UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL REI

SISTEMAS OPERACIONAIS

Ronan Souza Castro
Matheus Alexandre de Jesus
Luccas Guidio

Dezembro 2019

1 Introdução

Este trabalho prático da disciplina de Sistemas Operacionais consiste na implementação de um simulador de um sistema de arquivos simples baseado em tabela de alocação de 16 bits (*fat*) e um *shell* usado para realizar operações sobre este sistema de arquivos.

2 Desenvolvimento

O programa tem um fluxo de funcionamento simples. Percorrer o caminho necessários e executar uma determinada função no diretório/arquivo final. Desta forma, boa parte do código é similar, sofrendo pequenas alterações para executar trechos específicos. Foi utilizado uma estrutura para representar cada entrada da *fat*, chamada *datacluster*. O diretório raíz utiliza de uma versão desta mesma estrutura, porém com uma modificação na quantidade de elementos que podem ser criados.

1. **init( ):**
   Cria um arquivo binário que representará a *fat*, que será utilizado pelo *shell*.
2. **load( ):**
   Carrega o arquivo binário que simula a *fat* já existente para ser utilizado pelo *shell*.
3. **attArquivo( ):**
   Função que atualiza os dados criados/alterados nas funções de criação de arquivo/diretório e funções de alteração de arquivo para a *fat*.
4. **ls(char param[ ]):**
   Verifica se o parametro passado em seguida percorre os diretórios na *fat* a partir do diretório *root* até encontrar o diretório pedido, em seguida ele imprime na tela o diretório atual e os diretórios e arquivos contidos dentro dele.
5. **mkdir(char param[ ]):**
   Função que cria diretórios, verificando a entrada dada no *shell*, em seguida ele percorre os diretórios ja existentes a partir do *root* e assim que não encontra o diretório digitado, cria este diretório no primeiro espaço vazio na *fat* como filho do último diretório ja existente.
6. **create(char param[ ]):**
   Esta função funciona da mesma maneira genérica do mkdir, uma vez que percorrerá a *fat* de acordo com a entrada e no final criará um arquivo no último diretório percorrido, também salvando no primeiro espaço vazio na *fat* como filho do último diretório ja existente.
7. **unlink(char param[ ]):**
   Função encarregada de excluir arquivos e diretórios da *fat*, dada a entrada pelo usuário ele percorre o caminho dado até a última casa digitada, quando encontrado

o arquivo ou diretório ele verifica se é possivel ser deletado. Se for arquivo ele é removido, porém se for um diretório, primeiro a função verifica se ele está vazio para ser deletado, caso contrário ele informa que o diretório deve estar vazio para ser removido.

8. **read(char param[ ]):**
   Função percorre na *fat* o caminho dado pelo usuário no *shell* até encontrar o arquivo desejado, e em seguida imprime na tela o conteúdo do arquivo.
9. **write(char str[ ], char param[ ]):**
   Função percorre na *fat* o caminho dado pelo usuário no *shell* até encontrar o arquivo desejado, e em seguida apaga o conteúdo antigo, caso tenha algum, e salva um novo conteúdo no arquivo que o usuário digitou.
10. **append(char str[ ], char param[ ]):**
    Função que funciona da mesma forma que a função write ,porém ao invés de apagar e escrever, ela simplesmente agrega o que o usuário digitou ao arquivo.
11. **freeSpace*fat*( )**
    Função que verifica se há uma posição na *fat* vazia e se tiver retorna qual é a posição que está vazia.
12. **help( )**
    Função que mostra as funções existentes no *shell* e como usá-las.
13. **struct datacluster**
    Estrutura que representa virtualmente os dados do sistema *fat*.
14. **struct direntryt**
    Estrutura que comporta as informações de uma partição do *datacluster*. Sendo informações como nome e tipo de dado, diretório ou arquivo.
15. **struct *root*data**
    Estrutura que tem com função representar o diretório *root* na *fat*.

3 Análise de Resultados

```
ronan@ronan-VirtualBox:~/Área de Trabalho/tp3_so$ ./tp3
'help' para mais informações
$ mkdir /a/
$ ls /
<D> a
$ mkdir /a/b/
$ ls /a/
<D> b
$ create /a/b/c
$ write calejado /a/b/c
$ read /a/b/c
calejado
$ append s /a/b/c
$ read /a/b/c
calejados
$ unlink /a/b/
Diretorio precisa estar vazio.
$ unlink /a/b/c
$ ls /a/b/
Caminho especificado nao existe ou esta vazio.
$ ls /a/
<D> b
$ unlink /a/b/
$ ls /a/
$ unlink /a/
$ ls /
$ exit
Salvando os dados...
```

Figura 1: Teste feito no terminal do linux

4 Referências

- https://pt.wikipedia.org/wiki/File-Allocation-Table
- https://en.wikipedia.org/wiki/Data-cluster