Universidade Federal do Rio Grande do Sul
Instituto de Informática

Bacharelado em Ciência da Computação
INF01142 – Sistemas Operacionais IN
Prof. Sergio Luis Cechin

Relatório do Trabalho 2
Implementação de um Sistema de Arquivos T2FS

Lisandro Procedi
Otávio Carvalho
Tagline Treichel

Porto Alegre, Julho de 2014

1 - Introdução

Nesse trabalho, construímos um sistema de arquivos virtual, que interage com um disco virtual, fornecido pelo professor, e que realiza diversas operações. Dentre elas, podemos destacar as seguintes operações sobre arquivos: Abertura, Fechamento Criação, Deleção, Escrita, Leitura e Alteração do posicionamento do ponteiro para leitura e escrita.

A construção do sistema de arquivos baseia-se na sua integração com o disco virtual, com o qual interagimos atráves de funções de leituras e escritas de setores, que devem ser mapeados para uma estrutura virtual de blocos (também denominados clusteres em alguns sistemas de arquivos). A implementação considera até 20 arquivos abertos simultaneamente. Isto é, podemos manter em memória, para manipulação, até 20 arquivos diferentes.

Para realizar a manipulação dos arquivos abertos, foi construída uma estrutura do tipo **file**, que guarda algumas informações chave, como a posição do ponteiro para leitura/escrita em um dado instante de tempo. Esta estrutura possui proporciona a criação de uma lista encadeada de elementos desse tipo. Dessa forma, podemos manter os 20 arquivos abertos ao mesmo tempo.

A implementação começou pelas funções **t2fs_identify**, que é a mais simples do sistema e apenas retorna o nome dos componentes. Aproveitamos a sua existência para testar a inicialização das variáveis globais através da leitura dos dados do ***superblock***.

Em seguida, foram implementadas as funções **t2fs_create**, **t2fs_open**, **t2fs_close** e **t2fs_delete**, que demandaram uma atenção maior, devido às inconsistências que elas podem criar no diretório e na área de bitmap.

Por fim, foram implementadas as funções **t2fs_write**, **t2fs_read** e **t2fs_seek**, que apresentaram as maiores dificuldades de implementação, devido ao mapeamento em setores em blocos, e os casos complexos que a alocação indexada causa, onde deve ser tratadas a escrita em 2 níveis diferentes de abstração.

2 - Questionário

2.1 - Sem alterar a quantidade de ponteiros de alocação indexada, quais outros fatores influenciam no maior tamanho de arquivo T2FS possível? Como esses fatores influenciam nesse tamanho?

Uma vez que a estrutura é formada em 2 níveis, com um ponteiro para blocos diretos e um para bloco de índices que apontam para outros índices, e essa estrutura segue até terminar de indexar o tamanho máximo do disco, as únicas variantes que podem influenciar são: o tamanho dos blocos, o tamanho do superbloco e o tamanho dos registros que são indexados por esses blocos, no nosso caso a estrutura t2fs_record. Dentre outros fatores menores, podemos citar o tamanho em bytes dos ponteiros e a forma como é representado o bitmap.

2.2 - Supondo que você desejasse melhorar o T2FS, permitindo a criação de vínculos estritos (hardlinks). Que alterações seriam necessárias no T2FS? Há necessidade da criação de novas funções? Se sim, quais? Se não, porque não.

Seria possível a implementação de *hardlinks* no T2FS, mas deveríamos adicionar um contador de referências (*links*) para cada arquivo, adicionando um contador na estrutura t2fs_record.

Como o sistema adota uma estratégia Unix-like, não precisam ser feitas grandes modificações, uma vez que ele trata todos os arquivos e diretórios como arquivos, desalocando os arquivos somente quando não existem mais ponteiros apontando para eles, através da remoção das referências a eles no bitmap, e então seus antigos blocos podem ser sobreescritos novamente.

2.3 - As estruturas de controle do T2FS contêm informações que permitem verificar a consistência de alguns de seus elementos. Isso é possível graças a um nível de redundância de informação (por exemplo, no registro de arquivo, nas entradas do diretório, o número total de blocos usados por um arquivo e o tamanho do arquivo – em bytes – permitem uma verificação). Identifique quais outros elementos são redundantes e discuta como seria possível usar essa redundância para aumentar a confiabilidade do T2FS.

A redundância dá robustez ao sistema no sentido de que as informações estão disponíveis para conferência de diversas maneiras, e podem ser obtidas de forma mais direta, ou indireta, dependendo de que área do sistema estamos tratando.

Por exemplo, na área de superbloco vem descritos os tamanhos dos blocos, bem como outras áreas do sistema, o que se torna redundante, uma vez que é possível mensurá-las bit a bit, mas isso é menos direto do que ter essas informações guardadas em uma área, para utilizar quando necessário. Através do bitmap, é possível inferir quantos blocos podem ser alocados diretamente no disco, bem como outras interações que acontecem em estruturas utilizadas ao longo do desenvolvimento das funções da API.

Esses elementos tornam o gerenciamento do sistema mais complexo, pois uma mudança em uma parte do sistema deve garantir em diversos outros lugares, para garantir que o sistema seja válido como um todo.

Por outro lado, essa redundância de informações pode causar complicações em situações limite, como é o caso das recuperações do sistema, quando o sistema sofre uma pane e reinicia, por exemplo. Podemos ter informações conflitantes em partes diferentes do sistema, na cache um bloco pode ter parado de escrever em um arquivo em um dado ponto, mas como essa informação não foi escrita no disco antes da pane, não podemos ter certeza de onde devemos proseguir ao reiniciar o sistema.

Para evitar esses problemas, podemos aumentar a complexidade do sistema realizando check-points de tempos em tempos, garantindo que as informações gravadas até um certo ponto são válidas caso ocorra uma falha. Podemos também, implementar mecanismos de rollback e jornalização, gerando um log a cada operação atômica efetuada sobre o disco. É possível também alterar o disco a cada alteração, eliminando a necessidade de uma cache em memória, mas uma implementação desse tipo acarretaria em um grande número de acessos ao disco e, por consequência, em uma diminuição no desempenho do sistema como um todo devido ao tempo gasto com I/O.

2.4 - Como você implementou a atribuição dos identificadores dearquivos (file handler) pelas funções t2fs_create e t2fs_open? Discuta a questão da reutilização dos mesmos.

Os mecanismos são armazenados em uma área alocada no heap, que utiliza uma lista encadeada que, por sua vez, são compostas pelo t2fs_record e demais variáveis, que refletem a sua situação em um dado instante de tempo (posição do ponteiro de leitura, dentre outras).

Essa implementação simplifica o acesso aos arquivos na memória, uma vez que todos estão disponíveis para o acesso uma vez carregados, e são gravados no disco uma vez criados. Por outro lado, no caso de situações inesperadas, como queda de energia, o sistema não teria mecanismos para se recuperar dessas falhas, e o estado dos arquivos que estão sendo manipulados seria perdido, e não poderia ser recuperado.

Do ponto de vista da implementação, essas estruturas são plenamente reutilizáveis, pois os elementos são adicionados ou eliminados da lista de acordo com a implementação das funções **t2fs_open** e **t2fs_close**, que controlam a adição ou remoção desses elementos na lista. É utilizado também um vetor de “bytemap”, que guarda posições de 0 à 20 com elementos binários, para consistir a alocação de no máximo 20 arquivos (distintos) em um dado instante de tempo.

2.5 - Como você implementou a gerência do contador de posição (current pointer) usado pela função t2fs_seek?

A função t2fs_seek é implementada de maneira simplificada, apenas atualizando a estrutura descrita na questão anterior, com a posição atual do ponteiro que determina a posição atual do offset, a partir da posição de início do arquivo. Portanto, uma leitura ou uma escrita atualizam o offset atual em memória, mas não no disco. Isto é feito somente no fechamento do arquivo através da **t2fs_close**, que salva efetivamente a quantidade de bytes escritos, em um determinado record, no disco.

2.6 - A escrita em um arquivo (realizada pela função t2fs_write) requer uma sequência de leituras e escritas de blocos de dados e de blocos de controle. Qual é a sequência usada por essa função? Se essa sequência for interrompida (por falta de energia, por exemplo) entre duas operações de escrita de bloco, qual será o efeito na consistência dos dados no disco? É possível projetar uma sequência de escritas no disco que minimize a eventual perda de dados?

A função de escrita em um arquivo funciona na seguinte sequência: Acesso ao arquivo; Definição do tamanho a ser escrito com relação ao limite máximo disponível (do sistema e do arquivo em questão); Escritas nos dois blocos apontados diretamente; Escritas nos blocos apontados indiretamente; E, por fim, escritas nos blocos apontados indiretamente por outros blocos de índice.

Essa sequência não é ótima, pois faz diversas escritas parciais no disco, sem poder garantir que todo o conteúdo seja escrito, na sua totalidade, no caso de essa sequência ser interrompida. Idealmente, poderíamos procurar implementar um sistema que utilizasse as técnicas de banco de dados, utilizando a ideia de transações para implementar sistemas jornalizados (baseados em log), dessa forma, escreveríamos todas as etapas de escrita necessárias em um arquivo e, ao final, efetuaríamos essas escritas de uma única vez, diminuindo o tempo em que o sistema ficaria exposto às falhas de inconsistência.

2.7 - Algumas estruturas gravadas no disco são mais facilmente manipuláveis se estiverem na memória principal (como se fosse uma cache). Por outro lado, isso aumenta a possibilidade de perda de dados, pois as informações existentes nessa cache e que não foram escritas no disco, podem ser perdidas, caso ocorra alguma interrupção de operação do sistema. Quais informações do disco você está mantendo (e gerenciando) na memória principal e porque você as escolheu? Qual a política que você usou para decidir quando escrevê-las no disco?

Como foi dito anteriormente, os mecanismos são armazenados em uma área alocada no heap, que utiliza um vetor de estruturas que, por sua vez, são compostas pelo t2fs_record e demais variáveis, que refletem a sua situação em um dado instante de tempo. Nessa estrutura consta o handle para o arquivo que está sendo mantido na memória, a posição corrente dentro arquivo (em bytes, para leitura/escrita/seek), a posição do arquivo com relação aos blocos e a posição do arquivo com relação ao início do seu bloco, bem como a sua estrutura t2fs_record.

A essas informações é aplicada uma política de escrita semelhante ao write-through, principalmente no caso da função de escrita no disco, pois uma vez que é atualizada a “cache” virtual, são atualizados os dados no disco conjuntamente. Essa forma de manipulação foi escolhida devido a simplicidade de implementação, principalmente devido ao número pequeno de arquivos que podem ser mantidos abertos em um mesmo instante de tempo (20 arquivos).

2.8 - Todas as funções implementadas funcionam corretamente? Relatepara cada uma das funções desenvolvidas, como elas foram testadas?

t2fs_identify: É a função de identificação, a mais simples do trabalho, foi testada apenas com uma chamada de função normal, que imprime na tela normalmente.

t2fs_create: Cria o arquivo e armazena o seu primeiro bloco no disco, está funcionando corretamente para os blocos endereçados pelos 2 ponteiros diretos. Foi testada estressando-se a criação do total de arquivos possíveis endereçados por estes ponteiros.

t2fs_open: Está funcionando corretamente, abre o arquivo, aloca na estrutura que simula uma cache e retorna os parâmetros da forma correta. Foi testada para abertura de múltiplos arquivos criados, até o máximo possível de 20 simultâneos.

t2fs_close: Função de fechamento do arquivo, que elimina a referência existente a um arquivo na estrutura auxiliar utilizada para simular uma “cache”. Está funcionando corretamente, elimina a referência da cache.

t2fs_delete: Deleta os arquivos corretamente, está funcionando apenas para os blocos endereçados pelos 2 ponteiros diretos. Foi testada estressando-se a criação do total de arquivos possíveis endereçados por estes ponteiros e sua subsequente deleção.

t2fs_write: Funciona adequadamente para os blocos de alocação direta, escrevendo no ponteiros apontados pela estrutura *t2fs_record*, até atingir o seu limite máximo de 2 blocos.

t2fs_read: Funciona adequadamente para os blocos de alocação direta, lendo os blocos apontados pela estrutura *t2fs_record*, até atingir o seu limite máximo de 2 blocos.

copy2t2: A função está completamente desenvolvida, foi testada para cópia de um arquivo de 2048 bytes e se comportou corretamente.

mkdirt2: A função cria uma nova pasta no caminho indicado, mas a função de escrita *t2fs_write* não está adaptada para alocar um novo bloco para a pasta ao escrever a primeira vez.

2.9 - Relate as suas maiores dificuldades no desenvolvimento deste trabalho e como elas foram contornadas.

As maiores dificuldades se deram com relação à compreensão do funcionamento da estrutura de mapeamento em blocos/setores, descrita através da especificação fornecida. Isto se deu através de exaustivos testes utilizando o arquivo de disco fornecido e larga leitura da documentação. Toda a implementação foi compreendida, até mesmo à implementação dos blocos de indireção simples e dupla, através de blocos de índice, mas que não foram efetivamente implementados. A extensão do trabalho também foi de difícil abrangência, uma vez que as possibilidades do sistema de arquivos, ainda que simulado, são grandes.

As principais dificuldades contornadas foram o mapeamento dos setores em blocos, que uma vez resolvidos podemos reutilizar através de todo o código.

As estruturas encadeadas que representam a cache em memória também forneceu um certo trabalho, mas uma vez implementada com um mapa para as áreas utilizadas, pôde ser reutilizada na t2fs_open e t2fs_close sem maiores problemas.

Finalmente, a maior complexidade se deu com relação à persistência das estruturas dos arquivos alterados no disco, uma vez que a estrutura do t2fs_record de um arquivo não faz referência à pasta na qual ele está localizado, o que precisou ser implementado.