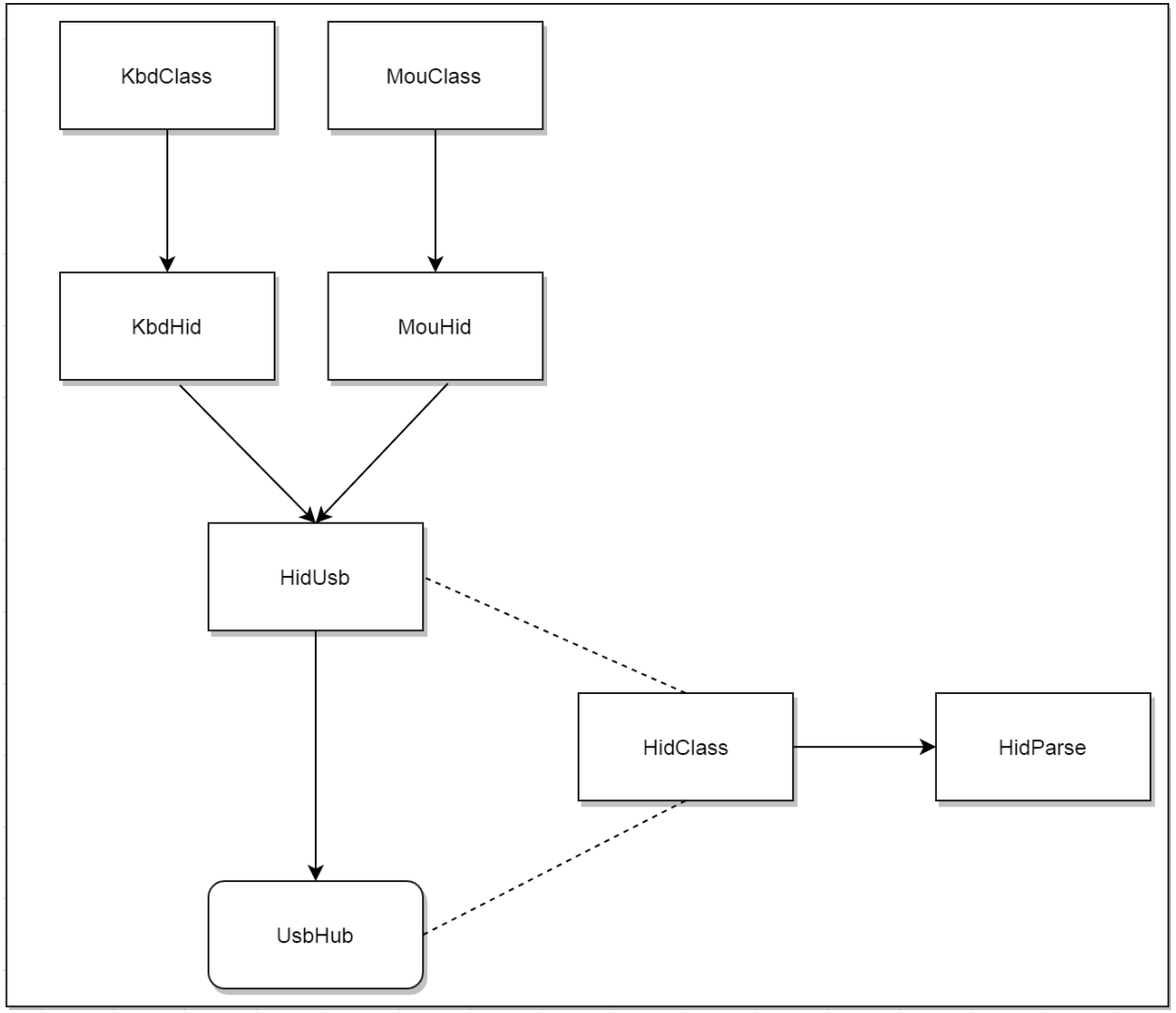
Drive device stack

- A driver can have more than one device object
- Each device object is serialised using the NextDevice domain
- The device object created by the upper-level driver attaches the device object of the underlying driver, forming the device stack
- Upper layer can generate open/read/READ/IO requests to device objects through file objects, etc.
-



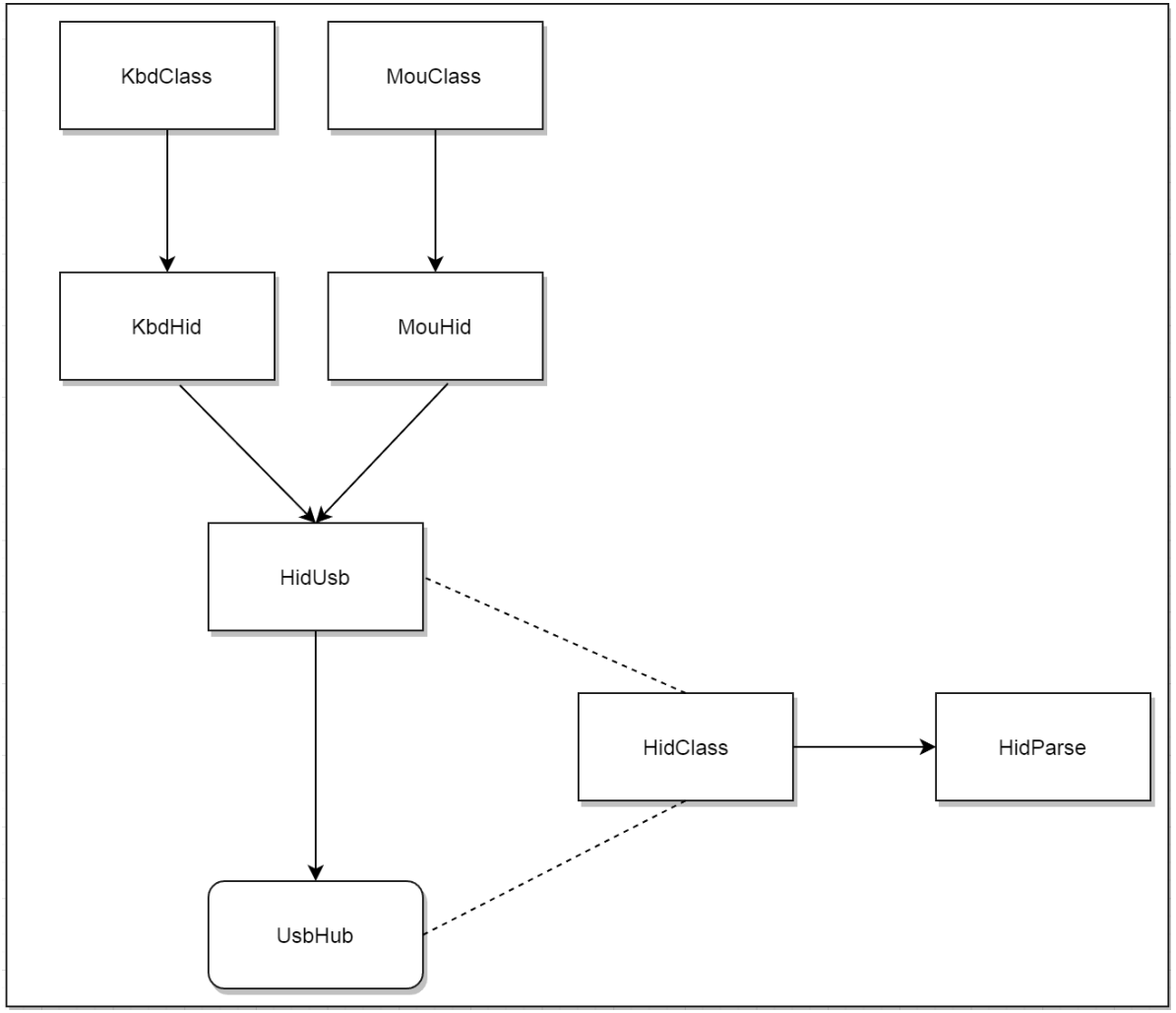
Windows Input System – Mice/Keyboard

- KbdClass , MouClass is responsible for all keyboard/mouse devices in the input system, interacting with the upper layer.
-
- KbdHid, MouHid is responsible for interacting with HID class drivers, such as receiving data, converting data formats, etc., to convert the data obtained by hid into the format required by the system
-
- HIDUSB manages all systems that can be used for HIDUSB class devices, with keyboard/mouse, to sense which upper-level driver needs to be loaded
-



Windows Input System – Mice/Keyboard

- HIDClass concentrates on simplifying all HID-class devices - not limited to USB
-
- HIDParse provides the most commonly used external drivers for HIDClass with the ability to resolve the deconstruction-related functions of HID descriptions
-
- HIDClass's chain of behaviors is also transparent to the upper and lower classes
-
- Manufacturers only need to provide a standard HID descriptor, the system can resolve and adapt
-



HID Transport MiniDriver

- Simply call HidRegisterMiniDriver to HIDCLASS to register as one of the hidMiniDrivers in the system, as provided by the system HidUsb.sys

```
NTSTATUS HidRegisterMinidriver(
  _In_ PHID_MINIDRIVER_REGISTRATION MinidriverRegistration
);
```

```
typedef struct _HID_MINIDRIVER_REGISTRATION {
  ULONG           Revision;
  PDRIVER_OBJECT  DriverObject;
  PUNICODE_STRING RegistryPath;
  ULONG           DeviceExtensionSize;
  BOOLEAN         DevicesArePolled;
  UCHAR           Reserved[3];
} HID_MINIDRIVER_REGISTRATION, *PHID_MINIDRIVER_REGISTRATION;
```

-
- MiniDriver needs to provide the system with its own drive objects, as well as device-related information, such as whether the device is polled
- HIDClass automatically requests all descriptors from the hardware, resolves them well, and creates corresponding functional device objects (such as keyboard/mouse...) for the upper layer. etc.)
- HIDClass provides features so that upper-level drivers don't need to understand that Windows is the underlying system of the Usb stack
- The upper and lower control flows can be taken over.
- Good support for HID device abstraction

How HIDClass works

- HidUsb is just one of the HID Transport MiniDrivers, or MiniDrivers, preset under Windows
- HidUSB is mostly supported by HidClass.
- Its main workflow is divided into two parts:
- IRP Hook
- AddDevice Hook
- All IRP in MiniDriver is redirected to HIDClass driver takeover
- HIDClass uses the drive objects provided by MiniDriver
- For MiniDriver, all I/O operations are also after-knowledge, need to wait for HIDClass to complete before arriving at their own post-operation
-
-

Code prototype of HidRegisterMiniDriver:

```
NTSTATUS HidRegisterMinidriver(
  _In_ PHID_MINIDRIVER_REGISTRATION MinidriverRegistration
);
```

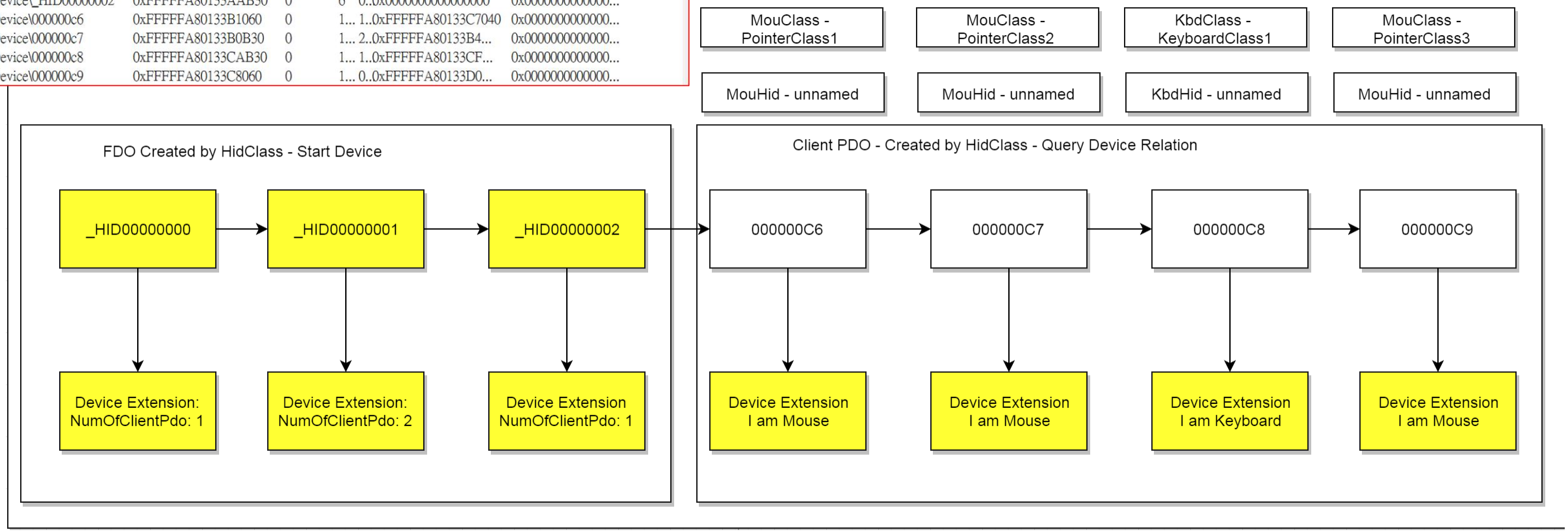
HidRegister MiniDriver's code snippets:

```
RtlCopyUnicodeString((PUNICODE_STRING)((char *)v3 + 8), v1->RegistryPath);
HidpGetFastResumeDisableState(DriverObjectExtension);
v6 = v1->DriverObject;
memmove((char *)DriverObjectExtension + 32, v6->MajorFunction, 0xE0ui64);
v7 = v6->DriverExtension;
*((_BYTE *)DriverObjectExtension + 296) = v1->DevicesArePolled;
v6->MajorFunction[23] = (PDRIVER_DISPATCH)&HidpMajorHandler;
v6->MajorFunction[4] = (PDRIVER_DISPATCH)&HidpMajorHandler;
v6->MajorFunction[3] = (PDRIVER_DISPATCH)&HidpMajorHandler;
v6->MajorFunction[22] = (PDRIVER_DISPATCH)&HidpMajorHandler;
v6->MajorFunction[27] = (PDRIVER_DISPATCH)&HidpMajorHandler;
v6->MajorFunction[15] = (PDRIVER_DISPATCH)&HidpMajorHandler;
v6->MajorFunction[14] = (PDRIVER_DISPATCH)&HidpMajorHandler;
v6->MajorFunction[0] = (PDRIVER_DISPATCH)&HidpMajorHandler;
v6->MajorFunction[2] = (PDRIVER_DISPATCH)&HidpMajorHandler;
*((_QWORD *)DriverObjectExtension + 32) = v7->AddDevice;
v7->AddDevice = (PDRIVER_ADD_DEVICE)HidpAddDevice;
*((_QWORD *)DriverObjectExtension + 33) = v6->DriverUnload;
v6->DriverUnload = (PDRIVER_UNLOAD)HidpDriverUnload;
*((_DWORD *)DriverObjectExtension + 68) = 0;
if ( !(unsigned __int8)EnqueueDriverExt(DriverObjectExtension) )
  v2 = -1073741438;
```

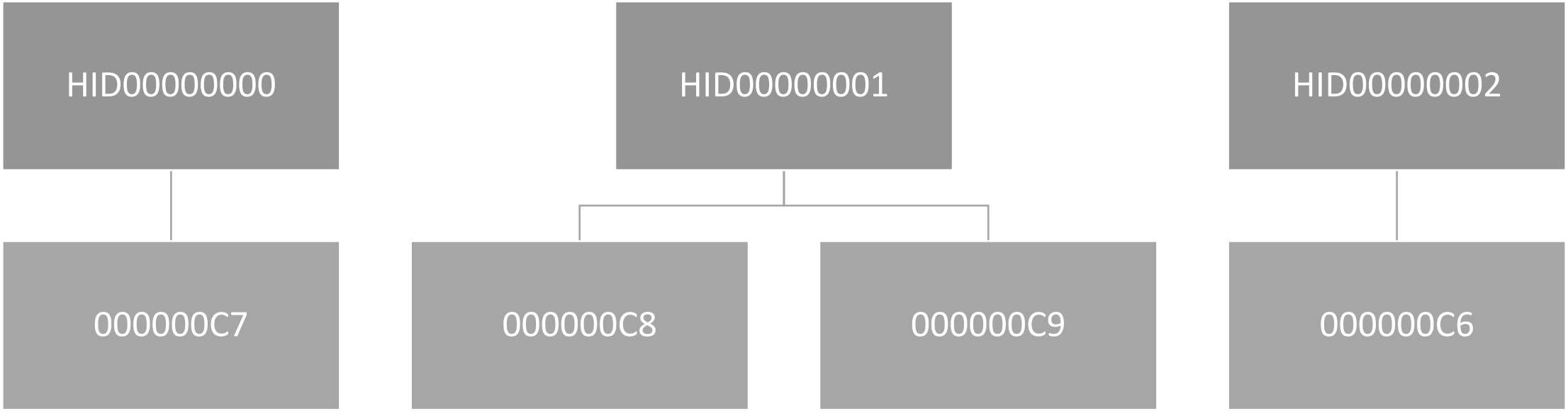
HIDUSB Device stack

Device List:

Device_Name	Device Object	Handles	P..	R	Attached	FSD
\Device_HID00000000	0xFFFFFA80133A1760	0	6		0..0x0000000000000000	0x0000000000000...
\Device_HID00000001	0xFFFFFA80133A8360	0	6		0..0x0000000000000000	0x0000000000000...
\Device_HID00000002	0xFFFFFA80133AAB30	0	6		0..0x0000000000000000	0x0000000000000...
\Device\000000c6	0xFFFFFA80133B1060	0	1...		1..0xFFFFFA80133C7040	0x0000000000000...
\Device\000000c7	0xFFFFFA80133B0B30	0	1...		2..0xFFFFFA80133B4...	0x0000000000000...
\Device\000000c8	0xFFFFFA80133CAB30	0	1...		1..0xFFFFFA80133CF...	0x0000000000000...
\Device\000000c9	0xFFFFFA80133C8060	0	1...		0..0xFFFFFA80133D0...	0x0000000000000...



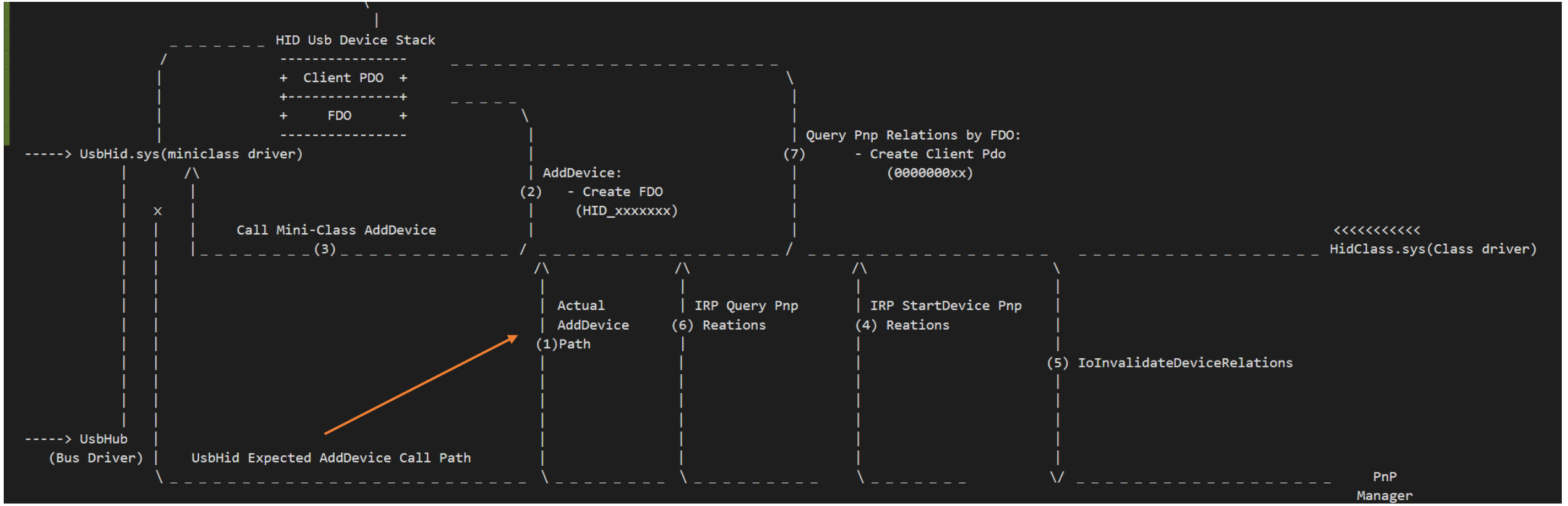
HIDUSB actual device relationship



The process of establishing hidUSB device stack

HIDUSB's AddDevice

- AddDevice was intercepted by HIDClass, and only HIDClass enables HID-class devices to simply receive Notification from AddDevice
- HIDClass uses its MiniDriver (HidUsb) drive object to create the FDO (HID000000X) it represents
-



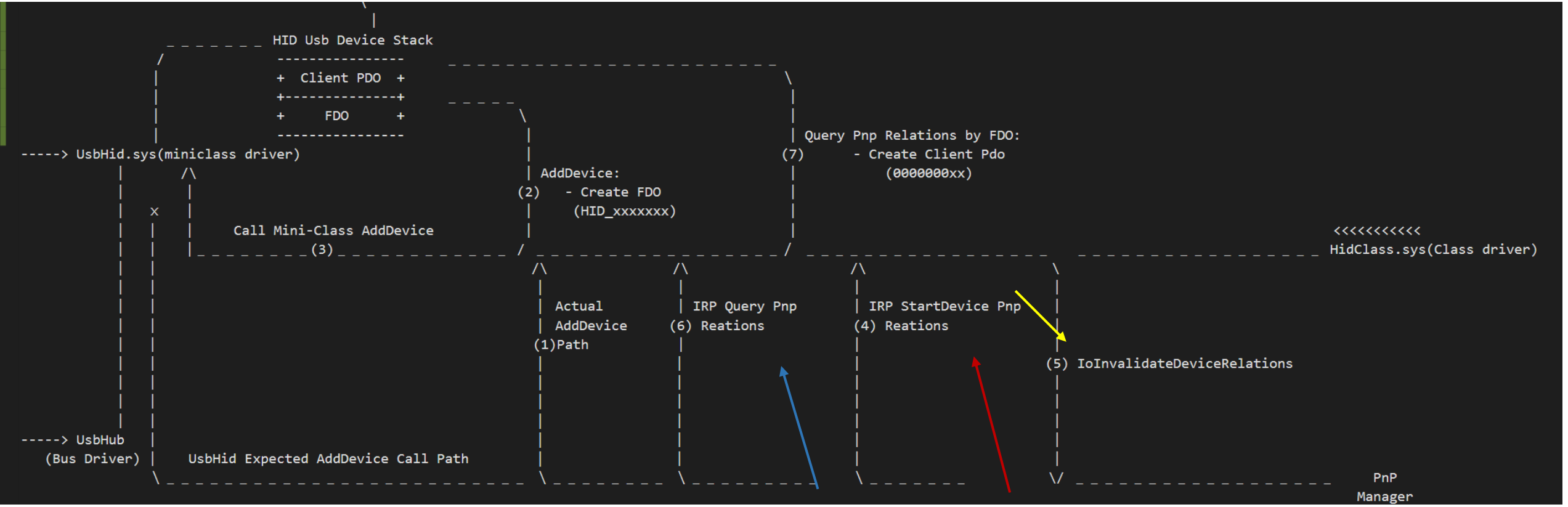
The process of establishing hidUsb device stack

HIDUSB's Start IRP

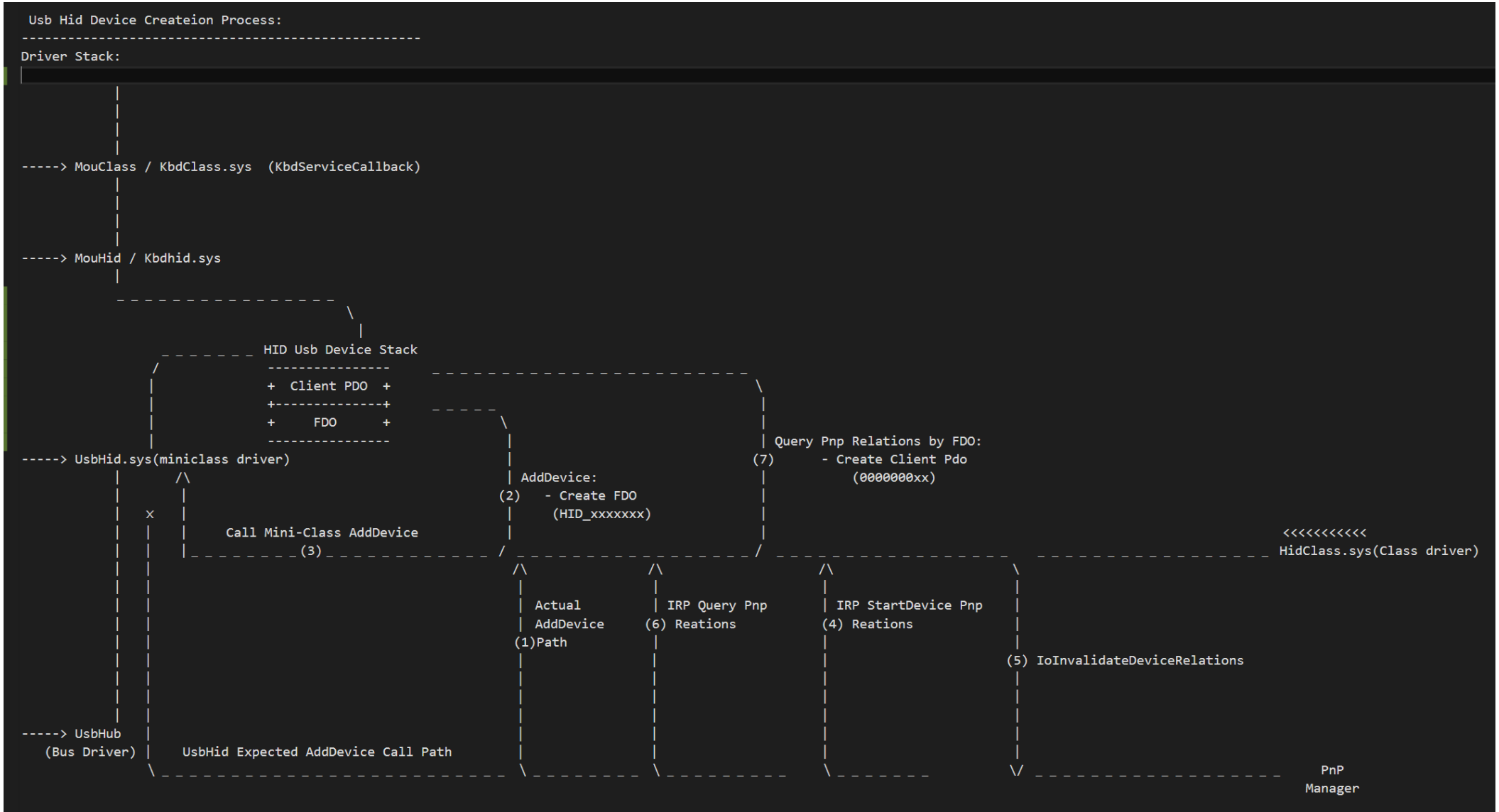
- HidClass intercepts its IRP, and assists HidUsb in initializing its FDO device extension, as well as Client PDO data, and finally calls IoInvalidateDeviceRelations to notify PnP Manager

HIDUSB's BusRelation IRP

– PnP Manager sends BusRelation IRP request, HIDClass intercepts again, and enumerates all of its Client PDO



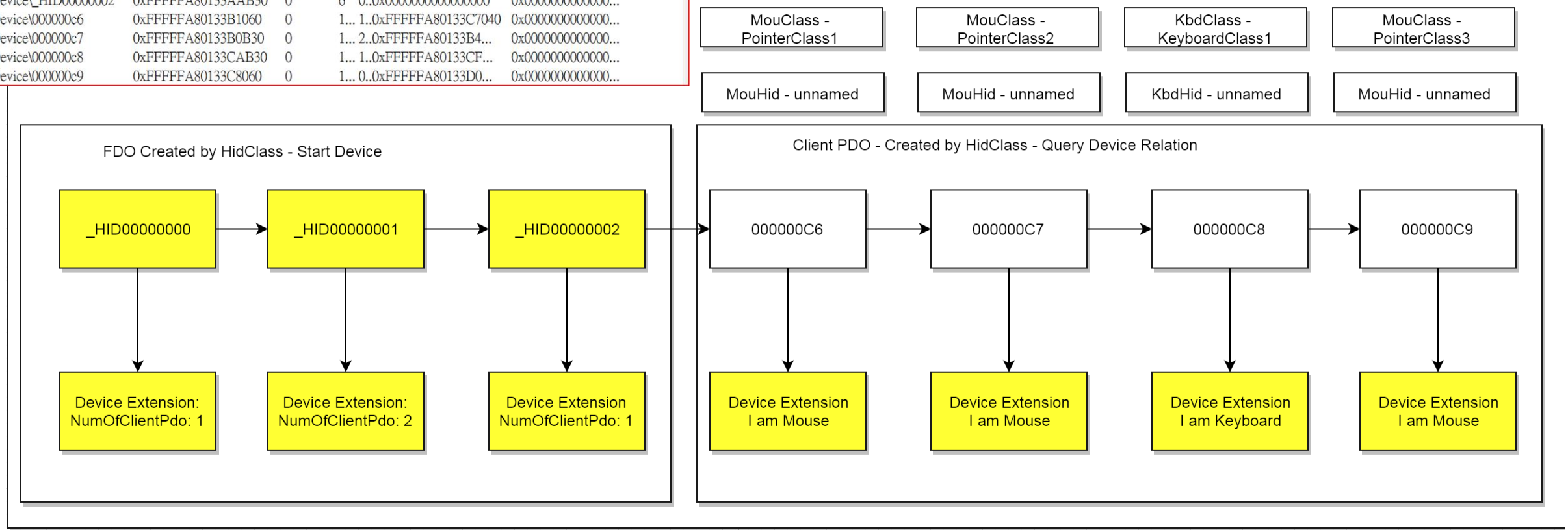
The process of establishing hidUsb device stack



HidUsb’s special Device Stack

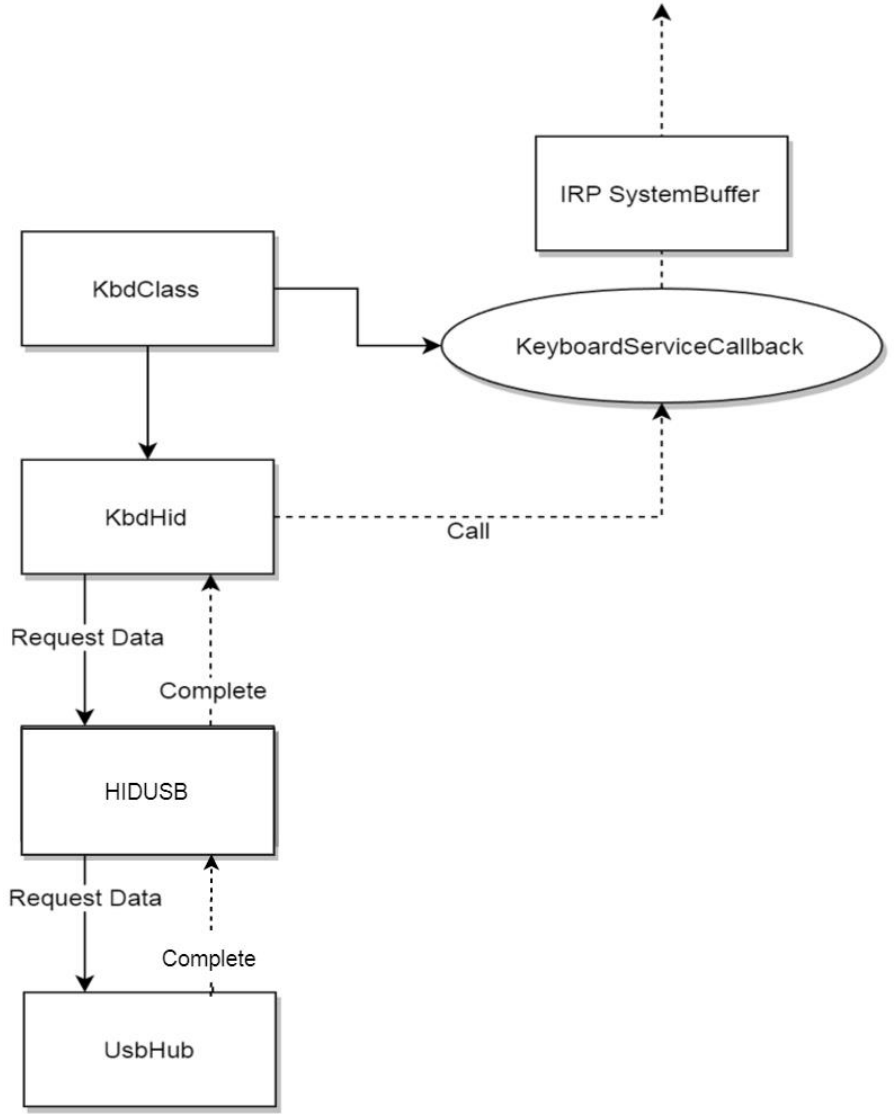
Device List:

Device_Name	Device Object	Handles	P..	R	Attached	FSD
\Device_HID00000000	0xFFFFFA80133A1760	0	6		0..0x0000000000000000	0x0000000000000...
\Device_HID00000001	0xFFFFFA80133A8360	0	6		0..0x0000000000000000	0x0000000000000...
\Device_HID00000002	0xFFFFFA80133AAB30	0	6		0..0x0000000000000000	0x0000000000000...
\Device\000000c6	0xFFFFFA80133B1060	0	1...		1..0xFFFFFA80133C7040	0x0000000000000...
\Device\000000c7	0xFFFFFA80133B0B30	0	1...		2..0xFFFFFA80133B4...	0x0000000000000...
\Device\000000c8	0xFFFFFA80133CAB30	0	1...		1..0xFFFFFA80133CF...	0x0000000000000...
\Device\000000c9	0xFFFFFA80133C8060	0	1...		0..0xFFFFFA80133D0...	0x0000000000000...



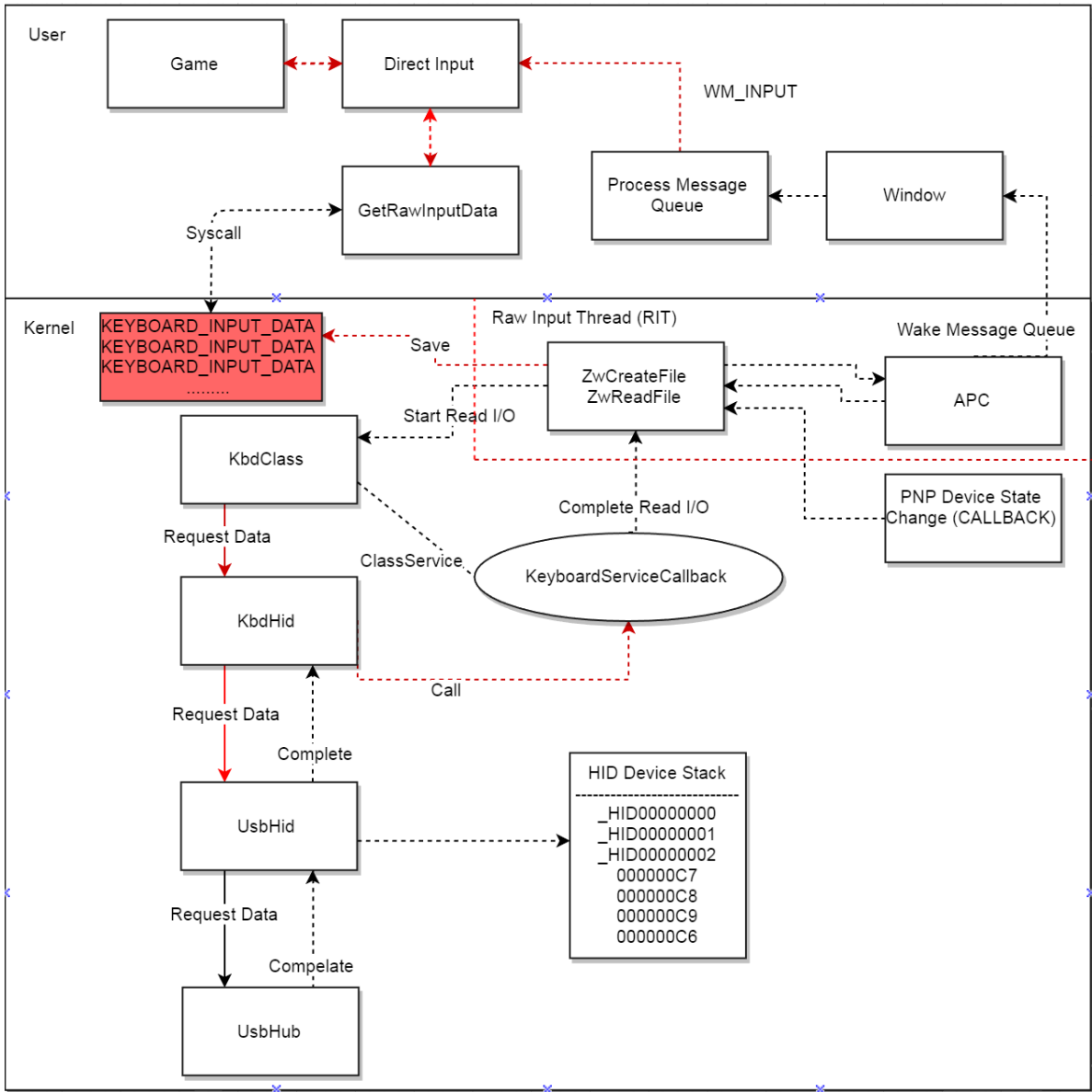
The data stream of the keyboard in the kernel

- KbdClass Create a read IRP request and process I/O through StartIo serialization
- KbdHid Start creating a loop to read data, finally calling ClassService to cancel IRP, and copying data to the I/O buffer
- UsbHid Complete KBDHID request
- UsbHub Complete HIDUSB request
-



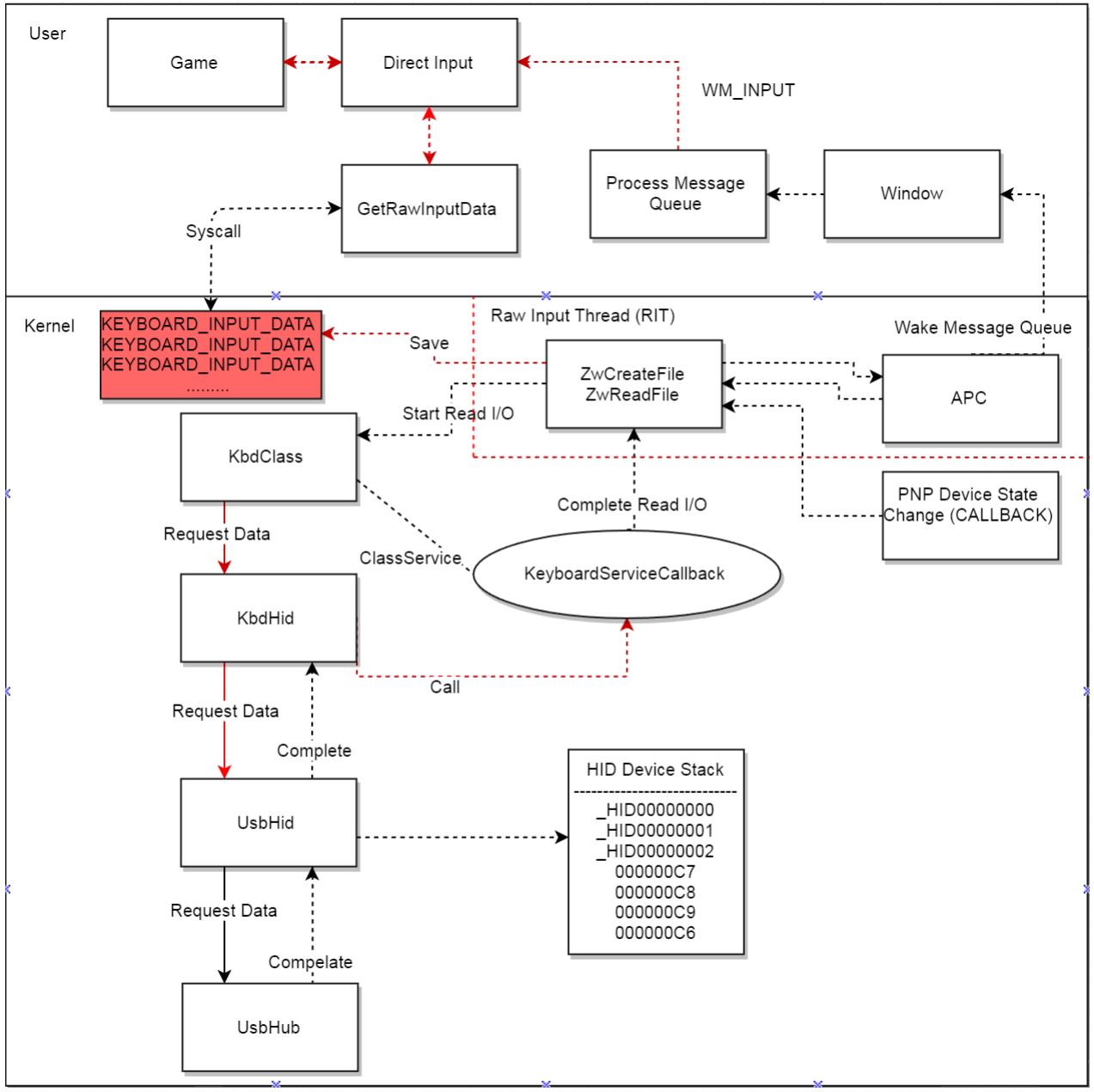
Kernel-to-application-level global resolution

- Raw Input Thread (RIT) is WIN32K's kernel thread responsible for sending and receiving input data
- After the keyboard is plugged in, the PNP Manager notifies the Win32k subsystem, starts the first ZwReadFile and arranges the insertion of the APC
- In fact, ZwReadFile will create an IRP delivery to KbdClass and wait until it's complete
- ZwReadFile arranges two jobs at the same time
- APC distributes messages to the application layer;
- Update to kernel buffer after IRP is complete
- KbdClass serializes all IRP series sits when it receives IRP
- Until KbdHid's Class Service is called, complete IRP
- Until UsbHub completes the above request, RIT will be able to store two pieces of data, one for RawInput's buffer in the kernel, and the other for the thread waiting for the wake-up message.
- (1) Call GetRaw Input Data to get
- (2) Call GetMessage / PeekMessage to get



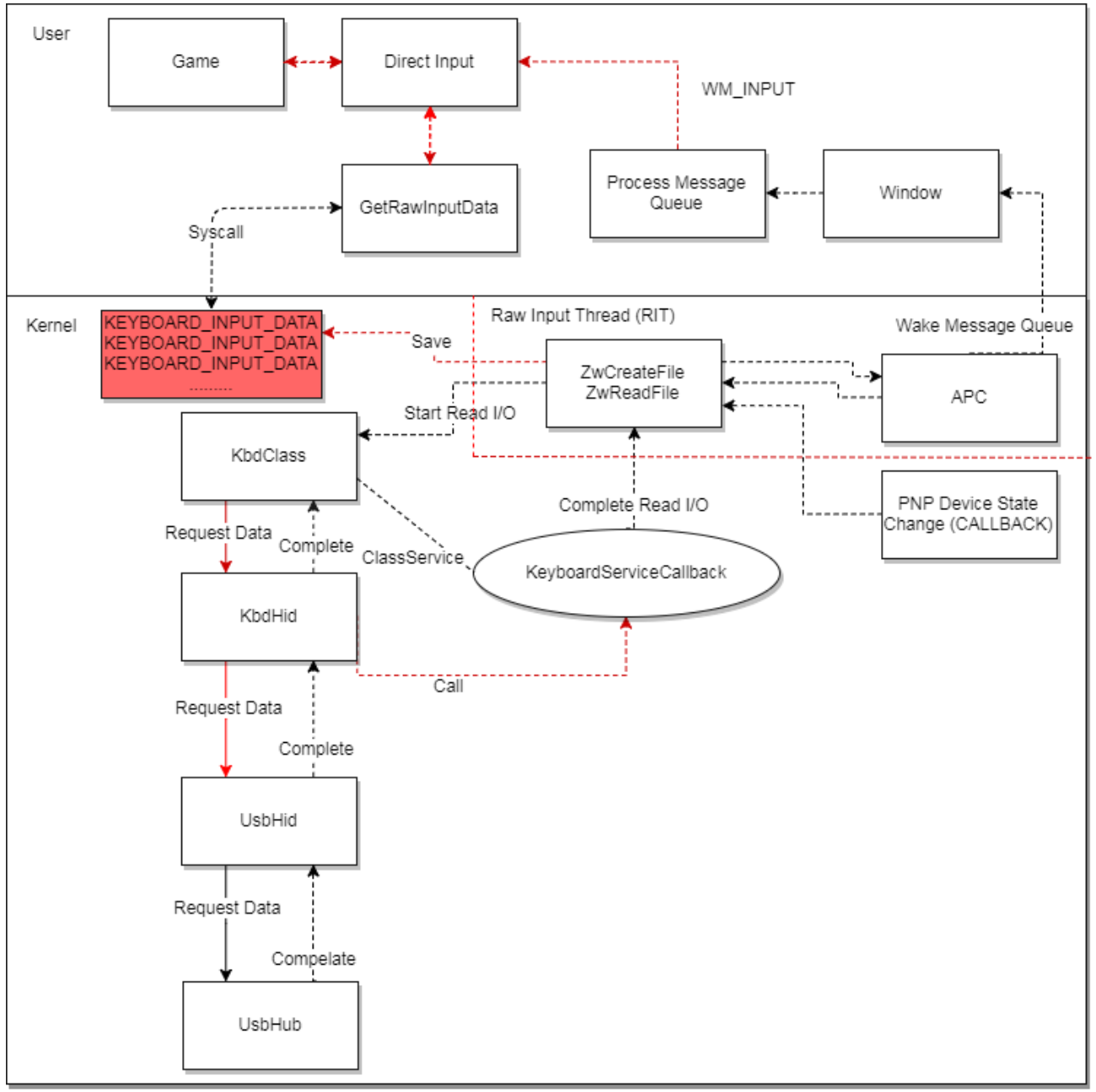
DirectInput

- The game uses DirectInput as an input data source
- DirectInput calls RegisterInput Device to register RawInput's data message (WM_INPUT)
- DirectInput calls GetRawInputData to buffer the kernel with a copy of the data when it receives a WM_INPUT message
- Finalbuffer to direct input device private buffer
- Waiting for the game to read
-



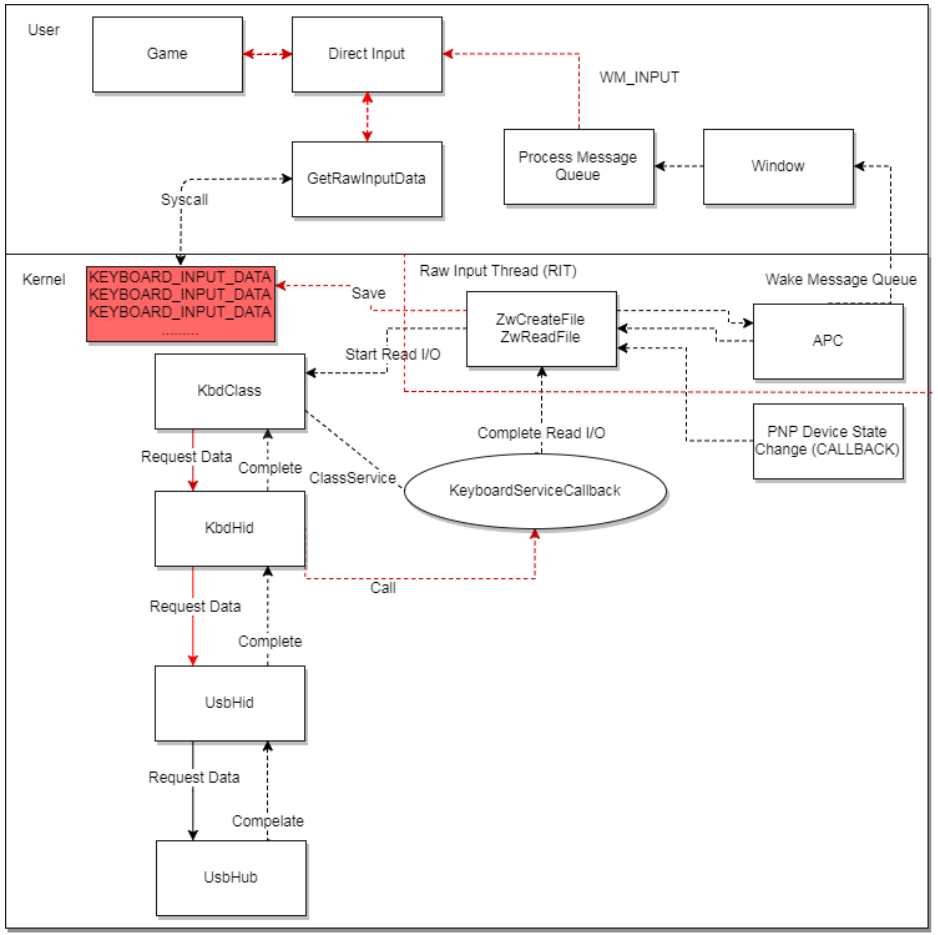
Cheat Attack Surface

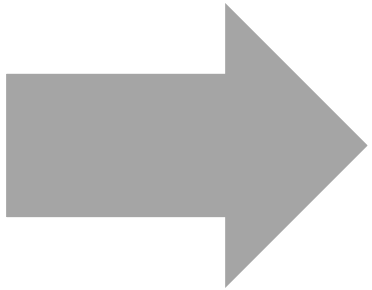
- Insert data into DirectInput buffer
- Insert data into RawInput buffer
- Simulated call Class Service
- Insert data into kernel buffer
- Insert your own filter driver in the Process of RequestData to change the package
- Virtual HID device, construct a usbhid of your own
- The common practice of simulated key input, such as SendInput, SendMessage, PostMessage is actually based on the above concept that data cache insertion is the one you want to simulate by data
- ….

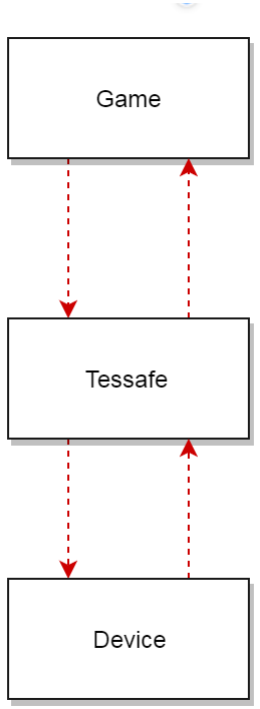


Solution

- Self-built input system, allowing the game to use a self-implemented private input channel







Summarize

- The analog keys have a lot to do with Windows' input system itself.
- The complexity of the input system itself increases the attack surface against the analog keys
- There's more attack on the outside.
- Understanding the underlying architecture helps to combat the various categories of external slots

End