Mestrado Integrado em Engenharia Informática
Fundamentos de Sistemas de Operação
Ano Lectivo 2013/2014 – 1º Semestre

Relatório do trabalho prático nº 2

Primeiro tópico: Implementação

O objetivo deste trabalho era desenvolver uma simulação de um sistema de ficheiros *fs-miei01* baseado em *inodes*, típico ao utilizado por um sistema operativo UNIX/LINUX. O nosso sistema de ficheiros, embora simples, possui diversas funções para armazenar e realizar operações sobre uma determinada quantidade de ficheiros.

O que nos era proposto no enunciado do trabalho era implementar as funções do ficheiro "*fs.c*", sendo que nos eram fornecidos os ficheiros referentes à simulação do disco "*disk.h*" e "*disk.c*" e os ficheiros relativos à interpretação de comandos, "*shell.h*" e "*shell.c*", de modo a interagir com o sistema de ficheiros "*fs.h"*. Tivémos também acesso um conjunto de imagens fornecidas no CLIP para testar a execução do nosso programa.

Para garantir o funcionamento do nosso sistema de ficheiros foi necessário adoptar uma estratégia típica de um sistema de ficheiros: A criação de um *bitmap* para ser possível saber quais os blocos de dados que estão ocupados por ficheiros. Esse *bitmap* é criado sempre que se monta o disco e possui um número de entradas igual ao número de blocos existentes no disco, sendo que as entradas que correspondem a blocos de dados livres são colocadas a '0' e as que correspondem a blocos ocupados são colocadas a '1'. O *bitmap* sofre alterações nas suas entradas sempre que se realiza alguma operação que envolva alocar ou libertar blocos de dados do disco para algum ficheiro, nomeadamente nas funções *fs_delete* e *fs_write*.

O sistema de ficheiros implementado é baseado numa atribuição de blocos indexada por *inodes.* Esses *inodes* possuem vários atributos a considerar, nomeadamente o tamanho do ficheiro a que são referentes, a validade do *inode* e os seus endereçamentos relativamente a blocos de dados do disco. Considerámos ainda que quando os endereçamentos dos *inodes* estivessem com um valor igual a '0' seria quando os mesmos se referissem a um ficheiro com zero *bytes,* ou seja, um ficheiro vazio.

Segue em baixo um exemplo de como funciona o nosso sistema de ficheiros *fs-miei01* para uma das imagens de teste (*image.5* simulada com 5 blocos), incluindo a representação do *bitmap* de blocos ocupados:

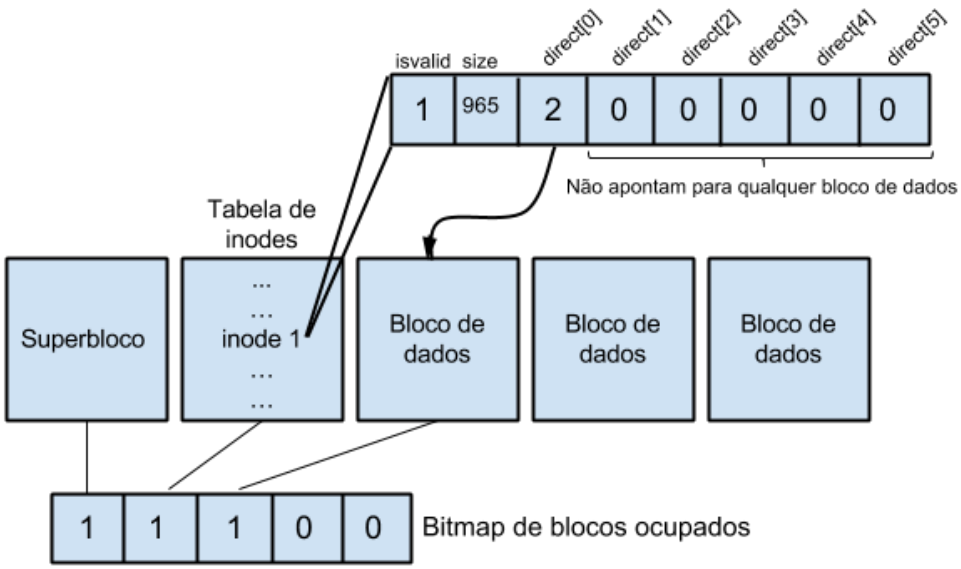

Fig.1 – Conteúdos da imagem 5 após o disco ser montado.

É possível referir que na atribuição de blocos de dados a um determinado *inode*, a mesma é feita de forma contígua sempre que possível (isto é realizado ao percorrer o *bitmap* iterativamente). No entanto, se durante a atribuição de blocos a um *inode*, a posição seguinte correspondente ao próximo bloco de dados livre se encontrar ocupada, então ter-se-á que continuar a percorrê-lo até se encontrar a próxima posição livre, criando situações em que os blocos de dados de um ficheiro não estarão imediatamente uns a seguir aos outros no disco, mas sim distribuídos por vários locais do mesmo.

Para realizarmos este trabalho baseámo-nos nos conceitos apresentados pelo livro da cadeira "*Operating Systems: Three Easy Pieces*".

Descrição da implementação feita por cada operação:

fs_debug() – Lê o superbloco do disco e verifica se o número mágico do superbloco está correcto, averiguando assim se o sistema de ficheiros está correcto. Se isto se verificar imprime na consola o número de blocos do disco, número de blocos ocupados pela tabela de *inodes* e número total de *inodes*. Posteriormente, percorre a tabela de *inodes* lendo cada bloco da mesma no disco. Em cada bloco da tabela irá então verificar todas as suas entradas e imprimir na consola as informações referentes a cada *inode* válido. Caso não se verifique a condição descrita, o método devolve uma mensagem de erro e termina.

fs_format() – Começa por formatar os valores do superbloco, escrevendo sobre este, alocando o número de blocos necessários para a tabela de *inodes*. Depois, sobre cada bloco da tabela de *inodes*, coloca o *bit* de validade de cada *inode* a '0', indicando que este já não se encontra ocupado, e escreve o bloco em disco. Falha caso o disco se encontre montado.

fs_mount() – Lê o superbloco do disco e verifica se o número mágico corresponde ao do sistema de ficheiros e se o disco não se encontra montado; caso alguma condição destas não se comprove a função termina retornando uma mensagem de erro. Cria então um *bitmap* de blocos ocupados com todas as entradas preenchidas a '0'. Marca as posições do superbloco e dos blocos da tabela de *inodes* como ocupadas no *bitmap*. Finalmente, lê os blocos da tabela de *inodes* um a um, percorrendo todos as entradas, e para cada *inode* percorre todos os seus endereços directos, procurando por blocos de dados endereçados pelo mesmo (com o valor do endereço diferente de '0'); os blocos cujas posições estão endereçadas por um *inode* são então marcados no *bitmap*.

fs_create() – Se o disco ainda não tiver sido montado é retornada uma mensagem de erro e a função termina. Caso contrário lê-se o superbloco e percorrem-se todos os blocos da tabela de *inodes*, lendo bloco a bloco do disco. Dentro de cada bloco procura-se por *inodes* que estejam livres. A partir do momento em que se encontra o primeiro *inode* inválido, marca-se o mesmo como válido e chama-se a função *reinitialize_inode*.

fs_delete(int inumber) – Se o disco ainda não tiver sido montado é retornada uma mensagem de erro e a função termina. Se existir, utilizamos três funções auxiliares que criámos, *inode_load* (a função termina e retorna erro caso o *inode* não seja válido) e *reinitialize_inode* de modo a que encontremos o *inode* procurado e que possamos limpar todos os endereços do mesmo, libertando os blocos do bitmap associados a esses endereços . Após estas operações o *inode* marcamos então o *inode* como inválido e é então gravado através da função *inode_save*.

fs_getsize(int inumber) – Se o disco ainda não tiver sido montado é retornada uma mensagem de erro e a função termina. Caso contrário encontra o *inode* procurado, pela função *inode_load*, e se o *inode* for válido retorna o seu tamanho, senão retorna uma mensagem de erro.

*fs_read(int inumber, char *data, int length, int offset)* – Carrega o *inode* pretendido através de *inode_load*, se o disco ainda não tiver sido montado, o deslocamento inserido exceder o tamanho máximo de um ficheiro em *bytes* ou o *inode* não for válido termina a execução da função, retornando um valor de erro. Caso contrário, calcula qual o bloco de dados a ler e lê-o do disco. Posteriormente calcula o máximo de *bytes* a ler na invocação actual e transfere o conteúdo do bloco de dados para o vector dado até completar esse máximo. Retorna então o número de *bytes* lidos.

*fs_write(int inumber, const char *data, int length, int offset)* – Carrega o *inode* pretendido através de *inode_load*, se o disco ainda não tiver sido montado, o deslocamento inserido exceder o tamanho máximo de um ficheiro em *bytes* ou o *inode* não for válido, termina a execução da função, retornando um valor de erro. Caso o *inode* tenha blocos de dados endereçados chama a função auxiliar *reinitialize_inode*. Caso contrário, calcula qual o bloco de dados inicial para escrever no mesmo. Após isto, inicia uma escrita do conteúdo do vector dado para o bloco de dados *byte* a *byte*, passando para o bloco de dados seguinte sempre que for necessário e escrevendo em disco sempre que tal acontecer. Por fim adiciona o número de *bytes* escritos ao tamanho do *inode* e grava-o em disco, devolvendo o número de *bytes* escritos.

Descrição da implementação das funções auxiliares:

*void inode_load(int inumber, struct fs_inode *inode)* – Calcula o bloco da tabela de *inodes* e a posição do *inode* dentro do bloco onde o mesmo é suposto se encontrar. Após este cálculo, lê-se o bloco correspondente do disco e guarda-se o *inode* pretendido.

*void inode_save(int inumber, struct fs_inode *inode)* – Calcula o bloco da tabela de *inodes* e a posição do *inode* dentro do bloco onde o mesmo é suposto se encontrar. Após este cálculo, lê-se o bloco correspondente do disco, altera os valores do *inode* anteriormente presente em disco para os valores do novo *inode* e volta-se a escrever o bloco em disco.

*void reinitialize_inode(struct fs_inode *inode)* – Coloca o tamanho do *inode* pretendido a '0' e percorre os endereços dos mesmos actualizando o *bitmap* de modo a que as posições dos blocos de dados antes referenciados possam ficar livres e apagando esse valor do endereço do *inode*.

int get_free_datablock() – Percorre o *bitmap* procurando por uma posição livre que corresponda a um bloco de dados. Se encontrar, sinaliza essa posição do *bitmap* como ocupada e devolve a mesma. Caso não encontre, retorna um valor de erro, sinalizando que o disco está cheio.

Segundo tópico: Suposições feitas

1. No ficheiro "*shell.c*" fornecido para este trabalho prático encontrava-se o que a gente supôs ser uma gralha, pois na invocação da função *fs_delete*, os valores de retorno descritos no enunciado não correspondiam aos que eram interpretados pela *shell*, causando uma mensagem de erro sempre que se tentava eliminar um *inode*. Neste ponto, mantivémo-nos fiéis ao enunciado na parte do retorno dos valores da função e alterámos a condição no ficheiro "*shell.c*" (de "*if(fs_delete(inumber))*" passou para *"if(fs_delete(inumber) == 0)"*).

2. Na função *fs_write* é referido que só se pode gravar o conteúdo de um ficheiro para um *inode* previamente inicializado. Assumimos que por *inode* inicializado se entende qualquer *inode*, tenha este já blocos de dados endereçados ou não, desde que o seu *bit* de validade se encontre a '1'. Através desta suposição definimos que se o *inode* já tivesse alguns blocos de dados endereçados eliminaríamos esses endereçamentos e reinicializávamos o valor do tamanho do ficheiro a '0'.

Terceiro tópico: Possíveis limitações

1. Na nossa implementação do sistema de ficheiros não adoptámos o uso de um *bitmap* de *inodes*, isto implica que cada vez que se pretende verificar se um dado ficheiro existe no disco, é necessário percorrer os blocos da tabela de *inodes* e percorrer os *inodes* para o encontrar. Algo que se torna menos eficiente quando comparado com um sistema que utiliza um *bitmap* para tal.

2. O sistema de ficheiros implementado tem implicitamente capacidade para suportar ficheiros de 24K's, pois tem apenas capacidade para endereçar 6 blocos de dados com 4K's cada um. Logo, todos os testes realizados sobre o sistema foram sempre concluídos com ficheiros com um tamanho inferior a esse valor.

3. Aquando a criação do *bitmap* de blocos ocupados do disco, é feita a alocação do mesmo no *heap* do processo. Esta alocação mantém-se ao longo da execução do programa, nunca sendo libertado o espaço atribuído ao *bitmap*. Para evitar consumos desnecessários de memória física, criámos uma condição extra para a função *fs_mount* de modo a evitar que se possa chamar a mesma com sucesso mais que uma vez. Desta forma, garantimos que a alocação só é realizada uma vez por cada instância de execução do programa. O espaço em questão é libertado pelo sistema operativo após terminação da execução do programa.

Realizado pelos alunos Francisco Godinho (nº 41611) e Pedro Carolina (nº 41665) do turno prático 5.