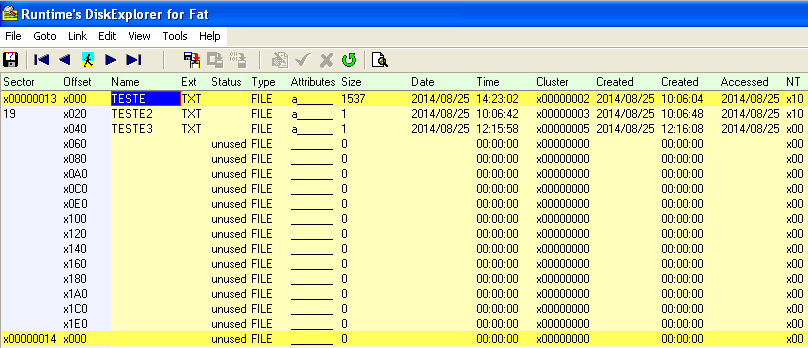
**Resumo de Leitura do Floppy utilizando a FAT 12**

Usando a Bios Parameter Block (Bpb), que são parâmetros que delimitam a utilização do disco e servem como base de cálculo para acesso utilizando LBA e CHS.

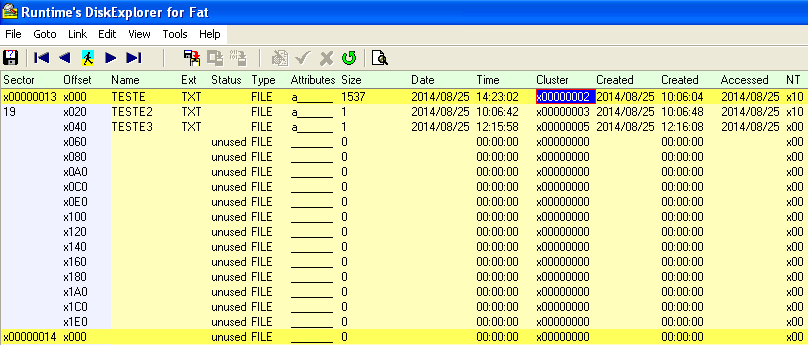
Utilizando a tabela a baixo, os dados dão organizados no disco.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Setor de Boot | Setores extra reservado | Tabela de alocação de arquivo 1 | Tabela de alocação de arquivo 2 | Diretório Raiz (FAT12/FAT16 Somente) | Região de dados contendo arquivos e diretórios. |

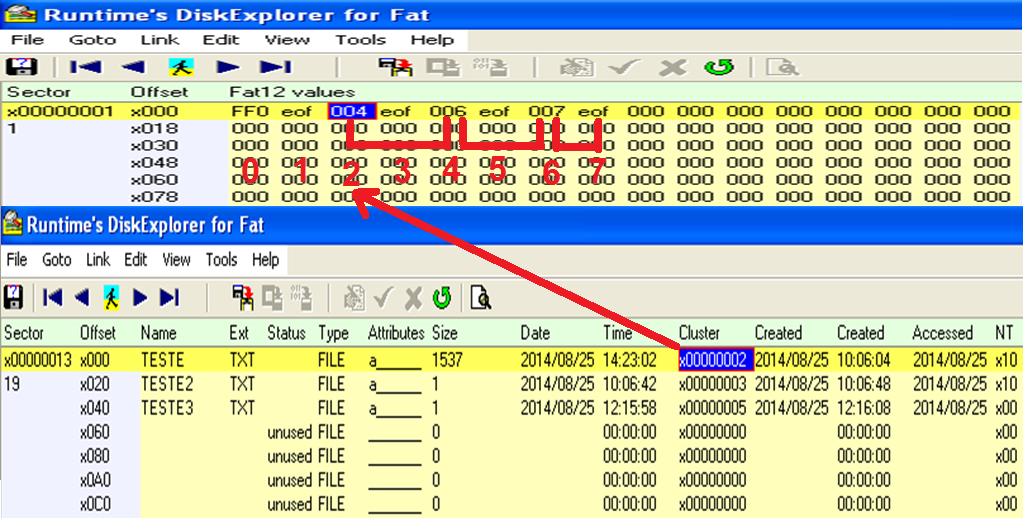
Iniciamos a leitura no diretório Raiz, buscamos o arquivo pelo nome, como destacado abaixo:



Na sequência, identificamos o cluster [Setores] onde se inicia a área de dados do arquivo, como destacado abaixo.



Após a identificação do cluster inicial do arquivo, iremos ler a tabela FAT, para identificarmos todos os clusters utilizados pelos dados contidos no arquivo. A leitura da FAT é feita da seguinte forma:



O cluster inicial do arquivo é 02, vamos até a FAT e lemos o cluster 02. Cluster 0 e 1 [FF0 eof] são reservado para identificação do tipo de disco.

Então a leitura transcorre da seguinte forma: Leio o cluster 2, ele me manda ir para o cluster 4, o 4 me manda ir para o cluster 6 e o 6 me manda para o cluster 7 que identifica o fim do arquivo.

Então a leitura dos dados seguem a sequência: Leitura do Cluster -> Conversão para LBA -> Conversão para CHS -> Leitura do próximo Cluster ..........

Sistema de arquivos FAT

FAT é a sigla para *File Allocation Table* (traduzindo: Tabela deAlocação de Arquivos). A primeira versão do FAT surgiu em 1977, para trabalhar com o sistema operacional MS-DOS, mas foi padrão até o Windows 95.

Trata-se de um sistema de arquivos que funciona com base em uma espécie de tabela que indica onde estão os dados de cada arquivo. Esse esquema é necessário porque o espaço destinado ao armazenamento é dividido em blocos, e cada arquivo gravado pode ocupar vários destes, mas não necessariamente de maneira sequencial: os blocos podem estar em várias posições diferentes. Assim, a tabela acaba atuando como um "guia" para localizá-los.

Com o surgimento de dispositivos de armazenamento mais sofisticados e com maior capacidade, o sistema FAT foi ganhando revisões, identificadas pelos nomes FAT12 e FAT16, sendo o primeiro quase um desconhecido e o último padrão dos sistemas operacionais da Microsoft por muito tempo. As versões surgem com o intuito de eliminar determinadas limitações do sistema de arquivos anterior. O próprio FAT16, por exemplo, passou por isso: esta versão só trabalha com, no máximo, 2 GB, assim, para aplicá-lo em um disco de 5 GB, seria necessário dividi-lo em 3 partições (2 GB + 2 GB + 1 GB, por exemplo) para ser possível o aproveitamento de toda a capacidade da unidade.

Diante deste e de outros problemas, a Microsoft lançou, em 1996, o FAT32, que se tornou o sistema de arquivos do Windows 95 (versão OSR 2) e do Windows 98, sendo também compatível com versões lançadas posteriormente, como Windows 2000 e Windows XP, embora estes tenham um sistema de arquivos mais avançado, o NTFS.

Entendendo os sistemas de arquivos FAT

Em um disco rígido, a área de armazenamento é dividida em trilhas. Cada trilha é subdividida em setores (saiba mais [neste artigo sobre HDs](http://www.infowester.com/hd.php)), cada um com 512 bytes, geralmente. Desse modo, é de se presumir que os sistemas de arquivos FAT trabalhem diretamente com esses setores. Mas não é bem assim.

Na verdade, o FAT trabalha com grupos de setores, onde cada um recebe a denominação **cluster**(ou unidade de alocação). No caso do FAT16, cada cluster pode ter, comumente, um dos seguintes tamanhos: 2 KB, 4 KB, 8 KB, 16 KB e, por fim, 32 KB. A definição desse tamanho é uniforme, ou seja, não pode haver tamanhos variados de clusters em uma mesma unidade de armazenamento.

Cada arquivo gravado utiliza tantos clusters quanto forem necessários para cobrir o seu tamanho. Se, por exemplo, tivermos um arquivo com 50 KB, é possível guardá-lo em dois clusters de 32 KB cada. Você deve ter percebido então que, neste caso, um cluster ficou com espaço sobrando. Esta área pode ser destinada a outro arquivo, correto? Errado! Acontece que cada cluster só pode ser utilizado por um único arquivo. Se sobrar espaço, este permanecerá vazio. Esse é um dos problemas do sistema FAT: desperdício.

Normalmente, o tamanho dos clusters é definido no procedimento de instalação do sistema operacional, na etapa de formatação da unidade de armazenamento.

Diferenças entre FAT16 e FAT32

O FAT16 utiliza 16 bits para endereçamento dos dados (daí o número 16 na sigla), o que, na prática, significa que o sistema de arquivos pode trabalhar com até 65536 clusters, no máximo. Para chegar a este número, basta fazer 2 elevado a 16 (65536).

Se temos então até 65536 clusters e cada um pode ter até 32 KB de tamanho, significa que o sistema FAT16 é capaz de trabalhar com discos ou partições com até 2 GB: 65536 x 32 = 2.097.152 KB, que corresponde a 2 GB.

O sistema de arquivos FAT32 consegue solucionar esse problema por utilizar 32 bits no endereçamento de dados (novamente, aqui você pode perceber o porquê do número na sigla). No FAT16, quanto maior o espaço em disco (considerando o limite de até 2 GB, é claro), maior o tamanho do cluster. Com o FAT32, é possível usar clusters menores - geralmente de 4 KB - mesmo com a unidade oferecendo maior capacidade de armazenamento. Desta forma, o desperdício acaba sendo menor.

O limite do FAT32 é de 2 TB (terabytes). Perceba, no entanto, que se você fizer o cálculo anterior considerando 32 em vez de 16 (2 elevado a 32) e, posteriormente, multiplicar o resultado pelo tamanho máximo do cluster (também 32), o valor obtido será de 128 TB. Então, qual o motivo do limite de 2 TB?

Na verdade, cada endereçamento tem tamanho de 32 bits, mas, no FAT32, o número máximo de clusters é calculado considerando apenas 28 bits, fazendo a conta ser 2 elevado a 28, que é igual a 268.435.456, ou seja, pouco mais de 268 milhões de clusters. Multiplicando esse número por 32, teremos então 8 TB.

Ok, novamente a conta não fechou, afinal, 8 TB para 2 TB é uma diferença muito grande. A explicação está no fato de que a Microsoft limitou o FAT32 a ter 2 elevado a 32 como quantidade máxima de setores, não de clusters (se fosse diferente, poderia haver problemas com a inicialização do sistema operacional devido a limitações na área de boot). Como cada setor, geralmente, possui 512 bytes (ou 0,5 kilobyte), a conta seria 2 elevado a 32 (4.294967296) multiplicado por 0,5, que é igual 2.147.483.648 KB ou 2 TB.

Fragmentação

Para os usuários do Windows 95/98, uma recomendação era frequente: utilizar um aplicativo de desfragmentação de disco regularmente. Isso tem um bom motivo: toda vez que um arquivo é apagado, seus clusters ficam disponíveis para nova utilização. Acontece que o sistema operacional sempre ocupa os primeiros clusters livres e, se houver áreas ocupadas no caminho, continuará utilizando os clusters livres subsequentes.

O resultado é que, com o passar do tempo, há fragmentos de dados por todo o disco. Isso torna o acesso aos arquivos mais lento, uma vez que o sistema precisa procurar "pedaço por pedaço". A desfragmentação consegue amenizar este problema porque reorganiza os arquivos em clusters sequenciais, deixando-os acessíveis mais rapidamente.

VFAT

VFAT é a sigla para *Virtual File Allocation Table*. Trata-se de uma espécie de extensão para FAT16 introduzido no Windows 95 para que este sistema operacional possa suportar arquivos com mais de 11 caracteres no nome. É que, por padrão, o FAT16 limita o tamanho dos nomes para 8 caracteres mais 3 destinados à extensão, por exemplo, *daniella.txt*. Caso nomes maiores sejam utilizados, estes aparecem de forma abreviada no sistema.

Com o uso do VFAT, as características do FAT16 são mantidas e, ao mesmo tempo, nomes maiores podem ser utilizados sem qualquer dificuldade. Isso porque o VFAT faz o sistema enxergar o nome abreviado, mas guarda o nome original em um área separada.

O FAT32 trabalha com VFAT por padrão.

Finalizando

Para os padrões atuais, os sistemas de arquivo FAT se mostram como ultrapassados. Isso se deve principalmente ao surgimento do sistema de arquivos NTFS, que é mais seguro e eficiente. Não por menos, é utilizado nas versões mais recentes do Windows, como XP, Vista e 7, por exemplo.

**Desenvolvimento de Sistemas Operacionais - Bootloaders 4**   
**by Mike 2009**

Esta série tem a intenção de demonstrar e ensinar o desenvolvimento do sistema operacional a partir do zero.

**Introdução**

Bem-vindo! No tutorial anterior falamos sobre como carregar e executar um setor. Também demos uma olhada em Anéis em linguagem assembly, e um olhar detalhado sobre o BIOS Parameter Block (BPB).

Neste tutorial, vamos usar tudo o que aprendi para analisar o sistema de arquivos FAT12, e carregar o segundo estágio do carregador pelo nome,

Este tutorial vai ter um monte de código nele. Eu farei o meu melhor possível para explicar tudo em detalhes. Além disso, este tutorial vai ter um pouco de matemática nele também.

Preparado?

**cli e hlt**

Você pode estar curioso em saber porque eu acabo todos os meus programas de demonstração com as instruções "cli" e "HLT". É realmente muito simples. Se não houver nenhuma maneira de parar o programa de alguma maneira, a CPU só vai para além do seu programa e executar instruções aleatórias. Isto, em última análise, acabam em uma falha tripla.

CLI – desabilita interrupções

STI - habilitar as interrupções

HLT - parar a CPU até que uma interrupção ocorra. Se as interrupções são desativadas, irá parar a CPU até que um NMI ocorre.

**Sistema de Arquivos - Teoria**

Agora é hora de falarmos de Sistema de Arquivos

Um sistema de arquivos não é nada mais do que uma especificação. Ela ajuda a criar o conceito de um "arquivo" em um disco.

Um arquivo é um conjunto de dados que representa algo. Estes dados podem ser qualquer coisa que nós queremos que ele seja. Tudo depende da forma como interpretamos os dados.

Como você sabe, um setor tem 512 bytes de tamanho. Um arquivo é armazenado nestes sectores no disco. Se o arquivo for superior a 512 bytes, temos de dar-lhe mais setores. Como nem todos os arquivos são uniformemente 512 bytes de tamanho, teremos de preencher o resto dos bytes (se o arquivo não usa). Mais ou menos como o que fizemos para o nosso gerenciador de inicialização.

Se um arquivo se estende por diversos setores, nós chamamos esses setores de cluster nos sistemas de arquivos FAT. Por exemplo, nosso kernel provavelmente terá um monte de setores. Para colocar o nosso kernel, teremos de carregar o cluster (Os setores) de onde ele está localizado.

Se um arquivo se estende por diferentes setores (não contínuos) em diferentes grupos, diz-se que é fragmentado. Vamos precisar de recolher as diferentes partes do arquivo.

Há um monte de diferentes tipos de sistemas de arquivos. Alguns são amplamente utilizados (como FAT12, FAT16, FAT32, NTFS, ext (Linux), HFS (Utilizado em Macs mais antigos); outros sistemas de arquivos são apenas usados ​​por empresas específicas para uso em casa (como o GFS - Google File System).

Um monte de desenvolvedores de OS também utiliza versões dos sistemas de arquivos FAT (ou mesmo algo completamente novo). Estes geralmente não são tão bons como os sistemas de arquivos mais comuns (como FAT e NTFS).

Ok, então vamos conhecer um pouco sobre sistemas de arquivos agora. Nós vamos usar o sistema FAT12 por sua simplicidade. Se decidirmos, podemos sempre usar um diferente.

**FAT12 Sistema de Arquivos - Teoria**

FAT12 é o primeiro FAT (File Allocation Table) sistema de arquivos lançado em 1977, e utilizado no Microsoft Disk BASIC. FAT12, como sendo um sistema de arquivos mais velhos geralmente liberado para disquetes, tinha uma série de limitações.

* FAT12 não tem suporte para diretórios hierárquicos. Isto significa que há apenas um diretório no diretório raiz.
* Endereços de cluster eram apenas 12 bits de comprimento, o que limita o número máximo de clusters para 4096
* Os nomes dos arquivos são armazenados no FAT como um identificador de 12 bits. Os endereços de cluster representam os clusters a partir dos arquivos.
* Devido ao tamanho limitado cluster, o número máximo de arquivos possíveis é 4.077
* O tamanho do disco é armazenado apenas como uma contagem de setores 16 bits, limitando-o a 32 MB de tamanho
* FAT12 usa o valor "0x01" para identificar partições

Estas são algumas das grandes limitações. Por que queremos FAT12, então?

FAT16 tem suporte para diretórios, e mais de 64.000 arquivos, uma vez que utiliza um conjunto de 16 bits (arquivo) endereço, ao invés de FAT16. No entanto, FAT16 e FAT12 são muito semelhantes**.**

Para tornar as coisas simples, vamos usar FAT12. Podemos enfeitar as coisas com FAT16 (ou mesmo usar FAT32) depois :) (FAT32 é bastante diferente, em seguida, FAT 12/16, para que possamos usar FAT16 mais tarde.)

**FAT12 Sistema de Arquivos – Armazenamento de Disco**

Para entender mais sobre FAT12, e como ele funciona, é melhor olhar para a estrutura de um disco típico quando formatado.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Setor de Boot | Setores extra reservado | Tabela de alocação de arquivo 1 | Tabela de alocação de arquivo 2 | Diretório Raiz (FAT12/FAT16 Somente) | Região de dados contendo arquivos e diretórios. |

Este é um disco típico, no formato FAT12, do setor de boot até o último setor no disco.

Compreender esta estrutura será importante ao carregar e procurar o nosso arquivo.

**Note que há duas FATs em um disco. Ele está localizado \* logo após \* os setores reservados (ou o bootloader, se não houver nenhum).**

**Além disso, note: O diretório raiz é certo, afinal das FATs. Isso significa ...**

**se somarmos o número de setores por FAT, e os setores reservados, podemos obter o primeiro setor para o diretório raiz. Ao pesquisar o diretório raiz para uma seqüência simples (nosso nome), podemos efetivamente encontrar exatamente o setor do arquivo no disco :)**

Vamos olhar mais de perto ...

**Setor de Boot**

Esta seção contém a BIOS Parameter Block e o bootloader. O BIOS Parameter Block contém informações que ajuda descrever o nosso disco.

**Setores extra reservados**

Lembre-se o membro do nosso bpbReservedSectors BPB? Quaisquer sectores extra reservados são armazenados aqui, logo após o setor de inicialização.

**Tabelas de Alocação Arquivo (FATs)**

Lembre-se que um cluster representa uma série de setores situados no disco. O tamanho de cada cluster é normalmente 2 KB até 32 KB. Os pedaços de arquivos são ligados (de um cluster para outro usando uma estrutura de dados comum, como uma lista ligada.

Existem duas FATs. No entanto, é apenas uma cópia do primeiro para fins de recuperação de dados. Geralmente, não é utilizado.

**A tabela de alocação (FAT) é uma lista de entradas que são mapeadas para cada um desses grupos. Eles ajudam a identificar informações importantes para ajudar no armazenamento de dados a estes clusters.**

Cada entrada é um valor de 12 bits que representa um cluster. O FAT é uma estrutura de lista semelhante ligada a essas entradas que ajuda a identificar os clusters que estão em uso.

Para entender melhor, vamos olhar para os valores possíveis:

* **Este valor marca os clusters livres :** 0x00
* **Este valor marca os clusters ocupados :** 0x01
* **Este cluster está em uso - o valor representa próximo cluster:** 0x002 até 0xFEF
* **Valores reservados :** 0xFF0 até 0xFF6
* **Este valor marca clusters defeituosos:** 0xFF7
* **Valor assinala este cluster como o último no arquivo:** 0xFF8 até 0xFFF

A FAT é apenas uma matriz desses valores - isso é tudo. Quando encontramos o setor começando formar o diretório raiz, podemos olhar através do FAT para encontrar que agrupa para carregar. Como? Nós simplesmente verificamos o valor. Se o valor for entre 0x02 e 0xff, este valor representa o próximo cluster para carregar o arquivo.

Vamos olhar para isso de uma maneira mais profunda. Um cluster, como se sabe, representa uma série de setores. Nós definimos a quantidade de setores que representa a partir do BIOS Parameter Block:

bpbBytesPerSector: DW 512

bpbSectorsPerCluster: DB 1

No nosso caso, cada cluster é um setor. Quando temos o primeiro setor da Fase 2 (Recebemos esta no diretório raiz), usamos esse setor como o número de cluster a partir do FAT. Uma vez que nós encontramos o conjunto de partida, nós apenas referenciamos o FAT para determinar o cluster (O FAT é apenas um conjunto de números de 32 bits. Nós apenas comparamos este número com a lista acima para determinar o que fazer com ele.)

**Tabela de Diretório Raiz**

Agora, isso vai ser importante para nós:)

O diretório raiz é uma tabela de valores de 32 bytes que representam informações sobre arquivos e diretórios. Este valor de 32 bytes usa o formato:

* **Bytes 0-7 : DOS - Nome do arquivo (preenchido com espaços)**
* **Bytes 8-10 : DOS – Extensão de arquivo (preenchido com espaços)**
* **Bytes 11 :** Atributos de arquivo. Este é um padrão de bits
  + **Bit 0 :** Somente Leitura
  + **Bit 1 :** Oculto
  + **Bit 2 :** Arquivo de Sistema
  + **Bit 3 :** Volume
  + **Bit 4 :** Subdiretório
  + **Bit 5 :** Arquivo
  + **Bit 6 :** Dispositivo (uso interno)
  + **Bit 7 :** Não usado
* **Bytes 12 :** Não usado
* **Bytes 13 :** Criado em: tempo (ms)
* **Bytes 14-15 :** Criado. Horário, usando o seguinte formato:
  + **Bit 0-4 :** Segundos (0-29)
  + **Bit 5-10 :** Minutos (0-59)
  + **Bit 11-15 :** Horas (0-23)
* **Bytes 16-17 :** Criado. Ano, usando o seguinte formato:
  + **Bit 0-4 :** Ano (0=1980; 127=2107)
  + **Bit 5-8 :** Mês (1=January; 12=December)
  + **Bit 9-15 :** Horas (0-23)
* **Bytes 18-19 :** Última data de acesso (Usa mesmo formato acima)
* **Bytes 20-21 :** EA Index (Used in OS/2 and NT, não se preocupe com isso)
* **Bytes 22-23 :** Horário da Última Modificação (Veja bytes 14-15 para o formato)
* **Bytes 24-25 :** Data da Última Modificação (Veja bytes 16-17 for format)
* **Bytes 26-27 :** **Primeiro Cluster**
* **Bytes 28-31 :** **Tamanho do Arqivo**

Em negrito as partes importantes - tudo o resto é apenas lixo Microsoft acrescentou que podemos acrescentar a quando criamos um driver FAT12, muito mais tarde.

Espere um minuto. Lembre-se que os nomes de arquivos DOS estão limitados a 11 bytes? É por isso que:

* **Bytes 0-7 : DOS - Nome do arquivo (preenchido com espaços)**
* **Bytes 8-10 : DOS – Extensão de arquivo (preenchido com espaços)**

De 0 a 10, isso é 11 bytes. Ter um nome de arquivo a menos de 11 bytes vai perder-se a entrada de dados (A tabela de entrada de 32 bytes exibido acima). Isto, naturalmente, é ruim :) Devido a isso, nós temos que preencher os nomes de arquivos com caracteres, e segurá-lo até 11 bytes.

Lembre-se, em um tutorial anterior eu expliquei como são os seus nomes de arquivos internos e externos? A estrutura de arquivo eu expliquei é o nome do arquivo interno. Como é limitada a 11 bytes, o nome do arquivo "Stage2.sys" tem de se tornar

"STAGE2 SYS" (Observe os espaçõs!)

**Buscando e Lendo FAT12 - Teoria**

depois de ler todos os itens acima, você provavelmente está cansado de me dizer "FAT12":)

Como é uma informação útil para nós?

Vamos ser referência a um monte BIOS Parameter Block (BPB). Aqui está o BPB que criamos com os tutoriais anteriores para referencia:

bpbBytesPerSector: DW 512

bpbSectorsPerCluster: DB 1

bpbReservedSectors: DW 1

bpbNumberOfFATs: DB 2

bpbRootEntries: DW 224

bpbTotalSectors: DW 2880

bpbMedia: DB 0xF0

bpbSectorsPerFAT: DW 9

bpbSectorsPerTrack: DW 18

bpbHeadsPerCylinder: DW 2

bpbHiddenSectors: DD 0

bpbTotalSectorsBig: DD 0

bsDriveNumber: DB 0

bsUnused: DB 0

bsExtBootSignature: DB 0x29

bsSerialNumber: DD 0xa0a1a2a3

bsVolumeLabel: DB "MOS FLOPPY "

bsFileSystem: DB "FAT12 "

Por favor, veja o tutorial anterior para uma explicação detalhada de cada membro.

O que estamos tentando fazer é carregar um segundo estagio. Vamos olhar para o que temos de fazer em detalhes:

**Começando com um nome de arquivo**

A primeira coisa a fazer é criar um bom nome. Lembre-se: **os nomes de arquivo deve ser exatamente 11 bytes para garantir que não corrompem o diretório raiz.**

Eu estou usando "STAGE2.SYS", para o meu segundo estágio. Você pode olhar para um exemplo de seu nome de arquivo interno na seção acima.

**Criando Stagio 2**

Ok, Stage2 é um programa separado, em seguida, ao bootloader. Nosso Stage2 será muito similar a um programa DOS COM, parece legal?

O Stage2 irá imprimir uma mensagem e parar. Tudo o que você já tenha visto a partir do bootloader:

; Note: Here, we are executed like a normal

; COM program, but we are still in Ring 0.

; We will use this loader to set up 32 bit

; mode and basic exception handling

; This loaded program will be our 32 bit Kernel.

; We do not have the limitation of 512 bytes here,

; so we can add anything we want here!

org 0x0 ; offset to 0, we will set segments later

bits 16 ; we are still in real mode

; we are loaded at linear address 0x10000

jmp main ; jump to main

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*;

; Prints a string

; DS=>SI: 0 terminated string

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*;

Print:

lodsb ; load next byte from string from SI to AL

or al, al ; Does AL=0?

jz PrintDone ; Yep, null terminator found-bail out

mov ah, 0eh ; Nope-Print the character

int 10h

jmp Print ; Repeat until null terminator found

PrintDone:

ret ; we are done, so return

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*;

; Second Stage Loader Entry Point

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*;

main:

cli ; clear interrupts

push cs ; Insure DS=CS

pop ds

mov si, Msg

call Print

cli ; clear interrupts to prevent triple faults

hlt ; hault the system

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*;

; Data Section

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*;

Msg db "Preparing to load operating system...",13,10,0

Para montar com NASM, apenas montá-lo como um programa binário (programas COM são binários) e copiá-lo para a imagem de disquete. por exemplo:

nasm -f bin Stage2.asm -o STAGE2.SYS

copy STAGE2.SYS A:\STAGE2.SYS

Não é necessário usar o PARTCOPY :)

**Passo 1: Carregando a Tabela Diretório Raiz**

Agora é hora de carregar a Fase 2. Diz! Nós faremos muita referência a tabela de diretório raiz, junto com o bloco de parâmetros do BIOS para obter informações do disco.

**Passo 1: Pegando o tamanho do diretório raiz**

Ok, primeiro precisamos obter o tamanho do diretório raiz.

Para obter o tamanho, basta multiplicar o número de entradas no diretório raiz. Parece bastante simples :)

No Windows, sempre que você adicionar um arquivo ou diretório para um disco FAT12 formatado, o Windows adiciona automaticamente as informações do arquivo para o diretório raiz, por isso não precisa se ​​preocupar com isso. Isso torna as coisas muito mais simples.

Aqui está um exemplo:

mov ax, 0x0020 ; 32 byte directory entry

mul WORD [bpbRootEntries] ; number of root entrys

div WORD [bpbBytesPerSector] ; get sectors used by root directory

Lembre-se de que a tabela de diretório raiz é uma tabela de valores de 32 bytes (entradas) que representam as informações do arquivo.

Ok, nós sabemos quantos setores precisamos carregar para o diretório raiz. Agora, vamos encontrar a partir de que setor devemos carregar ☺

**Passo 2: Obter início do diretório raiz**

Este é mais fácil. Primeiro, vamos olhar para um disco FAT12 formatado novamente:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Setor de Boot | Setores extra reservado | Tabela de alocação de arquivo 1 | Tabela de alocação de arquivo 2 | Diretório Raiz (FAT12/FAT16 Somente) | Região de dados contendo arquivos e diretórios. |

Ok, note que o **diretório raiz está localizado logo após as duas FATs e setores** reservados. Em outras palavras, basta adicionar as FATs + setores reservados, e você encontrou o diretório raiz!

Por exemplo...

mov al, [bpbNumberOfFATs] ; Get number of FATs (Useually 2)

mul [bpbSectorsPerFAT] ; number of FATs \* sectors per FAT; get number of sectors

add ax, [bpbReservedSectors] ; add reserved sectors

; Now, AX = starting sector of root directory

Muito fácil, não é? Agora, acabamos de ler o setor para algum local na memória:

mov bx, 0x0200 ; load root directory to 7c00:0x0200

call ReadSectors

**Diretório Raiz – Exemplo Completo**

Este exemplo de código é levado diretamente a partir do gerenciador de inicialização é o fim do tutorial. Ele carrega o diretório raiz:

LOAD\_ROOT:

; compute size of root directory and store in "cx"

xor cx, cx

xor dx, dx

mov ax, 0x0020 ; 32 byte directory entry

mul WORD [bpbRootEntries] ; total size of directory

div WORD [bpbBytesPerSector] ; sectors used by directory

xchg ax, cx

; compute location of root directory and store in "ax"

mov al, BYTE [bpbNumberOfFATs] ; number of FATs

mul WORD [bpbSectorsPerFAT] ; sectors used by FATs

add ax, WORD [bpbReservedSectors] ; adjust for bootsector

mov WORD [datasector], ax ; base of root directory

add WORD [datasector], cx

; read root directory into memory (7C00:0200)

mov bx, 0x0200 ; copy root dir above bootcode

call ReadSectors

**Passo 2: Localizar o Stage 2**

Ok, agora a tabela de diretório raiz está carregada. Olhando para o código acima, **podemos carregá-la para 0x200**. Agora, para encontrar o nosso arquivo.

Vamos olhar de volta ao diretório raiz de 32 bytes novamente (Seção Root Table Directory. Lembre-se os primeiros 11 bytes representam o nome do arquivo. Lembre-se também que, **que cada entrada de diretório raiz é de 32 bytes, a cada 32 bytes será o início da próxima entrada - Apontando-nos de volta para os primeiros 11 bytes da próxima entrada.**

Por isso, tudo o que precisamos fazer é comparar os nomes de arquivo, e saltar para a próxima entrada (32bytes), e testar novamente até chegar ao final do setor. por exemplo...

; browse root directory for binary image

mov cx, [bpbRootEntries] ; the number of entrys. If we reach 0, file doesnt exist

mov di, 0x0200 ; Root directory was loaded here

.LOOP:

push cx

mov cx, 11 ; eleven character name

mov si, ImageName ; compare the 11 bytes with the name of our file

push di

rep cmpsb ; test for entry match

pop di

je LOAD\_FAT ; they match, so begin loading FAT

pop cx

add di, 32 ; they dont match, so go to next entry (32 bytes)

loop .LOOP

jmp FAILURE ; no more entrys left, file doesnt exist :(

Agora iremos para a próxima etapa...

**Passo 3: Carregando a FAT**

**Passo 1: Obter o Cluster Inicial**

Ok, então o diretório raiz é carregado e encontramos a entrada de arquivos. Como é que vamos obter seu conjunto de partida?

* Bytes 26-27 : Primeiro Cluster
* Bytes 28-32 : Tamanho do Arquivo

Para obter o cluster inicial, byte 26 apenas referência a entrada do arquivo:

mov dx, [di + 0x001A] ; di contains starting address of entry. Just refrence byte 26 (0x1A) of entry

; Yippe--dx now stores the starting cluster number

O cluster inicial será importante para nós durante o carregamento do arquivo.

**Passo 2: Obter o tamanho da FAT**

Vamos olhar para o bloco de parâmetro BIOS novamente. mais especificamente...

bpbNumberOfFATs: DB 2

bpbSectorsPerFAT: DW 9

Ok, então como é que vamos descobrir quantos setores existem em ambas as FATs? Apenas multiplique setores por FAT pelo número de setores :) Parece simples, ...porém...

xor ax, ax

mov al, [bpbNumberOfFATs] ; number of FATs

mul WORD [bpbSectorsPerFAT] ; multiply by number of sectors per FAT

; ax = number of sectors the FATs use!

não se preocupe, é simples ^^

**Passo 3: Carregar a FAT**

Agora que sabemos quantos setores para ler. Assim, é só ler:)

mov bx, 0x0200 ; address to load to

call ReadSectors ; load the FAT table

Agora, com o material FAT fora do caminho (não completamente!), Carregamos o estágio 2!

**FAT – Exemplo Completo**

Aqui está o código completo, carregado diretamente a partir do gerenciador de inicialização:

LOAD\_FAT:

; save starting cluster of boot image

mov si, msgCRLF

call Print

mov dx, WORD [di + 0x001A]

mov WORD [cluster], dx ; file's first cluster

; compute size of FAT and store in "cx"

xor ax, ax

mov al, BYTE [bpbNumberOfFATs] ; number of FATs

mul WORD [bpbSectorsPerFAT] ; sectors used by FATs

mov cx, ax

; compute location of FAT and store in "ax"

mov ax, WORD [bpbReservedSectors] ; adjust for bootsector

; read FAT into memory (7C00:0200)

mov bx, 0x0200 ; copy FAT above bootcode

call ReadSectors

**LBA e CHS**

Em carregar a imagem, tudo o que precisamos fazer é carregar cada cluster, fazendo referência a FAT.

Há um pequeno problema que ainda não discutimos. Ok, temos um número de cluster do FAT. **Mas, Como podemos usá-lo?**

O problema é que este cluster representa um endereço linear, enquanto que, a fim de carregar os setores, vamos precisar de um endereço **segment / track / head (Interrupt 0x13)**

Há duas maneiras de acessar um disco. Seja através de **Cylinder/Head/Sector endereçamento (CHS) ou endereçamento** **Logical Block (LBA)**.

O **LBA** representa um local indexado no disco. O primeiro bloco sendo 0, em seguida, 1, e assim por diante. LBA simplesmente representa os setores são numerados sequencialmente com LBA 0. Então você precisa saber como converter CHS para LBA.

**Converter CHS para LBA**

**A fórmula para converter CHS para LBA:**

LBA = (cluster - 2 ) \* sectors per cluster

Isso é bastante simples. :) Aqui está um exemplo:

sub ax, 0x0002 ; subtract 2 from cluster number

xor cx, cx

mov cl, BYTE [bpbSectorsPerCluster] ; get sectors per cluster

mul cx ; multply

**Converter LBA para CHS**

Este é um pouco mais complexo, mas ainda é relativamente fácil:

absolute sector = (logical sector / sectors per track) + 1

absolute head = (logical sector / sectors per track) MOD number of heads

absolute track = logical sector / (sectors per track \* number of heads)

Aqui está um exemplo...

LBACHS:

xor dx, dx ; prepare dx:ax for operation

div WORD [bpbSectorsPerTrack] ; divide by sectors per track

inc dl ; add 1 (obsolute sector formula)

mov BYTE [absoluteSector], dl

; these forumlas are very simular...

xor dx, dx ; prepare dx:ax for operation

div WORD [bpbHeadsPerCylinder] ; mod by number of heads (Absolue head formula)

mov BYTE [absoluteHead], dl ; everything else was already done from the first formula

mov BYTE [absoluteTrack], al ; not much else to do :)

ret

Não é muito difícil, espero:)

**Load the cluster**

Ok, no carregamento do Stage 2, primeiro precisamos fazer referência ao conjunto da FAT. Muito simples. Em seguida, converter o número de cluster para LBA para que possamos lê-lo em:

mov ax, [cluster] ; cluster to read

pop bx ; buffer to read into

call ClusterLBA ; convert cluster to LBA

xor cx, cx

mov cl, [bpbSectorsPerCluster] ; sectors to read

call ReadSectors ; read in cluster

push bx

**Obtendo o próximo cluster**

Isto é complicado.

**Ok, lembre-se cada número de cluster na entrada do FAT é de 12 bits. Isto é um problema. Se lemos em 1 byte, estamos apenas copiando uma parte do número de cluster!**.

Devido a isso, temos que ler um WORD (2 bytes).

No entanto, mais uma vez, nos deparamos com um problema. Copiando 2 bytes (de um valor de 12 bits) significa que vamos copiar uma **parte da próxima entrada cluster**. Por exemplo, imagine que esta é a sua FAT:

Note: Binary numbers seperated in bytes.

Each 12 bit FAT cluster entry is displayed. Cada entrada de cluster FAT de 12 bits é apresentado

| |

01011101 0111010 01110101 00111101 0011101 0111010 0011110 0011110

| | | | | |

| |1st cluster | |3rd cluster-| |

|-0 cluster ----| |2nd cluster---| |4th cluster----|

**Observe, o mesmo Cluster ocupa todo o primeiro byte, mas parte da segundo. Além disso, observe que todos os clusters ímpares ocupa uma parte do seu primeiro byte, mas todos do segundo!**

Ok, então o que temos de fazer é ler 2 bytes (WORD) do FAT (Este é o nosso cluster).

Se é o mesmo cluster é ainda, **Máscaramos os 4 bits superiores, uma vez que pertence ao próximo cluster.**

Se for ímpar**, transferimos para baixo de 4 bits (para descartar os bits usados ​​pelo primeiro cluster.)** Por exemplo,...

; compute next cluster

mov ax, WORD [cluster] ; identify current cluster from FAT

;is the cluster odd or even? Just divide it by 2 and test! | é o conjunto par ou ímpar? Apenas dividi-lo por 2 e testar!

mov cx, ax ; copy current cluster

mov dx, ax ; copy current cluster

shr dx, 0x0001 ; divide by two

add cx, dx ; sum for (3/2)

mov bx, 0x0200 ; location of FAT in memory

add bx, cx ; index into FAT

mov dx, WORD [bx] ; read two bytes from FAT

test ax, 0x0001

jnz .ODD\_CLUSTER

; Remember that each entry in the FAT is a 12 but value. If it represents

; Lembre-se que cada entrada na FAT é um valor de 12 bits. Se ela representa

; a cluster (0x002 through 0xFEF) then we only want to get those 12 bits

; um cluster (0x002 através 0xFEF), então nós só queremos obter esses 12 bits

; that represent the next cluster

.EVEN\_CLUSTER:

and dx, 0000111111111111b ; take low twelve bits

jmp .DONE

.ODD\_CLUSTER:

shr dx, 0x0004 ; take high twelve bits

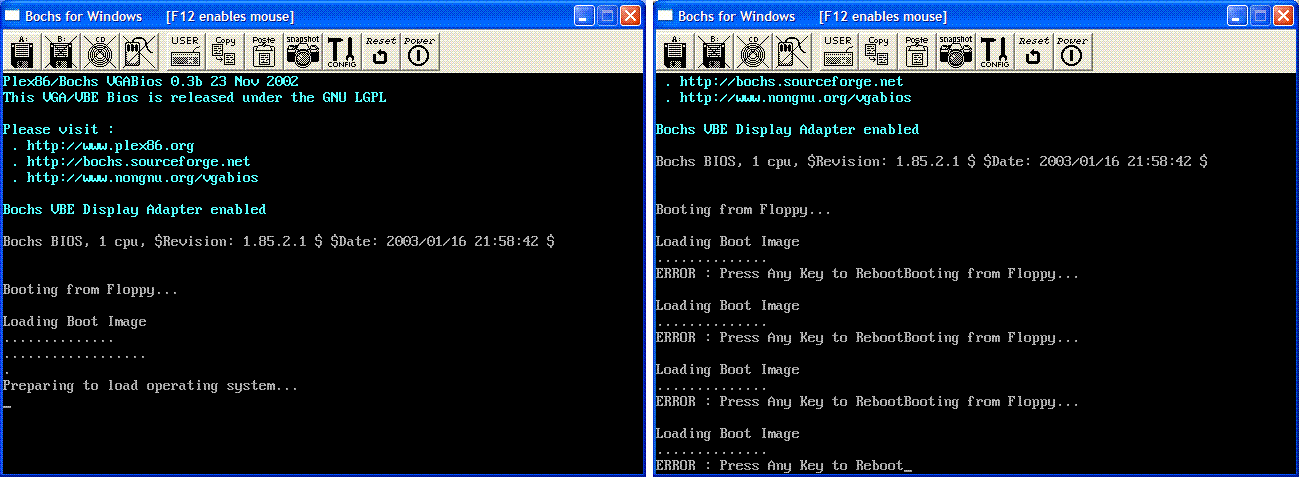
.DONE:

mov WORD [cluster], dx ; store new cluster

cmp dx, 0x0FF0 ; test for end of file

jb LOAD\_IMAGE ; we are not done yet--go to next cluster

**Demo**



O primeiro estagio é parte do bootloader que foi carregado com sucesso. O Stagio 2 imprime a mensagem carregamento do sistema operacional.

O segundo estagio exibe uma mensagem de erro quando ele não consegue encontra o arquivo Stage 2 (dentro do diretório raiz)..

Esta demo contém a maior parte do código nesta lição, dois arquivos de origem, 2 diretórios, e dois programas de lote. O primeiro diretório contém o programa de estágio 1 - o nosso bootloader, o segundo diretório contém o nosso programa de estágio 2 - STAGE2.SYS.

[DEMO DOWNLOAD HERE](http://www.brokenthorn.com/Resources/Demos/Demo1.zip)

**Conclusão**

Uau, este tutorial foi difícil de escrever. Simplesmente porque é difícil explicar um tema tão complexo em cada detalhe, enquanto ainda continuo tentando torná-lo muito fácil de seguir. Espero ter conseguido:)

Se você tem alguma sugestão que iria melhorar este tutorial, por favor me avise:)

Bem ... Eu acho que é isso: Adeus bootloader**!**

No próximo tutorial, vamos começar a construir no Palco 2. Vamos falar sobre A20, e olhar para o **modo protegido** com mais detalhes...

Nos vemos la!