INF01142: Sistemas Operacionais I N

Trabalho #2

01 de julho de 2014

Fernando Bombardelli da Silva(218324), William Bombardelli da Silva(218323)

Questão 1

Desconsiderando a quantidade de ponteiros de alocação indexada, o tamanho do bloco (BlockSize) do sistema de arquivos influencia no maior tamanho de arquivo T2FS possível. Quanto maior o tamanho de bloco, mais dados cabem nos blocos apontados diretamente pelo registro de diretório (campo dataPtr do t2fs_record) e mais blocos são apontados pelos ponteiros de indireção (pois os blocos de índice são maiores).

Questão 2

Para criar *hardlinks* é necessário criar um novo tipo de registro de diretório (nova possibilidade para o campo TypeVal do t2fs_record); alterar a função de criação de arquivo (t2fs_create); e naturalmente alterar a função de navegação nos diretórios, para que seja possível usar o *hardlink*.

Questão 3

Alguns dos elementos redundante são:

- 1: A relação entre o tamanho do arquivo em bytes (campo bytesFileSize de t2fs_record) e a quantidade de blocos usados (campo blocksFileSize de t2fs_record).
- 2: A relação entre a quantidade de blocos realmente usadas (verificadas através da contagem dos blocos de um arquivo) e o valor no campo blockFileSize de t2fs_record.
- 3: A relação entre a quantidade de blocos de um arquivo (campo blocksFileSize de t2fs_record) e a quantidade de blocos da partição (campo NofBlocks de t2fs_superbloco).
- 4: A relação entre o tamanho de um arquivo (campo bytesFileSize de t2fs_record) e o tamanho da partição (campo DiskSize de t2fs_superbloco).
- 5: A relação entre a quantidade de espaço ocupado indicado pelo bitmap e a quantidade real de espaço ocupado (contagem dos ponteiros para blocos do sistema de arquivos).

Poderia-se criar métodos de verificação em cada um dos itens enumerados acima, que seria chamado a cada operação e verificaria a consistência do sistema de arquivos. Em especial, para o item 2 pode ser criado um utilitário de verificação (file system check), que contaria a quantidade de blocos de cada arquivo e verificaria a consistencia dos valores no registro.

Questão 4

Utilizou-se duas *arrays* para controlar os arquivos abertos:

- *openRecords*: Contem os registros abertos, isto é, mantem as entradas de diretório abertas atualmente. Cada arquivo aberto contem apenas um registro na *openRecords*.

Questão 4 continuação na próxima página...

- *openFiles*: Contem as instâncias de arquivos abertos, isto é, guarda uma instância da estrutura *OpenFile*, que contem o índice do registro do arquivo em *openRecords* e a posição atual no arquivo. Um arquivo aberto pode ter mais de uma entrada em *openFiles*.

```
typedef struct s_OpenFile{
        int recordIndex;
    unsigned int currentPosition;
} OpenFile;
```

Notar que esta organização possibilita a abertura de um arquivo simultaneamente por mais de um usuário. Criou-se uma função (FS_createHandle) utilizada por ambas funções t2fs_create e t2fs_open. A FS_createHandle cria uma entrada em *openRecords* para o arquivo requisitado (se este já não estiver aberto) e cria uma entrada em *openFiles* que aponta para o *openRecord* do arquivo requisitado. Retorna o índice da nova entrada de *openFiles* em caso de sucesso.

Questão 5

Para controlar a posição do ponteiro no arquivo, basta acessar a entrada desejada em *openFiles* e acessar o campo *currentPosition*. A função *t2fs_seek* atribui um novo valor para este campo. Ou posiciona para o fim do arquivo se *offset = -1*.

Questão 6

Como o t2fs_record está em memória (mantido em *openRecords*), não é necessário navegar na estrutura de arquivos, porém é necessário carregar o bloco no qual se deseja escrever. Para isso diversas leituras de blocos de indireção podem ocorrer.
No caso do bloco em que se deseja escrever não existe no disco (p.e. aumento de um arquivo), é necessário criar um bloco de dados novo, e possivelmente criar novos blocos de indireção ou atualizar a entrada de diretório do arquivo (p.e. atualização do campo *dataPtr*). O sistema cria primeiramente o bloco de arquivo, depois os blocos de indireção (se houver necessidade) e finalmente, se todos os blocos foram criados com sucesso, atualiza a entrada de diretório do arquivo desejado.
Neste segundo caso, pode ocorrer inconsistência dos dados em disco pelo fato de, durante o processo, vários blocos de dados ou blocos de indireção podem ter sido criados e não terem sido "ligados"entre si, pois a sequência fora interrompida entre o salvamento de um bloco e outro.
Supor, por exemplo, que um novo bloco de dados suposto a ser apontado pelo *dataPtr* de uma entrada de diretório é criado e salvo no disco e, então, ocorre queda de energia. O bloco agora ocupa uma posição no disco (pois requisitou espaço ao bitmap) e não é apontado pela entrada de diretório do arquivo.
Portanto, ocorre inconsistência entre os dados do bitmap e a verdadeira quantidade de blocos utilizados. Para minimizar este problema, deve-se minimizar o tempo entre a alocação do bloco de dados e a atualização do registro de dados do arquivo.

Questão 7

Apenas os registros de diretório (t2fs_record) abertos são mantidos e gerenciados em memória. Porém, cada alteração destes dados são replicados para o disco no momento da alteração. Assim, se esses dados guardados em memória fossem vistos como uma cache, a política de escrita seria write-through.
Esta escolha foi feita para facilitar a implementação do sistema de arquivos, pois elimina a necessidade de implementação de politicas sofisticadas de controle de atualização e aumenta a segurança, diminuindo chance perda de dados.

Questão 8

Todas as funções implementadas funcionam corretamente de acordo com a especificação. Foi desenvolvida uma suíte de teste unitário para cada módulo principal do sistema. Cada suíte de testes unitários testa cada função do módulo. A estratégia para construção dos casos de teste foi o whitebox e blackbox, isto é, baseia-se no código-fonte e na especificação. Os códigos-fonte dos testes estão na pasta *teste*.

Questão 9

A primeira dificuldade encontrada está na dificuldade de modelagem do sistema, isto é, a dificuldade de abstrair o problema − que é tipicamente de baixo nível − em um sistema de alto nível que seja capaz de resolvê-lo de maneira a permitir a reusabilidade do código. A implementação do sistema também mostra-se complexa pelas estruturas de dados do sistema de arquivos, que não permitem a fácil reusabilidade do código, criando assim grande reescrita de código.
Por estes e outros motivos a implementação apresenta grande volume de código escrito, o que representa muito trabalho braçal e custoso, aumentando consideravelmente o tempo necessário para cumprir todos os requisitos da especificação.