# Implementação de sistema de arquivos utilizando FUSE

## S.A.D. File System

Arthur Alexsander Martins Teodoro  Davi Ribeiro Militani  Samuel Terra Vieira

Departamento de Ciência da Computação
Universidade Federal de Lavras

Professor: Prof. Tales Heimfarth

# Sumário

# Sumário

- Forma de organização de dados em algum meio de armazenamento;
- Armazenamento, recuperação, atribuição de nomes, compartilhamento e proteção de arquivos [Bzoch and Safarik, 2011];

- Arquivos:
  - Dados;
  - **Atributos:** Tamanho do arquivo, tipo de arquivo, identidade do proprietário e listas de controle de acesso;
- Diretório(atribuição de nomes):
  - Mapeamento dos nomes textuais para identificadores internos;
  - Atribuição hierárquica de nomes *(pathname)*;

- Controle do acesso aos arquivos
  - Chamada de sistema *Open*
  - Verifica os direitos permitidos à identidade do usuário.

- O sistema operacional precisa "conhecer" o sistema de arquivos para:
  - exibir conteúdo;
  - abrir arquivos;
  - salvar arquivos.
- O S.O. precisa "entender" o sistema de arquivos para que ele possa realizar as operações operações;
- Necessário para a troca de dados entre diferentes sistemas operacionais;
- Se o S.O. não "entender" o sistema de arquivos, basta instalar um *driver* de sistema de arquivos que forneça suporte, caso contrário não poderá usar o sistema de arquivos.

- Desenvolver um Sistema de Arquivos utilizando a biblioteca libfuse
- Armazenamento de arquivos em um cartão SD em um Arduino
- Cartão localizado em um Arduino que não esteja diretamente ligado ao computador onde o sistema de arquivo foi implementado.

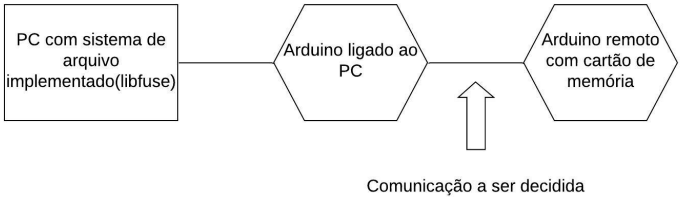


Figure: Objetivo do trabalho (elaborado pelo autor).

# Sumário

- *File Allocation Table* (FAT) é um sistema de arquivos simples originalmente desenvolvido para pequenos discos;
- Disco FAT é alocado em *clusters*, cujo tamanho é determinado pelo tamanho do volume.;
- Os arquivos vão para o primeiro local aberto na unidade;

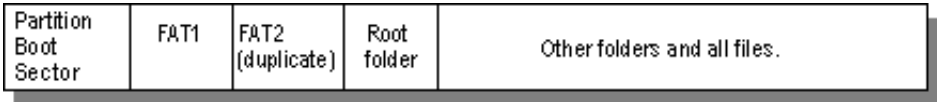


Figure: Organização do disco usando FAT FS.

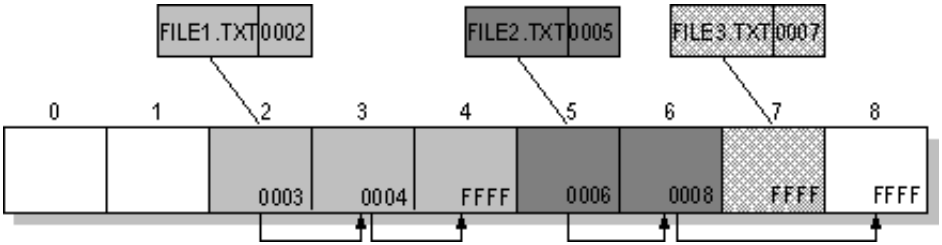


Figure: Funcionamento da tabela FAT (Adaptado de [Bzoch and Safarik, 2011])

- À medida que o tamanho do volume aumenta, o desempenho do FAT diminui rapidamente;
- Partições FAT são limitadas em tamanho para um máximo de 4 GB;
- Não é possível definir permissões nos arquivos.

# Sumário

## Componentes

- Ambiente:
  - Linux, distribuição Ubuntu 18.04;
  - wxHexEditor;
  - Arduino IDE.
- Dependências necessárias:
  - Cmake;
  - make;
  - FUSE $\geq$ 2.6;
  - FUSE development files (libfuse-dev);
  - GCC ou Clang.
- Hardware utilizado:
  - Arduino Uno;
  - Leitor de cartão micro-SD.

## Instalação das dependências

**sudo apt install gcc make cmake fuse libfuse-dev**

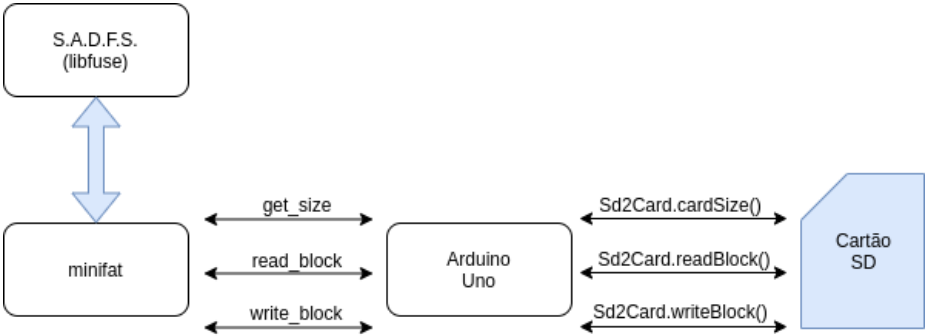


Figure: Fluxo de comunicação (elaborado pelo autor).

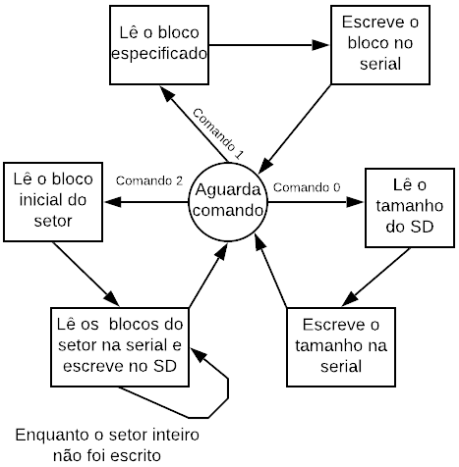


Figure: Fluxo de comunicação (elaborado pelo autor).

# Sumário

- Versão simplificada do FAT FS;
- Faz uso de uma tabela para definir quais setores podem ser utilizados ou não;
- É capaz de criar e remover arquivos e diretórios;
- Nos arquivos, é capaz de escrever e alocar dinamicamente espaço para um arquivo, além de os ler;
- Além disso, implementa algumas questões que o FAT não implementa, como permissões dos arquivos;
- Possui algumas limitações, para tornar a implementação mais simples, como um limite na quantidade de arquivos e sub-diretórios por diretório;

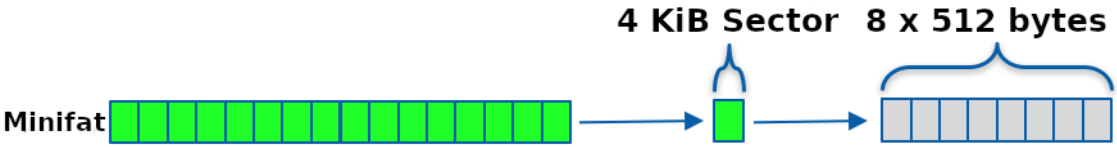


Figure: Organização dos blocos do *minifat.h* (elaborado pelo autor).

| dir_entry_t |
| --- |
| **name:** char[64]; |
| **uid:** uid_t; |
| **gid:** gid_t; |
| **mode:** mode_t; |
| **create:** date_t; |
| **update:** date_t; |
| **size:** uint32_t; |
| **first_block:** uint32_t; |

| info_entry_t |
| --- |
| **total_blocks:** uint32_t |
| **available_blocks:** uint32_t; |
| **block_size:** uint32_t; |
| **block_per_sector:** uint32_t; |
| **sector_per_fat:** uint32_t; |
| **dir_entry_number:** uint32_t; |

| dir_descriptor_t |
| --- |
| **dir_infos:** dir_entry_t; |
| **entry:** char[SECTOR_SIZE]; |

| date_format_t |
| --- |
| **day:** unsigned int:6; |
| **month:** unsigned int:5; |
| **year:** unsigned int:13; |
| **hour:** unsigned int:6; |
| **minutes:** unsigned int:7; |
| **seconds:** unsigned int:7; |

Figure: Estruturas usadas pelo Mini-FAT (elaborado pelo autor).

## Principais funções implementas para Mini-FAT

- **write_sector** → Escreve setor no SD;
- **read_sector** → Lê setor no SD;
- **create_empty_file** → Cria um arquivo vazio;
- **create_empty_dir** → Cria um diretório vazio;
- **write_file** → Escreve no arquivo;
- **read_file** → Lê no arquivo;
- **delete_file** → Deleta o arquivo;
- **delete_dir** → Deleta o diretório;
- **resize_file** → Muda tamanho do arquivo.

### *Write sector*

```
void write_sector(uint32_t sector, void *buffer) {
        uint32_t first_block = sector *
                                    (SECTOR_SIZE / BLOCK_SIZE);
        for (int i = 0; i < SECTOR_SIZE / BLOCK_SIZE; i++) {
                write_block(first_block + i, buffer +
                                    (i * BLOCK_SIZE));
        }
}
```

*Read sector*

```
void read_sector(uint32_t sector, void *buffer) {
        uint32_t first_block = sector *
                                    (SECTOR_SIZE / BLOCK_SIZE);
        for (int i = 0; i < SECTOR_SIZE / BLOCK_SIZE; i++) {
                read_block(first_block + i, buffer +
                                        (i * BLOCK_SIZE));
        }
}
```

### *Create empty file*

```
int create_empty_file(dir_entry_t *dir, dir_entry_t *dir_entry_list,
                      info_entry_t *info, fat_entry_t *fat,
                      const char *name, mode_t mode, uid_t uid,
                      gid_t gid);
```

- **dir_entry_t* dir:** Entrada do diretório pai;
- **dir_entry_t* dir_entry_list:** Lista de entradas do diretório pai;
- **info_entry_t* info:** Informações do cartão;
- **fat_entry* fat:** Tabela FAT;
- **char* name:** Nome do arquivo;
- **mode_t mode:** Modo do arquivo (Permissões e tipo)
- **uid_t uid:** UID do dono do arquivo;
- **gid_t uid:** GID em que o dono do arquivo está;

## *Create empty file*

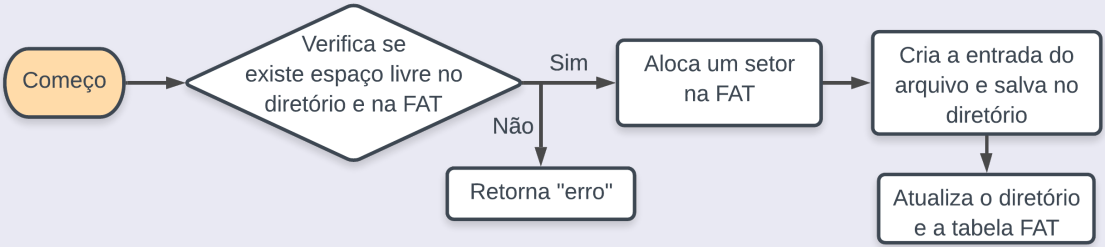


Figure: Elaborado pelo autor.

## *Create empty dir*

```
int create_empty_dir(dir_entry_t *dir, info_entry_t *info,
                     dir_entry_t *dir_entry_list,
                     fat_entry_t *fat, const char *name, mode_t mode,
                     uid_t uid, gid_t gid);
```

- **dir_entry_t* dir:** Entrada do diretório pai;
- **dir_entry_t* dir_entry_list:** Lista de entradas do diretório pai;
- **info_entry_t* info:** Informações do cartão;
- **fat_entry* fat:** Tabela FAT;
- **char* name:** Nome do diretório;
- **mode_t mode:** Modo do diretório (Permissões e tipo)
- **uid_t uid:** UID do dono do diretório;
- **gid_t uid:** GID em que o diretório do diretório está;

## *Write file*

```
int write_file(fat_entry_t *fat, const info_entry_t *info,
               dir_entry_t *dir, dir_entry_t *dir_entry_list,
               dir_entry_t *file, int offset, const char *buffer,
               int size);
```

- **fat_entry_t* fat:** Tabela FAT;
- **info_entry_t* info:** Informações sobre o SD;
- **dir_entry_t* dir:** Entrada do diretório pai;
- **dir_entry_t* dir_entry_list:** Lista de entradas do diretório pai;
- **dir_entry_t* file:** Entrada do arquivo no diretório pai;
- **int offset:** *Offset* para escrita no arquivo;
- **char* buffer:** Buffer com os valores a serem escritos;
- **int size:** Quantidade de *bytes* a escrever;

## *Write File*

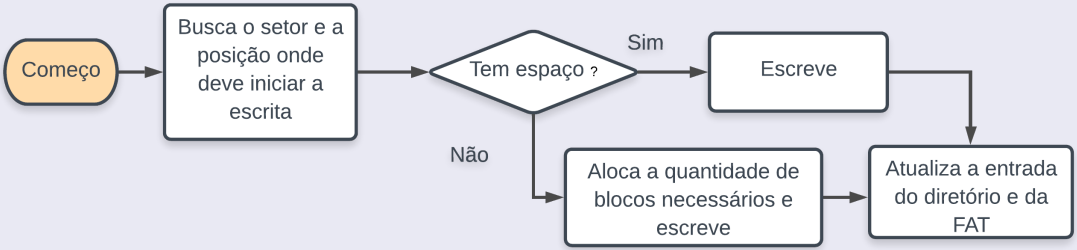


Figure: Elaborado pelo autor.

### *Read file*

```
int read_file(const fat_entry_t *fat, const info_entry_t *info,
              dir_entry_t *file, int offset, char *buffer,
                                 unsigned int size);
```

- **fat_entry_t* fat:** Tabela FAT;
- **info_entry_t* info:** Informações do FS;
- **dir_entry_t* file:** Entrada do arquivo;
- **int offset:** *Byte* de início de leitura;
- **char* buffer**: *Buffer* para leitura;
- **int size:** Tamanho do *buffer*;

## *Read File*

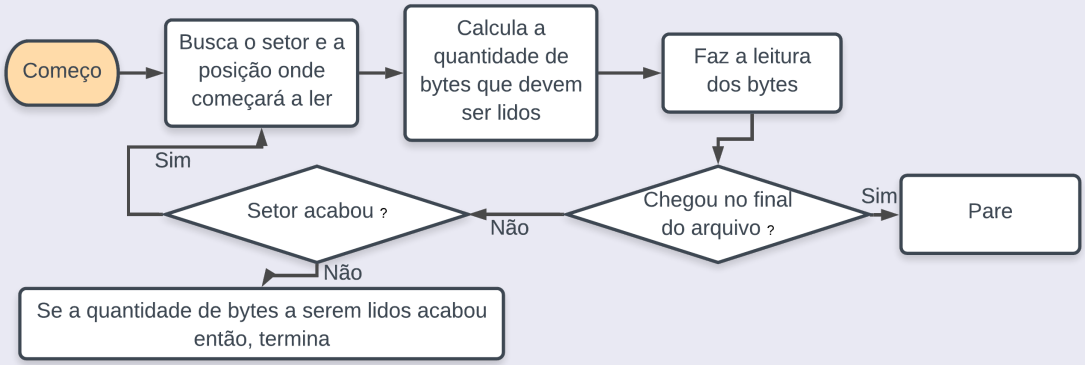


Figure: Elaborado pelo autor.

## *Delete file*

```
int delete_file(fat_entry_t *fat, const info_entry_t *info,
                dir_entry_t *dir, dir_entry_t *dir_entry_list,
                                  dir_entry_t *file);
```

- **fat_entry_t* fat:** Tabela FAT;
- **info_entry_t* info:** Informações do FS;
- **dir_entry_t* dir:** Entrada do diretório pai;
- **dir_entry_t* dir_entry_list:** Lista de entradas do diretório pai;
- **dir_entry_t* file:** Entrada do arquivo;

## *Delete File*

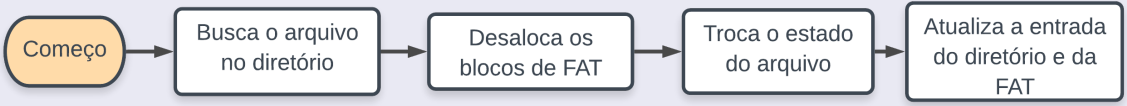


Figure: Elaborado pelo autor.

### *Delete dir*

```
int delete_dir(fat_entry_t *fat, const info_entry_t *info,
               dir_entry_t *father_dir, dir_entry_t *dir_entry_list,
                                        dir_entry_t *dir);
```

- **fat_entry_t* fat:** Tabela FAT;
- **info_entry_t* info:** Informações do FS;
- **dir_entry_t* father_dir:** Entrada do diretório pai;
- **dir_entry_t* dir_entry_list:** Lista de entradas do diretório pai;
- **dir_entry_t* dir:** Entrada do diretório;

### *Resize file*

```
int resize_file(fat_entry_t *fat, const info_entry_t *info,
                dir_entry_t *dir, dir_entry_t *dir_entry_list,
                dir_entry_t *file, int new_size);
```

- **fat_entry_t* fat:** Tabela FAT;
- **info_entry_t* info:** Informações do FS;
- **dir_entry_t* dir:** Entrada do diretório pai;
- **dir_entry_t* dir_entry_list:** Lista de entradas do diretório pai;
- **dir_entry_t* dir:** Entrada do diretório pai;
- **int new_size:** Novo tamanho do arquivo;

## *Resize File*

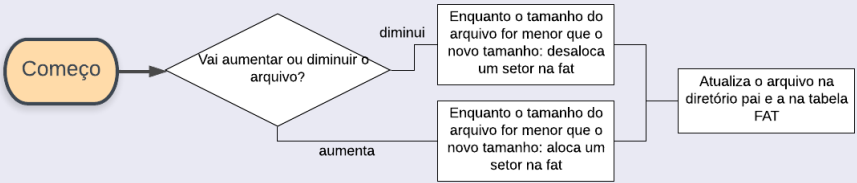


Figure: Elaborado pelo autor.

# Mini-FAT

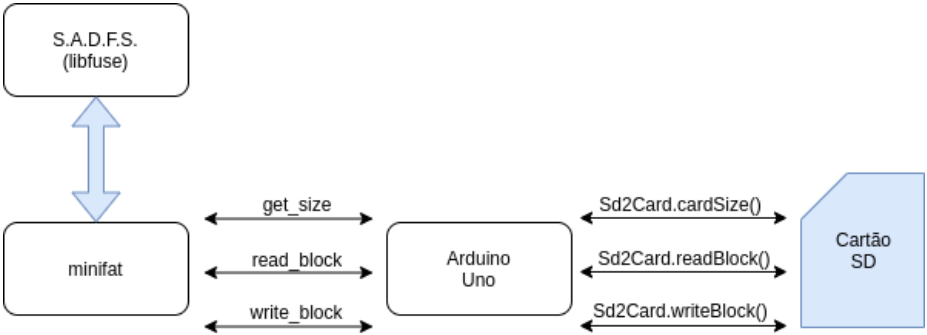


Figure: Fluxo de comunicação (elaborado pelo autor).

# Sumário

- FUSE - **F**ilesystem in **USE**rspace é a estrutura do sistema de arquivos de espaço do usuário mais amplamente utilizada [Libfuse, 2019];
- O projeto FUSE consiste em dois componentes: fuse *kernel module* e a *libfuse userspace library*.
- Possui suporte para linguagens:
  - Go
  - NodeJS
  - Python
  - Java
  - C

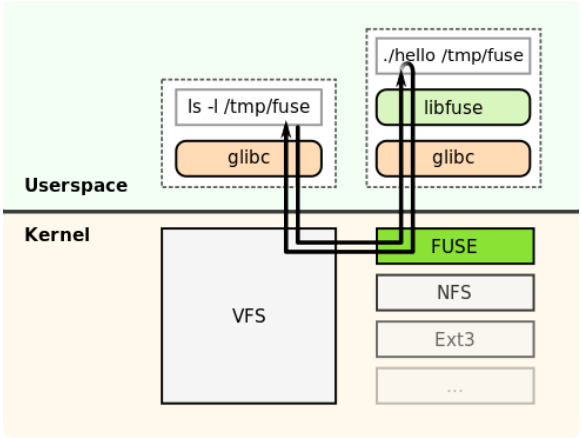


Figure: Fluxo da Libfuse (adaptado de [Fontana, 2016]).

# Sumário

Samuel Arthur Davi (S.A.D) File System

```
SAD-FileSystem
├── bin
│   └── SAD
├── cache
│   ├── cache.c
│   └── cache.h
├── minifat
│   ├── minifat.c
│   └── minifat.h
├── serial
│   ├── serial.c
│   └── serial.h
├── CMakeLists.txt
└── main.c
```

- **SAD-FileSystem:** Pasta *root* do projeto
- **bin:** Pasta para o binário;
- **cache:** Implementação para a cache;
- **minifat:** Implementação da mini-FAT;
- **serial:** Implementação para comunicação serial;
- **CMakeLists.txt:** "Roteiro" para a compilação do CMake;
- **main.c:** Implementação do sistema de arq. utilizando libfuse;

- Verificar dependências, configurar o ambiente e gerar Makefiles:
  - **cmake -DCMAKE_BUILD_TYPE=Debug .**
- Para 'buildar' o projeto:
  - **make -j**

- Crie um diretório para montagem, por exemplo:
  - **mkdir /tmp/example**

- Monte o sistema de arquivos:
  - **./bin/SAD -d -s -f /tmp/example**
- Argumentos:
  - **-d**: enable debugging
  - **-s**: run single threaded
  - **-f**: stay in foreground

- Verifique se ocorreu a montagem com sucesso:
  - **ls -al /tmp/example/**
  - **mount — grep SAD**

- Caso deseje desmontar o sistema de arquivos execute:
  - **fusermount -u /tmp/example**

## Principais funções implementas para o S.A.D.

- sad_getattr
- sad_readdir
- sad_mkdir
- sad_mknod
- sad_read
- sad_write
- sad_unlink
- sad_rmdir
- sad_rename
- sad_truncate

## Get atributes

```
static int sad_getattr(const char *path, struct stat *st);
```

- Responsável por ler os atributos de um arquivo e diretório;
- Recebe como parâmetro o nome do arquivo;
- Altera o parâmetro *st* com as informações;
- A *struct stat* st* é a mesma usada na chamada de sistema *stat()* e *get_dents()*
- Chama a função de buscar arquivo e diretório do Mini-FAT.

## Read directory

```
static int sad_readdir(const char *path, void *buffer,
                       fuse_fill_dir_t filler, off_t offset,
                       struct fuse_file_info *fi);
```

- Função responsável por ler as entradas de um diretório (usadas no *ls* por exemplo);
- *path* é o caminho do diretório a ser lido;
- *buffer* é onde as entradas devem ser salvas;
- *filler* é uma função da *libfuse* para fazer o empacotamento das entradas;
  - `filler(buffer, ".", NULL, 0);`
  - Escreve a entrada do próprio diretório.
- Chama a função de buscar entradas do diretório do Mini-FAT.

## Create directory

```
int sad_mkdir(const char *path, mode_t mode);
```

- Responsável por criar um novo diretório;
- Recebe o caminho onde o diretório deve ser criado;
- Além disso, recebe o modo em que o diretório é criado;
- É importante lembrar que o modo é usado para diferenciar um arquivo de um diretório;
- Chama a função `create_empty_dir()` do Mini-FAT.

## Make node

```
int sad_mknod(const char *path, mode_t mode, dev_t dev);
```

- Responsável por criar um novo arquivo;
- Recebe o caminho onde o novo arquivo deve ser criado;
- Além disso, recebe o modo em que o arquivo é criado;
- Chama a função `create_empty_file()` do Mini-FAT.

## Read file

```
static int sad_read(const char *path, char *buffer, size_t size,
                    off_t offset, struct fuse_file_info *fi);
```

- Implementa a função de leitura de um arquivo;
- Recebe o caminho do arquivo que deve ser lido;
- Também recebe o *buffer* onde os dados do arquivo devem ser salvos;
- O tamanho do *buffer* também é recebido;
- O *offset* é a quantidade de *bytes* que deve ser pulados;
- Deve retornar a quantidade de *bytes* lidos;
- Faz chamada da função `read_file()`.

## Write

```
int sad_write(const char *path, const char *buffer, size_t size,
              off_t offset, struct fuse_file_info *fi);
```

- Implementa a função de escrita de um arquivo;
- Recebe o caminho do arquivo que deve ser escrito;
- Também recebe o *buffer* onde os dados do arquivo devem ser lidos;
- O tamanho do *buffer* também é recebido;
- O *offset* é a quantidade de *bytes* que deve ser pulados antes de escrever;
- Deve retornar a quantidade de *bytes* escritos;
- Faz chamada da função `write_file()`.

## Remove file

```
int sad_unlink(const char *path);
```

- Implementa a função que exclui um arquivo;
- Recebe o caminho do arquivo que deve ser excluído;
- Deve retornar se o arquivo foi excluído ou não;
- Usa a função `delete_file()` da Mini-FAT.

## Remove directory

```
int sad_rmdir(const char *path);
```

- Similar à função de excluir um arquivo, porém, exclui um diretório;
- Recebe o caminho do diretório que deve ser excluído;
- Deve retornar se o diretório foi excluído ou não;
- Usa a função `delete_dir()` da Mini-FAT.

## Rename

```
int sad_rename(const char *path, const char *newpath);
```

- Função responsável por renomear (e/ou mover) um arquivo;
- Recebe o caminho de origem do arquivo;
- Também recebe o caminho de destino do arquivo;
- Deve retornar se o arquivo foi movido com sucesso ou não;
- Faz chamada de funções para copiar e atualizar arquivos;

## Truncate

```
int sad_truncate(const char *path, off_t newsize);
```

- Função responsável por trocar o tamanho do arquivo;
- Recebe o caminho do arquivo;
- Deve retornar se o tamanho do arquivo foi alterado com sucesso ou não;
- Chama a função `resize_file()` da Mini-FAT.echo

# Sumário

```
struct date_format {
        unsigned int day:6;
        unsigned int month: 5;
        unsigned int year:13;
        unsigned int hour:6;
        unsigned int minutes:7;
        unsigned int seconds:7;
} __attribute__((packed));
typedef struct date_format date_t;
```

**Dados**

- Dia: 1-31
  - $2^6 = 64$
- Mês: 1-12
  - $2^5 = 32$
- Ano: 1900-4096
  - $2^13 = 8192$
- Hora: 0-24
  - $2^6 = 64$
- Minutos: 0-59
  - $2^7 = 128$
- Segundos: 0-59
  - $2^7 = 128$

```
struct date_format {
        unsigned int day:6;
        unsigned int month: 5;
        unsigned int year:13;
        unsigned int hour:6;
        unsigned int minutes:7;
        unsigned int seconds:7;
} __attribute__((packed));
typedef struct date_format date_t;
```

**Dados**

- $sizeof(time\_t) = 8\ bytes$
- $44\ bits \div 8 \cong 6\ bytes$

- $sizeof(int) = 4\ bytes$
- $4\ bytes \times 6 = 24\ bytes$

- Velocidade em buscar informações de diretórios e arquivos;

- Velocidade em buscar informações de diretórios e arquivos;

## *Cache*

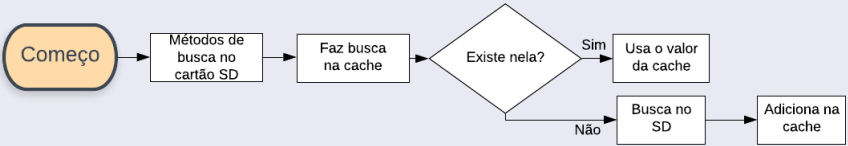


Figure: Elaborado pelo autor.

- Exemplo com:
  - 3 arquivos com respectivamente 0, 8 e 8903 bytes;
  - 1 diretório vazio.

**Time sem cache**

- real $\rightarrow$ 0m0,376s
- user $\rightarrow$ 0m0,001s
- sys $\rightarrow$ 0m0,000s

- Exemplo com:
  - 3 arquivos com respectivamente 0, 8 e 8903 bytes;
  - 1 diretório vazio.

**Time sem cache**

- real → 0m0,376s
- user → 0m0,001s
- sys → 0m0,000s

**Time com cache**

- real → 0m0,004s
- user → 0m0,003s
- sys → 0m0,000s

**Melhoria de 94x**

- Porque ouve a melhoria?
- A cache diminui a quantidade de acessos no SD;
- Menos acesso SD = Menos chamadas de sistema para entrada e saída;

- Exemplo: buscar o arquivo `file.txt` localizado em `/dir/`.

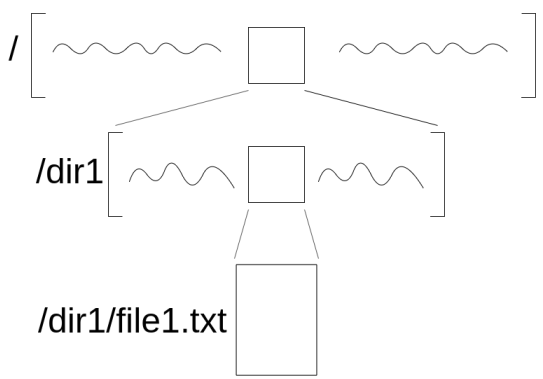


Figure: Elaborado pelo autor.

- O arquivo está dentro de um sub-diretório;
- No primeiro passo, é buscado no diretório / a entrada `dir1`;
- Como é um diretório, é consultado no SD o vetor de entradas do diretório `/dir1`;
- Agora, usando o vetor de entradas de `/dir1` que é buscado a entrada `file1.txt`.
- Quando a cache é usada, o vetor de entradas de `/dir1` está na memória, não sendo necessário buscar no SD.

# Sumário

- Aplicação da biblioteca libfuse;
- Implementação de uma versão simplista da FAT porém, eficaz para o que se propõe;
- Possibilidade de adicionar funções extras a FAT na Mini-FAT;
- Limitação de desempenho devido ao Arduino;
- Contorno do problema com o uso de memória *cache*;
- Fácil modificação do protocolo de leitura/escrita dos blocos.
  - *write*
  - *read*

https://github.com/samuelterra22/S.A.D.-File-System

# Sumário

# Referências bibliográficas I



Bzoch, P. and Safarik, J. (2011).
State of the art in distributed file systems: Increasing performance.
In *2011 Second Eastern European Regional Conference on the Engineering of Computer Based Systems*, pages 153–154. IEEE.


Fontana, L. (2016).
Write a filesystem with fuse.


Libfuse (2019).
libfuse/libfuse.

# Implementação de sistema de arquivos utilizando FUSE

## S.A.D. File System

Arthur Alexsander Martins Teodoro  Davi Ribeiro Militani  Samuel Terra Vieira

Departamento de Ciência da Computação
Universidade Federal de Lavras

Professor: Prof. Tales Heimfarth