Universidade federal São João Del Rei
Campus: Tancredo de Almeida Neves
Departamento de Ciência da Computação
Sistema Operacional

Trabalho Sistemas Operacionais: SISTEMA DE ARQUIVO SIMPLES EM FAT16

Alunos:

Alex de Andrade Soares, 182050080
Lucas Rômulo de Souza Resende, 172050036
Sidney de O. Felipe Júnior,7 172050095

Professor: Rafael Sachetto

ABRIL
2021

z

Conteúdo

1 Introdução

Levando em consideração nosso disco como uma sequencia linear que contem partes definidas, e que nessas partes definidas pode-se editar e armazenar dados. Com isso precisamos de um sistemas de arquivos para gerenciar essas informação.

Como definido por [Tanenbaum,] em seu livro de *sistema operacionais moderno* "Arquivo é um mecanismos de abstração". Sendo usado para armazenar informações no disco, tomando o cuidado com o usuário para que ele não saiba os detalhes de onde e de que forma os dados foram armazenados no disco, e do funcionamento real do disco.

Uma forma adotada pela maioria dos sistemas operacionais permite que seja valido nomes de arquivos com cadeia de caracteres do tipo strings, com o seu tamanho de até 255 caracteres.

Para gerir os arquivos de um disco rígido pessoal a microsoft por um bom tempo no seu sistema operacional Windows 95, 98 e ME usou o MS-DOS o qual foi o primeiro gerenciador de sistema de arquivo FAT. Uma tabela Fat pode-se armazenar quais blocos(suas partes definias) de disco pertence a cada arquivo. Podendo assim operar uma busca à partir de um Nó inicial da memoria, podendo ler todo arquivo através da tabela FAT. Mesmo existindo varias restrições de tamanho o sistema FAT durou algumas gerações mas com o passar dos anos o sistema Fat foi encontrando dificuldades para operar em discos rígidos maiores, sendo então não usado computadores pessoais atualmente mas ainda sendo usado em câmeras digitais, em sistema embarcado, usamos também o Fat em sua grande parte em sistemas atuais de televisão, pendrive, etc. pois além de ser um sistema de gerenciamento de arquivos simples para dispositivos com tamanha limitado acaba sendo eficiente.

Depois de um breve resumo sobre o que tanenbaum diz sobre arquivo e como o FAT foi usado e ainda e usado atualmente, nosso trabalho consiste em implementar um simulador de gerenciamento de sistema de arquivo simples com baseada na tabela de alocação de 16 bits conhecida como FAT, e seu respectivo shell para que o usuário possa então interagir com sistema desenvolvido. O simulador tem o modelo que devemos seguir:

- **Partição virtual**

  - 512 bytes por setor;
  - Cluster: 1024 bytes, ou seja 2 setores por cluster.
  - 4096 clusters.
  - Tamanho = 1 setor * 1 cluster * 4096 clusters = 4 MB

- Mínimo : 1 clusters

- **Boot Block**

  - 1 clusters que tem tamanho de 1024bytes preenchido com oxfffd

- **FAT**

  - 4096 * 2 bytes por entrada, 16bits que e igual 8192 bytes (8 cluster).

- **Root Dir**

  - 1 cluster (32 entradas de diretório)

- **Restante**

  tendo o cluster final com 4086 clusters

2 Metodologia

A simulação do gerenciador de sistema de arquivo se consiste em o uso de um Shell desenvolvido para FAT16, o inicio da simulação e feita pelo Shell.

Essa simulação para ser feita foi preciso um editor de texto, como o shell deve se comportar e precisamos saber sobre o funcionamento do FAT16 e de um sistema operacional Linux. sendo assim para a execução do simulador pelo terminal

Caminhe ate o diretório do trabalho
$ cd caminho_do_sistema/FAT16/

Para Compilar o programa
$ make
$ make install

Para Executar o simulador do gerenciador de processos
$ ./fat_sys

Para Limpar os arquivos .o da pasta
$ make clean

3 Desenvolvimento

O programa consiste basicamente em simular nosso sistemas de arquivos FAT16, criando ou carregando um arquivo binário, com a ajuda do uso de um Shell que foi criado especificamente parta gerir o FAT. Toda execução do simulador se dar por interações com o nosso Shell

3.1 Shell

O shell foi criado para ser onde nosso usuário interage com os seus arquivos tendo 11 comandos.

Comando	Descrição
help	imprime ajuda
init	inicializar o sistema de arquivos
load	carregar o sistema de arquivos do disco
ls	lista diretório
mkdir	cria diretório
creat	cria arquivo
unlink	excluir arquivo ou diretório
write "string"[path]	escrever dados em um arquivo
append "string"[path]	anexar dados em um arquivo
read	lê o conteúdo de um arquivo
exit	Sai do simulador

Tabela 1: Comandos do Shell

- help: Imprime uma tabela como a tabela que vimos assim, com nome da função e uma descrição simples dela para que o usuário possa operar no shell.
- init: Inicia nosso simulado de gerenciamento de arquivo em branco, simulando uma formatação do sistema de memoria virtual.
- load: Carrega a tabela FAT e o seu diretório raiz para a memoria, se houver falha nesse processo e por algum fator, e a tabela não seja salva, a mesma perde a validade no disco e todos os dados armazenado ali perdem seu significado na hora da leitura, já que seus dados não são atuais.
- ls: Lista para o usuário as entradas de cada diretório existente.

- mkdir: Cria um novo diretor dentro do diretório raiz, permitindo que seja criado diretórios com uma cadeia de caracteres de ate 8 letras, permitindo o uso de espaço, se neste caso usar o espaço o mesmo ocupa na cadeia de carácter o tamanho relativo a um espaço(1 letra).

- creat: Cria um arquivo de texto dentro do diretório, permitindo o uso de espaço.

- unlink: Deleta um diretório,

- write "string"[path] : reescreve/escreve uma cadeia de caracteres(string) em um arquivo de texto.

- append string"[path]: concatena uma cadeia de caracteres(string) em um arquivo de texto.

- read: Imprime, ou seja, faz um leitura do arquivo, para o usuário e retorna no terminal todo texto escrito no arquivo.

- exit: Encerra o simulador do shell.

Para o funcionamento do shell, o arquivo "entry.c"faz a leitura do comando, lendo a primeira palavra, assumindo essa palavra como seu comando, em seguida validando se o comando digitado pelo usuário existe, em caso do comando digitado existir, verifica-se a que tipo esse comando pertence. Em algumas funções e permitido a passagem de mais uma informação, sendo elas uma cadeia de caracteres(string) ou o path(caminho do diretório), assim permitido a leitura de uma string e um path ao mesmo tempo. Entretanto caso o comando digitado pelo usuário seja invalido, retorna para o mesmo uma mensagem de erro.

Na Tad "entry"ela e definida com 4 atributos sendo eles :

- um inteiro, definido como type

- um ponteiro do tipo char, definido como command

- um ponteiro do tipo char, definido como path

- um ponteiro do tipo char, definido como string

Apos a validação o shell começa executa o FAT, recebendo um dos atributos definido acima, buscando executar sua função de acordo com o comando definido para ele, em alguns caso podendo receber um comando, um [path] e uma [string].

3.2 Estrutura de Dados

3.2.1 Fat

Fat possui 3 tipos abstratos de dados, sendo eles: table_t, dir_t e cluster_t que serão detalhados abaixo:

table_t:
Tipo abstrato que armazena a tabela que contém os 4096 cluster da FAT.

```
typedef struct table_t
    uint16_t fat[FAT];
table_t;
```

- **fat[FAT]:** vetor do tipo unsigned int de 16 bits que armazena o endereço dos 4096 clusters da FAT.

dir_t:
Tipo abstrato que armazena todas as informações de um entrada de diretório.

```
typedef struct dir_t
    uint8_t filename[18];
    uint8_t attributes;
    uint8_t reserved[7];
    uint16_t first_block;
    uint32_t size;
dir_t;
```

- **filename[18]:** Vetor do tipo unsigned int de 8 bits que armazena o código ascii dos caracteres que compõem o nome do arquivo.

- **attributes:** vetor de bits
  Bit 0 – somente leitura;
  Bit 1 – oculto;
  Bit 2 – arquivo de sistema;
  Bit 3 – rótulo de volume;
  Bit 4 – subdiretório;
  Bit 5 – archive;
  Bits 6 e 7 – não utilizados.

- **reserved:** vetor do tipo unsigned int de 16 bits que armazena o endereço dos 4096 clusters da FAT.
- **first_block:** unsigned int de 16 bits que armazena o cluster inicial do arquivo;
- **size:** unsigned int de 32 bits que armazena o tamanho do arquivo.

cluster_t:
Tipo abstrato que armazena os arquivos e dados do diretório.

```
typedef union cluster_t
    dir_t dir[CLUSTER/sizeof(dir_t)];
    uint8_t data[CLUSTER];
table_t;
```

- **dir[CLUSTER/sizeof(dir_t)]:** vetor de tamanho 32 que armazena informações de diretórios;
- **data[CLUSTER]:** vetor do tipo unsigned int de 8 bits que armazena conteúdo de um arquivo.

entry_t:
Tipo abstrato que armazena comando, tipo, caminho e string extraídos de um comando inserido pelo usuário.

```
typedef struct entry_t
    int type;
    char* command;
    char* path;
    char* string;
entry_t;
```

- **type:** tipo inteiro que armazena o tipo dos comandos;
- **command:** string que armazena o nome do comando inserido;
- **path:** string que armazena o caminho do comando inserido caso possua;
- **string:** string que armazena o texto extraído do comando inserido para escrita em arquivo, caso possua.

3.3 Globais e Macros

3.3.1 Macros de tamanho

As macros de tamanho são utilizadas para valores que indicam o tamanho de vetores substituindo números fixos por nomes que facilitam a leitura e compreesão do código. As macros de tamanho podem ser visualizadas abaixo.

- **INPUT_SIZE 100:** Tamanho máximo permitido para cada instrução inserida pelo usuário.
- **COMMAND_SIZE 10:** Tamanho máxima para um comando extraído de um entry_t
- **STRING_SIZE 50:** Tamanho máximo permitido das strings inseridas como argumentos em cada instrução.
- **FAT 4096:** Quantidade de clusters existentes na FAT.
- **CLUESTER 1024:** Tamanho de cada clusters na FAT..
- **BOOT 2:** Indica a quatidade de seções que o boot block ocupa na FAT.
- **TABLE 8:** Quantidade de clusters que a tabela FAT ocupa.
- **OTHERS (FAT-BOOT-TABLE):** Quantidade de clusters livres para utilização.

3.3.2 Macros de Retorno de navegação do diretório

Na linguagem C, dados booleanos são representados como zero para falso e qualquer número diferente de zero. Também por motivos de legibilidade e entendimento do código, foram criadas as macros abaixo para retorno de funções:

- **TRUE 1:** Utilizado para representar o Boolean true;
- **FALSE 0:** Utilizado para representar o Boolean false;

3.3.3 Macros de pedido de navegação

Assim como as macros já apresentadas, as macros de pedido de navegação foram criadas para facilitar a leitura e compreensão do código e são utilizadas para definição dos tipos de comandos após serem extraídos das instruções inseridas pelo usuário.

- **EXEC_TYPE 1:** Valor atribuido ao tipo para instruções que apenas
- **DIR_TYPE 2:** Valor atribuido ao tipo para instruções que executam operações envolvendo diretórios.
- **DOC_TYPE 3:** Valor atribuido ao tipo para instruções que executam operações envolvendo documentos.
- **STR_TYPE 4:** Valor atribuido ao tipo para instruções que lêem, escrevem.
- **EXIT_TYPE 5:** Valor atribuido ao tipo para a instrução "exit" que finaliza o programa.

3.3.4 Globais

Para a execução do simulador foram criadas duas variáveis globais:

- **table_t* table:** Ponteiro para uma variável do tipo abstrato table_t utilizada para armazenar o estado da tabela FAT.
- **cluster_t* current:** Ponteiro para uma variável do tipo abstrato cluster_t criada para armazenar o diretorio que se encontra aberto no momento.
- **char* file_path[50]:** String contendo o nome da partição virtual que armazena o estado do sistema de arquivos
- **int fat_loaded:** Flag que indica se a FAT está carregada na memória.

3.4 Funções

3.4.1 Shell

- **int main():**

  Função principal do programa. Cria uma variável do tipo entry_t que será usada para armazenar as instruções inseridas pelo usuário e executa o comando da instrução. A leitura de novos comandos é feita até que o usuário insira o comando exit.

  A função retorna "0" se o programa finaliza sem ocorrência de erros.

- **void execute(entry_t entry):**

  Recebe uma variável do tipo entry_t como argumento.

  Função que executa os comandos de acordo com o nome do comando e seu tipo.

3.4.2 Entry

- **entry_t new_entry():**

  Cria e aloca espaço na memória para um ponteiro do tipo entry_t. Os atributos da variável são alocados na memoria em um espaço do tamanho máximo determinado para cada um através das macros correspondentes e recebem seus valores padrão.

  A função retorna a variável entry.

- **entry_t read_entry():**

  Declara um vetor de char para armazenar uma intrução, um ponteiro do tipo entry_t para armazenar o comando com seus atributos e um inteiro que será utilizado para armazenar o tipo da instrução.

  O comando é extraído da instrução lida e atualizado na variável entry, seu tipo é verificado e em seguida, caso o tipo do comando seja valido, entry tem seu atributo tipo atualizado.

Os atributos necessários para cada tipo de comando são extraídos da instrução e entry é retornado com os atributos atualizados para os que foram lidos da instrução.

- **int verify_command(char* command):**

  A função recebe uma string command como argumento.

  É realizada a comparação do nome do comando com os comandos disponíveis para um tipo. Caso o nome do comando seja igual a algum comando, a função retorna o tipo a qual o comando pertence, se não retorna "0" que não corresponde a nenhum tipo.

- **char* extract_command(char* input, int*i):**

  A função recebe uma string input contendo uma instrução e um ponteiro para um inteiro contendo o índice da posição onde a leitura da instrução foi finalizada na última extração.

  É criado e alocado espaço na memoria para uma string command. Em um loop, o comando é extraído da função copiando os caracteres até que se encontre um espaço ou '\n' indicando o fim do comando na instrução.

  A string command é retornada contendo apenas o nome do comando da instrução.

- **char* extract_paths(char* input, int* i):**

  A função recebe uma string input contendo uma instrução e um ponteiro para um inteiro contendo o índice da posição onde a leitura da instrução foi finalizada na última extração.

  É criado e alocado espaço na memoria para uma string path. A execução da função é encerrada retornando uma string vazia se o próximo caractere da instrução for '' se não, o restante da instrução é percorrido realizando assim a extração do caminho para um diretorio ou arquivo.

  A variável path é retornada pela função.

- **char* extract_string(char* input, int* i):**

  A função recebe uma string input contendo uma instrução e um ponteiro para um inteiro contendo o índice da posição onde a leitura da instrução foi finalizada na última extração.

  É criado e alocado espaço na memoria para uma string string. A execução da função é encerrada retornando uma string vazia de o próximo caractere da instrução for “”’, se não, o restante da string é percorrido realizando a copia do conteúdo para string até que se chegue ao final da string da instrução.

  A variável string é retornada pela função. ornada pela função.//

3.4.3 Fat_TAD

- **int get_fat_loaded():**

  A função retorna o valor variável global fat_loaded que indica se o sistema de arquivos foi carregado previamente.

- **void set_fat_loaded(int i):**

  Recebe um inteiro como argumento.

  A função altera o valor da variável global fat_loaded que indica se o sistema de arquivos foi carregado previamente.

- **void check_sys_load():**

  A função verifica se o sistema está carregado na memória. Caso esteja, retorna TRUE, se não, imprime uma mensagem solicitando que o usuário carrege o sistema primeiro.

- **uint8_t ctohex(char* name, uint8_t* filename):**

  Recebe uma string nome e um vetor de inteiros de 8 bits contendo a representação hexadecimal de uma string.

  Realiza a conversão de name para a representação hexadecimal.

- **char* hextoc(uint8_t filename, char* name):**

  Recebe um vetor de inteiros de 8 bits filename contendo a representação hexadecimal de uma string name.

  Realiza a conversão de filename para o tipo string.

- **cluster_t init_cluster():**

  É criado e alocado espaço na memoria para um ponteiro empty_cluster do tipo cluster_t. Todas a posições do atributo data são preenchidas com 0.

  A variável empty_cluster é retornado pela função.

- **cluster_t* init_boot():**

  É criado e alocado espaço na memoria para um ponteiro boot do tipo cluster_t. As posições do atributo data são preenchidas com o valor 0x00bb.

  A variável boot é retornada pela função.

- **table_t* init_fat():**

  É criado e alocado espaço na memoria para um ponteiro table do tipo table_t. As posições do atributo fat são preenchidas com 0.

  A variável table é retornada pela função.

- **dir_t* Init_dir_entry(char* name, uint8_t attributes, uint16_t first_block):**

  Recebe uma string name, um inteiro de 8 bits atributes de um inteiro de 16 bits first_block.

  É criado e alocado espaço na memoria para um ponteiro dir do tipo dir_t. A atributos são preenchidos com seus devidos valores.

  A variável dir é retornada pela função.

- **cluster_t init_dir(uint16_t root, uint16_t first_block):**

  A função recebe dois inteiros de 16 bit root e block.

  É declarada a variável cluster do tipo cluster_t e inicializados os dois setores de cluster com o endereço para o diretório root e para o primeiro bloco. Os demais diretórios são inicializados como vazios.

  A variável cluster é retornada pela função.

- **get_first_fat_empty(table_t* table):**

  Recebe a FAT como argumento.

  A função percorre a FAT procurando e retornando o endereço do primeiro cluster vazio. Caso não termine de percorrer e não tenha encontrado um cluster vazio, retorna a mensagem "Theres no more space in the system!!".

- **get_fisrt_dir_empty(cluster_t* current):**

  Recebe um cluster como argumento.

  Percorre os diretórios do cluster procurando e retornando o índice do primeiro diretório vazio.

  Caso termine de percorrer e não tenha encontrado um cluster vazio, retorna a mensagem "Theres no more space in the directory!!".

- **cluster_t get_cluster(int offset)::**

  Recebe um inteiro offset como argumento.

  A função realiza a leitura da partição e retorna o cluster na posição definida por offset considerando o tamanho do bloco de boot, da tebela e de cada cluster.

- **table_t get_fat():**

  A função realiza a leitura da partição e retorna a fat do sistema.

- **void set_fat(table_t table):**

  Recebe uma table_t table como argumento.

  A função realiza a abertura do arquivo da partição e atualuza a FAT para a variável table.

- **void set_cluster(int offset, cluster_t* cluster):**

  Recebe um inteiro offset e um ponteiro para uma variável cluster do tipo cluster_t.

  A função realiza a abertura do arquivo da partição e atualiza o cluster na posição offset considerando o tamanho do bloco de boot, da table e de cada cluster.

3.4.4 Fat

- **void init():**

  A função inicializa o sistema alocando na memoria e inicializando os endereços para seus valores padrão.

  Após serem inicializados, o arquivo de partição é atualizado para os valores criados

- **void load():**

  A função realiza a leitura do arquivo de partição, salva a FAT em uma variável table_t table e o cluster atual na variável cluster_t current. Por fim a variável global fat_loaded é atualizada para TRUE indicando que a fat esta carregada na memoria.

- **void mkdir(char* name):**

  Recebe uma string nome como argumento.

  A função cria um novo diretório no diretório atual, cria um cluster diretório atual para o diretório criado.

A fat é atualizada inserindo o endereço do novo diretório. O cluster do diretório atual é atualizado inserindo o cluster do diretório criado.

O diretório é criado apenas caso não exista um diretório com o mesmo nome já criado.

- **void create(char* name):**

  Recebe uma string nome como argumento.

  A função cria e adiciona um novo diretório no diretório aberto no momento. É criado então um cluster para o novo diretório e adicionado à fat.

  O diretório é criado apenas caso não exista um diretório com o mesmo nome já criado.

- **void unlink(char* nome):**

  Recebe uma string nome como argumento.

  A função percorre os diretórios do diretório aberto no momento procurando por uma cominação entre o nome do diretório e o valor da string nome.

  Ao encontrar, atualiza o cluster para um novo cluster vazio e também os diretórios para novos diretórios vazios e atualiza o diretório atual.

- **void ls():**

  A função percorre os diretórios do diretório aberto no momento e imprime o nome de cada diretório encontrado.

- **void read(char* path):**

  Recebe uma string path como argumento.

  A função percorre os diretórios do diretório atual comparando seu nome com o valor da string nome. Caso sejam iguais e o diretório seja uma pasta, é exibida uma mensagem dizendo que não é possível ler um diretório, se não verifica se são iguais novamente e se é do tipo

arquivo convertendo o texto de hexadecimal para string e imprimindo, caso contrário, imprime uma mensagem dizendo que não foi possível encontrar um arquivo.

- **void write(char* path, char* string):**

  Recebe duas strings path e string como argumentos.

  A função percorre os diretórios do diretório atual comparando seu nome com o valor da string nome. Caso sejam iguais e o diretório seja uma pasta, é exibida uma mensagem dizendo que não é possível escrever em um diretório, se não verifica se são iguais novamente e se é do tipo arquivo, atualiza o tamanho do arquivo e seu conteudo para o tamanho de string e seu conteúdo já convertido para hexadecimal.

  As alterações são salvas na FAT e é impresso uma mensagem informando se houve sucesso da tarefa ou não.

- **void append(char* path, char* string):**

  Recebe duas strings path e string como argumentos.

  Executa uma tarefa parecida com a da função write. O que difere as duas é o fato da função append concatenas as o conteúdo do arquivo com string.

  Vale ressaltar que um arquivo pode ocupar apenas um cluster, logo, se a concatenação resultar em um arquivo que ocupe mais que um cluster, então os sistema não permite a atualização do conteúdo do arquivo.

- **void help():**

  A função imprime uma tabela contendo todos os comandos aceitos pelo shell informando para que serve cada um deles.

3.5 Analise de Dados

O simulador Fat16 mostrado a seguir, exemplifica como um usuário usaria nosso sistema em seu cotidiano, mensagens de erro são exibidas caso ele insira um comando invalido.

```
   $ ls                  ---> retorna mensagem de erro sistema não inicializado
Please, load the system first!!
$ help                   --> imprime uma descrição curta de cada comando
| ---------------------- | ------------------------------------------------ |
| Comando                | Descrição                                        |
| ---------------------- | ------------------------------------------------ |
| help                   | Imprime ajuda                                    |
| ---------------------- | ------------------------------------------------ |
| init                   | Inicializar o sistema de arquivos                |
| ---------------------- | ------------------------------------------------ |
| load                   | Carregar o sistema de arquivos do disco          |
| ---------------------- | ------------------------------------------------ |
| ls                     | Lista diretório                                  |
| ---------------------- | ------------------------------------------------ |
| mkdir                  | Cria diretório                                   |
| ---------------------- | ------------------------------------------------ |
| create                 | Cria arquivo                                     |
| ---------------------- | ------------------------------------------------ |
| unlink                 | Excluir arquivo ou diretório                     |
| ---------------------- | ------------------------------------------------ |
| write "string" [path]  | Escrever dados em um arquivo                     |
| ---------------------- | ------------------------------------------------ |
| append "string" [path] | Anexar dados em um arquivo                       |
| ---------------------- | ------------------------------------------------ |
| read                   | Lê o conteúdo de um arquivo                      |
| ---------------------- | ------------------------------------------------ |
| exit                   | Sai do simulador                                 |
| ---------------------- | ------------------------------------------------ |
$ init                 ---> inicia sistema
FAT system created!!
$ load                 ---> carrega sistema
FAT system loaded!!
$ creat conta          ---> retorna erro pois comando foi escrito errado
Invalid command!!
$ create conta         ---> cria um arquivo com nome de conta
$ ls                   ---> lista os diretórios
```

```
.       ..      conta.txt
$ write "luz 250" conta ---> comando de escrita "string" [path]
Doc writed!!
$ read conta            ---> le o arquivo
luz 250
$ append " vence dia 15" conta
Doc writed!!            ---> concatena a string com o que ja esta salvo
$ read conta
luz 250 vence dia 15
$ mkdir loja            ---> cria diretórios
$ mkdir casa            ---> cria diretórios
$ mkdir contas          ---> cria diretórios
$ ls
.       ..      conta.txt       loja    casa    contas
$ unlink conta          ---> apaga arquivo
Unlinked!!
$ ls
.       ..      loja    casa    contas
$ unlink loja           ---> apaga diretório
Unlinked!!
$ ls
.       ..      casa    contas
$ unlink contas         ---> apaga diretório
Unlinked!!
$ ls
.       ..      casa
$ exit                  ---> sai do simulador
See ya!!
```

4 Conclusão

O objetivo deste trabalho foi construir um algoritmo de simulação de gerenciamento de sistema de arquivo simples com baseada na tabela de alocação de 16 bits - FAT16, porém houve etapas que a implementação na linguagem C, ficou confusa e um pouco complexa de ser resolvido mas no decorrer do desenvolvimento conseguimos atingir o objetivo e fazer um simulador de gerenciamento de sistema de arquivo simples com baseada na tabela de alocação de 16 bits(FAT16) com isso nos possibilitando fazer uma analise critica de todo comportamento do simulador. Portanto, este trabalho abre uma uma oportunidade de simular e compreender métodos de gerenciamento do arquivos binários.

Referências

[Tanenbaum,] Tanenbaum, A. S. W. A. S. *Sistemas Operacionais Modernos*. 3 edition.