

UNIVERSIDADE FEDERAL DE GOIÁS – UFG  
INSTITUTO DE INFORMÁTICA – INF

Arthur de Oliveira Barbosa Lacerda  
Murillo Rodrigues de Paula

**Relatório do desenvolvimento do sistema de arquivos FAT16**

Linguagens de Programação  
Prof. Dr. Bruno Oliveira Silvestre

Goiânia, 2017

## SUMÁRIO

<b>1. CONVENÇÕES DE CÓDIGO.....</b>	<b>2</b>
<b>2. ESTRUTURAS E TIPOLOGIAS UTILIZADAS .....</b>	<b>2</b>
<b>3. FUNÇÃO sector_read .....</b>	<b>3</b>
<b>4. IMPLEMENTAÇÃO FAT16 .....</b>	<b>4</b>
<b>4.1. Funções utilizadas .....</b>	<b>4</b>
<b>4.1.1. <i>path_treatment</i> .....</b>	<b>4</b>
<b>4.1.2. <i>fat16_init</i> .....</b>	<b>5</b>
<b>4.1.3. <i>find_root</i> .....</b>	<b>5</b>
<b>4.1.4. <i>fat_entry_by_cluser</i> .....</b>	<b>6</b>
<b>4.1.5. <i>find_subdir</i> .....</b>	<b>6</b>
<b>4.2. Funcionamento .....</b>	<b>7</b>
<b>5. FUSE .....</b>	<b>8</b>
<b>5.1. Funções necessárias do FUSE .....</b>	<b>8</b>
<b>5.2. Funcionamento .....</b>	<b>9</b>
<b>5.3. Reutilização das funções da FAT16 .....</b>	<b>9</b>
<b>5.4. Funções implementadas .....</b>	<b>9</b>
<b>5.4.1. <i>fat16_decode</i> .....</b>	<b>9</b>
<b>5.4.2. <i>fat16_init</i> .....</b>	<b>10</b>
<b>5.4.3. <i>fat16_getattr</i> .....</b>	<b>10</b>
<b>5.4.4. <i>fat16_readdir</i> .....</b>	<b>10</b>
<b>5.4.5. <i>fat16_read</i> .....</b>	<b>11</b>
<b>6. REFERÊNCIAS .....</b>	<b>11</b>

## 1. CONVENÇÕES DO CÓDIGO

- Todas as variáveis utilizadas foram nomeadas na língua inglesa e apresentam formato em camel case.
  - Ex: variableNameExample
- Toda função foi nomeada na língua inglesa, utilizando apenas letras minúsculas e tem o caracter “\_” como separador.
  - Ex: function\_example
- Todo o comentário do código também está na língua inglesa.
  - Padrão de comentário de funções:

```
/**
 * Description: This is the function description
 * =====
 * Return
 * @returnname: What the return means.
 * =====
 * Parameters
 * @param1name: What parameter 1 mean.
 * @param2name: What parameter 2 mean.
 */
```

- Padrão de comentário dentro das funções:

```
/* This is a single line comment */

/* This is the format of a multiple
 * line comment */
```

- Tamanho 2 de tabulação.

## 2. ESTRUTURAS E TIPOLOGIAS UTILIZADAS

Por questão de compatibilidade e uniformidade com a FAT16, todas os tipos são unsigned por padrão.

O tipo BYTE é definido como uma quantidade de 8 bits sem sinal.

O tipo WORD é definido como uma quantidade de 16 bits sem sinal.

O tipo DWORD é definido como uma quantidade de 32 bits sem sinal.

Segue abaixo às estruturas e definições utilizadas:

```
typedef uint8_t BYTE;
typedef uint16_t WORD;
typedef uint32_t DWORD;
```

```
typedef struct {
    BYTE BS_jmpBoot[3];
    BYTE BS_OEMName[8];
    WORD BPB_BytsPerSec;
```

```

    BYTE BPB_SecPerClus;
    WORD BPB_RsvdSecCnt;
    BYTE BPB_NumFATS;
    WORD BPB_RootEntCnt;
    WORD BPB_TotSec16;
    BYTE BPB_Media;
    WORD BPB_FATSz16;
    WORD BPB_SecPerTrk;
    WORD BPB_NumHeads;
    DWORD BPB_HiddSec;
    DWORD BPB_TotSec32;
    BYTE BS_DrvNum;
    BYTE BS_Reserved1;
    BYTE BS_BootSig;
    DWORD BS_VollID;
    BYTE BS_VollLab[11];
    BYTE BS_FilSysType[8];
    BYTE Reserved[448];
    WORD Signature_word;
} __attribute__((packed)) BPB_BS;

typedef struct {
    BYTE DIR_Name[11];
    BYTE DIR_Attr;
    BYTE DIR_NTRes;
    BYTE DIR_CrtTimeTenth;
    WORD DIR_CrtTime;
    WORD DIR_CrtDate;
    WORD DIR_LstAccDate;
    WORD DIR_FstClusHI;
    WORD DIR_WrtTime;
    WORD DIR_WrtDate;
    WORD DIR_FstClusLO;
    DWORD DIR_FileSize;
} __attribute__((packed)) DIR_ENTRY;

typedef struct {
    DWORD FirstRootDirSecNum;
    DWORD FirstDataSector;
    BYTE *Fat;
    BPB_BS Bpb;
} VOLUME;

```

### 3. FUNÇÃO `sector_read`

A função `sector_read` à ser implementada em `sector.c` simplesmente deveria acessar o posicionamento da FAT16 referente ao parâmetro `secnum` (número do setor) e efetuar a leitura.

Para isso foi necessário apenas utilizar a função *fseek* de tal forma que o descritor, à partir do `SEEK_SET` (Ponto inicial da FAT16), desse um salto de  $512 \times \text{secnum}$ , e logo após a leitura dos próximos 512 bytes seria efetuada e armazenada no buffer.

Segue abaixo a implementação de *sector\_read*.

```
void sector_read(FILE *fd, unsigned int secnum, void *buffer)
{
    fseek(fd, BYTES_PER_SECTOR * secnum, SEEK_SET);
    fread(buffer, BYTES_PER_SECTOR, 1, fd);
}
```

## 4. IMPLEMENTAÇÃO FAT16

Primeiramente escrevemos o código `run_fat16.c` afim de entender e efetuar o funcionamento da FAT16.

O programa recebe como parâmetro a imagem da FAT16 e o caminho à ser percorrido, e é apresentado na saída se o caminho foi ou não percorrido com sucesso na imagem.

Para compilar e executar o programa, deve-se utilizar os comandos à seguir:

```
gcc run_fat16.c sector.c -o run_fat16
./run_fat16 <Imagem FAT16> <Diretório>
```

### 4.1. Funções utilizadas

#### 4.1.1. *path\_treatment*

retorno:

`char** pathFormatted`: Vetor de strings com cada string referente à um arquivo do caminho em uma posição, na devida formatação da FAT16.

parâmetros:

1. `char* pathInput` : String dada de entrada ao programa com o caminho de diretórios.
2. `int* pathSz`: Endereço da variável que armazenará a quantidade de arquivos presentes no caminho.

Descrição:

A função trata a string dada de entrada adaptando para o formato FAT16.

O caminho dado de entrada (`pathInput`) é dividido em um vetor de strings (`path`) que eram anteriormente delimitadas pelo token `"/"`, a função *strtok* da biblioteca *string.h* é utilizada para este fim.

Após esse procedimento, é criado um vetor de strings (`pathFormatted`), cada uma com tamanho 11, que armazenará o nome de cada arquivo no formato da FAT16.

O tratamento das strings então é realizada, de forma que a informação passada de `path` seja transferida para `pathFormatted` com as devidas alterações, sendo estas que

aceite apenas os caracteres aceitos pela especificação, troque caracteres em caixa baixa por caixa alta, que os 8 primeiros espaços das string sejam referentes ao nome do arquivo e os três últimos à extensão e qualquer espaço inutilizado é preenchido com o caracter de espaço “ ”, também é tratado o caso em que os arquivos possam ser “.” ou “..”.

#### **4.1.2. fat16\_init**

retorno:

VOLUME\* Vol: Estrutura com dados essenciais da FAT16

parâmetros:

1. FILE\* fd: Descritor de arquivo.

descrição:

A leitura do primeiro setor é efetuada, que é referente ao setor BPB, e o grava no volume (Vol->Bpb).

A posição do primeiro setor do diretório raiz (Vol->FirstRootDirSecNum) é calculado, tal calculo é feito baseado no salto da área reservada (BPB\_RsvdSecCnt), acrescido do tamanho de uma FAT (BPB\_FATSz16) multiplicado pela quantidade de FATs (BPB\_NumFATS).

O número de setores do diretório raiz então é calculado baseado na quantidade de entradas do diretório raiz. Com esse valor em memória, calcula-se então a posição do primeiro setor da região de dados (Vol->FirstDataSector), que é dada pelo primeiro setor do diretório raiz acrescido do número de setores do diretório raiz.

#### **4.1.3. find\_root**

retorno:

void

parâmetros:

1. FILE\* fd: Descritor do arquivo.
2. VOLUME Vol: Estrutura com dados essenciais da FAT16.
3. DIR\_ENTRY Root: Variável que armazenará entradas de diretório situadas no diretório raiz.
4. char \*\*path: Vetor de arquivos do caminho à ser percorrido.
5. int pathDepth: Profundidade (ou índice) do caminho.
6. int pathSize: Tamanho total do caminho.

descrição:

A função percorre as entradas de diretório raiz buscando o arquivo da atual profundidade do caminho.

Com a função *sector\_read*, lê-se todas às entradas de diretórios da root, até que encontre o arquivo especificado ou até que chegue em seu fim.

A comparação de string é feita caracter por caracter entre o path e Root.DIR\_Name.

Se o arquivo for encontrado e for o último do caminho, então ele é apresentado, se for encontrado e não for o último, então a função *find\_subdir* é chamada incrementando a profundidade do caminho.

#### 4.1.4. fat\_entry\_by\_cluster

retorno:

void

parâmetros:

1. FILE\* fd: Descritor do arquivo.
2. VOLUME\* Vol: Estrutura com dados essenciais da FAT16.
3. WORD ClusterN: Cluster em que será usado como base para obter a entrada da FAT.

Descrição:

Dado um cluster N, é determinada a entrada da FAT.

Primeiramente define-se o FATOffset da FAT como  $\text{ClusterN} * 2$ , já que uma entrada da fat tem 2 bytes (16bits).

Calcula-se então o número de setor da FAT em que a leitura deve ser realizada, saltando a área reservada acrescida do número de setores que o offset da fat passa ( $\text{FATOFFSET} / \text{Vol} \rightarrow \text{Bpb.BPB\_BytsPerSec}$ ).

Calcula-se então o offset de entrada, de acordo com o resto da divisão do FATOffset com  $\text{Vol} \rightarrow \text{Bpb.BPB\_BytsPerSec}$ .

Lê-se o setor correspondente à ao numero do setor calculado e retorna o valor encontrado no offset de entrada.

#### 4.1.5. find\_subdir

retorno:

void

parâmetros:

1. FILE\* fd: Descritor do arquivo.
2. VOLUME Vol: Estrutura com dados essenciais da FAT16.
3. DIR\_ENTRY Dir Variável que armazenará entradas de diretório situadas no subdiretório.
4. char\*\* path
5. int pathSize: Vetor de arquivos do caminho à ser percorrido.

6. int pathDepth: Profundidade (ou índice) do caminho.
7. int rootDepth: Profundidade atual do caminho em relação ao diretório raiz.

descrição:

A função percorre as entradas do subdiretório buscando o arquivo da atual profundidade do caminho.

Com a função *fat\_entry\_by\_cluster* determina-se a entrada da FAT para qual o subdiretório terá continuidade na busca. Caso o cluster seja o último então terá valor 0xFFFF.

Com a função *sector\_read*, lê-se todas as entradas do subdiretório, até que encontre o arquivo especificado ou até que chegue ao fim do cluster.

Se chegar ao fim do cluster, o valor do cluster atual passa a ser a entrada da FAT anterior, e então se recalcula a próxima entrada da FAT, reiniciando o processo e dando continuidade à leitura do subdiretório.

A comparação de string é feita caracter por caracter entre o path e Dir.DIR\_Name.

Se o arquivo for encontrado e for o último do caminho, então ele é apresentado, se for encontrado e não for o último, então a função *find\_subdir* é chamada recursivamente incrementando a profundidade do caminho, além disso, se o arquivo da profundidade atual for ".", a profundidade em relação ao diretório raiz não muda, e se for ".." a profundidade em relação ao diretório raiz diminui. Se chegar ao ponto de que a profundidade em relação ao diretório raiz chegar a zero, então em vez de chamar a função *find\_subdir*, chamaremos *find\_root* para retornar ao diretório raiz e continuar a percorrer o caminho..

## 4.2. Funcionamento

Na função main do programa desenvolvido, primeiramente abre-se a imagem passada como primeiro parâmetro, e logo após, trata-se a string passada como segundo parâmetro com a função *path\_treatment*, inicializa-se então o Volume com a função *fat16\_init*, afim de armazenar o BPB e determinar os setores de início do diretório raiz e da área de dados, e então chama-se a função de busca *find\_root* que iniciará a busca pela primeira profundidade do caminho, que chamará *find\_subdir* ao encontrar o primeiro arquivo, e entrará em recursão até encontrar o último (ou voltará a *find\_root* caso a profundidade relativa ao diretório raiz seja zero. Ao encontrar o arquivo final do caminho, ele é mostrado na tela, se o arquivo em alguma profundidade não for encontrado ao varrer o subdiretório (ou diretório raiz) atual, então uma mensagem é mostrada na tela informando que não há tal arquivo.

Abordando em um nível estrutural, primeiramente é lido o primeiro setor da FAT16, e então salta-se para o primeiro setor diretório raiz, e então busca-se entrada por entrada, àquela que seja correspondente ao arquivo encontrado, então move-se para a região de dados, e busca a entrada do próximo cluster na região da fat, e assim segue-se a leitura, fazendo consultas entra a região de dados e a fat, e ocasionalmente dependendo do caminho, poderá voltar à região do diretório raiz.



## 5. FUSE

O FUSE (Filesystem in USErspace) é uma ferramenta que permite ao usuário criar seus próprios sistemas de arquivo sem que seja necessário editar o modo kernel.

Para nosso trabalho, implementamos o sistema de arquivo FAT16 no FUSE, e para isso, é necessário implementar funções que o FUSE requisita ao usuário ao fazer a comunicação do nível usuário ao kernel.

### 5.1. Funções necessárias do FUSE

*init* inicializa o sistema de arquivos

```
void>(* fuse_operations::init)(struct fuse_conn_info *conn, struct fuse_config *cfg)
```

*getattr* é responsável por obter os atributos do arquivo.

```
int(* fuse_operations::getattr)(const char *, struct stat *, struct fuse_file_info *fi)
```

*readdir* lê um diretório

```
int(* fuse_operations::readdir)(const char *, void *, fuse_fill_dir_t, off_t, struct fuse_file_info *, enum fuse_readdir_flags)
```

*read* lê os dados de um arquivo aberto

```
int(* fuse_operations::read)(const char *, char *, size_t, off_t, struct fuse_file_info *)
```

*destroy* limpa o sistema de arquivos, chamado quando sair do sistema de arquivos.

```
void(* fuse_operations::destroy)(void *)
```

Em nosso trabalho usamos a nomenclatura da função, utilizando seu próprio nome com o prefixo “fat16”.

Ex: para a função *readdir* implementamos *fat16\_readdir*.

Para anexar estas funções implementadas ao fuse, devermos então atribuí-las à estrutura *fuse\_operations* como mostrado à seguir:

```
struct fuse_operations fat16_oper = {  
    .init      = fat16_init,  
    .getattr   = fat16_getattr,  
    .readdir   = fat16_readdir,  
    .read      = fat16_read,  
    .destroy   = fat16_destroy  
};
```

## 5.2. Funcionamento

Para compilar e executar o programa, utiliza-se os seguintes comandos:

```
make
./mount_fat16 <diretório de montagem>
```

Ao executar estes comandos, a FUSE chama a função *init* primeiramente, inicializando o sistema de arquivo, logo depois, o *getattr*, obtendo os atributos do diretório raiz.

Ao efetuar uma listagem com o comando *ls*, será chamada a função *readdir* que percorrerá as entradas do diretório, listando cada entrada de diretório correspondente à arquivos e diretórios encontrados. Quanto ao *readdir*, ele executa diferentemente no diretório raiz e em outros diretórios, já que no diretório raiz, ele deve percorre-lo acessando apenas a região reservada, e em outros diretórios, que se encontram na região de dados, deverão fazer acesso a FAT para consultar se há mais informação a ser lida ou se o último cluster é o que está sendo lido.

Para efetuar uma cópia com o comando *cp* de dentro da imagem para fora, deve-se encontrar o diretório com *find\_root*, e então ler o seu conteúdo com a função *read*, e assim os dados já estarão em buffer e serão transferidos para o destino, onde o FUSE fará a comunicação com o kernel para que a escrita seja feita no outro sistema de arquivos.

No fechamento do sistema de arquivo, deverá ser chamada a função *destroy*.

## 5.3. Reutilização das funções da FAT16

As funções *find\_root*, *find\_subdir* e *path\_treatment* foram reutilizadas na implementação do FUSE, porém com algumas alterações:

*path\_treatment* não precisa mais verificar se a entrada é válida, apenas realiza o tratamento para o formato padrão.

*find\_root* e *find\_subdir* não imprimem mais o diretório, mas o armazenam, passando um ponteiro *DIR\_ENTRY* como parâmetro, e retornam 0 em caso de sucesso e 1 em caso de fracasso.

*fat\_entry\_by\_cluser* foi reutilizada sem alterações.

## 5.4. Funções Implementadas

### 5.4.1. path\_decode

retorno:

char \*pathDecoded: Nome do arquivo no formato comum, com nome seguindo de ponto e extensão em letras minúsculas.

parâmetros:

1. path: Nome do arquivo no formato FAT16

descrição:

A função passa o formato de nomenclatura da FAT para letras minúsculas e nome e extensão separados por ponto.

Ex: "ARQ        TXT" para "arq.txt"

#### 5.4.2. fat16\_init

retorno:

void\* context->private\_data: ponteiro para os dados privados retornados pela função init.

parâmetros:

1. struct fuse\_conn\_info \*conn: fornece informação sobre quais características tem suporte pelo FUSE

descrição:

realiza uma preparação inicial de única vez e recebe o contexto.

#### 5.4.3. fat16\_getattr

retorno:

retorna o valor 0 no fim da função.

parâmetros:

1. const char \*path: caminho do arquivo que os atributos devem ser obtidos.
2. struct stat \*stbuf: estrutura que armazena os atributos do arquivo.

descrição:

A função ao receber o caminho, verifica primeiramente se o path é "/", ou seja, se o arquivo à obter os atributos é o diretório raiz, se sim, atribui-se os devidos valores, se não, realiza o *path\_treatment*, e então chama *find\_root* que retornará a entrada de diretório correspondente ao path, e então os atributos são obtidos à partir da entrada de diretório.

#### 5.4.4. fat16\_readdir

retorno:

retorna o valor 0 no fim da função.

parâmetros:

1. const char \*path: caminho do diretório a ser lido.
2. void \*buffer: responsável para passar as informações para a FUSE.
3. fuse\_fill\_dir\_t filler: preenche o buffer com os nomes dos arquivos do diretório.

4. `off_t` `offset`: não utilizado
5. `struct fuse_file_info` `*fi`: não utilizado

descrição:

Se o `path` for `"/"`, ele percorrerá o diretório raiz e passará para o `filler` todo nome de diretório ou arquivo que forem encontrados em entradas do diretório raiz.

Se o `path` não for `"/"`, o `path` passará pelo `path_treatment` e percorrerá com `find_root` o `path` dado de entrada e assim obtém a entrada de diretório destino. Com a entrada de diretório, encontra-se o primeiro cluster, e inicia a leitura das entradas, chamando o `filler` para cada uma encontrada (em cada nome, usa-se `path_decode`), se tiver mais de um cluster, a cada finalização de leitura de cluster, a entrada de fat é acessada para obter o novo cluster e continuar a leitura, até que a entrada da fat indique que não há mais clusters a serem lidos para aquele determinado diretório.

#### 5.4.5. `fat16_read`

retorno:

`int DIR_FileSize`: Tamanho do arquivo à ser lido

parâmetros:

1. `const char *path`: caminho do arquivo à ser lido.
2. `char *buffer`: buffer que guardará os dados a serem gravados.
3. `size_t size`: tamanho do buffer em bytes.
4. `off_t offset`: offset de leitura.
5. `struct fuse_file_info` `*fi`: não utilizado

descrição:

Com o `path`, usa-se `find_root` para encontrar o arquivo, e então acessa-se seu primeiro setor do cluster, a cada chamada `read`, `sector_read` é chamado (`size/BytesPerSector`) vezes, até que preencha o buffer, ou até que não tenha mais conteúdo à ser lido.

A cada chamada de `read`, lê-se `size` bytes do arquivo encontrado em `path`. Os `size` bytes são gravados no buffer.

Com o `offset`, nas chamadas subsequentes o arquivo continuará a ser lido do seu ponto de interrupção.

## 6. REFERÊNCIAS

- `libfuse`: `fuse_operations` Struct Reference  
([http://libfuse.github.io/doxygen/structfuse\\_\\_operations.html](http://libfuse.github.io/doxygen/structfuse__operations.html))
- Write a filesystem with FUSE  
(<https://engineering.facile.it/blog/eng/write-filesystem-fuse/>)

- Microsoft FAT Specification, Microsoft Corporation, August 30 2005
- CS135 FUSE Documentation  
([https://www.cs.hmc.edu/~geoff/classes/hmc.cs135.201001/homework/fuse/fuse\\_doc.html](https://www.cs.hmc.edu/~geoff/classes/hmc.cs135.201001/homework/fuse/fuse_doc.html))