EP3 - Sistema de Arquivos

Sistemas Operacionais (MAC0422)

Nome: Lara Ayumi Nagamatsu NUSP: 9910568
Nome: Lucy Anne de Omena Evangelista NUSP:11221776

Visão Geral

Este EP trata da simulação de sistema de arquivos de 100MB com mapeamento de memória livre por meio de bitmap e mapeamento de memória por uma tabela fat. A estrutura possui blocos de memória de 4KB para armazenar os dados.

Para termos o controle sobre o que e quanto está sendo ocupado pelo sistema de dados, utilizamos uma variável global admin, que é um vetor de 24414 posições, para armazenar o que está em cada posição.

A tabela de bitmap ocupa 7 blocos na memória.

A tabela fat ocupa 49 blocos.

Decisões de Projeto

- **FAT**

  Nossa FAT possui 24414 posições de arquivos do tipo Bloco, que possuem:

```
 typedef struct node {
    void * endereco; // endereço do bloco de dados no vetor admin
    int prox; // indica o índice do próximo bloco (-1 se não tiver próximo)
} Bloco;
```

- **BITMAP**

  É um vetor simples de 0s e 1s com tamanho 24414. Tanto o FAT quando o BITMAP se autorepresentam.

Decisões de Projeto

- **METADADOS DE DIRETÓRIOS E ARQUIVOS**

  Criamos a estrutura Celula para armazenar as informações dos diretórios e arquivos

```
typedef struct celula {
    char * nome; // Nome do arquivo txt/diretório
    char tipo; // D: diretório, A: arquivo texto -> 1B
    unsigned int t_criado; // instante de tempo em que o arquivo foi criado  ->> 4B
    unsigned int t_alterado; // instante de tempo em que o arquivo foi alterado  -> 4B
    unsigned int t_acesso; // instante de tempo em que o arquivo foi acessado -> 4B
    int pos_fat; // posição inicial no FAT -> 4B
    int filhos; // conta o número de filhos do diretório
    int tamanho; // tamanho em bytes, só para não diretórios -> 4B
    struct celula * node_filho; // Diretório abaixo na hierarquia -> 4B
    struct celula * node_prox; // Diretório na mesma hierarquia -> 4B
} Celula;
```

Decisões de Projeto

- **METADADOS DE DIRETÓRIOS E ARQUIVOS**

  Como pode ser observado pela variável filhos, armazenamos a quantidade de filhos que determinada pasta pode possuir, a fim de restringir o uso de memória e evitar que não haja espaço para os dados.

  Em operações como ls, mkdir e etc, é necessário colocar uma / ao final do endereço do diretório.

Decisões de Projeto

- **SALVANDO ESTADO DO SISTEMA E O RECUPERANDO**

  Para registrar o estado do sistema, armazenamos, em ordem:

  - Os bits do bitmap em uma linha;
  - Os valores da tabela FAT em outra linha;
  - Os valores dos metadados, um por linha, com dados separados por “|”;
  - Para as linhas seguintes, armazenamos os conteúdos dos dados, uma linha por bloco ocupado, escritos como texto.

Decisões de Projeto

- **SALVANDO ESTADO DO SISTEMA E O RECUPERANDO**

  Para recuperar os dados, armazenamos:

  - Não conseguimos carregar os arquivos no EP. Para simular a cópia de um arquivo, criação de diretório, remoção de diretório e remoção de arquivo, é necessário não possuir o arquivo “arquivo.txt” no computador. Basta rodar, no bash, “pwd”, pegar o caminho dado e rodar como “mount _pwd_”, sendo _pwd_ o caminho dado por pwd.

Decisões de Projeto

- **SALVANDO ESTADO DO SISTEMA E O RECUPERANDO**

  Para recuperar os dados, armazenamos:

  - Não conseguimos carregar os arquivos no EP. Para simular a cópia de um arquivo, criação de diretório, remoção de diretório e remoção de arquivo, é necessário não possuir o arquivo “arquivo.txt” no computador. Basta rodar, no bash, “pwd”, pegar o caminho dado e rodar como “mount _pwd_”, sendo _pwd_ o caminho dado por pwd.

Conclusões e considerações

Para este EP nem todas as funções foram implementadas. As implementadas são:

- mount arquivo
- cp origem destino
- mkdir diretorio
- rmdir diretorio
- cat arquivo
- rm arquivo
- ls diretorio
- umount
- sai

A função sai também realiza o umount do sistema. A função mount está parcialmente completa, não recuperando o sistema.

Conclusões e considerações

Inicialmente, pensamos em inserir o sistema de diretorios no vetor admin para melhor gestão do espaço. Entretanto, tivemos sérios problemas com essa implementação principalmente pelo fato de não conseguirmos excluir os metadados de algum registro ou arquivo que fosse excluido, pois teríamos que ter a respectiva posição do metadado na tabela admin.

Com isso, mudamos a implementação com muito pouco tempo restante e não conseguimos ajustar a função de mount para fazer o carregamento devido, e sofremos com problemas com o parse de ponteiro do tipo void *. Fizemos poucos testes preliminares com arquivos de tamanhos variados, mas não realizamos nenhum experimento propriamente dito.

Obrigada!