键盘驱动系列---JIURL键盘驱动 4

转载

cosmoslife 于 2012-08-10 23:04:35 发布 1592 收藏
分类专栏： 驱动开发学习 文章标签： keyboard struct object initialization c system

6.2 初始化与注册表

在驱动的初始化中，注册表起着非常重要的作用。

6.2.1 重要认识

每台计算机的硬件配置可能是不同的，系统是如何知道需要为哪些硬件载入驱动？系统是如何知道要载入的驱动文件是哪个文件？答案是通过注册表，注册表中保存着这些信息。那么注册表中的这些信息是哪里来的？答案是在安装驱动的时候，安装程序放到注册表中的。

6.2.2 确定驱动载入顺序的基本知识

在系统初始化的时候，决定驱动程序在什么时候被载入的信息保存在注册表中。最早的一批驱动是由ntldr载入内存的（仅仅是载入），第二批是由IO管理器载入内存的，第三批是由 SCM(Service Control Manager) 载入的。一个驱动在第几批中被载入是由 HKLM\SYSTEM\CurrentControlSet\Services\驱动名\Start 的值来决定。该值为0，第一批被载入。该值为1，第二批被载入。该值为2，第三批被载入。对于同一批驱动中的驱动，按驱动所在组的先后载入。组的先后顺序由 HKLM\SYSTEM\CurrentControlSet\Control\ServiceGroupOrder\List 决定。每个驱动的 HKLM\SYSTEM\CurrentControlSet\Services\驱动名\Group 决定了驱动所属的组，如果没有这个 Services\驱动名\Group ，那么就在所有组之后。对于同一个组中的驱动，按驱动的Tag的先后载入。Tag的先后顺序由 HKLM\SYSTEM\CurrentControlSet\Control\GroupOrderList\组名 决定。每个驱动的 HKLM\SYSTEM\CurrentControlSet\Services\驱动名\Tag 决定了驱动在组中的Tag。

关于系统初始化过程和如何确定驱动载入的顺序的更详细内容，可以参考下面两个资料：

Inside the Boot Process Part 1 ， Inside the Boot Process Part 2 。

http://osg.informatik.tu-chemnitz.de/lehre/seminar/lwb/Doku/SystemInitiation.pdf 。

"6.2.2 确定驱动载入顺序的基本知识"的内容就是从这两个资料中翻译的。

6.2.3 注册表与设备栈

设备栈的建立，是靠注册表中的信息。这些信息放在以下两个键下。

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class\

具体请看，稍后对注册表中决定键盘设备栈的信息的讨论。

参考 ...\NTDDK\src\general\toaster\toaster.htm

6.2.4 驱动文件的路径

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\驱动名\ImagePath

6.2.5 键盘驱动与注册表

6.2.5.1 键盘驱动载入顺序

查看注册表中决定键盘驱动 i8042prt, kbdclass 载入顺序的内容。

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\i8042prt]

"Type"=dword:00000001

"Start"=dword:00000001

"Group"="Keyboard Port"

"ErrorControl"=dword:00000001

"DisplayName"="i8042 Keyboard and PS/2 Mouse Port Driver"

"ImagePath"="System32\DRIVERS\i8042prt.sys"

"Tag"=dword:00000004

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Kbdclass]

"ErrorControl"=dword:00000001

"Group"="Keyboard Class"

"Start"=dword:00000001

"Tag"=dword:00000001

"Type"=dword:00000001

"DisplayName"="Keyboard Class Driver"

"ImagePath"="System32\DRIVERS\kbdclass.sys"

两个的 Start 类型都是 1，即 SERVICE_SYSTEM_START ，所以他们都将由 IO 管理器载入。还可以看到 i8042prt 属于组 "Keyboard Port" ，kbdclass 属于组 "Keyboard Class" 。

查看这两个组的先后顺序。组的先后顺序保存在 HKEY_LOCAL_MACHINE\CurrentControlSet\Control\ServiceGroupOrder\List 中，由于使用 regedit.exe 来看这个值不是很方便，我们使用系统中另一个注册表编辑器regedt32.exe 。 可以看到下面的内容。

System Reserved

...

Keyboard Port

Pointer Class

Keyboard Class

...

MS Transactions

组 "Keyboard Port" 在 组 "Keyboard Class" 之前，所以 i8042prt 在 kbdclass 之前被载入。

确定了顺序之后，HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\驱动名\ImagePath 是驱动程序文件的路径，通过这个值，就可以在硬盘上找到驱动程序文件，也就可以把它读入内存。

6.2.5.2 键盘设备栈与注册表

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\ACPI\PNP0303\4&5289e18&0]

"Capabilities"=dword:00000020

"HardwareID"=hex(7):41,00,43,00,50,00,49,00,5c,00,50,00,4e,00,50,00,30,00,33,\

00,30,00,33,00,00,00,2a,00,50,00,4e,00,50,00,30,00,33,00,30,00,33,00,00,00,\

00,00

"Service"="i8042prt"

"ClassGUID"="{4D36E96B-E325-11CE-BFC1-08002BE10318}"

"ConfigFlags"=dword:00000000

"Driver"="{4D36E96B-E325-11CE-BFC1-08002BE10318}\\0000"

"Mfg"="(标准键盘)"

"DeviceDesc"="Standard 101/102-Key or Microsoft Natural PS/2 Keyboard"

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class\{4D36E96B-E325-11CE-BFC1-08002BE10318}]

"Class"="Keyboard"

"Icon"="-3"

"Installer32"="SysSetup.Dll,KeyboardClassInstaller"

"UpperFilters"="kbdclass"

@="键盘"

"NoInstallClass"="1"

"TroubleShooter-0"="tshoot.chm,hdw_keyboard.htm"

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\ACPI\PNP0303\4&5289e18&0\Service

通过这里找到键盘设备栈中的 i8042prt

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\ACPI\PNP0303\4&5289e18&0\ClassGUID

通过这里找到类的GUID，用这个GUID就可以在 HKLM\SYSTEM\CurrentControlSet\Control\Class\ 下找到键盘驱动类，进而找到键盘类的过滤程序

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class\{4D36E96B-E325-11CE-BFC1-08002BE10318}\UpperFilters

通过这里找到键盘设备栈中的 kbdclass

6.3 键盘驱动初始化分析

6.3.1 键盘驱动本身的初始化部分

系统初始化进行到下图所示阶段

[i8042prt!DriverEntry]

此时的 Call Stack

ChildEBP RetAddr Args to Child

00 f901f510 804a4431 fe4f69f0 fe4f5000 00000000 i8042prt!DriverEntry(struct _DRIVER_OBJECT * DriverObject = 0xfe4f69f0, struct _UNICODE_STRING * RegistryPath = 0xfe4f5000) (CONV: stdcall)

01 f901f5d8 80426f6d 80000044 fe4f5000 f901f6e0 nt!IopLoadDriver+0x672 (FPO: [Non-Fpo])

02 f901f608 8049ccaa fe4f7348 80000044 00000001 nt!IopCallDriverAddDeviceQueryRoutine+0x356 (FPO: [Non-Fpo])

03 f901f654 80497df1 00000012 00000001 f901f6b0 nt!RtlpCallQueryRegistryRoutine+0x349 (FPO: [Non-Fpo])

04 f901f6b8 804a4ac0 00000000 00000082 00000001 nt!RtlQueryRegistryValues+0x1ed (FPO: [Non-Fpo])

05 f901f77c 8054f1bb 80000048 00000001 f901f864 nt!IopCallDriverAddDevice+0x38f (FPO: [Non-Fpo])

06 f901f798 804e504e fe4fed68 f901f864 fe4d6368 nt!IopProcessAddDevicesWorker+0x6c (FPO: [2,0,3])

07 f901f7a8 8054f184 fe4d6368 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

08 f901f7c4 804e504e fe4d6368 f901f864 fe4e7c28 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

09 f901f7d4 8054f184 fe4e7c28 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0a f901f7f0 804e504e fe4e7c28 f901f864 fe5181a8 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

0b f901f800 8054f184 fe5181a8 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0c f901f81c 804e504e fe5181a8 f901f864 fe51b5e8 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

0d f901f82c 8054f184 fe51b5e8 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0e f901f848 8054f114 fe51b5e8 f901f864 00000003 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

0f f901f86c 8054dfdf fe51baa8 0000ffff 00000003 nt!IopProcessAddDevices+0x59 (FPO: [Non-Fpo])

10 f901f8c0 8054c5c9 00000000 00000032 00000000 nt!IopInitializeSystemDrivers+0x25 (FPO: [Non-Fpo])

11 f901fa58 8054b35a 80087000 00000000 00000000 nt!IoInitSystem+0x644 (FPO: [Non-Fpo])

12 f901fda8 804524f6 80087000 00000000 00000000 nt!Phase1Initialization+0x71b (FPO: [Non-Fpo])

13 f901fddc 80465b62 8054aca6 80087000 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

可以看到是 IO 管理器调用 nt!IopLoadDriver 载入了驱动文件 i8042prt.sys，并调用 i8042prt!DriverEntry。

介绍一下 nt!IopLoadDriver 的执行过程

nt!IopLoadDriver 会遍历 PsLoadedModuleList ，查看是否已经有叫 i8042prt.SYS 的模组。也就是查看是否 i8042prt.SYS 已经被载入。发现没有载入，就读取 "ImagePath" 下的数据，获得了路径 "System32\DRIVERS\i8042prt.sys"。使用获得的路径做参数，调用 nt!MmLoadSystemImage，载入驱动。使用 nt!ObCreateObject 创建一个驱动对象。按下面的叙述顺序初始化这个驱动对象。给 DRIVER_OBJECT 的 +18 struct _DRIVER_EXTENSION *DriverExtension 赋值，驱动对象的首地址加上0xa8，即紧跟在 DRIVER_OBJECT 之后的地址。将 DRIVER_EXTENSION 的 +00 struct _DRIVER_OBJECT *DriverObject 赋值为驱动对象的首地址。用 nt!IopInvalidDeviceRequest 初始化驱动对象的整个 MajorFunction[28] 。初始化驱动对象的 Type，Size，Flags。用 i8042prt!DriverEntry 初始化驱动对象的 +2c function *DriverInit 。初始化 +14 void *DriverSection 。用 i8042prt 在内存中的首地址初始化 +0c void *DriverStart 。用 i8042prt 的大小初始化 +10 uint32 DriverSize 。用 nt!ObInsertObject 将 DRIVER_OBJECT 插入命名对象空间，即在插入前 \driver\ 下是没有 i8042prt 的，插入后就有了。初始化 +24 struct _UNICODE_STRING *HardwareDatabase 。从 PagedPool 分配内存，用刚分配的内存初始化 +1c struct _UNICODE_STRING DriverName，放入 "\Driver\i8042prt"。从 NonPagedPool 分配内存，初始化 DRIVER_EXTENSION 的 ServiceKeyName，放入 "i8042prt"。将 PUNICODE_STRING RegistryPath ，PDRIVER_OBJECT DriverObject 压入堆栈，然后调用 DriverObject+0x2c 即 DriverInit 即 DriverEntry。从 DriverEntry 中返回之后，nt!IopLoadDriver 继续执行，会释放为 RegistryPath 所申请的空间。

DriverEntry(

IN PDRIVER_OBJECT DriverObject,

IN PUNICODE_STRING RegistryPath

)

RegistryPath 指向的 unicode 字符串为 "\REGISTRY\MACHINE\SYSTEM\ControlSet001\Services\i8042prt"

此时我的 ControlSet001 是CurrentControlSet 。

在 i8042prt!DriverEntry 中，

初始化一些全局变量。

从 HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\i8042prt\Parameters 下读取一些参数，放在全局变量中。

设置驱动对象的各种例程。

DriverObject->DriverStartIo = I8xStartIo;

DriverObject->DriverUnload = I8xUnload;

DriverObject->DriverExtension->AddDevice = I8xAddDevice;

DriverObject->MajorFunction[IRP_MJ_CREATE] = I8xCreate;

DriverObject->MajorFunction[IRP_MJ_CLOSE] = I8xClose;

DriverObject->MajorFunction[IRP_MJ_INTERNAL_DEVICE_CONTROL] =

I8xInternalDeviceControl;

DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = I8xDeviceControl;

DriverObject->MajorFunction[IRP_MJ_FLUSH_BUFFERS] = I8xFlush;

DriverObject->MajorFunction[IRP_MJ_PNP] = I8xPnP;

DriverObject->MajorFunction[IRP_MJ_POWER] = I8xPower;

DriverObject->MajorFunction[IRP_MJ_SYSTEM_CONTROL] = I8xSystemControl;

从注册表中读数据的函数 RtlQueryRegistryValues 的用法值得一提

NTSTATUS

RtlQueryRegistryValues(

IN ULONG RelativeTo,

IN PCWSTR Path,

IN PRTL_QUERY_REGISTRY_TABLE QueryTable,

IN PVOID Context,

IN PVOID Environment OPTIONAL

);

可以一次获得一个注册表键(Key)下的多个注册表值(Value)的数据。在从注册表中取得驱动的参数时很方便。

typedef struct _RTL_QUERY_REGISTRY_TABLE {

PRTL_QUERY_REGISTRY_ROUTINE QueryRoutine;

ULONG Flags;

PWSTR Name;

PVOID EntryContext;

ULONG DefaultType;

PVOID DefaultData;

ULONG DefaultLength;

} RTL_QUERY_REGISTRY_TABLE, *PRTL_QUERY_REGISTRY_TABLE;

对于 QueryTable 如果 Flags 为 RTL_QUERY_REGISTRY_DIRECT 时

QueryRoutine 被忽略，并且 EntryContext 指向存储结果的地方。

如果访问的键值不存在的话，会使用 DefaultData 做为结果存储到 EntryContext 中。

[kdbclass!DriverEntry]

此时的 Call Stack

ChildEBP RetAddr Args to Child

00 f901f510 804a4431 fe4f6310 fe4f5000 00000000 kbdclass!DriverEntry(struct _DRIVER_OBJECT * DriverObject = 0xfe4f6310, struct _UNICODE_STRING * RegistryPath = 0xfe4f5000) (CONV: stdcall)

01 f901f5d8 80426f6d 80000044 fe4f5000 f901f6e0 nt!IopLoadDriver+0x672 (FPO: [Non-Fpo])

02 f901f608 80506535 fe4f72c8 80000044 00000001 nt!IopCallDriverAddDeviceQueryRoutine+0x356 (FPO: [Non-Fpo])

03 f901f654 80497df1 e129eec6 00000007 f901f6b0 nt!RtlpCallQueryRegistryRoutine+0x1a2 (FPO: [Non-Fpo])

04 f901f6b8 804a4b67 00000000 00000082 00000001 nt!RtlQueryRegistryValues+0x1ed (FPO: [Non-Fpo])

05 f901f77c 8054f1bb 80000048 00000001 f901f800 nt!IopCallDriverAddDevice+0x499 (FPO: [Non-Fpo])

06 f901f798 804e504e fe4fed68 f901f864 fe4d6368 nt!IopProcessAddDevicesWorker+0x6c (FPO: [2,0,3])

07 f901f7a8 8054f184 fe4d6368 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

08 f901f7c4 804e504e fe4d6368 f901f864 fe4e7c28 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

09 f901f7d4 8054f184 fe4e7c28 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0a f901f7f0 804e504e fe4e7c28 f901f864 fe5181a8 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

0b f901f800 8054f184 fe5181a8 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0c f901f81c 804e504e fe5181a8 f901f864 fe51b5e8 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

0d f901f82c 8054f184 fe51b5e8 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0e f901f848 8054f114 fe51b5e8 f901f864 00000003 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

0f f901f86c 8054dfdf fe51baa8 0000ffff 00000003 nt!IopProcessAddDevices+0x59 (FPO: [Non-Fpo])

10 f901f8c0 8054c5c9 00000000 00000032 00000000 nt!IopInitializeSystemDrivers+0x25 (FPO: [Non-Fpo])

11 f901fa58 8054b35a 80087000 00000000 00000000 nt!IoInitSystem+0x644 (FPO: [Non-Fpo])

12 f901fda8 804524f6 80087000 00000000 00000000 nt!Phase1Initialization+0x71b (FPO: [Non-Fpo])

13 f901fddc 80465b62 8054aca6 80087000 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

DriverEntry(

IN PDRIVER_OBJECT DriverObject,

IN PUNICODE_STRING RegistryPath

)

RegistryPath 指向的 unicode 字符串为 "\REGISTRY\MACHINE\SYSTEM\ControlSet001\Services\kbdclass"

在 kbdclass!DriverEntry 中，

初始化一些全局变量。

从 HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\kbdclass\Parameters 下读取一些参数，放入全局变量中。

设置驱动对象的各种例程。

DriverObject->DriverStartIo = KeyboardClassStartIo;

DriverObject->MajorFunction[IRP_MJ_CREATE] = KeyboardClassCreate;

DriverObject->MajorFunction[IRP_MJ_CLOSE] = KeyboardClassClose;

DriverObject->MajorFunction[IRP_MJ_READ] = KeyboardClassRead;

DriverObject->MajorFunction[IRP_MJ_FLUSH_BUFFERS] = KeyboardClassFlush;

DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = KeyboardClassDeviceControl;

DriverObject->MajorFunction[IRP_MJ_INTERNAL_DEVICE_CONTROL] =

KeyboardClassPassThrough;

DriverObject->MajorFunction[IRP_MJ_CLEANUP] = KeyboardClassCleanup;

DriverObject->MajorFunction[IRP_MJ_PNP] = KeyboardPnP;

DriverObject->MajorFunction[IRP_MJ_POWER] = KeyboardClassPower;

DriverObject->MajorFunction[IRP_MJ_SYSTEM_CONTROL] = KeyboardClassSystemControl;

DriverObject->DriverExtension->AddDevice = KeyboardAddDevice;

[i8042prt!I8xAddDevice]

此时的 Call Stack

ChildEBP RetAddr Args to Child

00 f901f6c4 8048e8f2 fe4f69f0 fe4dd730 00000000 i8042prt!I8xAddDevice(struct _DRIVER_OBJECT * Driver = 0xfe4f69f0, struct _DEVICE_OBJECT * PDO = 0xfe4dd730) (CONV: stdcall)

01 f901f77c 8054f1bb 80000048 00000002 f901f800 nt!IopCallDriverAddDevice+0x52d (FPO: [Non-Fpo])

02 f901f798 804e504e fe4fed68 f901f864 fe4d6368 nt!IopProcessAddDevicesWorker+0x6c (FPO: [2,0,3])

03 f901f7a8 8054f184 fe4d6368 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

04 f901f7c4 804e504e fe4d6368 f901f864 fe4e7c28 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

05 f901f7d4 8054f184 fe4e7c28 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

06 f901f7f0 804e504e fe4e7c28 f901f864 fe5181a8 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

07 f901f800 8054f184 fe5181a8 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

08 f901f81c 804e504e fe5181a8 f901f864 fe51b5e8 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

09 f901f82c 8054f184 fe51b5e8 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0a f901f848 8054f114 fe51b5e8 f901f864 00000003 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

0b f901f86c 8054dfdf fe51baa8 0000ffff 00000003 nt!IopProcessAddDevices+0x59 (FPO: [Non-Fpo])

0c f901f8c0 8054c5c9 00000000 00000032 00000000 nt!IopInitializeSystemDrivers+0x25 (FPO: [Non-Fpo])

0d f901fa58 8054b35a 80087000 00000000 00000000 nt!IoInitSystem+0x644 (FPO: [Non-Fpo])

0e f901fda8 804524f6 80087000 00000000 00000000 nt!Phase1Initialization+0x71b (FPO: [Non-Fpo])

0f f901fddc 80465b62 8054aca6 80087000 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

10 00000000 00000000 00000000 00000000 00000000 nt!KiThreadStartup+0x16

IO 管理器调用了 i8042prt!I8xAddDevice。

i8042prt!I8xAddDevice 中，

调用 IoCreateDevice 创建设备对象。

调用 IoAttachDeviceToDeviceStack 将产生的 i8042prt 的设备对象放入键盘设备栈。

初始化自定义的设备扩展中的一些域。

IoCreateDevice 的参数 DeviceName ，调用时为空，说明这个设备对象没有名字。

IoAttachDeviceToDeviceStack 中的参数，TargetDevice 决定了产生的 i8042prt 的设备对象将被放入哪个设备栈。而调用时参数 TargetDevice ，为 i8042prt!I8xAddDevice 传入参数 PDO。系统如何知道给 i8042prt!I8xAddDevice 传入哪个 PDO 呢？答案是，键盘设备栈的 PDO 通过注册表中保存的键盘设备栈的信息，找到了 i8042prt，才调用了 i8042prt 的 DriverObject->DriverExtension->AddDevice，所以当然知道 PDO 。

设备栈上的设备对象，从上自下，通过 DEVICE_OBJECT 的 DEVOBJ_EXTENSION 的 +18 struct _DEVICE_OBJECT *AttachedTo 联系在一起。设备栈上的设备对象，从下自上，通过 DEVICE_OBJECT 的 +10 struct _DEVICE_OBJECT *AttachedDevice 联系在一起。

IoAttachDeviceToDeviceStack 会把一个设备对象，放入设备栈的栈顶，也就是把这个设备对象的 DEVICE_OBJECT 的 DEVOBJ_EXTENSION 的 +18 struct _DEVICE_OBJECT *AttachedTo 和 DEVICE_OBJECT 的 +10 struct _DEVICE_OBJECT *AttachedDevice 这两个域和原来设备栈的栈顶的设备对象的这两个域发生联系。还会相应的设置这个新放入设备栈的设备对象的 +30 char StackSize。

[kbdclass!KeyboardAddDevice]

此时的 Call Stack

ChildEBP RetAddr Args to Child

00 f901f6c4 8048e8f2 fe4f6310 fe4dd730 00000000 kbdclass!KeyboardAddDevice(struct _DRIVER_OBJECT * DriverObject = 0xfe4f6310, struct _DEVICE_OBJECT * PhysicalDeviceObject = 0xfe4dd730) (CONV: stdcall)

01 f901f77c 8054f1bb 80000048 00000004 f901f800 nt!IopCallDriverAddDevice+0x52d (FPO: [Non-Fpo])

02 f901f798 804e504e fe4fed68 f901f864 fe4d6368 nt!IopProcessAddDevicesWorker+0x6c (FPO: [2,0,3])

03 f901f7a8 8054f184 fe4d6368 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

04 f901f7c4 804e504e fe4d6368 f901f864 fe4e7c28 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

05 f901f7d4 8054f184 fe4e7c28 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

06 f901f7f0 804e504e fe4e7c28 f901f864 fe5181a8 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

07 f901f800 8054f184 fe5181a8 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

08 f901f81c 804e504e fe5181a8 f901f864 fe51b5e8 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

09 f901f82c 8054f184 fe51b5e8 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0a f901f848 8054f114 fe51b5e8 f901f864 00000003 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

0b f901f86c 8054dfdf fe51baa8 0000ffff 00000003 nt!IopProcessAddDevices+0x59 (FPO: [Non-Fpo])

0c f901f8c0 8054c5c9 00000000 00000032 00000000 nt!IopInitializeSystemDrivers+0x25 (FPO: [Non-Fpo])

0d f901fa58 8054b35a 80087000 00000000 00000000 nt!IoInitSystem+0x644 (FPO: [Non-Fpo])

0e f901fda8 804524f6 80087000 00000000 00000000 nt!Phase1Initialization+0x71b (FPO: [Non-Fpo])

0f f901fddc 80465b62 8054aca6 80087000 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

10 00000000 00000000 00000000 00000000 00000000 nt!KiThreadStartup+0x16

kbdclass!KeyboardAddDevice 中，

调用 IoCreateDevice 创建设备对象。

初始化自定义的设备扩展中的一些域。

从 NonPagedPool 为 kbdclass 的输入数据队列，分配内存。

初始化设备扩展中的使用输入数据队列的相关域。

调用 IoAttachDeviceToDeviceStack 将产生的 kbdclass 的设备对象放入键盘设备栈。

调用 IoRegisterDeviceInterface ，暴露驱动的接口给应用层。

调用函数 kbdclass!KbdSendConnectRequest ，把自己(kbdclass)的处理输入数据的回调函数告诉下一层(i8042prt)。

IoCreateDevice 的参数 DeviceName ，调用时不为空，说明这个设备对象有名字。在我这里这个名字是 "\Device\KeyboardClass0"。

调用 ExAllocatePool 为 kbdclass 的输入数据队列分配内存，分配内存的大小为 deviceExtension->KeyboardAttributes.InputDataQueueLength。而 deviceExtension->KeyboardAttributes.InputDataQueueLength 是由 Globals.InitExtension.KeyboardAttributes.InputDataQueueLength 赋值的。而 Globals.InitExtension.KeyboardAttributes.InputDataQueueLength 是在 kbdclass 的 DriverEntry 中初始化的。即读取注册表 HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Kbdclass\Parameters\KeyboardDataQueueSize ，然后乘以 sizeof(KEYBOARD_INPUT_DATA)。

deviceExtension->InputData 保存申请空间的首地址。

然后调用 KbdInitializeDataQueue，初始化，

deviceExtension->InputCount = 0;

deviceExtension->DataIn = deviceExtension->InputData;

deviceExtension->DataOut = deviceExtension->InputData;

也就是开始的时候，DataIn,DataOut 都指向输入数据队列的开头。

调用 IoAttachDeviceToDeviceStack 将产生的 kbdclass 的设备对象放入键盘设备栈。现在键盘设备栈就已经连接完了。

kbdclass!KbdSendConnectRequest 以一个 CONNECT_DATA ，IOCTL_INTERNAL_KEYBOARD_CONNECT 为参数调用 IoBuildDeviceIoControlRequest，创建一个 IRP_MJ_INTERNAL_DEVICE_CONTROL 的 IRP ，调用 IoCallDriver 传给 i8024prt。这个 CONNECT_DATA 中有kbdclass处理输入数据的回调函数 kbdclass!KeyboardClassServiceCallback 的地址。IoCallDriver 根据这个 IRP ，调用 i8042prt!I8xInternalDeviceControl。在 i8042prt!I8xInternalDeviceControl 中会把这个 CONNECT_DATA 保存在 kbExtension->ConnectData 中。这样，以后i8042prt得到了按键的数据以后，就可以调用kbdclass指定的这个回调函数，让 kbdclass 作出处理。这里 i8042prt!I8xInternalDeviceControl 处理 IOCTL_INTERNAL_KEYBOARD_CONNECT 的部分，还早在这时就给上一层一个设置hook机会，如果在这时设置hook，可能会对初始化的某些过程hoot，不过 kbdclass 对此没有处理。

[IRP_MJ_PNP IRP_MN_QUERY_LEGACY_BUS_INFORMATION]

ChildEBP RetAddr Args to Child

00 f901f638 8041f54b fe4f5df0 fe4fea08 fe4f5df0 kbdclass!KeyboardPnP(struct _DEVICE_OBJECT * DeviceObject = 0xfe4f5df0, struct _IRP * Irp = 0xfe4fea08) (CONV: stdcall)

01 f901f64c 8049cb91 f901f6bc fe4dd730 fe4fedc0 nt!IopfCallDriver+0x35 (FPO: [0,0,2])

02 f901f678 8048f328 fe4f5df0 f901f698 f901f6c4 nt!IopSynchronousCall+0xca (FPO: [Non-Fpo])

03 f901f6bc 804a4bd1 fe4dd730 00000000 fe4fedc0 nt!IopQueryLegacyBusInformation+0x2c (FPO: [Non-Fpo])

04 f901f77c 8054f1bb 80000048 00000005 f901f800 nt!IopCallDriverAddDevice+0x634 (FPO: [Non-Fpo])

05 f901f798 804e504e fe4fed68 f901f864 fe4d6368 nt!IopProcessAddDevicesWorker+0x6c (FPO: [2,0,3])

06 f901f7a8 8054f184 fe4d6368 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

07 f901f7c4 804e504e fe4d6368 f901f864 fe4e7c28 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

08 f901f7d4 8054f184 fe4e7c28 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

09 f901f7f0 804e504e fe4e7c28 f901f864 fe5181a8 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

0a f901f800 8054f184 fe5181a8 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0b f901f81c 804e504e fe5181a8 f901f864 fe51b5e8 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

0c f901f82c 8054f184 fe51b5e8 8054f147 f901f864 nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0d f901f848 8054f114 fe51b5e8 f901f864 00000003 nt!IopProcessAddDevicesWorker+0x3d (FPO: [2,0,3])

0e f901f86c 8054dfdf fe51baa8 0000ffff 00000003 nt!IopProcessAddDevices+0x59 (FPO: [Non-Fpo])

0f f901f8c0 8054c5c9 00000000 00000032 00000000 nt!IopInitializeSystemDrivers+0x25 (FPO: [Non-Fpo])

10 f901fa58 8054b35a 80087000 00000000 00000000 nt!IoInitSystem+0x644 (FPO: [Non-Fpo])

11 f901fda8 804524f6 80087000 00000000 00000000 nt!Phase1Initialization+0x71b (FPO: [Non-Fpo])

12 f901fddc 80465b62 8054aca6 80087000 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

13 00000000 00000000 00000000 00000000 00000000 nt!KiThreadStartup+0x16

IO 管理器向键盘设备栈发 IRP_MJ_PNP IRP_MN_QUERY_LEGACY_BUS_INFORMATION 的 IRP。

这将导致 kbdclass!KeyboardPnP 被执行，我们从 kbdclass!KeyboardPnP 的传入参数 Irp 可以知道，这个 IRP 是哪种 IRP。

kbdclass 和 i8042prt 对 IRP_MJ_PNP IRP_MN_QUERY_LEGACY_BUS_INFORMATION 的 IRP 都没有处理，向下传。

[IRP_MJ_PNP IRP_MN_FILTER_RESOURCE_REQUIREMENTS]

ChildEBP RetAddr Args to Child

00 f901f6f4 8041f54b fe4f5df0 fe4fea08 fe4f5df0 kbdclass!KeyboardPnP(struct _DEVICE_OBJECT * DeviceObject = 0xfe4f5df0, struct _IRP * Irp = 0xfe4fea08)+0x9 (CONV: stdcall)

01 f901f708 8048f468 e12bf298 fe4fed68 f901f7c0 nt!IopfCallDriver+0x35 (FPO: [0,0,2])

02 f901f734 8048f2dc fe4f5df0 e12b4ee8 f901f794 nt!IopFilterResourceRequirementsCall+0xdb (FPO: [Non-Fpo])

03 f901f79c 8048dff9 fe4dd730 e12b4ee8 e12bf201 nt!IopQueryDeviceResources+0x211 (FPO: [Non-Fpo])

04 f901f7d0 8048df26 e12bf288 e12bf5c4 f901f7f8 nt!IopGetResourceRequirementsForAssignTable+0xc6 (FPO: [Non-Fpo])

05 f901f800 8048f0ac f901f84c f901f850 00000000 nt!IopAllocateResources+0xa0 (FPO: [Non-Fpo])

06 f901f844 8048edf6 00000017 e12bf288 00000001 nt!IopAssignResourcesToDevices+0xd9 (FPO: [Non-Fpo])

07 f901f86c 8054e005 fe51baa8 00000000 00000001 nt!IopProcessAssignResources+0xe8 (FPO: [Non-Fpo])

08 f901f8c0 8054c5c9 00000000 00000032 00000000 nt!IopInitializeSystemDrivers+0x4b (FPO: [Non-Fpo])

09 f901fa58 8054b35a 80087000 00000000 00000000 nt!IoInitSystem+0x644 (FPO: [Non-Fpo])

0a f901fda8 804524f6 80087000 00000000 00000000 nt!Phase1Initialization+0x71b (FPO: [Non-Fpo])

0b f901fddc 80465b62 8054aca6 80087000 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

0c 00000000 00000000 00000000 00000000 00000000 nt!KiThreadStartup+0x16

IO 管理器向键盘设备栈发 IRP_MJ_PNP IRP_MN_FILTER_RESOURCE_REQUIREMENTS 的 IRP。

kbdclass!KeyboardPnP 对 IRP_MJ_PNP IRP_MN_FILTER_RESOURCE_REQUIREMENTS 没处理，向下传。

i8042prt!I8xPnP 处理 IRP_MJ_PNP IRP_MN_FILTER_RESOURCE_REQUIREMENTS 的部分中，首先把这个 IRP 向下传。之后会调用函数 i8042prt!I8xFilterResourceRequirements。i8042prt!I8xFilterResourceRequirements中会把传入的端口信息保存在 Globals.ControllerData->KnownPorts 中，除此之外，没有作什么值得注意的处理。

DDK 中说，pnp管理器发 IRP_MN_FILTER_RESOURCE_REQUIREMENTS IRP ，是给驱动一个调整和修改资源的一个机会。

我们来看一看系统打算给键盘驱动分配的资源。

使用 WinDbg 的 !irp 命令来查看传入 IRP。

kd> !irp fe4fea08

Irp is active with 6 stacks 6 is current (= 0xfe4feb2c)

No Mdl Thread fe4f47e0: Irp stack trace.

cmd cl Device File Completion-Context

[0, 0] 0 0 00000000 00000000 00000000-00000000

Args: 00000000 00000000 00000000 00000000

[0, 0] 0 0 00000000 00000000 00000000-00000000

Args: 00000000 00000000 00000000 00000000

[0, 0] 0 0 00000000 00000000 00000000-00000000

Args: 00000000 00000000 00000000 00000000

[0, 0] 0 0 00000000 00000000 00000000-00000000

Args: 00000000 00000000 00000000 00000000

[0, 0] 0 0 00000000 00000000 00000000-00000000

Args: 00000000 00000000 00000000 00000000

>[1b, d] 0 0 fe4f5df0 00000000 00000000-00000000

\Driver\Kbdclass

Args: e12b4ee8 00000000 00000000 00000000

看到当前 IO_STACK_LOCATION 的 MajorFunction 为 0x1b，MinorFunction 为 0xd。即 MajorFunction 为 IRP_MJ_PNP，MinorFunction 为 IRP_MN_FILTER_RESOURCE_REQUIREMENTS。

详细的看看当前的 IO_STACK_LOCATION，

kd> !strct io_stack_location fe4feb2c

struct _IO_STACK_LOCATION (sizeof=36)

+00 byte MajorFunction = 1b .

+01 byte MinorFunction = 0d .

+02 byte Flags = 00 .

+03 byte Control = 00 .

+04 union __unnamed19 Parameters

+04 struct __unnamed42 FilterResourceRequirements

+04 struct _IO_RESOURCE_REQUIREMENTS_LIST *IoResourceRequirementList = E12B4EE8

+14 struct _DEVICE_OBJECT *DeviceObject = FE4F5DF0

+18 struct _FILE_OBJECT *FileObject = 00000000

+1c function *CompletionRoutine = 00000000

+20 void *Context = 00000000

传入了一个指向 IO_RESOURCE_REQUIREMENTS_LIST 结构的指针。

我们使用 WinDbg 的 !ioreslist 看看这个 IO_RESOURCE_REQUIREMENTS_LIST 中的资源

kd> !ioreslist e12b4ee8

IoResList at 0xe12b4ee8 : Interface 0xf Bus 0 Slot 0

Reserved Values = {0x00000030, 0x00000201, 0x05000000}

Alternative 0 (Version 1.1)

Preferred Descriptor 0 - NonArbitrated/ConfigData (0x80) Shared (0x3)

Flags (0000) -

Data: : 0x1 0x5 0x180200

Preferred Descriptor 1 - Port (0x1) Device Exclusive (0x1)

Flags (0x11) - PORT_IO 16_BIT_DECODE

0x000001 byte range with alignment 0x000001

60 - 0x60

Preferred Descriptor 2 - Port (0x1) Device Exclusive (0x1)

Flags (0x11) - PORT_IO 16_BIT_DECODE

0x000001 byte range with alignment 0x000001

64 - 0x64

Preferred Descriptor 3 - Interrupt (0x2) Device Exclusive (0x1)

Flags (0x01) - LATCHED

0x1 - 0x1

可以看到，系统打算给键盘驱动分配的资源有，两个端口，一个 0x60，一个0x64。一个中断，IRQ为1。

[IRP_MJ_PNP IRP_MN_START_DEVICE]

ChildEBP RetAddr Args to Child

00 f901f6a4 8041f54b fe4f5df0 fe4fea08 fe4f5df0 kbdclass!KeyboardPnP(struct _DEVICE_OBJECT * DeviceObject = 0xfe4f5df0, struct _IRP * Irp = 0xfe4fea08)+0x9 (CONV: stdcall)

01 f901f6b8 8049cb91 00020000 fe4fed68 00000000 nt!IopfCallDriver+0x35 (FPO: [0,0,2])

02 f901f6e4 804289ce fe4f5df0 f901f704 f901f72c nt!IopSynchronousCall+0xca (FPO: [Non-Fpo])

03 f901f730 8048e06a fe4dd730 00000000 fe4fed68 nt!IopStartDevice+0x127 (FPO: [Non-Fpo])

04 f901f764 8048e040 fe4fed68 f901f88c 00000000 nt!IopStartAndEnumerateDevice+0x22 (FPO: [Non-Fpo])

05 f901f784 804e504e fe4fed68 f901f88c fe4d6368 nt!IopProcessStartDevicesWorker+0x72 (FPO: [Non-Fpo])

06 f901f794 804a4670 fe4d6368 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

07 f901f7b8 804e504e fe4d6368 f901f88c fe4e7c28 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

08 f901f7c8 804a4670 fe4e7c28 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

09 f901f7ec 804e504e fe4e7c28 f901f88c fe5181a8 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

0a f901f7fc 804a4670 fe5181a8 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0b f901f820 804e504e fe5181a8 f901f88c fe51b5e8 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

0c f901f830 804a4670 fe51b5e8 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0d f901f854 804a4607 fe51b5e8 f901f88c 00000003 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

0e f901f870 8054e017 fe51b2c8 f901f88c 80087000 nt!IopProcessStartDevices+0x43 (FPO: [EBP 0xf901f8c0] [2,0,4])

0f f901f8c0 8054c5c9 00000000 00000032 00000000 nt!IopInitializeSystemDrivers+0x5d (FPO: [Non-Fpo])

10 f901fa58 8054b35a 80087000 00000000 00000000 nt!IoInitSystem+0x644 (FPO: [Non-Fpo])

11 f901fda8 804524f6 80087000 00000000 00000000 nt!Phase1Initialization+0x71b (FPO: [Non-Fpo])

12 f901fddc 80465b62 8054aca6 80087000 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

13 00000000 00000000 00000000 00000000 00000000 nt!KiThreadStartup+0x16

pnp 管理器向键盘设备栈发 IRP_MN_START_DEVICE。

我们看看这个 IRP

kd> !irp fe4fea08

Irp is active with 6 stacks 6 is current (= 0xfe4feb2c)

No Mdl Thread fe4f47e0: Irp stack trace.

cmd cl Device File Completion-Context

[0, 0] 0 0 00000000 00000000 00000000-00000000

Args: 00000000 00000000 00000000 00000000

[0, 0] 0 0 00000000 00000000 00000000-00000000

Args: 00000000 00000000 00000000 00000000

[0, 0] 0 0 00000000 00000000 00000000-00000000

Args: 00000000 00000000 00000000 00000000

[0, 0] 0 0 00000000 00000000 00000000-00000000

Args: 00000000 00000000 00000000 00000000

[0, 0] 0 0 00000000 00000000 00000000-00000000

Args: 00000000 00000000 00000000 00000000

>[1b, 0] 0 0 fe4f5df0 00000000 00000000-00000000

\Driver\Kbdclass

Args: e12c1bc8 e12c97c8 00000000 00000000

看看当前 IO_STACK_LOCATION

kd> !strct io_stack_location fe4feb2c

struct _IO_STACK_LOCATION (sizeof=36)

+00 byte MajorFunction = 1b .

+01 byte MinorFunction = 00 .

+02 byte Flags = 00 .

+03 byte Control = 00 .

+04 union __unnamed19 Parameters

+04 struct __unnamed59 StartDevice

+04 struct _CM_RESOURCE_LIST *AllocatedResources = E12C1BC8

+08 struct _CM_RESOURCE_LIST *AllocatedResourcesTranslated = E12C97C8

+14 struct _DEVICE_OBJECT *DeviceObject = FE4F5DF0

+18 struct _FILE_OBJECT *FileObject = 00000000

+1c function *CompletionRoutine = 00000000

+20 void *Context = 00000000

我们看看 AllocatedResources

kd> !cmreslist E12C1BC8

CmResourceList at 0xe12c1bc8 Version 0.0 Interface 0xf Bus #0

Entry 0 - Port (0x1) Device Exclusive (0x1)

Flags (0x11) - PORT_MEMORY PORT_IO 16_BIT_DECODE

Range starts at 0x60 for 0x1 bytes

Entry 1 - Port (0x1) Device Exclusive (0x1)

Flags (0x11) - PORT_MEMORY PORT_IO 16_BIT_DECODE

Range starts at 0x64 for 0x1 bytes

Entry 2 - Interrupt (0x2) Device Exclusive (0x1)

Flags (0x01) - LATCHED

Level 0x1, Vector 0x1, Affinity 0xffffffff

我们看看 AllocatedResourcesTranslated

kd> !cmreslist E12C97C8

CmResourceList at 0xe12c97c8 Version 0.0 Interface 0xf Bus #0

Entry 0 - Port (0x1) Device Exclusive (0x1)

Flags (0x11) - PORT_MEMORY PORT_IO 16_BIT_DECODE

Range starts at 0x60 for 0x1 bytes

Entry 1 - Port (0x1) Device Exclusive (0x1)

Flags (0x11) - PORT_MEMORY PORT_IO 16_BIT_DECODE

Range starts at 0x64 for 0x1 bytes

Entry 2 - Interrupt (0x2) Device Exclusive (0x1)

Flags (0x01) - LATCHED

Level 0xa, Vector 0xb3, Affinity 0x1

键盘驱动对于 IRP_MJ_PNP IRP_MN_START_DEVICE 的处理

kbdclass!KeyboardPnP 中，首先将这个IRP向下传，导致 i8042prt!I8xPnP 被执行。下面处理返回之后，如果没有错误，调用 IoSetDeviceInterfaceState，enable 键盘驱动暴露给应用层的接口。

i8042prt!I8xPnP 中，首先将这个IRP向下传，下面处理返回没有错，就继续 i8042prt!I8xPnP 中的处理。调用 i8042prt!I8xKeyboardStartDevice。i8042prt!I8xKeyboardStartDevice 中，把IRP传入的转换过的资源的信息保存在全局变量和设备扩展中。从注册表 HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\i8042prt\Parameters 键下，读出参数，初始化设备扩展的一些域。从 NonPagedPool 为 i8042prt 的输入数据队列，分配内存。初始化设备扩展中的使用输入数据队列的相关域。初始化设备扩展中的一些域。

从注册表 HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\i8042prt\Parameters 键下，读出参数，初始化设备扩展的一些域。包括，如果 HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\i8042prt\Parameters\KeyboardDataQueueSize，

存在，那么读出这个值作为 i8042prt的输入数据队列中单元的个数，如果不存在使用默认值十进制100作为 i8042prt的输入数据队列中单元的个数。在我这里，HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\i8042prt\Parameters\KeyboardDataQueueSize 不存在，所以使用默认值十进制100作为i8042prt的输入数据队列中单元的个数，最终 KeyboardExtension->KeyboardAttributes.InputDataQueueLength 赋值为输入数据队列中单元的个数*sizeof(KEYBOARD_INPUT_DATA)。

调用 ExAllocatePool 为 i8042prt 的输入数据队列分配内存，分配内存的大小为 KeyboardExtension->KeyboardAttributes.InputDataQueueLength。

KeyboardExtension->InputData 保存申请空间的首地址。

KeyboardExtension->DataEnd 保存申请空间的尾地址。

然后调用 I8xInitializeDataQueue，初始化，

kbExtension->InputCount = 0;

kbExtension->DataIn = kbExtension->InputData;

kbExtension->DataOut = kbExtension->InputData;

也就是开始的时候，DataIn,DataOut 都指向输入数据队列的开头。

初始化设备扩展中的一些DPC，包括 KeyboardExtension->KeyboardIsrDpc 初始化为 i8042prt!I8042KeyboardIsrDpc。

硬件的初始化工作，对于ps/2的鼠标和ps/2的键盘都是向i8042发一些命令。所以驱动 i8042prt 把ps/2键盘鼠标的硬件初始化工作放在了一起进行。

在键盘启动的最后，会判断，如果没有ps/2鼠标，或者有ps/2鼠标但是已经启动了，那么现在执行硬件的初始化工作。如果有鼠标，并且鼠标还没有被启动，那么硬件的初始化工作，放在鼠标启动的最后进行。

我这里有ps/2鼠标，并且鼠标是在键盘之后被启动的，目前鼠标还没有被启动，所以硬件的初始化放在后面的鼠标启动时一起进行。

[IRP_MJ_PNP IRP_MN_QUERY_CAPABILITIES]

ChildEBP RetAddr Args to Child

00 f901f664 8041f54b fe4f5df0 fe4fea08 fe4f5df0 kbdclass!KeyboardPnP(struct _DEVICE_OBJECT * DeviceObject = 0xfe4f5df0, struct _IRP * Irp = 0xfe4fea08)+0x9 (CONV: stdcall)

01 f901f678 8049cb91 f901f6e0 fe4dd730 fe4fed68 nt!IopfCallDriver+0x35 (FPO: [0,0,2])

02 f901f6a4 80427122 fe4f5df0 f901f6bc f901f6ec nt!IopSynchronousCall+0xca (FPO: [Non-Fpo])

03 f901f6e0 8048ed08 fe4fed68 f901f6f0 00010040 nt!IopQueryDeviceCapabilities+0x4c (FPO: [Non-Fpo])

04 f901f730 8048e075 fe4fed68 00000000 fe4fed68 nt!IopDeviceNodeCapabilitiesToRegistry+0x12 (FPO: [Non-Fpo])

05 f901f764 8048e040 fe4fed68 f901f88c 00000000 nt!IopStartAndEnumerateDevice+0x2d (FPO: [Non-Fpo])

06 f901f784 804e504e fe4fed68 f901f88c fe4d6368 nt!IopProcessStartDevicesWorker+0x72 (FPO: [Non-Fpo])

07 f901f794 804a4670 fe4d6368 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

08 f901f7b8 804e504e fe4d6368 f901f88c fe4e7c28 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

09 f901f7c8 804a4670 fe4e7c28 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0a f901f7ec 804e504e fe4e7c28 f901f88c fe5181a8 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

0b f901f7fc 804a4670 fe5181a8 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0c f901f820 804e504e fe5181a8 f901f88c fe51b5e8 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

0d f901f830 804a4670 fe51b5e8 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0e f901f854 804a4607 fe51b5e8 f901f88c 00000003 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

0f f901f870 8054e017 fe51b2c8 f901f88c 80087000 nt!IopProcessStartDevices+0x43 (FPO: [EBP 0xf901f8c0] [2,0,4])

10 f901f8c0 8054c5c9 00000000 00000032 00000000 nt!IopInitializeSystemDrivers+0x5d (FPO: [Non-Fpo])

11 f901fa58 8054b35a 80087000 00000000 00000000 nt!IoInitSystem+0x644 (FPO: [Non-Fpo])

12 f901fda8 804524f6 80087000 00000000 00000000 nt!Phase1Initialization+0x71b (FPO: [Non-Fpo])

13 f901fddc 80465b62 8054aca6 80087000 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

kbdclass!KeyboardPnP 和 i8042prt!I8xPnP 中都没有做什么处理。

[IRP_MJ_PNP IRP_MN_QUERY_PNP_DEVICE_STATE]

ChildEBP RetAddr Args to Child

00 f901f6ac 8041f54b fe4f5df0 fe4fea08 fe4f5df0 kbdclass!KeyboardPnP(struct _DEVICE_OBJECT * DeviceObject = 0xfe4f5df0, struct _IRP * Irp = 0xfe4fea08)+0x9 (CONV: stdcall)

01 f901f6c0 8049cb91 fe4dd730 fe4dd730 fe4fed68 nt!IopfCallDriver+0x35 (FPO: [0,0,2])

02 f901f6ec 80428f92 fe4f5df0 f901f70c f901f738 nt!IopSynchronousCall+0xca (FPO: [Non-Fpo])

03 f901f730 8048e07b fe4dd730 00000000 fe4fed68 nt!IopQueryDeviceState+0x2c (FPO: [Non-Fpo])

04 f901f764 8048e040 fe4fed68 f901f88c 00000000 nt!IopStartAndEnumerateDevice+0x33 (FPO: [Non-Fpo])

05 f901f784 804e504e fe4fed68 f901f88c fe4d6368 nt!IopProcessStartDevicesWorker+0x72 (FPO: [Non-Fpo])

06 f901f794 804a4670 fe4d6368 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

07 f901f7b8 804e504e fe4d6368 f901f88c fe4e7c28 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

08 f901f7c8 804a4670 fe4e7c28 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

09 f901f7ec 804e504e fe4e7c28 f901f88c fe5181a8 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

0a f901f7fc 804a4670 fe5181a8 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0b f901f820 804e504e fe5181a8 f901f88c fe51b5e8 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

0c f901f830 804a4670 fe51b5e8 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0d f901f854 804a4607 fe51b5e8 f901f88c 00000003 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

0e f901f870 8054e017 fe51b2c8 f901f88c 80087000 nt!IopProcessStartDevices+0x43 (FPO: [EBP 0xf901f8c0] [2,0,4])

0f f901f8c0 8054c5c9 00000000 00000032 00000000 nt!IopInitializeSystemDrivers+0x5d (FPO: [Non-Fpo])

10 f901fa58 8054b35a 80087000 00000000 00000000 nt!IoInitSystem+0x644 (FPO: [Non-Fpo])

11 f901fda8 804524f6 80087000 00000000 00000000 nt!Phase1Initialization+0x71b (FPO: [Non-Fpo])

12 f901fddc 80465b62 8054aca6 80087000 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

13 00000000 00000000 00000000 00000000 00000000 nt!KiThreadStartup+0x16

kbdclass!KeyboardPnP 和 i8042prt!I8xPnP 中都没有做什么值得注意的处理。

[IRP_MJ_PNP IRP_MN_QUERY_DEVICE_RELATIONS]

ChildEBP RetAddr Args to Child

00 f901f674 8041f54b fe4f5df0 fe4fea08 fe4f5df0 kbdclass!KeyboardPnP(struct _DEVICE_OBJECT * DeviceObject = 0xfe4f5df0, struct _IRP * Irp = 0xfe4fea08)+0x9 (CONV: stdcall)

01 f901f688 8049cb91 f901f6f8 fe4fed68 80064b8c nt!IopfCallDriver+0x35 (FPO: [0,0,2])

02 f901f6b4 80428f1d fe4f5df0 f901f6d4 f901f724 nt!IopSynchronousCall+0xca (FPO: [Non-Fpo])

03 f901f6fc 804a470f 00000000 fe4dd730 00142100 nt!IopQueryDeviceRelations+0x13f (FPO: [Non-Fpo])

04 f901f728 8048e0ac fe4dd730 f901f88c 00142100 nt!IopEnumerateDevice+0xce (FPO: [Non-Fpo])

05 f901f764 8048e040 fe4fed68 f901f88c 00000000 nt!IopStartAndEnumerateDevice+0x1c7 (FPO: [Non-Fpo])

06 f901f784 804e504e fe4fed68 f901f88c fe4d6368 nt!IopProcessStartDevicesWorker+0x72 (FPO: [Non-Fpo])

07 f901f794 804a4670 fe4d6368 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

08 f901f7b8 804e504e fe4d6368 f901f88c fe4e7c28 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

09 f901f7c8 804a4670 fe4e7c28 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0a f901f7ec 804e504e fe4e7c28 f901f88c fe5181a8 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

0b f901f7fc 804a4670 fe5181a8 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0c f901f820 804e504e fe5181a8 f901f88c fe51b5e8 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

0d f901f830 804a4670 fe51b5e8 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0e f901f854 804a4607 fe51b5e8 f901f88c 00000003 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

0f f901f870 8054e017 fe51b2c8 f901f88c 80087000 nt!IopProcessStartDevices+0x43 (FPO: [EBP 0xf901f8c0] [2,0,4])

10 f901f8c0 8054c5c9 00000000 00000032 00000000 nt!IopInitializeSystemDrivers+0x5d (FPO: [Non-Fpo])

11 f901fa58 8054b35a 80087000 00000000 00000000 nt!IoInitSystem+0x644 (FPO: [Non-Fpo])

12 f901fda8 804524f6 80087000 00000000 00000000 nt!Phase1Initialization+0x71b (FPO: [Non-Fpo])

13 f901fddc 80465b62 8054aca6 80087000 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

kbdclass!KeyboardPnP 和 i8042prt!I8xPnP 中都没有做什么处理。

[启动ps/2鼠标的 IRP_MJ_PNP IRP_MN_START_DEVICE]

硬件的初始化工作，对于ps/2的鼠标和ps/2的键盘都是向i8042发一些命令。所以驱动 i8042prt 把ps/2键盘鼠标的硬件初始化工作放在了一起进行。

前面已经给键盘设备栈发 IRP_MJ_PNP IRP_MN_START_DEVICE 的 IRP 将键盘启动了。现在给鼠标设备栈发 IRP_MJ_PNP IRP_MN_START_DEVICE 的 IRP，启动鼠标，将在鼠标启动的最后初始化键盘和鼠标的硬件。

ChildEBP RetAddr Args to Child

00 f901f5c8 8041f54b fe4d3ba0 fe4fea08 fe4d3b18 i8042prt!I8xPnP(struct _DEVICE_OBJECT * DeviceObject = 0xfe4d3ba0, struct _IRP * Irp = 0xfe4fea08) (CONV: stdcall)

01 f901f5dc f91ff8c2 fe4d3a60 fe4fea08 fe4fea08 nt!IopfCallDriver+0x35 (FPO: [0,0,2])

WARNING: Stack unwind information not available. Following frames may be wrong.

02 f901f604 8041f54b fe4d3a60 fe4fea08 00000000 vmmouse+0x8c2

03 f901f618 f8ec26fd fe4fea08 fe4d3928 fe043801 nt!IopfCallDriver+0x35 (FPO: [0,0,2])

04 f901f638 f8ec12d8 fe4d3a60 fe4fea08 00000001 mouclass!MouseSendIrpSynchronously+0x57 (FPO: [Non-Fpo])

05 f901f6a4 8041f54b fe4feb50 fe4fea08 fe4d3870 mouclass!MousePnP+0x2b2 (FPO: [Non-Fpo])

06 f901f6b8 8049cb91 00020000 fe4fec88 00000000 nt!IopfCallDriver+0x35 (FPO: [0,0,2])

07 f901f6e4 804289ce fe4d3870 f901f704 f901f72c nt!IopSynchronousCall+0xca (FPO: [Non-Fpo])

08 f901f730 8048e06a fe4dd610 00000000 fe4fec88 nt!IopStartDevice+0x127 (FPO: [Non-Fpo])

09 f901f764 8048e040 fe4fec88 f901f88c 00000000 nt!IopStartAndEnumerateDevice+0x22 (FPO: [Non-Fpo])

0a f901f784 804e504e fe4fec88 f901f88c fe4d6368 nt!IopProcessStartDevicesWorker+0x72 (FPO: [Non-Fpo])

0b f901f794 804a4670 fe4d6368 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0c f901f7b8 804e504e fe4d6368 f901f88c fe4e7c28 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

0d f901f7c8 804a4670 fe4e7c28 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

0e f901f7ec 804e504e fe4e7c28 f901f88c fe5181a8 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

0f f901f7fc 804a4670 fe5181a8 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

10 f901f820 804e504e fe5181a8 f901f88c fe51b5e8 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

11 f901f830 804a4670 fe51b5e8 804a4618 f901f88c nt!IopForAllChildDeviceNodes+0x1f (FPO: [3,0,1])

12 f901f854 804a4607 fe51b5e8 f901f88c 00000003 nt!IopProcessStartDevicesWorker+0x55 (FPO: [Non-Fpo])

13 f901f870 8054e017 fe51b2c8 f901f88c 80087000 nt!IopProcessStartDevices+0x43 (FPO: [EBP 0xf901f8c0] [2,0,4])

我们只讨论键盘部分的内容。

i8042prt!I8xPnP 中，调用 I8xMouseStartDevice。

I8xMouseStartDevice 中都是鼠标相关的内容，我们并不关心，直到最后，判断，如果没有ps/2键盘，或者有ps/2键盘但是已经启动了，那么将调用 i8042prt!I8xMouseInitializeHardware 初始化硬件。我这里有ps/2键盘，并且已经启动了，i8042prt!I8xMouseInitializeHardware 将被调用。

i8042prt!I8xMouseInitializeHardware 中，

调用 I8xInitializeHardwareAtBoot ，初始化硬件。

调用 I8xKeyboardConnectInterrupt ，连接键盘中断。

I8xInitializeHardwareAtBoot 中，

调用 I8xSanityCheckResources ，检查必要的资源是否存在。

调用 I8xToggleInterrupts(FALSE) ，disable 键盘中断。

调用 I8xInitializeHardware ，初始化键盘硬件和鼠标硬件，我们只讨论键盘。

调用 I8xToggleInterrupts(TRUE) ，enable 键盘中断。

I8xSanityCheckResources 中，

将 Globals.ControllerData->DeviceRegisters[i] 设置为 Globals.ControllerData->Configuration.PortList[i].u.Port.Start.LowPart

驱动中把 0x60 叫数据端口。

驱动中把 0x64 叫命令端口。

关于各种命令的详细内容请参考"1 ps/2 键盘的硬件"。

I8xToggleInterrupts(FALSE) 中，

向命令端口发 0x20 ，取出i8042的命令字节。我这里取出的命令字节为 0x47，即二进制 01000111，看到Bit0为1，说明 enable 键盘中断。Bit1为1，说明 enable 鼠标中断。使用指定的 ByteMask: ~((UCHAR) CCB_ENABLE_KEYBOARD_INTERRUPT |(UCHAR) CCB_ENABLE_MOUSE_INTERRUPT) 执行一个指定的操作: AND_OPERATION。于是命令字节变为 0x44，即二进制 01000100。禁止了键盘和鼠标的中断。向命令端口发 0x60 ，写入i8042的命令字节，将修改过的命令字节作为参数发送到数据端口，从而禁止了键盘和鼠标的中断。

I8xInitializeHardware 中，调用 I8xInitializeKeyboard 初始化键盘硬件，I8xInitializeKeyboard 中，Reset Keyboard，关闭键盘的 Translate 模式，设置键盘的 Typematic Rate 和 Typematic Delay，设置键盘指示灯，重新打开键盘的 Translate 模式。

Reset Keyboard，向数据端口发 0xff ，来 Reset Keyboard。这个命令将使Keyboard，首先回复一个ACK，然后启动自身的Reset程序，并进行自身基本正确性检测（BAT-Basic Assurance Test）。等这一切结束之后，将返回给系统一个单字节的结束码（AAh=Success, FCh=Failed），并将键盘的Scan code set设置为2。

关闭键盘的 Translate 模式，首先向命令端口发 0x20 ，取出i8042的命令字节。我这里取出的命令字节为 0x44，即二进制 01000100。使用指定的 ByteMask: ~((UCHAR)CCB_KEYBOARD_TRANSLATE_MODE) 执行一个指定的操作: AND_OPERATION。于是命令字节变为 0x4，即二进制 00000100，Bit6被清0，关闭了 Translate 模式。向命令端口发 0x60 ，写入i8042的命令字节，将修改过的命令字写入i8042控制器命令字节，从而关闭了 Translate 模式。

设置 Typematic Rate 和 Typematic Delay，向数据端口发 0xf3，这是设置 Typematic Rate 和 Typematic Delay的命令，然后将 deviceExtension->KeyRepeatCurrent.Rate,deviceExtension->KeyRepeatCurrent.Delay 转换为1个字节的设置命令的参数。deviceExtension->KeyRepeatCurrent.Rate,deviceExtension->KeyRepeatCurrent.Delay 在前面的 I8xKeyboardConfiguration 中，被设置为 KEYBOARD_TYPEMATIC_RATE_DEFAULT 和 KEYBOARD_TYPEMATIC_DELAY_DEFAULT ，KEYBOARD_TYPEMATIC_RATE_DEFAULT 为 30，KEYBOARD_TYPEMATIC_DELAY_DEFAULT 为 250。将转换得到的设置参数，发到数据端口。

设置键盘指示灯，向数据端口发 0xed，这是设置LED命令，然后将deviceExtension->KeyboardIndicators.LedFlags 作为命令的参数，发到数据端口。deviceExtension->KeyboardIndicators.LedFlags 在前面的 I8xKeyboardConfiguration 中，被设置为 KEYBOARD_INDICATORS_DEFAULT，即 0。

重新打开键盘的 Translate 模式，键盘默认发送 scan code set 2，当设置了i8042命令字节中的translate 位时，i8042在把扫描码发送给 CPU 之前，转换为 scan code set 1。首先向命令端口发 0x20 ，取出i8042的命令字节。我这里取出的命令字节为 0x4，即二进制 00000100。使用指定的 ByteMask: CCB_KEYBOARD_TRANSLATE_MODE 执行一个指定的操作: OR_OPERATION。于是命令字节变为 0x44，即二进制 01000100，Bit6被置1，打开了 Translate 模式。向命令端口发 0x60 ，写入i8042的命令字节，将修改过的命令字写入i8042控制器命令字节，从而打开了 Translate 模式。

I8xToggleInterrupts(TRUE) 中，

向命令端口发 0xad，禁止键盘接口。向命令端口发0xa7，禁止鼠标接口。向命令端口发 0x20 ，取出i8042的命令字节。我这里取出的命令字节为 0x74，即二进制 01110100，看到Bit0为0，说明 disable 键盘中断。Bit1为0，说明 disable 鼠标中断。Bit4为1，Bit5为1，说明禁止了键盘和鼠标。向命令端口发 0xae，打开键盘接口。向命令端口发0xa8，打开鼠标接口。于是命令字节变为 0x44。使用指定的 ByteMask: CCB_ENABLE_KEYBOARD_INTERRUPT |CCB_ENABLE_MOUSE_INTERRUPT 执行一个指定的操作: OR_OPERATION。于是命令字节变为 0x47，即二进制 01000111。enable 了键盘和鼠标的中断。向命令端口发 0x60 ，写入i8042的命令字节，将修改过的命令字节作为参数发送到数据端口，从而打开了键盘和鼠标的中断。

I8xKeyboardConnectInterrupt 中，

以i8042prt键盘设备扩展中的一些域为参数，调用 IoConnectInterrupt 连接键盘中断。

IoConnectInterrupt(

&(KeyboardExtension->InterruptObject),

(PKSERVICE_ROUTINE) I8042KeyboardInterruptService,

self,

&KeyboardExtension->InterruptSpinLock,

KeyboardExtension->InterruptDescriptor.u.Interrupt.Vector,

(KIRQL) KeyboardExtension->InterruptDescriptor.u.Interrupt.Level,

configuration->InterruptSynchIrql,

KeyboardExtension->InterruptDescriptor.Flags

== CM_RESOURCE_INTERRUPT_LATCHED ? Latched : LevelSensitive,

(BOOLEAN) (KeyboardExtension->InterruptDescriptor.ShareDisposition

== CmResourceShareShared),

KeyboardExtension->InterruptDescriptor.u.Interrupt.Affinity,

configuration->FloatingSave

);

其中 IoConnectInterrupt 的参数 ServiceRoutine 为 I8042KeyboardInterruptService ，参数 Vector 为 KeyboardExtension->InterruptDescriptor.u.Interrupt.Vector ，值为 0xb3。也就是说，键盘中断IRQ1，将引起中断描述符表中 0xb3 中的中断服务例程被执行，最终将执行 i8042!prtI8042KeyboardInterruptService。驱动的中断有一些特殊，详细我们将在以后讨论。

之后会得到键盘设备栈的栈顶设备对象，然后向栈顶发一个 IRP_MJ_INTERNAL_DEVICE_CONTROL IOCTL_INTERNAL_I8042_KEYBOARD_START_INFORMATION 的 IRP，不过键盘设备栈对此不做什么处理。

[kbdclass!KeyboardClassFindMorePorts]

由于在 kbdclass!DriverEntry 中 IoRegisterDriverReinitialization(DriverObject,KeyboardClassFindMorePorts,NULL);

所以现在 kbdclass!KeyboardClassFindMorePorts 被调用了。

ChildEBP RetAddr Args to Child

00 f901f614 804d92bc fe4f6330 00000000 00000001 kbdclass!KeyboardClassFindMorePorts(struct _DRIVER_OBJECT * DriverObject = 0xfe4f6330, void * Context = 0x00000000, unsigned long Count = 1) (CONV: stdcall)

01 f901f638 8052a72f f901f668 f901f6c4 f901f740 nt!IopLoadUnloadDriver+0x7a (FPO: [EBP 0xf901f6b8] [1,1,4])

02 f901f6b8 80461691 f901f760 8000003c 80000040 nt!NtLoadDriver+0x199 (FPO: [Non-Fpo])

03 f901f6b8 80400d15 f901f760 8000003c 80000040 nt!KiSystemService+0xc4 (FPO: [0,0] TrapFrame @ f901f6c4)

04 f901f734 fe22a8bd f901f760 fe4da470 fe4fea08 nt!ZwLoadDriver+0xb (FPO: [1,0,0])

05 f901f774 8041f54b fe4da470 fe4fea9c fe4da470 NDIS!ndisPnPDispatch+0x40e (FPO: [Non-Fpo])

06 f901f788 8049cb91 00020000 fe519408 00000000 nt!IopfCallDriver+0x35 (FPO: [0,0,2])

07 f901f7b4 804289ce fe4da470 f901f7d4 f901f7fc nt!IopSynchronousCall+0xca (FPO: [Non-Fpo])

08 f901f800 8048e06a fe519510 00000000 fe519408 nt!IopStartDevice+0x127 (FPO: [Non-Fpo])

09 f901f834 8048e040 fe519408 f901f88c 00000000 nt!IopStartAndEnumerateDevice+0x22 (FPO: [Non-Fpo])

0a f901f854 804a4607 fe519408 f901f88c 00000003 nt!IopProcessStartDevicesWorker+0x72 (FPO: [Non-Fpo])

0b f901f870 8054e017 fe519208 f901f88c 80087000 nt!IopProcessStartDevices+0x43 (FPO: [EBP 0xf901f8c0] [2,0,4])

0c f901f8c0 8054c5c9 00000000 00000032 00000000 nt!IopInitializeSystemDrivers+0x5d (FPO: [Non-Fpo])

0d f901fa58 8054b35a 80087000 00000000 00000000 nt!IoInitSystem+0x644 (FPO: [Non-Fpo])

0e f901fda8 804524f6 80087000 00000000 00000000 nt!Phase1Initialization+0x71b (FPO: [Non-Fpo])

0f f901fddc 80465b62 8054aca6 80087000 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

10 00000000 00000000 00000000 00000000 00000000 nt!KiThreadStartup+0x16

kbdclass!KeyboardClassFindMorePorts 中，没有什么值得注意的处理

6.3.2 应用层初始化引发键盘驱动函数被调用部分

系统初始化进行到下图所示阶段

[IRP_MJ_CREATE]

这是键盘驱动的应用层win32k!RawInputThread 调用 ZwCreateFile，来获得一个可以找到键盘设备栈的句柄。

ZwCreateFile 会创建一个 IRP_MJ_CREATE 的 IRP，并把这个 IRP 发给键盘设备栈的最顶端的设备对象。

ChildEBP RetAddr Args to Child

00 f90ef608 8041f54b fe4f5df0 fe442328 fe442338 kbdclass!KeyboardClassCreate(struct _DEVICE_OBJECT * DeviceObject = 0xfe4f5df0, struct _IRP * Irp = 0xfe442328) (CONV: stdcall)

01 f90ef61c 804a3e54 804a392a fe4dd718 f90ef90c nt!IopfCallDriver+0x35 (FPO: [0,0,2])

02 f90ef7a4 8044e27e fe4dd730 00000000 f90ef850 nt!IopParseDevice+0xa04 (FPO: [Non-Fpo])

03 f90ef810 804957ae 00000000 f90ef900 00000000 nt!ObpLookupObjectName+0x4c4 (FPO: [Non-Fpo])

04 f90ef920 804a78b8 00000000 00000000 00000000 nt!ObOpenObjectByName+0xc5 (FPO: [Non-Fpo])

05 f90ef9f4 804a0c5b e1970adc 00100001 f90efb14 nt!IoCreateFile+0x3ec (FPO: [Non-Fpo])

06 f90efa34 80461691 e1970adc 00100001 f90efb14 nt!NtCreateFile+0x2e (FPO: [Non-Fpo])

07 f90efa34 804009d1 e1970adc 00100001 f90efb14 nt!KiSystemService+0xc4 (FPO: [0,0] TrapFrame @ f90efa68)

08 f90efad8 a000e304 e1970adc 00100001 f90efb14 nt!ZwCreateFile+0xb (FPO: [11,0,0])

09 f90efb2c a000e192 e1970ac8 80400b46 00000001 win32k!OpenDevice+0x8e (FPO: [Non-Fpo])

0a f90efb58 a000eb74 00000001 00000000 00000000 win32k!ProcessDeviceChanges+0x92 (FPO: [EBP 0xf90efda8] [1,5,4])

0b f90efda8 804524f6 00000003 00000000 00000000 win32k!RawInputThread+0x463 (FPO: [Non-Fpo])

0c f90efddc 80465b62 a000e7cd f8d5f7d0 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

0d 00000000 f000ff53 f000e2c3 f000ff53 f000ff53 nt!KiThreadStartup+0x16

WARNING: Frame IP not in any known module. Following frames may be wrong.

0e f000ff53 00000000 00000000 00000000 00000000 0xf000ff53

kbdclass!KeyboardClassCreate 中，看看设备是否已经启动，deviceExtension->TrustedSubsystemCount++ 等，然后向下传。

i8042prt!I8xCreate 中，DeviceExtension->EnableCount 加1。然后结束这个 IRP。

[IRP_MJ_DEVICE_CONTROL IOCTL: IOCTL_KEYBOARD_QUERY_ATTRIBUTES]

这是键盘驱动的应用层win32k!RawInputThread 调用 ZwDeviceIoControlFile，IOCTL: IOCTL_KEYBOARD_QUERY_ATTRIBUTES 来查询键盘驱动的 kbExtension->KeyboardAttributes。

ZwDeviceIoControlFile 会创建一个 IRP_MJ_DEVICE_CONTROL 的 IRP，并把这个 IRP 发给键盘设备栈的最顶端的设备对象。

ChildEBP RetAddr Args to Child

00 f90ef908 8041f54b fe4f5df0 fe43e008 fe43e008 kbdclass!KeyboardClassDeviceControl(struct _DEVICE_OBJECT * DeviceObject = 0xfe4f5df0, struct _IRP * Irp = 0xfe43e008) (CONV: stdcall)

01 f90ef91c 804ba5e8 fe43e12c 00000000 fe43e008 nt!IopfCallDriver+0x35 (FPO: [0,0,2])

02 f90ef930 804ac5de fe4f5df0 fe43e008 fe42d908 nt!IopSynchronousServiceTail+0x60 (FPO: [Non-Fpo])

03 f90ef9fc 804a8f1e 000000d4 00000000 00000000 nt!IopXxxControlFile+0x5e4 (FPO: [Non-Fpo])

04 f90efa30 80461691 000000d4 00000000 00000000 nt!NtDeviceIoControlFile+0x28 (FPO: [Non-Fpo])

05 f90efa30 80400b51 000000d4 00000000 00000000 nt!KiSystemService+0xc4 (FPO: [0,0] TrapFrame @ f90efa60)

06 f90efad0 a000e355 000000d4 00000000 00000000 nt!ZwDeviceIoControlFile+0xb (FPO: [10,0,0])

07 f90efb00 a000e319 e196fd68 80400b46 00000000 win32k!QueryDeviceInfo+0x31 (FPO: [1,0,1])

08 f90efb2c a000e192 e196fd68 80400b46 00000001 win32k!OpenDevice+0x9f (FPO: [Non-Fpo])

09 f90efb58 a000eb74 00000001 00000000 00000000 win32k!ProcessDeviceChanges+0x92 (FPO: [EBP 0xf90efda8] [1,5,4])

0a f90efda8 804524f6 00000003 00000000 00000000 win32k!RawInputThread+0x463 (FPO: [Non-Fpo])

0b f90efddc 80465b62 a000e7cd f8d5f7d0 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

0c 00000000 f000ff53 f000e2c3 f000ff53 f000ff53 nt!KiThreadStartup+0x16

WARNING: Frame IP not in any known module. Following frames may be wrong.

0d f000ff53 00000000 00000000 00000000 00000000 0xf000ff53

kbdclass!KeyboardClassDeviceControl 中，没有处理，IRP 的下一个 IO_STACK_LOCATION 的 MajorFunction 被设置为 IRP_MJ_INTERNAL_DEVICE_CONTROL，向下传。

i8042prt!I8xInternalDeviceControl 中，

*(PKEYBOARD_ATTRIBUTES) Irp->AssociatedIrp.SystemBuffer = kbExtension->KeyboardAttributes，把i8042prt键盘设备对象的设备扩展中的KeyboardAttributes复制到 Irp->AssociatedIrp.SystemBuffer 中，然后结束这个 IRP。

[IRP_MJ_PNP IRP_MN_QUERY_DEVICE_RELATIONS]

ChildEBP RetAddr Args to Child

00 f90efa20 8041f54b fe4f5df0 fe43e008 fe4f5df0 kbdclass!KeyboardPnP(struct _DEVICE_OBJECT * DeviceObject = 0xfe4f5df0, struct _IRP * Irp = 0xfe43e008) (CONV: stdcall)

01 f90efa34 8049cb91 f90efaa0 fe42d908 00000000 nt!IopfCallDriver+0x35 (FPO: [0,0,2])

02 f90efa60 804c1800 fe4f5df0 f90efa7c f90efaa8 nt!IopSynchronousCall+0xca (FPO: [Non-Fpo])

03 f90efaa0 804a3306 fe42d908 f90efb04 00000000 nt!IopGetRelatedTargetDevice+0x4b (FPO: [Non-Fpo])

04 f90efafc a000e3a2 00000003 00000000 fe42d908 nt!IoRegisterPlugPlayNotification+0x71 (FPO: [Non-Fpo])

05 f90efb2c a000e09b fe42d908 80400b46 00000001 win32k!RegisterForDeviceChangeNotifications+0x46 (FPO: [Non-Fpo])

06 f90efb58 a000eb74 00000001 00000000 00000000 win32k!ProcessDeviceChanges+0x9c (FPO: [EBP 0xf90efda8] [1,5,4])

07 f90efda8 804524f6 00000003 00000000 00000000 win32k!RawInputThread+0x463 (FPO: [Non-Fpo])

08 f90efddc 80465b62 a000e7cd f8d5f7d0 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

09 00000000 f000ff53 f000e2c3 f000ff53 f000ff53 nt!KiThreadStartup+0x16

WARNING: Frame IP not in any known module. Following frames may be wrong.

0a f000ff53 00000000 00000000 00000000 00000000 0xf000ff53

kbdclass!KeyboardPnP 和 i8042prt!I8xPnP 中对此都没有处理。

[IRP_MJ_READ]

这是键盘驱动的应用层win32k!RawInputThread 向键盘驱动发来的第一个读请求，请求读入数据。从现在开始应用层开始处理键盘的输入了。

ChildEBP RetAddr Args to Child

00 f90ef960 8041f54b fe4f5df0 fe43e008 fe43e008 kbdclass!KeyboardClassRead(struct _DEVICE_OBJECT * Device = 0xfe4f5df0, struct _IRP * Irp = 0xfe43e008) (CONV: stdcall)

01 f90ef974 804ba5e8 fe43e12c fe43e008 00000000 nt!IopfCallDriver+0x35 (FPO: [0,0,2])

02 f90ef988 804a2d4c fe4f5df0 fe43e008 fe42d908 nt!IopSynchronousServiceTail+0x60 (FPO: [Non-Fpo])

03 f90efa5c 80461691 000000d4 00000000 a005c962 nt!NtReadFile+0x5f4 (FPO: [Non-Fpo])

04 f90efa5c 804011d5 000000d4 00000000 a005c962 nt!KiSystemService+0xc4 (FPO: [0,0] TrapFrame @ f90efa88)

05 f90efaf8 a005c91d 000000d4 00000000 a005c962 nt!ZwReadFile+0xb (FPO: [9,0,0])

06 f90efb2c a000e0ad e196fd68 80400b46 00000001 win32k!StartDeviceRead+0x8c (FPO: [1,0,3])

07 f90efb58 a000eb74 00000001 00000000 00000000 win32k!ProcessDeviceChanges+0xb0 (FPO: [EBP 0xf90efda8] [1,5,4])

08 f90efda8 804524f6 00000003 00000000 00000000 win32k!RawInputThread+0x463 (FPO: [Non-Fpo])

09 f90efddc 80465b62 a000e7cd f8d5f7d0 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

0a 00000000 f000ff53 f000e2c3 f000ff53 f000ff53 nt!KiThreadStartup+0x16

WARNING: Frame IP not in any known module. Following frames may be wrong.

0b f000ff53 00000000 00000000 00000000 00000000 0xf000ff53

键盘驱动对于应用层读请求的处理，我们将在后面讨论。

[IRP_MJ_DEVICE_CONTROL IOCTL: IOCTL_KEYBOARD_QUERY_INDICATORS]

键盘驱动的应用层win32k!RawInputThread 查询指示灯。

ChildEBP RetAddr Args to Child

00 fe14b940 8041f54b fe4f6df0 fe43e8a8 fe43e8a8 kbdclass!KeyboardClassDeviceControl(struct _DEVICE_OBJECT * DeviceObject = 0xfe4f6df0, struct _IRP * Irp = 0xfe43e8a8)+0x6 (CONV: stdcall)

01 fe14b954 804ba5e8 fe43e9cc 00000000 fe43e8a8 nt!IopfCallDriver+0x35 (FPO: [0,0,2])

02 fe14b968 804ac5de fe4f6df0 fe43e8a8 fe42c8a8 nt!IopSynchronousServiceTail+0x60 (FPO: [Non-Fpo])

03 fe14ba34 804a8f1e 000000d4 00000000 00000000 nt!IopXxxControlFile+0x5e4 (FPO: [Non-Fpo])

04 fe14ba68 80461691 000000d4 00000000 00000000 nt!NtDeviceIoControlFile+0x28 (FPO: [Non-Fpo])

05 fe14ba68 80400b51 000000d4 00000000 00000000 nt!KiSystemService+0xc4 (FPO: [0,0] TrapFrame @ fe14ba98)

06 fe14bb08 a000e10f 000000d4 00000000 00000000 nt!ZwDeviceIoControlFile+0xb (FPO: [10,0,0])

07 fe14bb58 a000eb74 00000001 00000000 00000000 win32k!ProcessDeviceChanges+0x190 (FPO: [EBP 0xfe14bda8] [1,5,4])

08 fe14bda8 804524f6 00000003 00000000 00000000 win32k!RawInputThread+0x463 (FPO: [Non-Fpo])

09 fe14bddc 80465b62 a000e7cd f8d5f7d0 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

0a 00000000 f000ff53 f000e2c3 f000ff53 f000ff53 nt!KiThreadStartup+0x16

WARNING: Frame IP not in any known module. Following frames may be wrong.

0b f000ff53 00000000 00000000 00000000 00000000 0xf000ff53

kbdclass 层向下传，i8042prt 层中，将 kbExtension->KeyboardIndicators 复制到

*(PKEYBOARD_INDICATOR_PARAMETERS) Irp->AssociatedIrp.SystemBuffer 中，结束 IRP。

[IRP_MJ_DEVICE_CONTROL IOCTL: IOCTL_KEYBOARD_SET_INDICATORS]

键盘驱动的应用层win32k!RawInputThread 设置指示灯。

ChildEBP RetAddr Args to Child

00 fe14b92c 8041f54b fe4f6df0 fe43e8a8 fe43e8a8 kbdclass!KeyboardClassDeviceControl(struct _DEVICE_OBJECT * DeviceObject = 0xfe4f6df0, struct _IRP * Irp = 0xfe43e8a8) (CONV: stdcall)

01 fe14b940 804ba5e8 fe42d64c 00000000 fe43e8a8 nt!IopfCallDriver+0x35 (FPO: [0,0,2])

02 fe14b954 804ac5de fe4f6df0 fe43e8a8 fe42c8a8 nt!IopSynchronousServiceTail+0x60 (FPO: [Non-Fpo])

03 fe14ba20 804a8f1e 000000d4 00000000 00000000 nt!IopXxxControlFile+0x5e4 (FPO: [Non-Fpo])

04 fe14ba54 80461691 000000d4 00000000 00000000 nt!NtDeviceIoControlFile+0x28 (FPO: [Non-Fpo])

05 fe14ba54 80400b51 000000d4 00000000 00000000 nt!KiSystemService+0xc4 (FPO: [0,0] TrapFrame @ fe14ba84)

06 fe14baf4 a0000503 000000d4 00000000 00000000 nt!ZwDeviceIoControlFile+0xb (FPO: [10,0,0])

07 fe14bb2c a000e206 00000000 80400b46 00000001 win32k!UpdateKeyLights+0xfe (FPO: [1,0,3])

08 fe14bb58 a000eb74 00000001 00000000 00000000 win32k!ProcessDeviceChanges+0x224 (FPO: [EBP 0xfe14bda8] [1,5,4])

09 fe14bda8 804524f6 00000003 00000000 00000000 win32k!RawInputThread+0x463 (FPO: [Non-Fpo])

0a fe14bddc 80465b62 a000e7cd f8d5f7d0 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

0b 00000000 f000ff53 f000e2c3 f000ff53 f000ff53 nt!KiThreadStartup+0x16

WARNING: Frame IP not in any known module. Following frames may be wrong.

0c f000ff53 00000000 00000000 00000000 00000000 0xf000ff53

[IRP_MJ_DEVICE_CONTROL IOCTL: IOCTL_KEYBOARD_SET_TYPEMATIC]

键盘驱动的应用层win32k!RawInputThread 设置 Typematic。

ChildEBP RetAddr Args to Child

00 fe14b96c 8041f54b fe4f6df0 fe43d1c8 fe43d1c8 kbdclass!KeyboardClassDeviceControl(struct _DEVICE_OBJECT * DeviceObject = 0xfe4f6df0, struct _IRP * Irp = 0xfe43d1c8) (CONV: stdcall)

01 fe14b980 804ba5e8 fe42d6ce 00000000 fe43d1c8 nt!IopfCallDriver+0x35 (FPO: [0,0,2])

02 fe14b994 804ac5de fe4f6df0 fe43d1c8 fe42c8a8 nt!IopSynchronousServiceTail+0x60 (FPO: [Non-Fpo])

03 fe14ba60 804a8f1e 000000d4 00000000 00000000 nt!IopXxxControlFile+0x5e4 (FPO: [Non-Fpo])

04 fe14ba94 80461691 000000d4 00000000 00000000 nt!NtDeviceIoControlFile+0x28 (FPO: [Non-Fpo])

05 fe14ba94 80400b51 000000d4 00000000 00000000 nt!KiSystemService+0xc4 (FPO: [0,0] TrapFrame @ fe14bac4)

06 fe14bb34 a000ea81 000000d4 00000000 00000000 nt!ZwDeviceIoControlFile+0xb (FPO: [10,0,0])

07 fe14bda8 804524f6 00000002 00000000 00000000 win32k!RawInputThread+0x40b (FPO: [Non-Fpo])

08 fe14bddc 80465b62 a000e7cd f8d5f7d0 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

09 00000000 f000ff53 f000e2c3 f000ff53 f000ff53 nt!KiThreadStartup+0x16

WARNING: Frame IP not in any known module. Following frames may be wrong.

0a f000ff53 00000000 00000000 00000000 00000000 0xf000ff53

[IRP_MJ_DEVICE_CONTROL IOCTL: IOCTL_KEYBOARD_SET_INDICATORS]

键盘驱动的应用层win32k!RawInputThread 设置指示灯。

ChildEBP RetAddr Args to Child

00 fe14b96c 8041f54b fe4f6df0 fe43d1c8 fe43d1c8 kbdclass!KeyboardClassDeviceControl(struct _DEVICE_OBJECT * DeviceObject = 0xfe4f6df0, struct _IRP * Irp = 0xfe43d1c8) (CONV: stdcall)

01 fe14b980 804ba5e8 fe42d6ac 00000000 fe43d1c8 nt!IopfCallDriver+0x35 (FPO: [0,0,2])

02 fe14b994 804ac5de fe4f6df0 fe43d1c8 fe42c8a8 nt!IopSynchronousServiceTail+0x60 (FPO: [Non-Fpo])

03 fe14ba60 804a8f1e 000000d4 00000000 00000000 nt!IopXxxControlFile+0x5e4 (FPO: [Non-Fpo])

04 fe14ba94 80461691 000000d4 00000000 00000000 nt!NtDeviceIoControlFile+0x28 (FPO: [Non-Fpo])

05 fe14ba94 80400b51 000000d4 00000000 00000000 nt!KiSystemService+0xc4 (FPO: [0,0] TrapFrame @ fe14bac4)

06 fe14bb34 a000eac3 000000d4 00000000 00000000 nt!ZwDeviceIoControlFile+0xb (FPO: [10,0,0])

07 fe14bda8 804524f6 00000002 00000000 00000000 win32k!RawInputThread+0x42f (FPO: [Non-Fpo])

08 fe14bddc 80465b62 a000e7cd f8d5f7d0 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

09 00000000 f000ff53 f000e2c3 f000ff53 f000ff53 nt!KiThreadStartup+0x16

WARNING: Frame IP not in any known module. Following frames may be wrong.

0a f000ff53 00000000 00000000 00000000 00000000 0xf000ff53

[IRP_MJ_DEVICE_CONTROL IOCTL: IOCTL_KEYBOARD_SET_TYPEMATIC]

键盘驱动的应用层win32k!RawInputThread 设置 Typematic。

ChildEBP RetAddr Args to Child

00 fe14b96c 8041f54b fe4f6df0 fe3dba48 fe3dba48 kbdclass!KeyboardClassDeviceControl(struct _DEVICE_OBJECT * DeviceObject = 0xfe4f6df0, struct _IRP * Irp = 0xfe3dba48) (CONV: stdcall)

01 fe14b980 804ba5e8 fe4455ee 00000000 fe3dba48 nt!IopfCallDriver+0x35 (FPO: [0,0,2])

02 fe14b994 804ac5de fe4f6df0 fe3dba48 fe42c8a8 nt!IopSynchronousServiceTail+0x60 (FPO: [Non-Fpo])

03 fe14ba60 804a8f1e 000000d4 00000000 00000000 nt!IopXxxControlFile+0x5e4 (FPO: [Non-Fpo])

04 fe14ba94 80461691 000000d4 00000000 00000000 nt!NtDeviceIoControlFile+0x28 (FPO: [Non-Fpo])

05 fe14ba94 80400b51 000000d4 00000000 00000000 nt!KiSystemService+0xc4 (FPO: [0,0] TrapFrame @ fe14bac4)

06 fe14bb34 a000ea81 000000d4 00000000 00000000 nt!ZwDeviceIoControlFile+0xb (FPO: [10,0,0])

07 fe14bda8 804524f6 00000002 00000000 00000000 win32k!RawInputThread+0x40b (FPO: [Non-Fpo])

08 fe14bddc 80465b62 a000e7cd f8d5f7d0 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

09 00000000 f000ff53 f000e2c3 f000ff53 f000ff53 nt!KiThreadStartup+0x16

WARNING: Frame IP not in any known module. Following frames may be wrong.

0a f000ff53 00000000 00000000 00000000 00000000 0xf000ff53

[IRP_MJ_DEVICE_CONTROL IOCTL: IOCTL_KEYBOARD_SET_INDICATORS]

键盘驱动的应用层win32k!RawInputThread 设置指示灯。

ChildEBP RetAddr Args to Child

00 fe14b96c 8041f54b fe4f6df0 fe3dba48 fe3dba48 kbdclass!KeyboardClassDeviceControl(struct _DEVICE_OBJECT * DeviceObject = 0xfe4f6df0, struct _IRP * Irp = 0xfe3dba48) (CONV: stdcall)

01 fe14b980 804ba5e8 fe3e060c 00000000 fe3dba48 nt!IopfCallDriver+0x35 (FPO: [0,0,2])

02 fe14b994 804ac5de fe4f6df0 fe3dba48 fe42c8a8 nt!IopSynchronousServiceTail+0x60 (FPO: [Non-Fpo])

03 fe14ba60 804a8f1e 000000d4 00000000 00000000 nt!IopXxxControlFile+0x5e4 (FPO: [Non-Fpo])

04 fe14ba94 80461691 000000d4 00000000 00000000 nt!NtDeviceIoControlFile+0x28 (FPO: [Non-Fpo])

05 fe14ba94 80400b51 000000d4 00000000 00000000 nt!KiSystemService+0xc4 (FPO: [0,0] TrapFrame @ fe14bac4)

06 fe14bb34 a000eac3 000000d4 00000000 00000000 nt!ZwDeviceIoControlFile+0xb (FPO: [10,0,0])

07 fe14bda8 804524f6 00000002 00000000 00000000 win32k!RawInputThread+0x42f (FPO: [Non-Fpo])

08 fe14bddc 80465b62 a000e7cd f8d5f7d0 00000000 nt!PspSystemThreadStartup+0x69 (FPO: [Non-Fpo])

09 00000000 f000ff53 f000e2c3 f000ff53 f000ff53 nt!KiThreadStartup+0x16

WARNING: Frame IP not in any known module. Following frames may be wrong.

0a f000ff53 00000000 00000000 00000000 00000000 0xf000ff53

6.4 键盘驱动初始化概要

驱动对象

IO 管理器根据注册表中安装驱动时写入的信息，载入键盘驱动文件 i8042prt.sys 和 kbdclass.sys 。为他们创建驱动对象，并调用他们的 DriverEntry。

设备对象

在 i8042prt!I8xAddDevice 中调用 IoCreateDevice 创建i8042prt的键盘设备对象。

在 kbdclass!KeyboardAddDevice 中，调用 IoCreateDevice 创建kbdclass的设备对象。

键盘设备栈

IO 管理器根据注册表中安装驱动时写入的信息，调用 i8042prt!I8xAddDevice 和 kbdclass!KeyboardAddDevice 来建立键盘设备栈。在 i8042prt!I8xAddDevice 和 kbdclass!KeyboardAddDevice 中，调用 IoCreateDevice 创建设备对象之后，调用 IoAttachDeviceToDeviceStack 将产生的设备对象连入键盘设备栈。

输入数据队列

kbdclass 的输入数据队列是在 kbdclass!KeyboardAddDevice 中，分配内存，并初始化设备扩展中的使用输入数据队列的相关域。i8042prt 的输入数据队列是在 i8042prt!I8xKeyboardStartDevice 中，分配内存，并初始化设备扩展中的使用输入数据队列的相关域。i8042prt!I8xKeyboardStartDevice 是 IO 管理器向键盘设备栈发 IRP_MJ_PNP IRP_MN_START_DEVICE 的 IRP，被 i8042prt 调用的。

将kbdclass处理输入数据的回调函数告诉i8042prt

在 kbdclass!KeyboardAddDevice 中，调用函数 kbdclass!KbdSendConnectRequest ，发一个 IRP_MJ_INTERNAL_DEVICE_CONTROL IOCTL_INTERNAL_KEYBOARD_CONNECT 的 IRP 给 i8042prt，i8042prt 从中得到kbdclass处理输入数据的回调函数的入口地址，把它保存在自己的设备扩展中。

初始化硬件，连接中断

硬件的初始化工作，对于ps/2的鼠标和ps/2的键盘都是向i8042发一些命令。所以驱动 i8042prt 把ps/2键盘鼠标的硬件初始化工作放在了一起进行。我这里既有ps/2鼠标又有ps/2键盘，而ps/2鼠标后启动，所以硬件的初始化工作在 ps/2 鼠标的启动中进行。IO 管理器发 IRP_MJ_PNP IRP_MN_START_DEVICE 的 IRP 给鼠标设备栈，初始化硬件，对于键盘，包括 Reset Keyboard，设置 Typematic Rate 和 Typematic Delay，设置键盘指示灯，打开键盘的 Translate 模式。调用 IoConnectInterrupt 连接键盘中断。

暴露驱动接口给应用层

在 kbdclass!KeyboardAddDevice 中，调用 IoRegisterDeviceInterface，注册键盘设备接口。在IO管理器发给键盘设备栈 IRP_MJ_PNP IRP_MN_START_DEVICE 的 IRP 而引起的 kbdclass!KeyboardPnP 中，调用 IoSetDeviceInterfaceState，enable 键盘驱动暴露给应用层的接口。

应用层获得句柄

可以看到键盘驱动的应用层win32k!RawInputThread 调用 ZwCreateFile，来获得一个可以找到键盘设备栈的句柄。ZwCreateFile 发一个 IRP_MJ_CREATE 的 IRP 给键盘设备栈的最顶端的设备对象。引起 kbdclass!KeyboardClassCreate 被执行。

应用层发第一个 IRP_MJ_READ 的 IRP

可以看到键盘驱动的应用层win32k!RawInputThread 向键盘设备栈发的第一个 IRP_MJ_READ 的 IRP。引起 kbdclass!KeyboardClassRead 被执行。