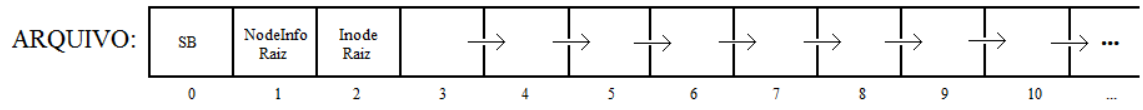
Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação
Sistemas Operacionais 2016/2
Professor: Italo Fernando Scota Cunha

TRABALHO PRÁTICO 2 B – SISTEMA DE ARQUIVOS

Aluno: Alison de Oliveira Souza – 2012049316
Aluno: Daniel Reis Souza – 2012049413

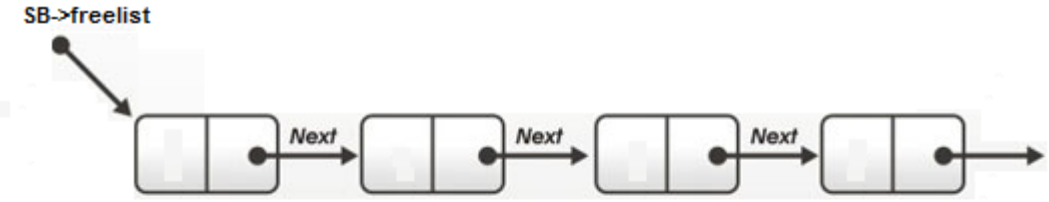
Neste trabalho implementamos as funções que simulam um sistema de arquivos, através da interface gerada pelo professor. O nosso sistema de arquivos, DCC605FS, é armazenado dentro de um arquivo previamente construído, e com tamanho pré-definido.

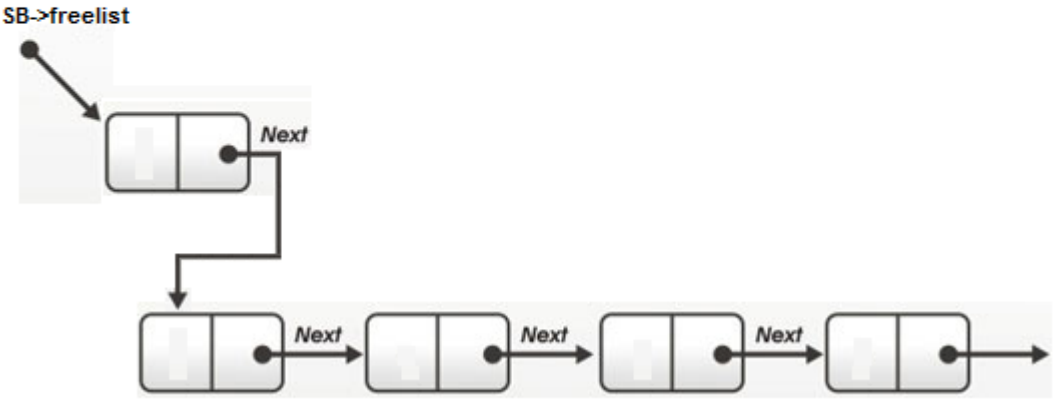
O primeiro procedimento a ser realizado, é formatar o arquivo que será utilizado como nosso disco, de modo que definimos a estrutura utilizada para gerenciar nosso sistema de arquivos. Isso é feito na função fs_format, e após a execução dessa função o arquivo deve ter essa aparência:



A função fs_format divide o arquivo (que simula o disco) em blocos de tamanhos iguais, definidos pelo parâmetro blocksize. O primeiro bloco do nosso arquivo (índice 0) será ocupado pelo superbloco (SB) do sistema de arquivos. Nele estarão contidas as informações sobre a quantidade de blocos, tamanho dos blocos, posição da pasta raiz, posição da lista de blocos vazia e etc. O segundo bloco (índice 1) será ocupado pelo NodeInfo da pasta raiz, onde estarão contidas as informações sobre o nome da pasta raiz, quantos elementos essa pasta tem e etc. O terceiro bloco (índice 2) será ocupado pelo Inode da pasta raiz, onde estarão as informações sobre a própria pasta e sobre os elementos presentes na pasta raiz. Nos outros blocos é criada uma estrutura de lista encadeada de blocos, utilizado para buscar blocos vazios no sistema, necessário em casos de escrita. Para isso, utilizamos a estrutura freepage onde temos um ponteiro que aponta para o próximo bloco vazio. Inicialmente, percorremos a lista de blocos escrevendo um ponteiro para o bloco seguinte, e assim temos nossa estrutura inicial do sistema.

A implementação da fs_format foi feita para simplificar ao máximo o sistema inicial. Utilizando apenas a lista encadeada de blocos vazios criada por meio do ponteiro next da estrutura freepage. Dessa forma, não precisamos de uma estrutura própria para controlar os blocos vazios evitando ocupar espaço no sistema de arquivos. Quando é necessário alocar um novo bloco para alguma escrita (usando a função fs_get_block) basta pegar o primeiro elemento da lista e rearranjar os ponteiros, alterando no superbloco o ponteiro do primeiro elemento da lista de blocos vazios. Já para retornar um bloco a lista (usando a função fs_put_block) basta colocar esse elemento no início da lista, escrever neste bloco um ponteiro para o primeiro elemento da lista vazia, e colocar esse bloco como o primeiro da lista no superbloco. Na imagem abaixo é mostrado a re-inserção de um bloco na lista de blocos livres, após um arquivo liberar esse bloco.





Na nossa implementação, todas as alterações no disco (arquivo) eram salvas imediatamente usando a chamada de sistema write, simulando uma gravação no disco. Ao sobrescrever um bloco antes utilizado, garantimos que os dados antigos não podem ser acessados por outro programa, já que escrevemos a estrutura freepage sobre o bloco a ser liberado.

Para simplificar nossa implementação, optamos por construir duas funções que são úteis para várias das funções principais do sistema. Elas são a find_block e a link_block. Falaremos um pouco sobre elas e também sobre como foi a implementação das funções do sistema.

uint64_t find_block(struct superblock *sb, const char *fname, int opmode)

Esta função pesquisa e retorna o índice do primeiro inode do arquivo com nome fname se opmode for igual a 0. Se opmode for igual a 1, então retorna o índice do inode do diretório pai do arquivo fname. Essa função é usada para simplificar as funções fs_unlink, fs_write_file, fs_read_file, fs_mkdir, fs_rmdir e fs_list_dir. Por exemplo, ao pesquisarmos pelo arquivo /home/user/file:

- Se opmode = 0, retorna o inode principal (IMREG) do arquivo file.
- Se opmode = 1, retorna o inode do diretório user/

int link_block(struct superblock *sb, struct inode *in, uint64_t in_n, uint64_t block)

Esta função recebe um inode, seu índice no sistema de arquivos e o índice que deseja ser linkado ao inode. Ela cria uma referência ao índice desejado no campo links[] do inode in.
Se não houver espaço no links[], é criado um novo inode do tipo child e dado a ele um bloco cujo índice é atribuído ao inode->next de in. Finalmente, é criado uma referencia a block no links[] do inode child.

A partir daqui falaremos das funções do sistema.

A função fs_open, responsável por abrir o sistema de arquivos, simplesmente lê o nome do arquivo passado como parâmetro e retorna o superbloco desse arquivo. Para garantir que um sistema de arquivos não seja aberto mais de uma vez, foi aplicado uma trava utilizando a função flock() da biblioteca sys/file.h.

A função fs_close remove a trava do arquivo gerada por fs_open e fecha o arquivo, de forma que mais tarde seja possível reabrir o sistema de arquivos novamente.

A função fs_get_block procura um bloco livre no sistema de arquivos. Para isso, utilizamos a lista de blocos livres criada na função fs_format, pegando o primeiro elemento da lista, atualizando o apontador da lista de blocos vazios do superbloco (da primeira posição da lista, para a segunda) e retornando o índice do bloco que será alocado por outra função.

A função fs_put_block insere um bloco, liberado por outra função, novamente na lista de blocos vazios. Para isso, realizamos a manobra adotada na imagem acima, onde inserimos o bloco no início da lista e atualizamos os ponteiros para a freelist do superbloco.

A função fs_write_file escreve um arquivo no sistema de arquivos. Primeiramente, ela verifica se já existe um arquivo no diretório corrente com o mesmo nome. Se houver, é feito uma chamada a função fs_unlink para apagar o arquivo existente. Após isso ela aloca novos blocos para o nodeinfo, inode e dados (quantos forem necessários) do arquivo a ser criado, adiciona esses blocos ao diretório pai utilizando a função link_block, escreve os dados nos blocos e salva os dados no sistema de arquivos.

A função fs_read_file lê de um arquivo no máximo uma quantidade de bytes passada por parâmetro, salva esses dados num buffer e retorna quantos bytes foram lidos realmente. A fs_read_file usa a função find_block para localizar o índice do inode principal do arquivo que precisa ser lido. Com o inode, localizamos os blocos de dados do arquivo e realizamos a leitura utilizando uma variável auxiliar para ler cada bloco e concatenando os dados no buffer de saída. Quando chegar no limite de bufsz bytes lidos, ou quando o arquivo acabar, a função retorna o número de bytes lidos.

A função fs_unlink utiliza a função find_block para localizar o índice do primeiro inode do arquivo que será apagado. Primeiramente, carregamos o inode principal numa variável auxiliar e apagamos o nodeinfo do arquivo. Após isso, apagamos os links de cada inode e deletamos os inodes. No final, atualizamos o nodeinfo do diretório que contém o arquivo apagado, atualizando seu tamanho e seus links.

A função fs_mkdir cria um diretório no caminho passado por dname. Utilizamos a função find_block para verificar se o diretório já existe, e também para verificar se o diretório pai, onde será criada a nova pasta também existe. Após isso, alocamos dois blocos para o inode e nodeinfo da pasta a ser criada, atualizamos os dados em cada bloco, atualizamos os dados do nodeinfo do diretório pai e escrevemos os dados no sistema de arquivos.

A função fs_rmdir procura a pasta a ser deletada e o diretório pai utilizando a função find_block. Após isso, ela deleta o nodeinfo da pasta, deleta o inode da pasta, e atualiza as informações do diretório pai.

A função fs_list_dir percorre os arquivos do diretório passado como parâmetro em dname imprimindo-os na tela. Isso foi feito percorrendo os links[i] do diretório e lendo o inode de cada arquivo ou pasta dentro do diretório dname. Após isso, pegamos o nome completo de cada arquivo ou pasta (por exemplo, /home/user/desktop/file1) e extraímos apenas a última parte do nome (no exemplo, file1). Se esse for uma pasta, é concatenado ao final do nome o caractere “/”. Ao final, concateno o nome da pasta ou arquivo a um vetor de char que será retornado, utilizando um espaço entre o nome de cada arquivo.