win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码

```
.386p

CSEG SEGMENT USE16

ASSUME CS:CSEG,DS:CSEG,ES:CSEG

START PROC

CodeStart:

;*************************************************************************

;主引导程序第一个扇区:

;完成将代码拷贝到 2000:0 处,接着转到该处继续执行.....

BOOTCodeSector1 proc

org 0 ;第一个扇区开始于 0

cli

xor ax,ax

mov ss,ax ;设置新堆栈 0:2000h

mov sp,2000h

sti

push cs

pop ds ;ds=cs

mov ax,2000h

mov es,ax ;es=2000h
```

```
mov si,offset CodeStart ;ds:si->CodeStart

xor di,di ;es:di->新缓冲区

; 注意:拷贝代码长度 2000h=16*512,这是因为 Ntldr 加载多扇区启动时.

; 要求的最小度为 16 个扇区长度,小于 16 个扇区时只加载 1 个扇区..

mov cx,2000h ;不用计算代码长度..

cld ;我们代码没这么长...

repz movsb ;把代码拷贝到 2000:0 处

push es

push offset BOOTCode10 - offset CodeStart

retf ;转到 BOOTStart 处执行

;*****************************************************************************

;完成在 NTFS 卷下查找文件 ghost.img(这是个可启动的软盘镜像文件)找到加载启动它..

BOOTCode10:

jmp BOOTCode20

file_name_length dw 9

file_name dw 'G', 'H', 'O', 'S', 'T','.','I','M','G'

filenamebuffer db 256 dup (0)

BOOTCode20:

push cs

pop ds

call NTFSMain ;安装 NTFS 参数等
```

```
movzx ecx, file_name_length

mov ax, offset file_name

call FindFile ;查找文件

or eax, eax

je BOOTCode99 ;没找到文件退出

mov ebp,0 ;获取开始 LBA

call GetFileDATAorLBA ;eax=开始 LBA

or eax, eax

je BOOTCode99 ;失败退出

mov ebp,eax ;ebp=开始 LBA

call HardDiskImageVirtualFloppy

xor ax,ax

mov es,ax

mov bx,7c00h

mov ax,0201h

mov cx,1

mov dx,0

int 13h

push es

push 7c00h

retf ;启动可启动软盘镜像
```

```
BOOTCode99:

int 18h ;出错退出

org 1ffh ;第一个扇区结束于 1ffh

BOOTCodeSector1 endp

;*****************************************************************

;引导程序第二个扇区:无条件转到第一个扇区开始处执行.....

;注意:Ntldr 加载长于一个扇区时:要求 1:全长代码不小于 2000h(也就是

; 16 个扇区长),要求 2:必须定义第二个扇区,而且第二个扇区的开始处

; 代码要无条件转到第一个扇区开始处执行.分析得来!没具体资料...

BOOTCodeSector2 proc

org 200h ;第二个扇区开始于 200h

jmp BOOTCodeSector1

org 3ffh ;第二个扇区结束于 3ffh

BOOTCodeSector2 endp

;*****************************************************************

;注意:其他代码从这里开始,Ntldr 要求所有代度长度最低要占用 16 个扇区.

; 我们这里不足尾部填 0.达到 Ntldr 要求长度....

org 400h

;*****************************************************************************

;*****************************************************************************

;*****************************************************************************
```

```
;NTFSDOSAPI: NTFS 文件操作 DOS 系统调用接口!
;入口: eax=要调用的 API 号 以及相应 API 参数
;出口: eax=零严重错误返回 以及相应 API 返回
NTFSDOSAPI proc
;push cs
;pop ds
;jmp FoundNTFSAPI
;*********************************************************************
;可以把 NTFS 封装成公共接口.安装成中断调用也可以..定义子函数入口偏移
NTFSAPIStart:
dw offset NTFSMain
dw offset FindFile
dw offset ReadFrs
;*********************************************************************
;杂乱的定义.注意:默认操作的是 C 盘.....
index_name dw '$', 'I', '3', '0'
index_name_length dw 4
DriveNumber db 80h ;默认操作的是 C 盘
SectorBase dd ? ;操作扇区开始 LBA
SectorCount dw ? ;操作的扇区数
NtfsSP dw ? ;保存堆栈
```

NtfsRET dw ? ;保存返回偏移

NtfsCS dw ? ;保存返回段值

NTFSFlags dw ? ;保存标志寄存器

;***

;NTFS 分区 BPB 这里需要的参数,以及计算机出来的参数...

BytesPerSector dw ? ;每扇区字节数

SectorsPerCluster db ? ;每簇扇区数

HiddenSectors dd ? ;隐藏扇区数

MftStartLcn dd ? ;MFT 开始 LCN

ClustersPerFrs dd ? ;每文件记录簇数

BytesPerCluster dd ? ;每簇字节数

BytesPerFrs dd ? ;每文件记录段字节数

SectorsPerFrs dd ? ;每文件记录段扇区数

BytesPerIndexBlock dd ? ;在根索引每索引块字节数

ClustersPerIndexBlock dd ? ;在根索引每索引块簇数

SectorsPerIndexBlock dd ? ;在根索引每索引块扇区数

;***

;NTFS 初始化要用的文件记录文件等缓冲及地址

AttrList dd 0e000h;属性链表缓冲首址

MftFrs dd 03000h;第一个 MFT 文件记录首址

SegmentsInMft dd ? ;文件记录号在 MFT 数据属性记录

```
RootIndexFrs dd ? ;根索引文件记录地址

AllocationIndexFrs dd ? ;分配索引文件记录地址

BitmapIndexFrs dd ? ;位图索引文件记录地址

IndexRoot dd ? ;根索引$INDEX_ROOT 属性地址

IndexAllocation dd ? ;根索引$INDEX_ALLOCATION 属性地址

IndexBitmap dd ? ;根索引$BITMAP 属性地址

FileFrs dd ? ;要查找的文件的文件记录地址

IndexBlockBuffer dd ? ;当前索引缓冲区地址

IndexBitmapBuffer dd ? ;索引位图缓冲区地址

NextBuffer dd ? ;在缓冲区的下一个空闲缓冲区

;数据定义结束.

;*****************************************************************

;*****************************************************************

;Main:  获取设置 NTFS 常用的参数,安装 NTFS 要用到的文件记录...

;  如:每簇扇区数,每文件记录扇区数,,如:根目录索引记录.

NTFSMain proc

push ds

pushad

push cs

pop ds

mov ax,0201h
```

```
mov ebx,MftFrs

mov cx,1

mov dh,01

mov dl,DriveNumber

int 13h ;读 NTFS 引导记录(LBA:3fh)

cmp dword ptr [bx+3],5346544eh ;是 NTFS?

jnz ErrorExit

; 获取 NTFS 资料.....

mov ax,[bx+0bh]

mov BytesPerSector,ax

mov al,[bx+0dh]

mov SectorsPerCluster,al

mov eax,[bx+1ch]

mov HiddenSectors,eax

mov eax,[bx+30h]

mov MftStartLcn,eax

mov eax,[bx+40h]

mov ClustersPerFrs,eax

movzx eax, BytesPerSector ;eax=每扇区字节数

movzx ebx, SectorsPerCluster ;ebx=每簇扇区数

mul ebx
```

```
mov BytesPerCluster,eax ;eax=每簇字节数
mov ecx, ClustersPerFrs
cmp cl, 00
jg NTFSMain1
neg cl
mov eax, 00000001
shl eax, cl ;eax=每 FRS 字节数
jmp NTFSMain2
NTFSMain1:
mov eax, BytesPerCluster
mul ecx ;eax=每 FRS 字节数
NTFSMain2:
mov BytesPerFrs,eax
movzx ebx,BytesPerSector
xor edx, edx
div ebx
mov SectorsPerFrs,eax
call SetupMft
; 下面是加载各个属性
mov ecx, NextBuffer
mov RootIndexFrs, ecx
```

```
add ecx, BytesPerFrs

mov AllocationIndexFrs,ecx

add ecx, BytesPerFrs

mov BitmapIndexFrs,ecx

add ecx, BytesPerFrs

mov FileFrs, ecx

add ecx,BytesPerFrs

mov IndexBlockBuffer,ecx

mov eax, 00000090h ;$INDEX_ROOT

mov ecx, AllocationIndexFrs

call LoadIndexFrs

or eax, eax

je ErrorExit

mov IndexRoot,eax

mov eax, 000000A0h ;$INDEX_ALLOCATION

mov ecx, AllocationIndexFrs

call LoadIndexFrs

mov IndexAllocation, eax

mov eax, 000000B0h ;$BITMAP

mov ecx, BitmapIndexFrs

call LoadIndexFrs
```

```
mov IndexBitmap,eax

mov eax,IndexRoot

or eax, eax

je ErrorExit

cmp byte ptr [eax+08h], 00

jne ErrorExit

; eax -> $INDEX_ROOT 属性记录

lea edx, dword ptr [eax+10h] ;edx->常驻信息

add ax, word ptr [edx+04h] ;eax -> value of $INDEX_ROOT

movzx ecx, byte ptr [eax+0Ch]

mov ClustersPerIndexBlock,ecx

mov ecx, dword ptr [eax+08h]

mov BytesPerIndexBlock,ecx

mov eax, BytesPerIndexBlock

movzx ecx, BytesPerSector

xor edx, edx

div ecx

mov SectorsPerIndexBlock,eax

mov eax, IndexBlockBuffer

add eax, BytesPerIndexBlock

mov IndexBitmapBuffer,eax
```

```
cmp IndexAllocation,00000000
je NTFSMain30
cmp IndexBitmap,00000000
je ErrorExit
mov ebx, IndexBitmap
push ds
pop es
mov edi, IndexBitmapBuffer
call ReadWholeAttribute
NTFSMain30:
popad
pop ds
ret
;*****************************************
; 所有 NTFS 严重错误都无条件转到这里处理.
ErrorExit:
int 18h
NTFSMain endp
;*************************************************************************
;GetFileDATAorLBA :根据文件记录索引入口获取在硬盘的位置(LBA)及数据
;入口: eax=文件记录索引入口 ebp=0 获取位置非 0 获取数据 es:edi=缓冲区
```

```
;出口: eax=文件开始 LBA(0 表示失败) es:edi=文件数据

GetFileDATAorLBA proc

pushad

push es

push edi

; 读文件记录的 DATA 属性

; eax -> 文件记录索引入口

mov eax, dword ptr [eax] ;.IE_FileReference.REF_LowPart

push cs

pop es

mov edi, FileFrs ;es:edi=缓冲

call ReadFrs

mov eax, FileFrs

mov ebx, 00000080h ;$DATA

mov ecx, 00000000 ;属性名长度

mov edx, 00000000 ;属性名

call LocateAttributeRecord

; eax -> $DATA 文件属性

or eax, eax

jz GetFileDATAorLBA99

movzx ebx, word ptr [eax+0Ch];.ATTR_Flags
```

```
and ebx, 000000FFh ;是不是压缩了的
jnz GetFileDATAorLBA99
mov ebx, eax
; 获取开始
cmp ebp,0 ;是获取位置还是获取数据
jnz GetFileDATA
; 获取文件开始 LBA
cmp byte ptr [ebx+08h], 00 ;常驻属性
jne GetFileLBA
jmp GetFileDATAorLBA99 ;文件在文件记录里可以返回在内存位置
GetFileLBA:
lea edx, dword ptr [ebx+10h] ;edx->非常驻属性信息
mov ecx, dword ptr [edx+08h] ;ecx = HighestVcn
inc ecx ;ecx=在属性的簇
sub eax, eax ;eax=开始 VCN
call ComputeLcn ;eax=开始 LCN
movzx ecx, SectorsPerCluster
mul ecx ;转换 LCN 到扇区编号
mov SectorBase, eax ;保存扇区编号(LBA)
mov SectorCount,1
pop edi
```

```
pop es

popad

mov eax,SectorBase

add eax,HiddenSectors ;文件开始 LBA

ret

; 获取文件数据

GetFileDATA:

mov ebx, eax

pop edi

pop es

call ReadWholeAttribute ;读数据到 es:edi

popad

ret

GetFileDATAorLBA99:

pop edi

pop es

popad

xor eax,eax

ret

GetFileDATAorLBA endp

;*************************************************************************
```

```
;ComputeLcn:  转换 VCN 到 LCN

;入口: eax->VCN ebx->属性

;出口: eax->LCN(为 0 没找到) ecx->剩余运行长度

ComputeLcn proc

cmp byte ptr [ebx+08h], 01 ;比较属性值,要是非常驻属性

je clcn10

sub eax, eax ;这是个常驻属性

ret

clcn10:

lea esi, dword ptr [ebx+10h] ;esi->非常驻属性信息

mov edx, dword ptr [esi+08h] ;edx = HighestVcn

cmp eax, edx

ja clcn15 ;VCN 是 HighestVcn

mov edx, dword ptr [esi] ;edx = LowestVcn

cmp eax, edx

jnb clcn20

clcn15:

sub eax, eax ;VCN  不在属性里

ret

clcn20:

add bx, word ptr [esi+10h] ;ebx ->  映射
```

```
sub esi, esi

clcn30:

; ebx -> Current mapping pair count byte

; edx == Current VCN

; esi == Current LCN

cmp byte ptr [ebx], 00

je clcn99

call LcnFromMappingPair

add esi, ecx ;esi =  当前映射 LCN

call VcnFromMappingPair

add ecx, edx ;ecx=下一个 VCN

cmp eax, ecx ;如果小于下一个 VCN...

jl clcn80 ;...  我们找到正确的映射.

mov edx, ecx ;当前 VCN =  下一个 VCN

push eax

movzx ecx, byte ptr [ebx] ;ecx =  字节数

mov eax, ecx

and eax, 0000000Fh ;eax = VCN 字节数

shr ecx, 04h

add ebx, ecx ;ecx = LCN 字节数

add ebx, eax
```

```
inc ebx ;ebx ->  下一个字节数

pop eax

jmp clcn30

clcn80:

;  找到我们要映射的值

sub ecx, eax ;ecx=剩余运行长度

sub eax, edx

add eax, esi ;eax=LCN

ret

clcn99:

; VCN 没有在这个属性里

sub eax, eax ;没找到

ret

ComputeLcn endp

;*****************************************************************************
******

;Findfile:  在根目录;下查找一个文件的索引入口

;入口: eax->要查找的文件名称  ecx=文件名长度

;出口: eax->文件索引入口  NULL 表示查找失败

FindFile proc

push eax

push ecx
```

; 开始, 搜索根目录

mov edx, eax

mov eax, IndexRoot ;eax ->INDEX_ROOT 属性

lea ebx, dword ptr [eax+10h] ;ebx -> 常驻属性

add ax, word ptr [ebx+04h] ;eax -> Index Root value

lea eax, dword ptr [eax+10h] ;eax -> Index Header

mov ebx, edx

call LocateIndexEntry

or eax, eax

je FindFile20 ;没找到

; 在根目录找到,结果保存在 eax,清除堆栈返回

pop ecx

pop ecx

ret

FindFile20:

; 在根目录没找到,接着在索引分配的缓冲区里查找.

mov eax, IndexAllocation

or eax, eax

jne FindFile30

; 若没有索引分配属性,清除堆并失败返回.

pop ecx

```
pop ecx
xor eax, eax
ret
FindFile30:
; 搜索索引分配块,搜索这个 B+树.....注意:算法.
mov edx, IndexAllocation ;edx->分配索引属性
lea edx, dword ptr [edx+10h] ;edx->非常属性
mov eax, dword ptr [edx+08h] ;eax = HighestVcn
inc eax ;eax = 簇在属性
mov ebx, BytesPerCluster
mul ebx ;字节在属性
xor edx, edx
div BytesPerIndexBlock ;转换为索引块
push eax
FindFile40:
; 搜索剩余的索引块
pop eax ;剩余索引块
or eax, eax
je FindFile90
dec eax ;下一个索引块
push eax
```

```
call IsBlockInUse
jb FindFile40
call ReadIndexBlock
pop edx ;搜索剩余缓冲
pop ecx ;edx=文件名长度
pop ebx ;ebx->文件名
push ebx
push ecx
push edx
; 索引缓冲搜索是在索引分配缓冲块
mov eax, IndexBlockBuffer ;eax->索引分配块
lea eax, dword ptr [eax+18h] ;eax->索引头
call LocateIndexEntry ;eax->找到入口
or eax, eax
je FindFile40
; 找到清除堆栈,返回.
pop ecx
pop ecx
pop ecx
ret
FindFile90:
```

```
; 文件没找到,失败回.

pop ecx

pop ecx

xor eax,eax

ret

FindFile endp

;*****************************************************************************
************
;IsBlockInUse:  索引分配块用来检查索引位图.

;入口: eax=块号

IsBlockInUse proc

push eax

push ebx

push ecx

mov ebx, IndexBitmapBuffer

mov ecx, eax

shr eax, 03h ;eax=字节号

and ecx, 00000007h ;ecx=在字节的位号

add ebx, eax ;ebx -> byte to test

mov eax, 00000001h

shl eax, cl ;eax=mask

test byte ptr [ebx], al
```

```
je IBU10
clc ;Block is not in use.
jmp IBU20
IBU10:
stc ; Block is in use.
IBU20:
pop ecx
pop ebx
pop eax
ret
IsBlockInUse endp
;*************************************************************************
;LcnFromMappingPair:
;入口: ebx->映射字节数
;出口: ecx->LCN 来自映射
LcnFromMappingPair proc
push ebx
push edx
sub edx, edx
mov dl, byte ptr [ebx] ;edx = count byte
and edx, 0000000Fh ;edi = v
```

```
sub ecx, ecx
mov cl, byte ptr [ebx]
```