

Windows引导过程

MIKE WANG (WMIKE@OUTLOOK.COM)

2016.06.20

部分细节详见“[Windows引导过程讲稿注释](#)”




修改历史记录

时间	作者	修改内容
2016-06-20	Mike Wang	第一个版本
2016-07-03	Mike Wang	增加PnP和驱动载入流程

范围和提要

- ▶ 分析范围
 - ▶ Windows 7.x, 8.x, and 10.x
 - ▶ 只关注64位系统
 - ▶ 通过BIOS从本地硬盘引导
 - ▶ 或通过UEFI从本地硬盘引导
 - ▶ 分析从开始引导到第一个用户态程序运行的整个过程
 - ▶ 主要关注磁盘I/O相关部分
- ▶ 提要
 - ▶ Windows 缺省磁盘布局
 - ▶ 典型引导过程分析 (MBR引导过程)
 - ▶ UEFI引导过程
 - ▶ 高级话题

Windows 缺省分区布局

 Disk 0 Basic 60.00 GB Online	System Reserved 500 MB NTFS Healthy (System, Active, Primary Partition)	(C:) 59.51 GB NTFS Healthy (Boot, Page File, Crash Dump, Primary Partition) Windows 10
 Disk 0 Basic 60.00 GB Online	System Reserved 350 MB NTFS Healthy (System, Active, Primary Partition)	(C:) 59.66 GB NTFS Healthy (Boot, Page File, Crash Dump, Primary Partition) Windows 8
 Disk 0 Basic 60.00 GB Online	System Reserved 100 MB NTFS Healthy (System, Active, Primary Partition)	(C:) 59.90 GB NTFS Healthy (Boot, Page File, Crash Dump, Primary Partition) Windows 7

▶ 系统保留分区

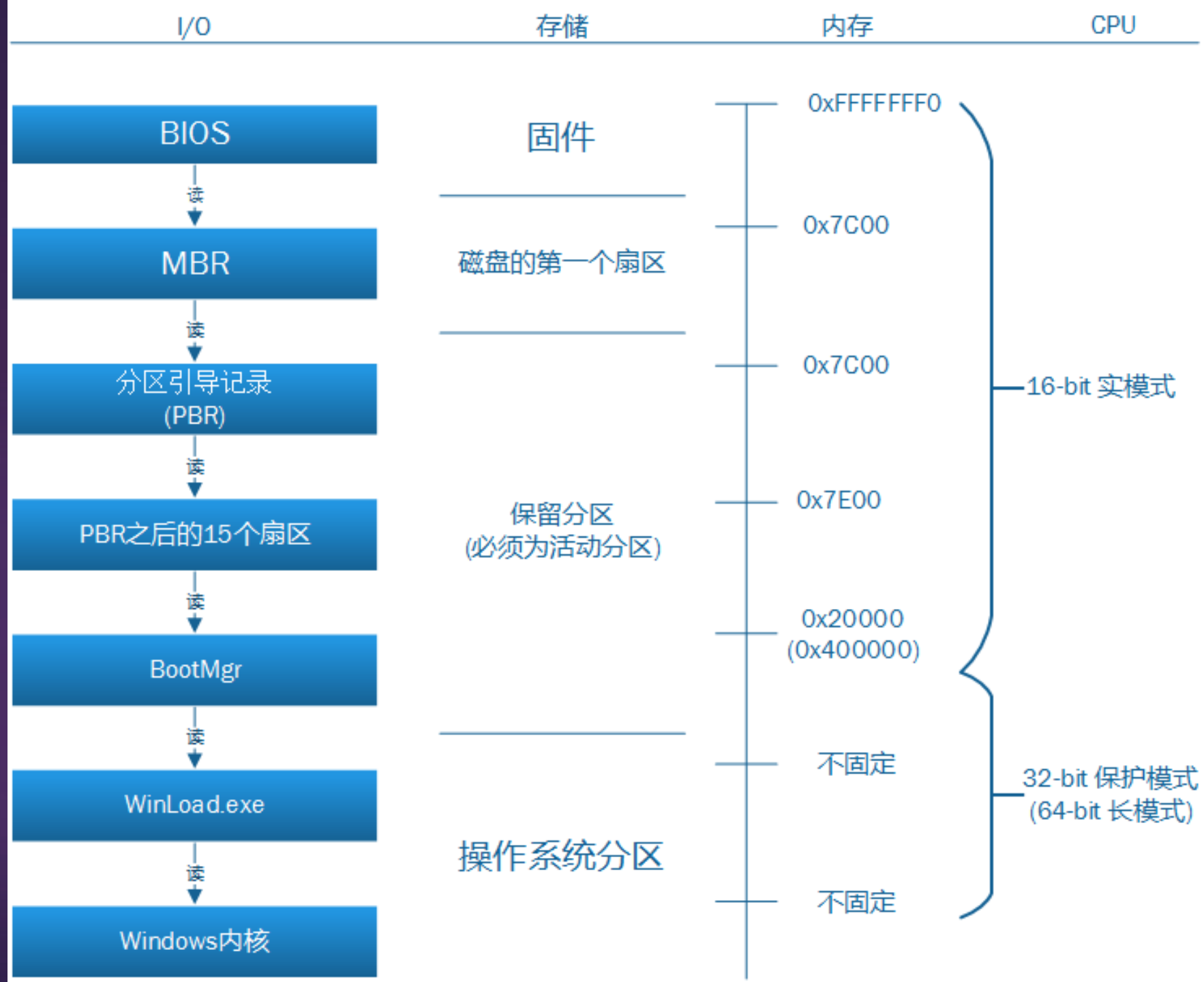
- ▶ 用于存放引导管理程序（包括分区引导记录PBR，引导程序扇区，BootMgr）
- ▶ 为满足BitLocker的需求，必须是一个单独的小分区
- ▶ 必须为活动分区
- ▶ 从Windows 8开始，WinRE也被存放在该分区；而到Windows 10的时候，WinRE需要更多存储空间

▶ 操作系统分区

- ▶ 用于存放操作系统载入程序（WinLoad.exe）和操作系统本身
- ▶ 有时候也被称作引导分区，容易误解

典型引导过程分析...

MBR引导过程



BIOS

- ▶ Intel x86 CPU 在启动后总是会先在实模式下运行地址FFFF:FFF0处的指令
- ▶ 那里一般为一个跳转指令，跳转到BIOS代码的入口
- ▶ BIOS会在配置的启动设备中找到第一个可以引导的设备
- ▶ 作为可引导设备，第一个扇区的最后两个字节必须是0xAA55
- ▶ BIOS 会将第一个扇区读入到内存地址0x7C00
- ▶ 如果该机器包含TPM 1.2+的支持
 - ▶ 计算MBR和分区表的哈希值，并存储在TPM的PCR寄存器中

注：如果硬件系统支持TPM，TPM模块会首先获得运行，计算自身和BIOS ROM的哈希值，并存于PCR寄存器中。

MBR

- ▶ 该部分代码工作在16位实模式下
- ▶ 它会先将自身复制到地址为0000:0600的位置，然后从那里继续执行代码
- ▶ 检查分区表中的每一个分区，直到找到一个活动分区
 - ▶ 只会检查主分区，而不会去检查扩展分区
- ▶ 将活动分区的第一个扇区（即分区引导记录PBR）载入到地址0000:7C00的位置
 - ▶ 先尝试用LBA模式读磁盘，如果失败则会尝试用CHS模式读取
- ▶ 如果检查到硬件支持TPM 1.2+
 - ▶ 打开A20地址线（以访问大于1MB的内存地址）
 - ▶ 通过BIOS中断运行TCG_CompactHashLogExtendEvent例程
 - ▶ 计算PBR的哈希值并将结果存储于TPM的PCR寄存器中

Offset	00 01 02 03 04 05 06 07	08 09 0A 0B 0C 0D 0E 0F	ASCII	Name	Offset	Value
0000000000	33 C0 8E D0 BC 00 7C 8E	C0 8E D8 BE 00 7C BF 00	3ÄŽĐ%. ŽÄŽĐ%. ž.	Bootstrap code	000	33 C0 8E D0 .
0000000010	06 B9 00 02 FC F3 A4 50	68 1C 06 CB FB B9 04 00	. ' . . üó×Ph..Ěû¹..	Disk serial number	1B8	27 4E D0 D9
0000000020	BD BE 07 80 7E 00 00 7C	0B 0F 85 0E 01 83 C5 10	%%.€~..fÅ.	(reserved)	1BC	00 00
0000000030	E2 F1 CD 18 88 56 00 55	C6 46 11 05 C6 46 10 00	ãñí. ^V.UÆF..ÆF..	Partition 1 (NTFS, 350 MB)	1BE	
0000000040	B4 41 BB AA 55 CD 13 5D	72 0F 81 FB 55 AA 75 09	'A»²UÍ.]r..ûU²u.	Active partition flag (80 = a...	1BE	0x80
0000000050	F7 C1 01 00 74 03 FE 46	10 66 60 80 7E 10 00 74	÷Á..t.pF.f`ë~..t	Start head	1BF	32
0000000060	26 66 68 00 00 00 00 66	FF 76 08 68 00 00 68 00	sfh....fÿv.h..h.	Start sector (bits 0-5), cylin...	1C0	0x21
0000000070	7C 68 01 00 68 10 00 B4	42 8A 56 00 8B F4 CD 13	h..h..`BŠV.<ôÍ.	Start cylinder (lower 8 bits)	1C1	0x00
0000000080	9F 83 C4 10 9E EB 14 B8	01 02 BB 00 7C 8A 56 00	ŸfÄ.žě.,...». ŠV.	File system ID	1C2	0x07
0000000090	8A 76 01 8A 4E 02 8A 6E	03 CD 13 66 61 73 1C FE	Šv.ŠN.Šn.Í.fas.p	End head	1C3	190
00000000A0	4E 11 75 0C 80 7E 00 80	0F 84 8A 00 B2 80 EB 84	N.u.ë~.ë.,,Š.²ë.,,	End sector (bits 0-5), cylin...	1C4	0x12
00000000B0	55 32 E4 8A 56 00 CD 13	5D EB 9E 81 3E FE 7D 55	U2äŠV.Í.]ěž.>þ}U	End cylinder (lower 8 bits)	1C5	0x2C
00000000C0	AA 75 6E FF 76 00 E8 8D	00 75 17 FA B0 D1 E6 64	²unyv.è..u.ú°Ñæd	First sector	1C6	2,048
00000000D0	E8 83 00 B0 DF E6 60 E8	7C 00 B0 FF E6 64 E8 75	èf.°Bæ`è .°ÿædèu	Total sectors	1CA	716,800
00000000E0	00 FB B8 00 BB CD 1A 66	23 C0 75 3B 66 81 FB 54	.û.,.»Í.f#Äu;f.ûT	Partition 2 (NTFS, 156 GB)	1CE	
00000000F0	43 50 41 75 32 81 F9 02	01 72 2C 66 68 07 BB 00	CPAu2.ù..r,fh.».	Active partition flag (80 = a...	1CE	0x00
0000000100	00 66 68 00 02 00 00 66	68 08 00 00 00 66 53 66	.fh....fh....fSf	Start head	1CF	190
0000000110	53 66 55 66 68 00 00 00	00 66 68 00 7C 00 00 66	SfUfh....fh. ..f	Start sector (bits 0-5), cylin...	1D0	0x13
0000000120	61 68 00 00 07 CD 1A 5A	32 F6 EA 00 7C 00 00 CD	ah...Í.Z2öê. ...Í	Start cylinder (lower 8 bits)	1D1	0x2C
0000000130	18 A0 B7 07 EB 08 A0 B6	07 EB 03 A0 B5 07 32 E4	. . .ë. ¶.ë. p.2ä	File system ID	1D2	0x07
0000000140	05 00 07 8B F0 AC 3C 00	74 09 BB 07 00 B4 0E CD	...<ð-<.t.»...´.Í	End head	1D3	254
0000000150	10 EB F2 F4 EB FD 2B C9	E4 64 EB 00 24 02 E0 F8	.ëòðëÿ+Éäde.\$.àø	End sector (bits 0-5), cylin...	1D4	0xFF
0000000160	24 02 C3 49 6E 76 61 6C	69 64 20 70 61 72 74 69	\$.ÄInvalid parti	End cylinder (lower 8 bits)	1D5	0xFF
0000000170	74 69 6F 6E 20 74 61 62	6C 65 00 45 72 72 6F 72	tion table.Error	First sector	1D6	718,848
0000000180	20 6C 6F 61 64 69 6E 67	20 6F 70 65 72 61 74 69	loading operati	Total sectors	1DA	326,963,200
0000000190	6E 67 20 73 79 73 74 65	6D 00 4D 69 73 73 69 6E	ng system.Missin	Partition 3 (NTFS, 310 GB)	1DE	
00000001A0	67 20 6F 70 65 72 61 74	69 6E 67 20 73 79 73 74	g operating syst	Active partition flag (80 = a...	1DE	0x00
00000001B0	65 6D 00 00 00 63 7B 9A	27 4E D0 D9 00 00 80 20	em...c{š'NDÛ..ë	Start head	1DF	254
00000001C0	21 00 07 BE 12 2C 00 08	00 00 00 F0 0A 00 00 BE	!..%.,.....ð...%	Start sector (bits 0-5), cylin...	1E0	0xFF
00000001D0	13 2C 07 FE FF FF 00 F8	0A 00 00 10 7D 13 00 FE	.,.þÿÿ.ø....}.þ	Start cylinder (lower 8 bits)	1E1	0xFF
00000001E0	FF FF 07 FE FF FF 00 08	88 13 00 48 B0 26 00 00	ÿÿ.þÿÿ..^...H°&..	File system ID	1E2	0x07
00000001F0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00U²	End head	1E3	254
				End sector (bits 0-5), cylin...	1E4	0xFF
				End cylinder (lower 8 bits)	1E5	0xFF

error message locator

check offset 0x163, 0x17B, 0x19A

分区引导记录(PBR)

- ▶ 每个分区的第一个扇区
- ▶ PBR开始的部分包含了很多重要的引导参数，例如：
 - ▶ 每个扇区包含的字节数
 - ▶ 每个簇包含的扇区数
 - ▶ MFT的簇号
 - ▶ 每个MFT记录包含的簇数
 - ▶ 每个索引块包含的簇数
- ▶ 检查当前分区是否使用NTFS文件系统
- ▶ 将PBR之后的15个扇区的数据载入到内存地址0x7E00的位置
 - ▶ 只会使用LBA模式读磁盘
- ▶ 如果检查到硬件支持TPM 1.2+
 - ▶ 通过BIOS中断运行TCG_CompactHashLogExtendEvent例程
 - ▶ 计算Boot Loader Area的哈希值并将结果存储于TPM的PCR寄存器中

如果当前卷被BitLocker加密, 那么OEM ID 将为 "-FVE-FS-", 并且 \$MFTMirr 将指向BitLocker元数据的块。

Offset	00 01 02 03 04 05 06 07	08 09 0A 0B 0C 0D 0E 0F	ASCII	Name	Offset	Value
0000100000	EB 52 90 4E 54 46 53 20	20 20 20 00 02 08 00 00	er.NTFS	JMP instruction	000	EB 52 90
0000100010	00 00 00 00 00 F8 00 00	3F 00 FF 00 00 08 00 002.v.....	OEM ID	003	NTFS
0000100020	00 00 00 00 80 00 80 00	FF EF 0A 00 00 00 00 00	...E.e.yi.....	BIOS Parameter Block	00B	
0000100030	AA 74 00 00 00 00 00 00	02 00 00 00 00 00 00 00	*t.....	Bytes per sector	00B	512
0000100040	F6 00 00 00 01 00 00 00	38 C8 6E 34 E7 6E 34 98	ö.....8Ën4çn4-	Sectors per cluster	00D	8
0000100050	00 00 00 00 FA 33 C0 8E	D0 BC 00 7C FB 68 C0 07ú3ÄžÐ%. ûhÄ.	Reserved sectors	00E	0
0000100060	1F 1E 68 66 00 CB 88 16	0E 00 66 81 3E 03 00 4E	..hf.Ë^...f.>..N	(always zero)	010	00 00 00
0000100070	54 46 53 75 15 B4 41 BB	AA 55 CD 13 72 0C 81 FB	TFSu.´A»*Uí.r..û	(unused)	013	00 00
0000100080	55 AA 75 06 F7 C1 01 00	75 03 E9 DD 00 1E 83 EC	U²u.+Ä..u.éÝ..fi	Media descriptor	015	248
0000100090	18 68 1A 00 B4 48 8A 16	0E 00 8B F4 16 1F CD 13	.h..´HŠ...<ô..í.	(unused)	016	00 00
00001000A0	9F 83 C4 18 9E 58 1F 72	E1 3B 06 0B 00 75 DB A3	ŸfÄ.žX.rá;...uŮ&	Sectors per track	018	63
00001000B0	0F 00 C1 2E 0F 00 04 1E	5A 33 DB B9 00 20 2B C8	..Á.....Z3Ů¹. +È	Number of heads	01A	255
00001000C0	66 FF 06 11 00 03 16 0F	00 8E C2 FF 06 16 00 E8	fÿ.....ŽÄÿ...è	Hidden sectors	01C	2,048
00001000D0	4B 00 2B C8 77 EF B8 00	BB CD 1A 66 23 C0 75 2D	K.+Èwī,.»Í.f#Äu-	(unused)	020	00 00 00 00
00001000E0	66 81 FB 54 43 50 41 75	24 81 F9 02 01 72 1E 16	f.ûTCPAu&.ù..r..	Signature	024	80 00 80 00
00001000F0	68 07 BB 16 68 52 11 16	68 09 00 66 53 66 53 66	h.».hR..h..fSfSf	Total sectors	028	716,799
0000100100	55 16 16 16 68 B8 01 66	61 0E 07 CD 1A 33 C0 BF	U...h,.fa..Í.3Äž	\$MFT cluster number	030	29,866
0000100110	0A 13 B9 F6 0C FC F3 AA	E9 FE 01 90 90 66 60 1E	..¹ö.üó²ép...f¹.	\$MFTMirr cluster number	038	2
0000100120	06 66 A1 11 00 66 03 06	1C 00 1E 66 68 00 00 00	.fj...f.....fh...	Clusters per File Record Segment	040	246
0000100130	00 66 50 06 53 68 01 00	68 10 00 B4 42 8A 16 0E	.fP.Sh..h..´BŠ..	Clusters per Index Block	044	1
0000100140	00 16 1F 8B F4 CD 13 66	59 5B 5A 66 59 66 59 1F	...<ôÍ.fY[ZfYfY.	Volume serial number	048	38 C8 6E 34 E7 6...
0000100150	0F 82 16 00 66 FF 06 11	00 03 16 0F 00 8E C2 FF	...fÿ.....ŽÄÿ	Checksum	050	0
0000100160	0E 16 00 75 BC 07 1F 66	61 C3 A1 F6 01 E8 09 00	...u»..faÄ;ö.è..	Bootstrap code	054	FA 33 C0 8E D0 ...
0000100170	A1 FA 01 E8 03 00 F4 EB	FD 8B F0 AC 3C 00 74 09	jú.è..ôëý<ð-<.t.	Signature (55 AA)	1FE	55 AA
0000100180	B4 0E BB 07 00 CD 10 EB	F2 C3 0D 0A 41 20 64 69	´.»...Í.èöÄ..A di			
0000100190	73 6B 20 72 65 61 64 20	65 72 72 6F 72 20 6F 63	sk read error oc			
00001001A0	63 75 72 72 65 64 00 0D	0A 42 4F 4F 54 4D 47 52	curred...BOOTMGR			
00001001B0	20 69 73 20 63 6F 6D 70	72 65 73 73 65 64 00 0D	is compressed..			
00001001C0	0A 50 72 65 73 73 20 43	74 72 6C 2B 41 6C 74 2B	.Press Ctrl+Alt+			
00001001D0	44 65 6C 20 74 6F 20 72	65 73 74 61 72 74 0D 0A	Del to restart..			
00001001E0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00001001F0	00 00 00 00 00 00 8A 01	A7 01 BF 01 00 00 55 AAŠ.\$.ž....U²			








jump to memory address 0x7F19 where is the code area of Boot Loader Area

error message locator
check offset 0x18A, 0x1A7, 0x1BF

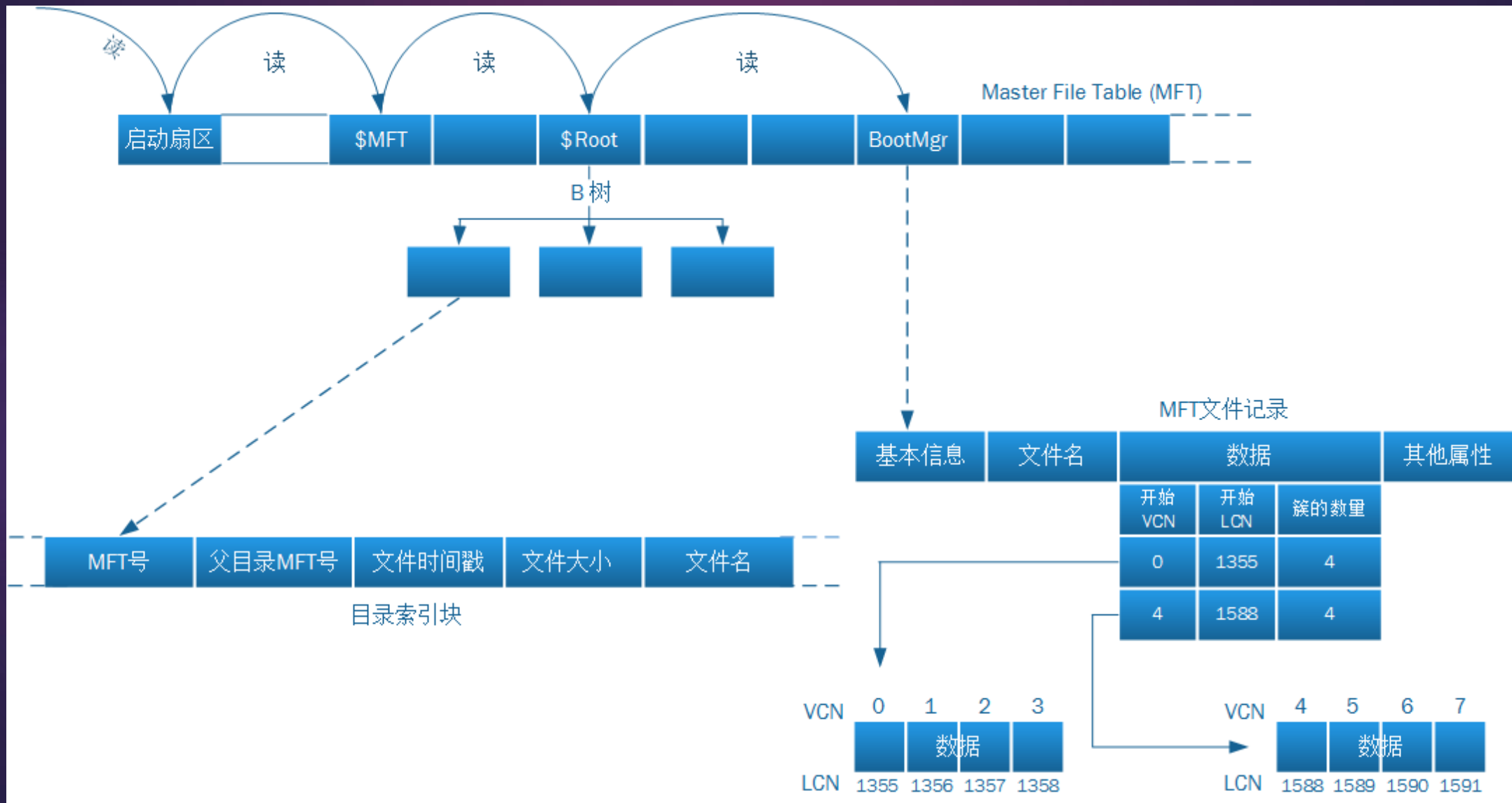
Clusters per FRS == "F6" == "-10" == "2^10" == 1024 bytes
Clusters per Index Block == "1" == "512*8*1" == 4096 bytes

引导区域（Boot Area）

- ▶ PBR之后的15个扇区
 - ▶ 目前只有大约4KB数据，剩余部分用零填充
- ▶ 该部分代码会调用PBR中的代码用于读磁盘，显示错误，执行TPM操作等
- ▶ 这部分代码能够理解基本的NTFS结构，可以从“系统保留分区”的根目录下读取文件
- ▶ (Windows 8+)尝试读取文件BOOTGT，如果成功，则继续读取文件BOOTNXT
 - ▶ 用于支持自动从USB启动（Windows To Go）
 - ▶ 遍历其他磁盘，寻找带有活动分区的磁盘，并将其MBR载入到内存地址0x7C00处，然后跳转到那里继续执行
- ▶ 尝试读取BootMgr
 - ▶ 将其载入到内存地址0x20000处
 - ▶ 如果没有找到BootMgr，则继续尝试读取NTLDR（用于和Windows Vista之前的系统保持兼容）
- ▶ 如果检查到硬件支持TPM 1.2+
 - ▶ 通过BIOS中断运行TCG_CompactHashLogExtendEvent例程
 - ▶ 计算BootMgr的哈希值并将结果存储于TPM的PCR寄存器中
- ▶ 跳转到BootMgr继续执行

	\$RECYCLE.BIN	6/9/2016 5:33 AM	File folder	
	Boot	6/9/2016 9:07 PM	File folder	
	Recovery	6/9/2016 8:15 PM	File folder	
	System Volume Information	6/9/2016 8:13 PM	File folder	
	bootmgr	7/10/2015 4:00 AM	System file	387 KB
	BOOTNXT	7/10/2015 4:00 AM	System file	1 KB
	BOOTSECT.BAK	6/9/2016 9:07 PM	BAK File	8 KB

引导区域 - 读取NTFS文件系统可以很简单



BootMgr

- ▶ 由两部分代码组成：16位的“startup.com”和32位的“bootmgr.exe”
 - ▶ 32位代码会被重新定位到内存地址0x400000处
 - ▶ 32位代码入口函数为“BmMain”
- ▶ 16位代码会将CPU切换到32位保护模式（即使64位的系统也是如此）
 - ▶ 仍然使用BIOS中断INT13进行磁盘读写
- ▶ 从系统保留分区载入BCD文件：\Boot\BCD
- ▶ 如果是系统睡眠的情况，则根据BCD文件的配置载入并运行winresume.exe
- ▶ 如果BCD文件中配置了多个启动选项，则会显示一个引导菜单
- ▶ 从操作系统分区载入WinLoad.exe（\Windows\System32\WinLoad.exe）
 - ▶ 如果操作系统分区被BitLocker保护，则可能会向用户询问解锁密码
- ▶ 更新引导状态文件\Boot\bootstat.dat (这是一个预分配空间的文件)
- ▶ 跳转到WinLoad.exe继续执行
 - ▶ 对于64位系统，在跳转至WinLoad.exe前会将CPU切换至64位长模式

BootMgr - BCD

- ▶ BCD文件位于系统保留分区上\Boot\BCD
 - ▶ 它是一个注册表数据库文件
- ▶ 它用于替换早期版本中的boot.ini文件（但BootMgr仍然可以读取boot.ini文件）
- ▶ BCD文件可以被认为是一系列BCD对象的容器
 - ▶ 一个BCD对象可以代表一个引导程序
 - ▶ 每个对象以GUID或者别名（例如：{bootmgr}，{default}）来识别
 - ▶ 每个BCD对象又可以被认为是若干BCD元素的集合
 - ▶ BCD元素即为一个引导程序的各种配置选项
- ▶ 命令行工具“bcdedit”用于编辑BCD文件（也可通过WMI编辑）
 - ▶ 可以设置启动选项和Windows内核参数
- ▶ 可以在注册表项HKEY_LOCAL_MACHINE\BCD00000000下查看BCD内部结构
 - ▶ 缺省只有SYSTEM账户可以修改BCD注册表项
 - ▶ BootMgr还会在载入BCD文件后进行完整性检查

BootMgr – 看看BCD是如何存储操作系统位置的

Windows Boot Manager

```
-----
identifier          {bootmgr}
device               partition=\Device\HarddiskVolume1
description          Windows Boot Manager
locale               en-US
inherit              {globalsettings}
default              {current}
resumeobject         {1cd97c1a-9581-11e3-8980-f0c52ae4d27b}
displayorder         {current}
toolsdisplayorder    {memdiag}
timeout              30
```

Windows Boot Loader

```
-----
identifier          {current}
device               partition=C:
path                 \Windows\system32\winload.exe
description          Windows 7
locale               en-US
inherit              {bootloadersettings}
recoverysequence     {1cd97c1c-9581-11e3-8980-f0c52ae4d27b}
recoveryenabled       Yes
osdevice              partition=C:
systemroot            \Windows
resumeobject         {1cd97c1a-9581-11e3-8980-f0c52ae4d27b}
nx                    OptIn
bootlog               Yes
sos                   Yes
```

Registry Editor

Computer

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_USER
- HKEY_LOCAL_MACHINE
 - BCD00000000
 - Objects
 - {0ce4991b-e6b3-4b16-b221-1a9c49-16ab-4a5c-9011-4636856e-540f-4170-a130}
 - {5189b25c-5558-4bf2-bca5-53dbf8ae-2505-11e5-adc0}
 - {53dbf8af-2505-11e5-adc0}
 - Elements
 - 11000001
 - 12000002
 - 12000004
 - 12000005
 - 14000006
 - 14000008
 - 1500004b
 - 16000009
 - 17000077
 - 21000001
 - 22000002
 - 23000003
 - 25000020
 - 250000c2
 - 250000f0
 - 26000090
 - 26000091
 - {53dbf8b0-2505-11e5-adc0}

Value name: Element

Value data:

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000001A0	67	20	6F	70	65	72	61	74	69	6E	67	20	73	78	73	74
00000001B0	65	6D	00	00	00	63	7B	9A	27	4E	D0	D9	00	00	80	20
00000001C0	21	00	07	BE	12	2C	00	08	00	00	00	F0	0A	00	00	BE
00000001D0	13	2C	07	FE	FF	FF	00	F8	0A	00	00	10	7D	13	00	FE
00000001E0	FF	FF	07	FE	FF	FF	00	08	88	13	00	48	B0	26	00	00
00000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	55	AA	AA
0000000200	45	46	49	20	50	41	52	54	00	00	01	00	5C	00	00	00

Hard Disk 1

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000001A0 67 20 6F 70 65 72 61 74 69 6E 67 20 73 78 73 74 g operating syst

00000001B0 65 6D 00 00 00 63 7B 9A 27 4E D0 D9 00 00 80 20 em...c(s\NBDU)...€

00000001C0 21 00 07 BE 12 2C 00 08 00 00 00 00 F0 0A 00 00 BE !..%.,.....8...%

00000001D0 13 2C 07 FE FF FF 00 F8 0A 00 00 10 7D 13 00 FE .,.pyy.ø....}..p

00000001E0 FF FF 07 FE FF FF 00 08 88 13 00 48 B0 26 00 00 yy.pyy...^..H^s..

00000001F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AAU*

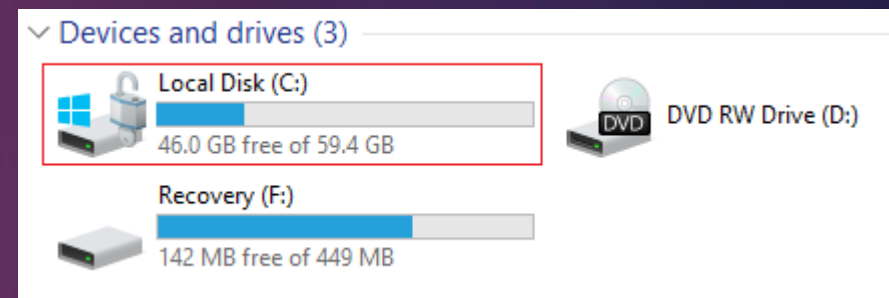
0000000200 45 46 49 20 50 41 52 54 00 00 01 00 5C 00 00 00 EFI PART....\...

disk signature

0xAF800 * 0x200 = 0x15F00000

BootMgr – BitLocker相关内容

- ▶ 加密整个磁盘卷（引导扇区和BitLocker元数据区域除外）
 - ▶ 加密解密工作是由一个过滤驱动完成的。该驱动工作在卷管理器驱动之上，文件系统驱动之下
 - ▶ （Windows 8+）可以只加密正在被使用的数据区域
- ▶ 整卷加密密钥（Full Volume Encryption Key, FVEK）
 - ▶ 可以选择AES-128 或 AES-256两种算法
 - ▶ 被用于加密磁盘卷上的数据
 - ▶ 它本身被VMK加密，并存储于它所保护的磁盘卷上
- ▶ 卷主密钥（Volume Master Key, VMK）
 - ▶ AES-256
 - ▶ 被用于加密FVEK
 - ▶ 它本身可能被以下密钥加密：恢复密钥（Recovery Key），用户密码，外部密钥（存储在USB设备上），或TPM
 - ▶ （Windows 10）如果以Microsoft账户登录机器，恢复密钥会被自动上载到OneDrive上，引起很多争议
 - ▶ 它也被存储在它所保护的磁盘卷上
- ▶ BootMgr需要VMK/FVEK来解密磁盘数据并载入操作系统载入程序(winload.exe)
 - ▶ 它会要求用户输入密码或者恢复密钥
- ▶ BootMgr会把解密后的VMK传给WinLoad.exe (而WinLoad.exe会最终把VMK传给ntoskrnl.exe)



WinLoad.exe

- ▶ 缺省位于操作系统分区\Windows\System32\WinLoad.exe
 - ▶ 从源代码角度来说，它的很多代码是和BootMgr共享的
 - ▶ 它也需要利用BIOS中断 INT13H进行磁盘读写
- ▶ 它与BootMgr共享内存中的BCD数据
- ▶ 它会读取文件\Windows\bootstat.dat以决定是否显示它特有的引导菜单
- ▶ 如果需要则显示引导菜单 (例如：安全引导选项)
- ▶ 从操作系统卷按顺序载入以下类型文件（不同Windows版本会略有差别。完整列表参见“高级话题”。）
 - ▶ 系统注册表文件\Windows\system32\config\SYSTEM
 - ▶ 用于代码完整性检查的相关文件
 - ▶ 关键的可执行程序：ntoskrnl.exe, HAL.dll, CI.dll, PSHEd.dll, BOOTVID.dll
 - ▶ 根据注册表设置，载入“引导类型”的驱动
 - ▶ 载入操作系统卷的文件系统驱动
 - ▶ 根据注册表设置，载入NLS文件
- ▶ 为内核做一些初始化操作
- ▶ 更新引导状态文件\Windows\bootstat.dat
- ▶ 跳转到内核代码开始执行
 - ▶ WinLoad.exe准备的引导数据会通过结构体LOADER_PARAMETER_BLOCK传给内核






Ntoskrnl.exe – 启动参数

```
kd> dt poi(nt!KeLoaderBlock) nt!_LOADER_PARAMETER_BLOCK
+0x000 OsMajorVersion      : 6
+0x004 OsMinorVersion      : 1
+0x008 Size                : 0xf0
+0x00c Reserved            : 0
+0x010 LoadOrderListHead  : _LIST_ENTRY [ 0xfffff800`00834380 - 0xfffff800`0086c4c0 ]
+0x020 MemoryDescriptorListHead : _LIST_ENTRY [ 0xfffff800`00a05000 - 0xfffff800`00a06450 ]
+0x030 BootDriverListHead : _LIST_ENTRY [ 0xfffff800`008644f0 - 0xfffff800`0081eff0 ]
+0x040 KernelStack         : 0xfffff800`01b5a000
+0x048 Prcb                 : 0xfffff800`02055e80
+0x050 Process              : 0xfffff800`02064180
+0x058 Thread               : 0xfffff800`02063cc0
+0x060 RegistryLength       : 0x1078000
+0x068 RegistryBase         : 0xfffff800`00ac4000 Void
+0x070 ConfigurationRoot    : 0xfffff800`008157f0 _CONFIGURATION_COMPONENT_DATA
+0x078 ArcBootDeviceName    : 0xfffff800`00824710 "multi(0)disk(0)rdisk(0)partition(2)"
+0x080 ArcHalDeviceName     : 0xfffff800`008244a0 "multi(0)disk(0)rdisk(0)partition(1)"
+0x088 NtBootPathName       : 0xfffff800`00812d40 "\\Windows\\"
+0x090 NtHalPathName        : 0xfffff800`008119d0 "\\"
+0x098 LoadOptions          : 0xfffff800`00819470 " BOOTDEBUG  DEBUG  DEBUGPORT=COM1  BAUDRATE=115200"
+0x0a0 NlsData               : 0xfffff800`008647e0 _NLS_DATA_BLOCK
+0x0a8 ArcDiskInformation    : 0xfffff800`008253e0 _ARC_DISK_INFORMATION
+0x0b0 OemFontFile           : 0xfffff800`01c731d0 Void
+0x0b8 Extension            : 0xfffff800`00824750 _LOADER_PARAMETER_EXTENSION
+0x0c0 u                     : <unnamed-tag>
+0x0d0 FirmwareInformation  : _FIRMWARE_INFORMATION_LOADER_BLOCK
```

- ▶ 此数据结构会在内核第一阶段初始化完成时被释放
- ▶ 与此数据结构对应的符号在Windows 8之后就找不到了，但数据结构本身仍然存在（只是增加了成员变量）

Ntoskrnl.exe – 第0阶段初始化

- ▶ 入口函数为“KiSystemStartup”
- ▶ 进行很多内部数据结构的初始化工作
- ▶ 一些硬件的初始化工作
 - ▶ 利用Hal.dll
 - ▶ CPU, Monitor, BIOS, PnP
- ▶ NLS(此前被winload.exe载入的) 初始化
 - ▶ 可以将Unicode转换为ANSI/OEM
 - ▶ 为BSOD准备错误信息
- ▶ 第一个可以连接内核调试器的位置
- ▶ 创建 “System Idle” 进程和 “System” 进程
 - ▶ 在进程“System” 内为第1阶段初始化工作创建线程










Name	PID [^]	User name	CPU	Memory (p...	Base priority	Threads	Description
 System interrupts	-	SYSTEM	00	0 K	N/A	-	Deferred procedure calls and interrupt service routines
 System Idle Process	0	SYSTEM	98	4 K	N/A	2	Percentage of time the processor is idle
 System	4	SYSTEM	00	45,388 K	N/A	133	NT Kernel & System
 smss.exe	256	SYSTEM	00	140 K	Normal	2	Windows Session Manager
 csrcs.exe	352	SYSTEM	00	460 K	Normal	9	Client Server Runtime Process

Ntoskrnl.exe – 第1阶段初始化

- ▶ Phase1 Initialization Discard
 - ▶ 将当前线程设置为最高优先级，避免其他线程抢占
 - ▶ 开始很多内核模块的初始化工作
 - ▶ 启动其他CPU处理器（此前只有一个处理器在运行）
- ▶ I/O 系统初始化
 - ▶ 初始化“引导型”的驱动
 - ▶ 等待操作系统卷可用
 - ▶ 如果失败，则将以错误码0x0000007B蓝屏
 - ▶ 载入动态库ntdll.dll
 - ▶ 载入并初始化“系统型”驱动
 - ▶ PnP管理器根据它探知的设备载入对应的驱动
- ▶ 更新文件\Windows\bootstat.dat
- ▶ 创建进程smss.exe
 - ▶ 这是第一个用户态进程
 - ▶ 内核会等待该进程5秒钟，如果该进程提前退出，则认为整个系统启动失败
- ▶ 在本阶段初始化完成时，屏幕会变成黑色(Windows 10)

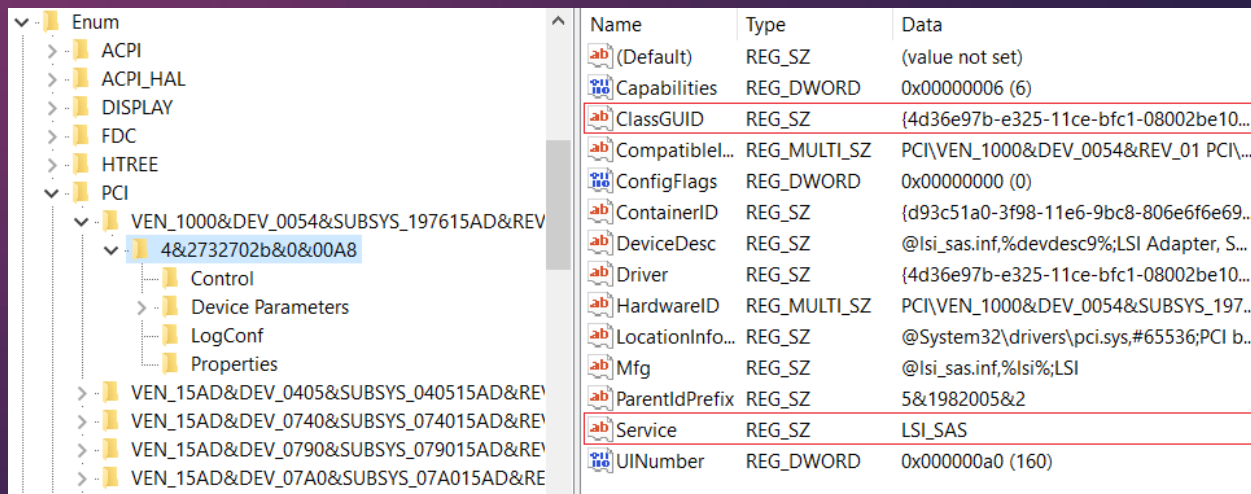
驱动的初始化顺序

- ▶ 驱动初始化类型（非动态载入型）
 - ▶ 引导型, SERVICE_BOOT_START, 被WinLoad.exe载入
 - ▶ 系统型, SERVICE_SYSTEM_START, 被内核载入
 - ▶ 针对那些无法被PnP管理器探知的设备或内核功能扩展
- ▶ 在SERVICE_BOOT_START和SERVICE_SYSTEM_START类型范围内：
 - ▶ ServiceGroupOrder
 - ▶ \HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control
 - ▶ 决定一组驱动的初始化顺序
 - ▶ 每个驱动注册项下都有一个叫“Group”的值，它的内容被定义在“ServiceGroupOrder”中
 - ▶ GroupOrderList
 - ▶ 每个驱动注册项都可以有一个名为“Tag”的值
 - ▶ 它的内容决定驱动在组内的初始化顺序
 - ▶ HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\GroupOrderList
 - ▶ Format: [Number of tags][Tag][Tag ...]
- ▶ 任何未设置成“无效”的驱动都会被PnP管理器按需载入

Name	Type	Data
 (Default)	REG_SZ	(value not set)
 DisplayName	REG_SZ	Volume Manager Driver
 DriverPackageId	REG_SZ	machine.inf_amd64_neutral_9e6bb86c3b39a3e9
 ErrorControl	REG_DWORD	0x00000003 (3)
 Group	REG_SZ	System Bus Extender
 ImagePath	REG_EXPAND_SZ	system32\drivers\volmgr.sys
 Start	REG_DWORD	0x00000000 (0)
 Tag	REG_DWORD	0x00000009 (9)
 Type	REG_DWORD	0x00000001 (1)

PnP管理器与设备驱动载入

- ▶ 每个硬件设备都被分配有设备ID
 - ▶ 对于PCI总线上的设备，这个ID的格式可能会是这样的：VEN_v(4)&DEV_d(4)&SUBSYS_s(4)n(4)&REV_r(2)
 - ▶ 硬件ID：<总线类型>\<设备ID>
- ▶ 为了区分多个同类型的设备，Windows会给每个设备分配一个实例ID
 - ▶ 这个ID的命名方法不固定，可能包含设备在总线上的位置，设备自身序列号，或者MAC地址（如果是网卡）
- ▶ 内核检测到新设备后，会到注册项HKLM\SYSTEM\CurrentControlSet\Enum下查找对应的驱动程序
 - ▶ <总线类型（如：PCI）>\<设备ID>\<实例ID>
- ▶ 对应的注册项下需要包含信息
 - ▶ ClassGUID：用于在注册项HKLM\SYSTEM\CurrentControlSet\Control\Class下进一步寻找对应该类型设备的过滤驱动
 - ▶ Service：用于在注册项HKLM\SYSTEM\CurrentControlSet\services下寻找当前设备的驱动在磁盘上的位置
 - ▶ LowerFilters 和 UpperFilters
- ▶ 驱动载入顺序
 - ▶ 注册于当前设备的底层过滤驱动
 - ▶ 注册于当前设备类型（device class）的底层过滤驱动
 - ▶ 当前设备的驱动
 - ▶ 注册于当前设备的上层过滤驱动
 - ▶ 注册于当前设备类型（device class）的上层过滤驱动

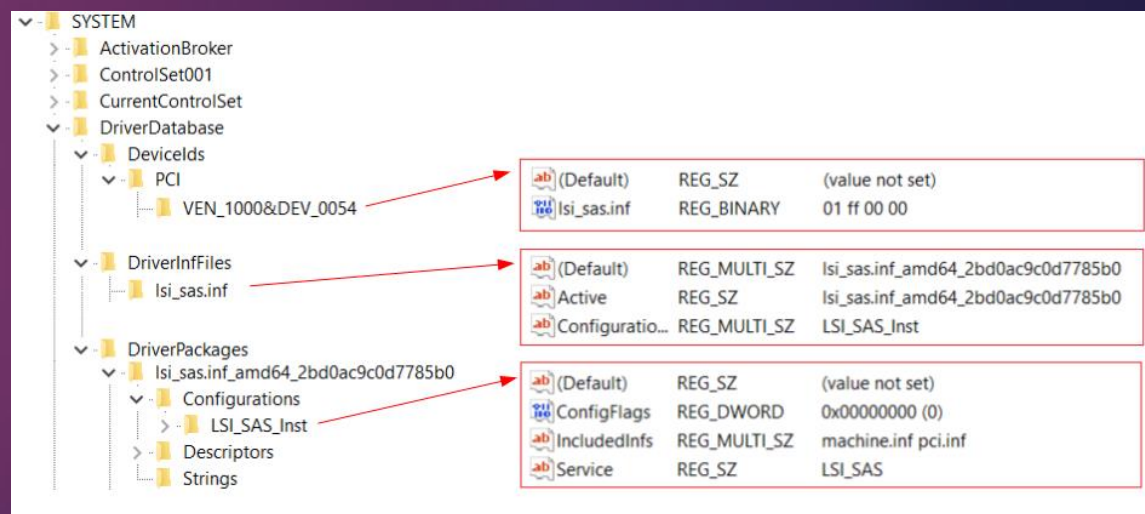


The screenshot shows the Windows Device Manager interface. On the left, the 'Enum' tree is expanded to 'PCI' > 'VEN_1000&DEV_0054&SUBSYS_197615AD&REV_4&2732702b&0&00A8'. On the right, the 'Properties' window for this device is displayed, showing various registry values.

Name	Type	Data
(Default)	REG_SZ	(value not set)
Capabilities	REG_DWORD	0x00000006 (6)
ClassGUID	REG_SZ	{4d36e97b-e325-11ce-bfc1-08002be10...
Compatible...	REG_MULTI_SZ	PCI\VEN_1000&DEV_0054&REV_01 PCI\...
ConfigFlags	REG_DWORD	0x00000000 (0)
ContainerID	REG_SZ	{d93c51a0-3f98-11e6-9bc8-806e6f6e69...
DeviceDesc	REG_SZ	@lsi_sas.inf,%devdesc9%;LSI Adapter, S...
Driver	REG_SZ	{4d36e97b-e325-11ce-bfc1-08002be10...
HardwareID	REG_MULTI_SZ	PCI\VEN_1000&DEV_0054&SUBSYS_197...
LocationInfo...	REG_SZ	@System32\drivers\pci.sys,#65536;PCI b...
Mfg	REG_SZ	@lsi_sas.inf,%lsi%;LSI
ParentIdPrefix	REG_SZ	5&1982005&2
Service	REG_SZ	LSI_SAS
UINumber	REG_DWORD	0x000000a0 (160)

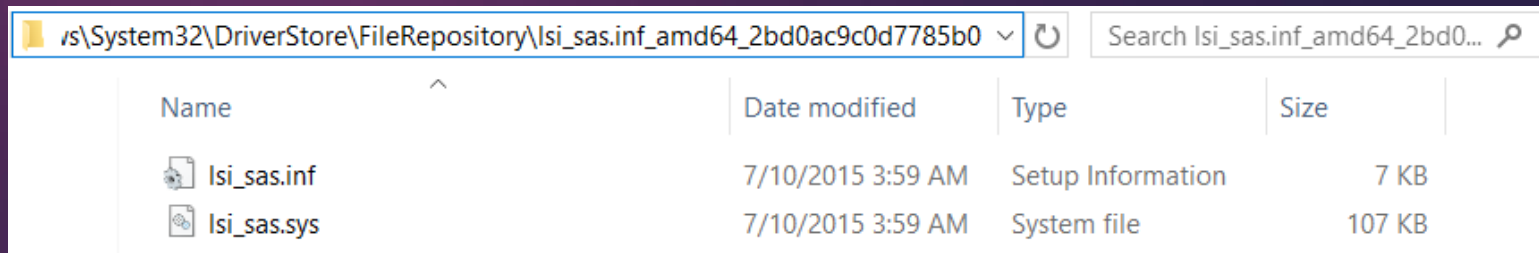
PnP管理器：引导型驱动匹配设备

- ▶ 根据<总线类型>\<设备ID>\<实例ID>在HKLM\SYSTEM\CurrentControlSet\Enum查找驱动名称
- ▶ 如果无法找到：
 - ▶ 为该设备实例在Enum下创建注册项并保存硬件相关的信息（但无驱动信息）
 - ▶ 根据“总线类型+设备ID”在DriverDatabase下查找(Windows 8之前在CriticalDeviceDatabase查找)
 - ▶ 首先在Devicelds注册项下找到INF文件名字
 - ▶ 从完全匹配查找到部分匹配查找
 - ▶ 然后在DriverInfFiles注册项下找到驱动包名字
 - ▶ 最后在DriverPackages注册项下找到驱动名称
 - ▶ 将找到的驱动信息更新到Enum下此前创建的注册项下
- ▶ 完成驱动和设备的匹配
- ▶ 设备工作后可能会枚举出新的子设备
 - ▶ 如果新设备的驱动尚未载入，则会推迟到系统初始化完成后再载入相应驱动



PnP管理器：普通设备驱动

- ▶ 如果内核没有在注册表中找到当前设备对应的驱动信息
 - ▶ 在HKLM\SYSTEM\CurrentControlSet\Enum下为当前设备创建一个注册项
 - ▶ 用户态的PnP管理器会主动获取等待驱动的设备列表，并尝试为设备安装驱动(服务“Device Install”调用DrvInst.exe)
 - ▶ 复制驱动程序到\Windows\System32\drivers
 - ▶ 复制INF文件到\Windows\INF
 - ▶ 更新驱动相关注册表项（例如：在Service下创建驱动自己的注册项）
 - ▶ 更新Enum下设备对应的注册项，然后通知内核载入驱动
- ▶ （Windows 8+）PnP管理器只会从DriverStore里安装驱动程序
 - ▶ 外部的驱动程序必须要先“stage”到DriverStore，然后才能被安装
- ▶ DriverStore
 - ▶ 目录\Windows\System32\DriverStore
 - ▶ 驱动包实际存于目录“FileRepository”下
 - ▶ 驱动文件包含到“\Windows\winsxs”的硬连接, PNF文件(预编译INF文件)除外
 - ▶ 命令“pnputil”可以被用于管理非Microsoft发布的驱动

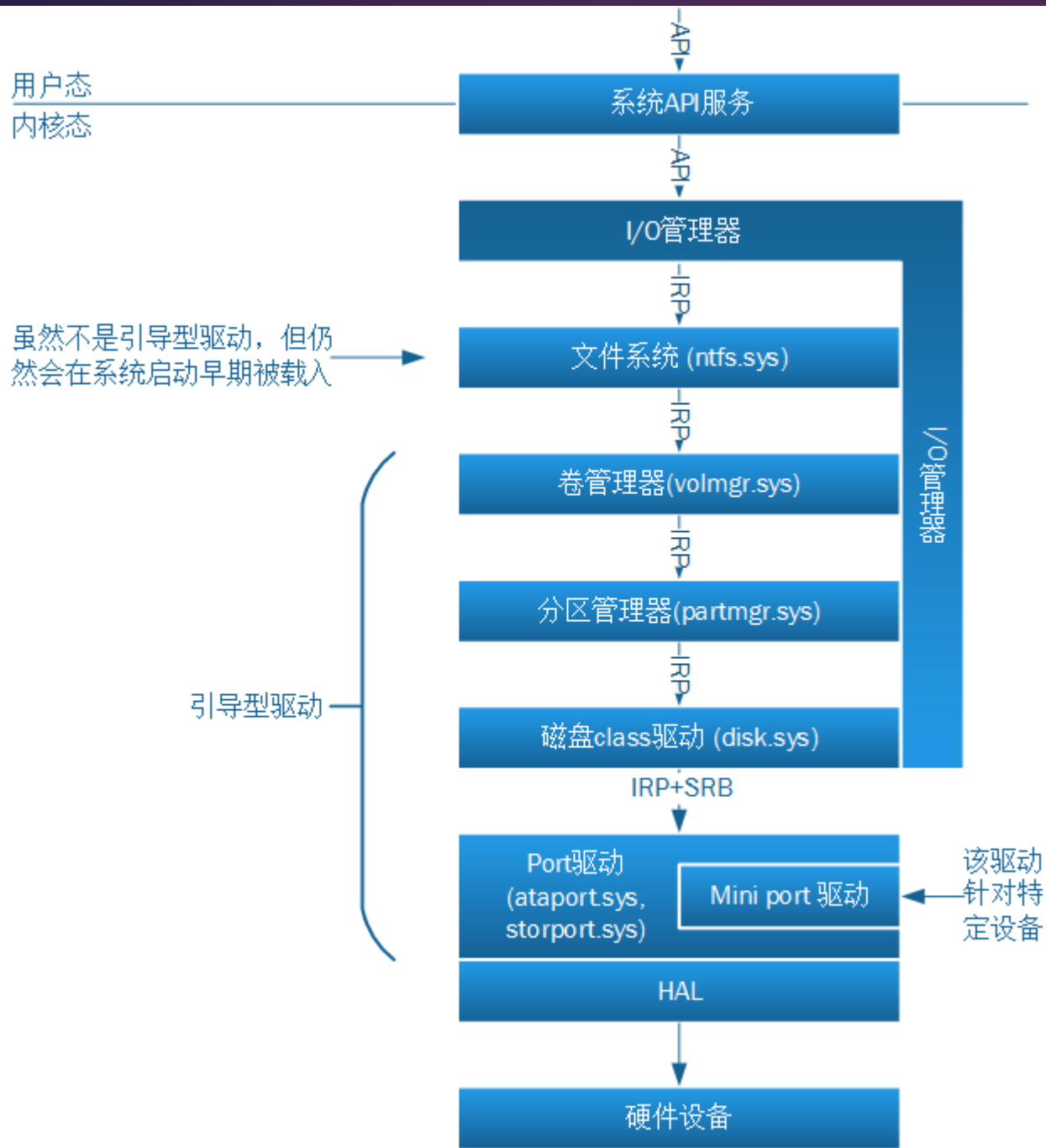


文件系统驱动的载入

- ▶ 操作系统卷的文件系统驱动由WinLoad.exe载入
- ▶ WinLoad.exe也会载入驱动程序fs_rec.sys
- ▶ Fs_rec.sys理解Windows支持的全部文件系统
 - ▶ 它将自己注册为文件系统驱动
 - ▶ 当有新的卷被装配时(mount)，I/O管理器会询问每个已经登记的文件系统驱动
 - ▶ 它检测新卷上的文件系统类型，载入相应驱动
 - ▶ 检查卷引导扇区上的文件系统标识
 - ▶ 新载入的文件系统驱动将声明自己将服务该卷设备
- ▶ 如果最终无法找到可用文件系统驱动，内核内置的RAW驱动将服务该卷
- ▶ 操作系统所在卷在全部引导型驱动初始化完成后开始工作
- ▶ 其他卷会在autochk（第二个用户态进程）做文件系统完整性检查时被装配

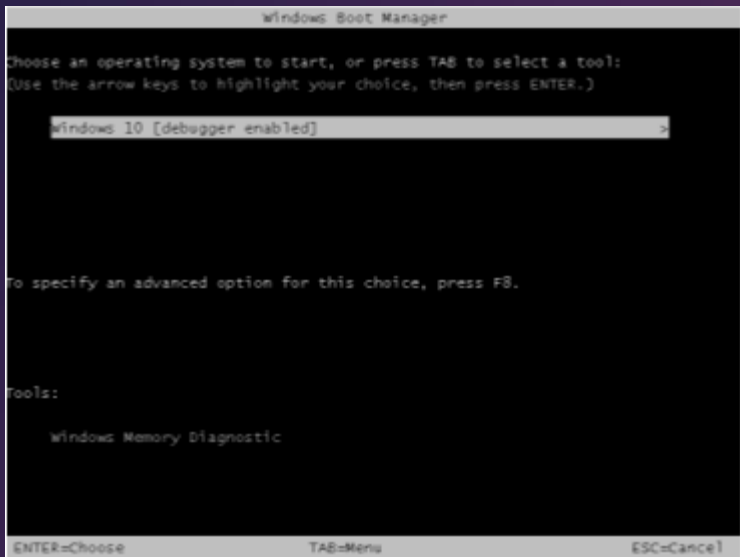
存储驱动栈

- ▶ 驱动使用I/O管理器提供的函数来提交IRP
- ▶ 过滤型驱动可以位于任何两个功能型驱动之间。例如：
 - ▶ BitLocker驱动fvevol.sys位于卷管理器和文件系统之间
 - ▶ Hyper-V child partition 虚拟磁盘过滤驱动vmstorfl.sys磁盘驱动和存储port驱动之间
- ▶ 硬件供应商可以为其硬件开发Mini port 驱动，但必须使用port驱动提供的功能来访问HAL（否则无法通过微软认证）

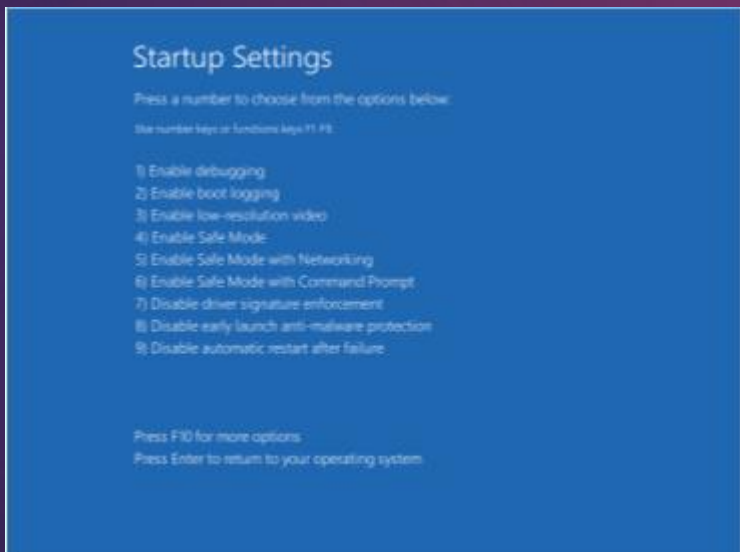


Smss.exe

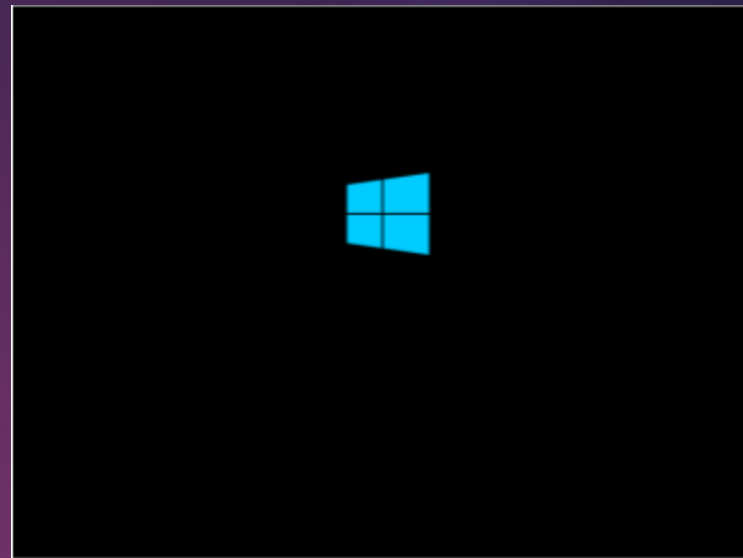
- ▶ Native application, 且为第一个用户态进程
- ▶ 先将自己设定为系统关键进程（这样的进程异常退出会导致整个系统停止运行）
- ▶ 根据注册表设定系统缺省环境变量
- ▶ 运行以下注册项下登记的程序
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\BootExecute
 - ▶ 一般至少包含 Autochk.exe，同为Native application
- ▶ 根据设置，写入引导日志到文件\Windows\ntbtlog.txt
- ▶ 处理上次系统运行时留下来的文件更名和删除操作
- ▶ 打开page files并一直持有他们的句柄（确保没有其他程序可以删除或修改这些文件）
- ▶ 从文件系统中载入其他的注册表，例如 SAM, SECURITY, 和SOFTWARE
- ▶ 根据注册表设置，载入已知的系统动态库（KnownDLL）
- ▶ 创建会话#0 (用于后台服务) 和会话#1 (用于用户交互)
 - ▶ 每个会话都会进一步运行csrss.exe



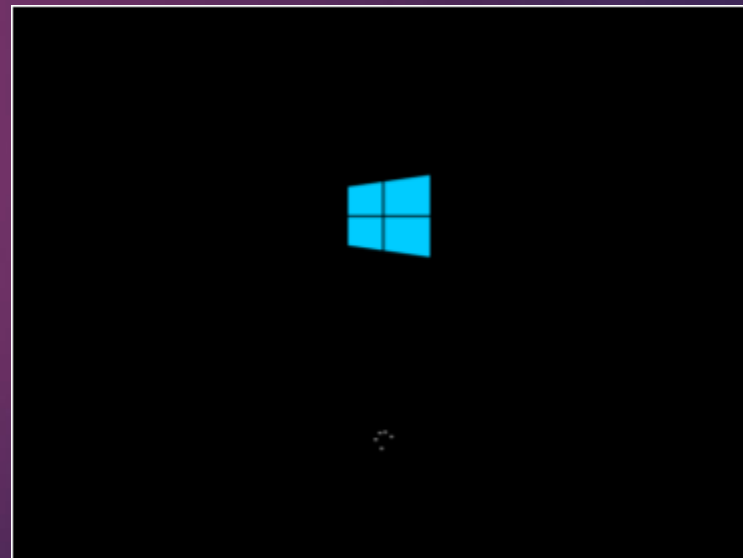
1. BootMgr菜单，只有在有多个启动选项时显示（或强制显示）



2. WinLoad.exe 菜单. 正常情况下一般不显示



3. 在WinLoad.exe 菜单被选择之后，在引导型驱动开始初始化之前

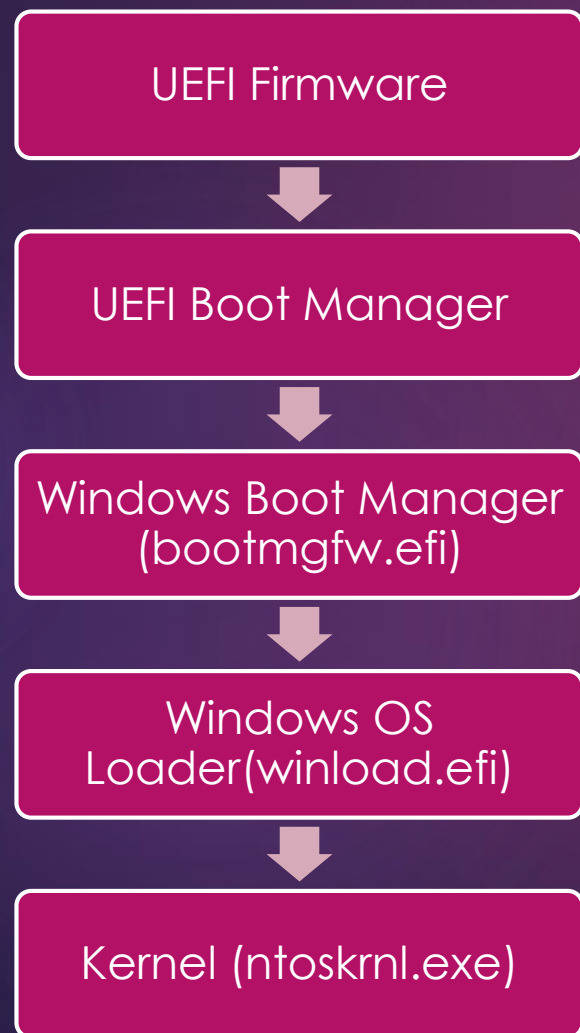


4. 从引导型驱动开始初始化，到内核第一阶段初始化完成。

UEFI 引导过程

关于UEFI早期引导过程和Secure Boot，可以参考《Grub2 引导过程》的部分内容。

典型引导过程



进行CPU和芯片初始化，载入必要的驱动等。

载入并运行安装时登记好的引导管理程序（bootmgfw.efi）

<ESP>\EFI\Microsoft\Boot\bootmgfw.efi
载入用户选择的操作系统载入程序（Winload.efi）

\Windows\system32\Winload.efi
载入Windows操作系统，调用ExitBootServices()结束UEFI服务。

引导管理器 (bootmgfw.efi)

- ▶ 全部代码会运行在CPU固有模式（64位长模式）下，并且提供内存分页支持
 - ▶ Bootmgfw.efi会被载入到内存地址0x10000000位置处
- ▶ 使用UEFI固件提供的服务进行磁盘读写(block I/O 协议)
- ▶ 从\EFI\Microsoft\Boot\BCD读取BCD文件

Disk 0 Basic 59.98 GB Online			
	450 MB Healthy (Recovery Partition)	99 MB Healthy (EFI System Partition)	(C:) 59.45 GB NTFS Healthy (Boot, Page File, Crash Dump, Primary Partition)

```
Windows Boot Loader
-----
identifier          {current}
device              partition=C:
path                \Windows\system32\winload.efi
description          Windows 8.1
locale              en-US
inherit              {bootloadersettings}
recoverysequence     {a0a31508-2ac7-11e6-97c1-8939003f9c5f}
integrityservices    Enable
recoveryenabled      Yes
isolatedcontext      Yes
allowedinmemorysettings 0x15000075
osdevice             partition=C:
systemroot           \Windows
resumeobject         {a0a31506-2ac7-11e6-97c1-8939003f9c5f}
nx                  OptIn
bootmenupolicy       Standard
```

Value name:	
Element	
Value data:	
0000	00 00 00 00 00 00 00 00
0008	00 00 00 00 00 00 00 00
0010	06 00 00 00 00 00 00 00
0018	48 00 00 00 00 00 00 00 H.....
0020	3B 34 1A 83 B1 E5 EE 48 :4...±âiH
0028	AB 62 7C C8 8B 87 32 6C <b Ë...21
0030	00 00 00 00 00 00 00 00
0038	D3 DE 8C 65 CF 1A C6 4B Óp.eI.ÆK
0040	8A BA A6 B2 4E A7 29 30 °l?NS)0
0048	00 00 00 00 00 00 00 00
0050	00 00 00 00 00 00 00 00
0058	00 00 00 00 00 00 00 00

Partition GUID

EFI disk unique ID

OK Cancel

高级话题

Windows Native Application

- ▶ 只能运行于Windows子系统之外
- ▶ 只能使用NTDLL.dll提供的函数（该动态库也提供了一些标准C函数）
 - ▶ 性能可能因此优于普通Windows程序
- ▶ 入口函数为“NtProcessStartup”
- ▶ 必须调用函数“NtProcessTerminate”结束自己（而不会像普通Windows程序那样退出main函数，则整个程序自动退出）
- ▶ 典型的 native applications
 - ▶ Smss.exe
 - ▶ Autochk.exe
 - ▶ Csrss.exe
 - ▶ 某些高级的磁盘管理程序
 - ▶ 某些病毒扫描程序

调式Windows引导过程

- ▶ 准备环境
 - ▶ 一个装有虚拟机软件的Windows机器
 - ▶ Windows 7.x: Virtual PC, 只支持32位客户机
 - ▶ Windows 8.x 或之后: 自带Hyper-V
 - ▶ 其他如: Vmware player
 - ▶ 从Microsoft网站安装Windbg
- ▶ 在虚拟机配置里, 将串口重定向到本地命名管道
- ▶ 在虚拟机内修改BCD以便通过串口进行调试
 - ▶ `bcdedit.exe /store <BCD_FILE> /dbgsettings serial debugport:1 baudrate:115200`
 - ▶ `bcdedit.exe /store <BCD_FILE> /bootdebug on`
- ▶ 在VM启动后通过命令行运行WinDbg:
 - ▶ `Windbg -b -k com:pipe,port=\\.\pipe\com1`
 - ▶ 对于Virtual PC/Hyper-V可尝试在以上命令行后附加 `“,resets=0”`

调式Windows引导过程（2）

- ▶ BootMgr到底访问了哪些文件？
 - ▶ `bp B!FileOpen "du /c80 edx; gc"`
 - ▶ 对于Windows 10:
 - ▶ `Bp B!FileOpen "du /c80 poi(esp+8); gc "`
 - ▶ 你看到的结果并不会包含BCD文件
- ▶ WinLoad.exe到底访问了哪些文件？
 - ▶ `bp B!FileOpen "du /c80 rdx; gc "`
- ▶ Winload.exe到底查询了哪些注册键？
 - ▶ `bp CmpFindValueByName ".echo {;da [rdx+0x4c];du poi(r8+8);.echo };gc"`
- ▶ ntoskrnl.exe到底访问了哪些文件？
 - ▶ `bp ZwOpenFile "dt _OBJECT_ATTRIBUTES ObjectName [r8]; gc"`
- ▶ 驱动是按什么样的顺序进行初始化的？
 - ▶ `bp nt!IoPlInitializeBuiltinDriver "dt _LDR_DATA_TABLE_ENTRY FullDllName [r9]; gc"`

一些工具

- ▶ Windbg
 - ▶ 可以用于调试Windows启动过程，并且有函数和部分结构体的符号支持（但不会有参数符号）
- ▶ BOCHS
 - ▶ 可用于调试启动过程早期的汇编模块，例如MBR、PBR
 - ▶ 自己集成了调试器，很方便 (bochsdbg.exe)
- ▶ IDA Pro 5.0 free edition
 - ▶ 用于反汇编以及分析汇编代码
- ▶ Active @ disk editor
 - ▶ 用于了解磁盘和文件系统的布局
- ▶ ReactOS
 - ▶ 一个非常像Windows的开源项目，像到很多内部函数名称都是一样的
 - ▶ 可以用于了解Windows启动的主体流程。但要注意很多代码相对于当前的Windows版本来说早已过时
 - ▶ 至少可以用来猜测一些内部函数的参数组成
- ▶ Xbootmgr in Windows Performance Toolkit (基于ETW技术)
 - ▶ Xbootmgr -trace boot -stackwalker @filename
 - ▶ 使用 xperfview 启动过程发生的一些事件 (还可以有函数符号支持)

谢谢！