Instituto Federal de Educação, Ciência e Tecnologia do Pará – Campus Belém

Disciplina: Sistemas Operacionais

Docente: Marcos Sadala

Curso: Tecnologia em Análise e Desenvolvimento de Sistemas

Discentes: Bruno Santana, Cassio Furtado, Emily Almeida, João Alves

Tema: Sistemas de arquivos

O armazenamento e recuperação de informações são necessários em diversas aplicações de computadores. No entanto, o espaço de endereçamento virtual utilizado por um processo é limitado em tamanho, o que pode ser insuficiente para aplicações que exigem grande capacidade de armazenamento, como sistemas corporativos e bancos de dados. Além disso, as informações armazenadas dentro desse espaço são perdidas quando o processo é encerrado, o que é inaceitável para muitas aplicações que requerem retenção de dados a longo prazo. A necessidade de permitir o acesso simultâneo a informações por vários processos também é um desafio. Para superar essas limitações, o armazenamento de longo prazo é tradicionalmente realizado por meio de discos magnéticos, mas unidades de estado sólido têm ganhado popularidade recentemente devido à sua confiabilidade e acesso rápido. Esses dispositivos podem ser vistos como sequências lineares de blocos de tamanho fixo, permitindo operações de leitura e escrita. No entanto, essas operações são inconvenientes em sistemas grandes e podem surgir várias questões, como localização de informações, controle de acesso e gerenciamento de blocos livres. A abstração do arquivo é uma nova solução para lidar com o armazenamento de informações em sistemas operacionais, assim como as abstrações de processos (e threads) e espaços de endereçamento. Os arquivos são unidades lógicas de informação criadas por processos, e um disco pode conter milhares ou até milhões deles, independentes uns dos outros. Os arquivos são usados para modelar o disco, diferentemente da RAM. Os processos podem ler arquivos existentes e criar novos, e as informações armazenadas em arquivos devem ser persistentes, ou seja, não devem ser afetadas pelo ciclo de vida dos processos. O sistema operacional gerencia os arquivos, incluindo sua estrutura, nomeação, acesso, proteção e implementação. O capítulo aborda a interface do usuário para arquivos e diretórios, detalhes sobre a implementação e gerenciamento do sistema de arquivos e exemplos de sistemas de arquivos reais.

Um arquivo é um mecanismo de abstração que permite armazenar e recuperar informações do disco, ocultando os detalhes de armazenamento e funcionamento dos discos. A nomeação de arquivos é uma característica crucial dos sistemas de arquivos, em que um processo pode criar um arquivo com um nome que persiste mesmo após a conclusão do processo, permitindo que outros processos o acessem. Os nomes de arquivos podem variar entre sistemas operacionais, mas geralmente são compostos por cadeias de letras, dígitos e caracteres especiais, com limites de tamanho e, em alguns casos, diferenciação entre maiúsculas e minúsculas. Além disso, as extensões de arquivo desempenham um papel importante em muitos sistemas operacionais, indicando o tipo de arquivo e influenciando o comportamento dos programas ao manipulá-los.

Em sistemas operacionais como o MS-DOS, as extensões de arquivo são opcionais e podem ser usadas para fornecer informações sobre o tipo de arquivo, enquanto em sistemas como o UNIX, as extensões podem ser arbitrárias ou seguir convenções. Algumas extensões de arquivo têm significados específicos e são reconhecidas pelo sistema operacional, associando-as a programas específicos. No Windows, as extensões de arquivo têm significado atribuído e ao clicar em um arquivo com determinada extensão, o programa associado a ela é iniciado para abrir o arquivo. Essas convenções são úteis quando um programa precisa lidar com diferentes tipos de arquivos, permitindo a identificação e processamento adequado com base nas extensões.

É importante destacar que diferentes sistemas operacionais possuem diferentes sistemas de arquivos, como FAT-16, FAT-32 e NTFS no caso do Windows, cada um com suas propriedades e recursos específicos. Além disso, existem extensões adicionais, como o sistema de arquivos exFAT, otimizado para flash drives e sistemas de arquivos grandes. O conhecimento desses conceitos de abstração de arquivos, nomeação, extensões e sistemas de arquivos é fundamental para compreender o funcionamento dos sistemas operacionais e a interação dos usuários com os arquivos.

Arquivos podem ser estruturados de diferentes maneiras, e uma abordagem comum é tratá-los como sequências de bytes desestruturadas. Nesse modelo, o sistema operacional não tem conhecimento ou preocupação com o conteúdo do arquivo, vendo-o apenas como uma sequência de bytes. A interpretação e significado dos dados são impostos por programas em nível de usuário. Essa abordagem oferece flexibilidade máxima, permitindo que os usuários coloquem qualquer tipo de informação nos arquivos e os nomeiem conforme desejarem. Tanto o UNIX quanto o Windows adotam essa abordagem, o que é especialmente útil para usuários que desejam realizar tarefas incomuns.

Outra forma de estruturação de arquivos é usando registros de tamanho fixo. Nesse modelo, um arquivo é uma sequência de registros, em que cada registro possui uma estrutura interna específica. A leitura de um arquivo retorna um registro, enquanto a escrita sobrepõe ou anexa um registro. Essa abordagem foi comumente usada em sistemas operacionais antigos, em que os arquivos eram baseados em cartões perfurados ou impressoras de linha. No entanto, atualmente, nenhum sistema de propósito geral utiliza esse modelo como principal forma de estruturação de arquivos.

Um terceiro tipo de estrutura de arquivo é a organização em forma de árvore de registros, em que cada registro contém um campo chave fixo. A árvore é ordenada por esse campo chave, permitindo uma busca rápida por uma chave específica. Nessa abordagem, a operação principal não é obter o próximo registro, mas sim aquele com uma chave específica. O sistema operacional é responsável por decidir onde adicionar novos registros. Esse tipo de estrutura é comumente usado em computadores de grande porte para processamento de dados comerciais e difere significativamente das sequências de bytes desestruturadas usadas no UNIX e no Windows.

Essas diferentes formas de estruturação de arquivos fornecem opções para lidar com informações de maneiras específicas, desde a flexibilidade máxima dos arquivos como sequências de bytes desestruturadas até a organização em árvore de registros para busca eficiente por chaves específicas. Cada modelo possui vantagens e é aplicável em contextos distintos, dependendo dos requisitos e finalidades do sistema operacional e dos usuários.

Muitos sistemas operacionais, como UNIX e Windows, suportam vários tipos de arquivos. Entre eles, encontramos arquivos regulares, diretórios, arquivos especiais de caracteres e arquivos especiais de blocos. Arquivos regulares são utilizados para armazenar informações do usuário, enquanto os diretórios mantêm a estrutura do sistema de arquivos. Em relação aos arquivos especiais de caracteres, são voltados para a modelagem de dispositivos de entrada/saída serial, como terminais, impressoras e redes, enquanto os arquivos especiais de blocos são usados para representar discos. Neste contexto, nosso foco estará nos arquivos regulares, que podem ser do tipo ASCII ou binários.

Arquivos ASCII são compostos por linhas de texto, podendo ser exibidos, impressos e editados facilmente com qualquer editor de texto. Eles são especialmente úteis quando diversos programas utilizam arquivos ASCII para entrada e saída, permitindo conectar a saída de um programa à entrada de outro por meio de pipelines. Já os arquivos binários não são legíveis em sua forma bruta e possuem uma estrutura interna conhecida pelos programas que os utilizam. Por exemplo, um arquivo binário executável contém seções como cabeçalho, texto, dados, bits de realocação e tabela de símbolos, necessários para o sistema operacional executá-lo corretamente. Outro exemplo de arquivo binário é um repositório que consiste em módulos compilados, cada um com um cabeçalho contendo informações como nome, data de criação, proprietário, código de proteção e tamanho.

Embora os diferentes tipos de arquivos sejam úteis em suas respectivas aplicações, a tipagem rígida de arquivos pode causar problemas quando os usuários realizam ações que não são esperadas pelos projetistas do sistema. Por exemplo, se um usuário escrever um programa formatador que transforma um arquivo .c (programa C) em um layout específico e salvar o arquivo transformado com uma extensão diferente, como .dat, ele pode enfrentar problemas ao tentar compilá-lo, já que o sistema reconhece a extensão errada. Essas restrições impostas pelo sistema operacional podem ser úteis para usuários iniciantes, mas podem ser um obstáculo para usuários experientes, que precisam contornar as limitações do sistema para realizar suas tarefas de forma eficiente.

Antigamente, os sistemas operacionais ofereciam apenas acesso sequencial aos arquivos, o que significa que os processos podiam ler os bytes ou registros em ordem, começando do início, mas não podiam pular ou ler os registros fora de ordem. No entanto, os arquivos sequenciais podiam ser reiniciados do início, permitindo que fossem lidos várias vezes. Esse tipo de acesso era adequado para armazenamento em fita magnética. No entanto, com a introdução dos discos como meio de armazenamento, tornou-se possível ler os bytes ou registros de um arquivo fora de ordem, ou acessá-los por meio de uma chave em vez da posição. Essa capacidade de leitura não sequencial ou acesso aleatório é essencial para muitas aplicações, como sistemas de bancos de dados.

Os arquivos de acesso aleatório são cruciais para aplicações em que é necessário acessar registros específicos sem ter que ler todos os outros registros anteriores. Por exemplo, em um sistema de reservas de companhias aéreas, quando um cliente liga para reservar um assento em um voo específico, o programa de reservas deve ser capaz de acessar diretamente o registro desse voo, sem a necessidade de ler todos os registros dos outros voos. Existem dois métodos para especificar onde iniciar a leitura em arquivos de acesso aleatório. No primeiro método, cada operação fornece a posição no arquivo em que a leitura deve começar. No segundo método, uma operação simples é usada para estabelecer uma posição atual, a partir da qual o arquivo pode ser lido sequencialmente. O segundo método é adotado pelos sistemas operacionais UNIX e Windows.

Cada arquivo possui um nome e uma data associada a ele. Além disso, os sistemas operacionais atribuem outras informações a cada arquivo, como a data e hora da última modificação e o tamanho do arquivo. Essas informações adicionais são chamadas de atributos do arquivo ou metadados. A lista de atributos varia de um sistema para outro, mas alguns exemplos incluem informações de proteção, como permissões de acesso e senhas, e sinalizações que controlam propriedades específicas, como se o arquivo está oculto ou se foi feito um backup recente do arquivo.

Em arquivos que podem ser lidos usando uma chave, são encontrados atributos relacionados ao tamanho do registro, posição da chave e tamanho dos campos-chave. Essas informações são necessárias para localizar e acessar os registros usando as chaves. Além disso, os atributos de tempo indicam quando o arquivo foi criado, acessado e modificado pela última vez, o que é útil para determinar se um arquivo precisa ser recompilado, por exemplo. O tamanho atual do arquivo indica seu tamanho atualmente, e em sistemas antigos era necessário especificar o tamanho máximo do arquivo no momento da criação para reservar memória antecipadamente, mas sistemas mais modernos não exigem esse atributo.

Arquivos existem para armazenar informações e permitir que elas sejam recuperadas depois. Sistemas diferentes proporcionam operações diferentes para permitir armazenamento e recuperação. A seguir uma discussão das chamadas de sistema mais comuns relativas a arquivos.

* Create: Cria um arquivo sem dados, estabelecendo seus atributos.
* Delete: Remove um arquivo que não é mais necessário para liberar espaço no disco.
* Open: Abre um arquivo, permitindo que o sistema busque seus atributos e lista de endereços no disco para acesso rápido posterior.
* Close: Fecha um arquivo quando todos os acessos são concluídos, liberando espaço na tabela interna do sistema.
* Read: Lê dados do arquivo, normalmente a partir da posição atual, com a quantidade de dados especificada pelo chamador e colocando-os em um buffer fornecido.
* Write: Escreve dados no arquivo, normalmente na posição atual. Se a posição atual for o final do arquivo, o tamanho aumentará.
* Append: Acrescenta dados somente ao final do arquivo.
* Seek: Reposiciona o ponteiro de arquivo para uma posição específica, permitindo o acesso aleatório aos dados do arquivo.
* Get attributes: Lê os atributos de um arquivo, como momentos de alteração, para que um processo possa realizar determinadas tarefas.
* Set attributes: Altera atributos do arquivo, como o modo de proteção.
* Rename: Renomeia um arquivo, permitindo que o usuário mude seu nome.

4.3.2 Implementando arquivos

A questão mais importante na implementação de arquivos é controlar quais blocos vão com quais arquivos onde vários métodos são usados em diferentes sistemas operacionais.

Alocação contígua

O esquema de alocação mais simples é armazenar cada arquivo como uma execução contígua de blocos de disco. Assim, em um disco com blocos de 1 KB, um arquivo de 50 KB seria alocado em 50 blocos consecutivos.

A alocação de espaço de disco contíguo tem duas vantagens significativas. Primeiro, ela é simples de implementar porque basta se lembrar do endereço em disco do primeiro bloco e o número de blocos no arquivo. Segundo, o desempenho da leitura é excelente, pois o arquivo inteiro pode ser lido do disco em uma única operação onde apenas uma busca é necessária, então os dados são lidos com a capacidade total do disco.

Infelizmente, a alocação contígua tem um ponto fraco importante, com o tempo, o disco torna-se fragmentado. Quando um arquivo é removido, seus blocos são naturalmente liberados, deixando uma lacuna de blocos livres no disco. O disco não é compactado imediatamente para eliminá-la, já que isso envolveria copiar todos os blocos seguindo essa lacuna, potencialmente milhões de blocos, o que levaria horas ou mesmo dias.

De início, essa fragmentação não é problema, já que cada novo arquivo pode ser escrito ao final do disco. No entanto, quando o disco estiver cheio, será necessário compactá-lo ou reutilizar os espaços livres nas lacunas. Reutilizar o espaço exige manter uma lista de lacunas, o que é possível, no entanto, quando um arquivo novo for criado, é necessário saber o seu tamanho final a fim de escolher uma lacuna do tamanho correto para alocá-lo.

Imagine as consequências de um projeto desses. O usuário inicializa um processador de texto, a primeira coisa que o programa pergunta é quantos bytes o documento final terá. Se o número em última análise se provar pequeno demais, o programa precisará ser terminado prematuramente, pois não haverá lugar para colocar o resto do arquivo. Se o usuário tentar evitar esse problema dando um número irrealisticamente grande como o tamanho final, digamos, 1 GB, o editor talvez não consiga encontrar uma lacuna tão grande e anunciará que o arquivo não pode ser criado. É claro, o usuário estaria livre para inicializar o programa novamente e dizer 500 MB dessa vez, e assim por diante até que uma lacuna adequada fosse localizada. Ainda assim, é pouco provável que esse esquema deixe os usuários felizes.

A alocação contígua já foi usada nos sistemas de arquivos de discos magnéticos anos atrás pela simplicidade e alto desempenho. Então a ideia foi abandonada por causa do incômodo de ter de especificar o tamanho final do arquivo no momento de sua criação. Mas com o advento dos CD-ROMs, DVDs, Blu-rays e outras mídias óticas para escrita única, subitamente arquivos contíguos eram uma boa ideia de novo.

Alocação por lista encadeada

O segundo método para armazenar arquivos é manter cada um como uma lista encadeada de blocos de disco. A primeira palavra de cada bloco é usada como um ponteiro para a próxima. O resto do bloco é reservado para dados. Diferente da alocação contígua, todos os blocos do disco podem ser usados nesse método. Nenhum espaço é perdido para a fragmentação de disco. Também, para a entrada de diretório é suficiente armazenar meramente o endereço em disco do primeiro bloco. O resto pode ser encontrado a partir daí.

Por outro lado, embora a leitura de um arquivo sequencialmente seja algo direto, o acesso aleatório é de extrema lentidão. Para chegar ao bloco n, o sistema operacional precisa começar do início e ler os blocos n − 1 antes dele, um de cada vez. Também, a quantidade de dados que um bloco pode armazenar não é mais uma potência de dois, pois os ponteiros ocupam alguns bytes do bloco, embora não seja fatal, é menos eficiente. Com os primeiros bytes de cada bloco ocupados por um ponteiro para o próximo bloco, a leitura de todo o bloco exige que se adquira e concatene a informação de dois blocos de disco, o que gera uma sobrecarga extra por causa da cópia.

Alocação por lista encadeada usando uma tabela na memória

Ambas as desvantagens da alocação por lista encadeada podem ser eliminadas colocando-se as palavras do ponteiro de cada bloco de disco em uma tabela na memória. Usando essa organização, o bloco inteiro fica disponível para dados. Além disso, o acesso aleatório é muito mais fácil. Embora ainda seja necessário seguir o encadeamento para encontrar um determinado deslocamento dentro do arquivo, o encadeamento está inteiramente na memória, portanto ele pode ser seguido sem fazer quaisquer referências ao disco.

A principal desvantagem desse método é que a tabela inteira precisa estar na memória o todo o tempo para fazê-la funcionar. Com um disco de 1 TB e um tamanho de bloco de 1 KB, a tabela precisa de 1 bilhão de entradas, uma para cada um dos 1 bilhão de blocos de disco. Cada entrada precisa ter no mínimo 3 bytes. Para aumentar a velocidade de consulta, elas deveriam ter 4 bytes. Desse modo, a tabela ocupará 3 GB ou 2,4 GB da memória principal o tempo inteiro.

I-nodes

Nosso último método para monitorar quais blocos pertencem a quais arquivos é associar cada arquivo a uma estrutura de dados, que lista os atributos e os endereços de disco dos blocos do disco. A grande vantagem desse esquema sobre os arquivos encadeados usando uma tabela na memória é que o i-node precisa estar na memória apenas quando o arquivo correspondente estiver aberto. Se cada i-node ocupa n bytes e um máximo de k arquivos puderem estar abertos simultaneamente, a memória total ocupada pelo arranjo contendo os i-nodes para os arquivos abertos é de apenas kn bytes.

Um problema com i-nodes é que se cada um tem espaço para um número fixo de endereços de disco, o que acontece quando um arquivo cresce além de seu limite? Uma solução é reservar o último endereço de disco para o endereço de um bloco contendo mais endereços de blocos de disco. Mais avançado ainda seria ter dois ou mais desses blocos contendo endereços de disco ou até blocos de disco apontando para outros blocos cheios de endereços.

4.3.3 Implementando diretórios

Antes que um arquivo possa ser lido, ele precisa ser aberto. O sistema operacional usa o nome do caminho fornecido pelo usuário para localizar a entrada de diretório no disco. A entrada de diretório fornece a informação necessária para encontrar os blocos de disco. A principal função do sistema de diretórios é mapear o nome do arquivo em ASCII na informação necessária para localizar os dados. Todo sistema de arquivos mantém vários atributos do arquivo, como o proprietário de cada um e seu momento de criação, e eles devem ser armazenados em algum lugar.

Para sistemas que usam i-nodes, outra possibilidade para armazenar os atributos é nos próprios i-nodes, em vez de nas entradas do diretório. Nesse caso, a entrada do diretório pode ser mais curta: apenas um nome de arquivo e um número de i-node.

Até o momento presumimos que os arquivos têm nomes curtos de tamanho fixo. No entanto, quase todos os sistemas operacionais modernos aceitam nomes de arquivos maiores e de tamanho variável. A abordagem mais simples é estabelecer um limite para o tamanho do nome dos arquivos, essa abordagem é simples, mas desperdiça muito espaço de diretório, já que poucos arquivos têm nomes tão longos.

Uma alternativa é abrir mão da ideia de que todas as entradas de diretório sejam do mesmo tamanho. Com esse método, cada entrada de diretório contém uma porção fixa, começando com o tamanho da entrada e, então, seguido por dados com um formato fixo, normalmente incluindo o proprietário, momento de criação, informações de proteção e outros atributos.

Outra maneira de lidar com nomes de tamanhos variáveis é tornar fixos os tamanhos das próprias entradas de diretório e manter os nomes dos arquivos em um heap no fim de cada diretório, é claro, o heap deve ser gerenciado e faltas de páginas ainda podem ocorrer enquanto processando nomes de arquivos. A vantagem é que não há mais nenhuma necessidade real para que os nomes dos arquivos comecem junto aos limites das palavras.

Em todos os projetos apresentados até o momento, os diretórios são pesquisados linearmente do início ao fim quando o nome de um arquivo precisa ser procurado. Para diretórios extremamente longos, a busca linear pode ser lenta. Uma maneira de acelerar a busca é usar uma tabela de espalhamento em cada diretório. A entrada da tabela correspondendo ao código de espalhamento é verificada. Entradas de arquivo seguem a tabela de espalhamento. Se aquela vaga já estiver em uso, uma lista encadeada é construída, inicializada naquela entrada da tabela e unindo todas as entradas com o mesmo valor de espalhamento. A procura por um arquivo segue o mesmo procedimento.

Usar uma tabela de espalhamento tem a vantagem de uma busca muito mais rápida, mas a desvantagem de uma administração mais complexa. Ela é uma alternativa realmente séria apenas em sistemas em que é esperado que os diretórios contenham de modo rotineiro centenas ou milhares de arquivos.

4.3.4 Arquivos compartilhados

Quando vários usuários estão trabalhando juntos em um projeto, eles muitas vezes precisam compartilhar arquivos. Em consequência, muitas vezes é conveniente que um arquivo compartilhado apareça simultaneamente em diretórios diferentes pertencendo a usuários distintos.

Compartilhar arquivos é conveniente, mas também apresenta alguns problemas. Para começo de conversa, se os diretórios realmente contiverem endereços de disco, então uma cópia desses endereços terá de ser feita no diretório de um dos usuários quando o arquivo for ligado. Se esse usuário adicionar blocos ao arquivo, os novos blocos serão listados somente no diretório do usuário que estiver realizando a adição. As mudanças não serão visíveis ao outro usuário, derrotando então o propósito do compartilhamento.

Esse problema apresenta algumas soluções, uma delas é, os blocos de disco não são listados em diretórios, mas em uma pequena estrutura de dados associada com o arquivo em si. Os diretórios apontariam então apenas para a pequena estrutura de dados. Essa é a abordagem usada em UNIX.

Com ligações simbólicas esse problema não surge, pois somente o verdadeiro proprietário tem um ponteiro para o i-node. Os usuários que têm ligações para o arquivo possuem apenas nomes de caminhos, não ponteiros de i-node. Quando o proprietário remove o arquivo, ele é destruído. Tentativas de usar o arquivo por uma ligação simbólica fracassarão quando o sistema for incapaz de localizá-lo. Remover uma ligação simbólica não afeta o arquivo de maneira alguma.

O problema com ligações simbólicas é a sobrecarga extra necessária. O arquivo contendo o caminho deve ser lido, então ele deve ser analisado e seguido, componente a componente, até que o i-node seja alcançado. Toda essa atividade pode exigir um número considerável de acessos adicionais ao disco. Ligações simbólicas têm a vantagem de poderem ser usadas para ligar os arquivos em máquinas em qualquer parte no mundo, simplesmente fornecendo o endereço de rede da máquina onde o arquivo reside, além de seu caminho naquela máquina.

4.3.5 Sistemas de arquivos estruturados em diário (log)

Mudanças na tecnologia estão pressionando os sistemas de arquivos atuais. Em particular, CPUs estão ficando mais rápidas, discos tornam-se muito maiores e baratos e as memórias crescem exponencialmente em tamanho. O único parâmetro que não está se desenvolvendo de maneira tão acelerada é o tempo de busca dos discos.

A combinação desses fatores significa que um gargalo de desempenho está surgindo em muitos sistemas de arquivos. Pesquisas realizadas em Berkeley tentaram minimizar esse problema projetando um tipo completamente novo de sistema de arquivos, o LFS.

A ideia que impeliu o design do LFS é de que à medida que as CPUs ficam mais rápidas e as memórias RAM maiores, caches em disco também estão aumentando rapidamente. Em consequência, agora é possível satisfazer uma fração muito substancial de todas as solicitações de leitura diretamente da cache do sistema de arquivos. Segue dessa observação que, no futuro, a maior parte dos acessos ao disco será para escrita, então o mecanismo de leitura antecipada usado em alguns sistemas de arquivos para buscar blocos antes que eles sejam necessários não proporciona mais um desempenho significativo.

A partir desse raciocínio, os projetistas do LFS decidiram reimplementar o sistema de arquivos UNIX de maneira que fosse possível utilizar a largura total da banda do disco, mesmo diante de uma carga de trabalho consistindo em grande parte de pequenas operações de escrita aleatórias. A ideia básica é estruturar o disco inteiro como um grande diário (log).

De modo periódico, e quando há uma necessidade especial para isso, todas as operações de escrita pendentes armazenadas na memória são agrupadas em um único segmento e escritas para o disco como um único segmento contíguo ao fim do diário. Se o segmento médio puder ser feito com o tamanho de cerca de 1 MB, quase toda a largura de banda de disco poderá ser utilizada.

Neste projeto, i-nodes ainda existem e têm até a mesma estrutura que no UNIX, mas estão dispersos agora por todo o diário, em vez de ter uma posição fixa no disco. Mesmo assim, quando um i-node é localizado, a localização dos blocos acontece da maneira usual. É claro, encontrar um i-node é muito mais difícil agora, já que seu endereço não pode ser simplesmente calculado a partir do seu i-número, como no UNIX. Para tornar possível encontrar i-nodes, é mantido um mapa do i-node, indexado pelo i-número. O registro i nesse mapa aponta para o i-node i no disco. O mapa fica armazenado no disco, e também é mantido em cache, de maneira que as partes mais intensamente usadas estarão na memória a maior parte do tempo.

Se discos fossem infinitamente grandes, dariam conta de toda a história. No entanto, discos reais são finitos, então finalmente o diário ocupará o disco inteiro, momento em que nenhum segmento novo poderá ser escrito para o diário. Felizmente, muitos segmentos existentes podem ter blocos que não são mais necessários. Por exemplo, se um arquivo for sobrescrito, seu i-node apontará então para os blocos novos, mas os antigos ainda estarão ocupando espaço em segmentos escritos anteriormente.

Para lidar com esse problema, o LFS tem um thread limpador que passa o seu tempo escaneando o diário circularmente para compactá-lo. Ele começa lendo o resumo do primeiro segmento no diário para ver quais i-nodes e arquivos estão ali. Então confere o mapa do i-node atual para ver se os i-nodes ainda são atuais e se os blocos de arquivos ainda estão sendo usados. Os i-nodes e blocos que ainda estão sendo usados vão para a memória para serem escritos no próximo segmento. O segmento original é então marcado como disponível, de maneira que o arquivo pode usá-lo para novos dados. Dessa maneira, o limpador se movimenta ao longo do diário, removendo velhos segmentos do final e colocando quaisquer dados ativos na memória para serem reescritos no segmento seguinte.

4.3.6 Sistemas de arquivos journaling

Muitos sistemas de arquivos diferentes estão em uso mesmo para o mesmo sistema operacional. Um sistema Windows pode ter um sistema de arquivos NTFS principal, mas também uma antiga unidade ou partição FAT-16 ou FAT-32, que contenha dados antigos, porém ainda necessários. O Windows lida com esses sistemas de arquivos díspares identificando cada um com uma letra de unidade diferente, como em C:, D: Não há uma tentativa de integrar sistemas de arquivos heterogêneos em um todo unificado.

Em comparação, todos os sistemas UNIX fazem uma tentativa muito séria de integrar múltiplos sistemas de arquivos em uma única estrutura. Um sistema Linux pode ter o ext2 como o diretório-raiz, com a partição ext3 montada em /usr e um segundo disco rígido com o sistema de arquivos ReiserFS montado em /home. Do ponto de vista do usuário, existe uma hierarquia de sistema de arquivos única. O fato de ela lidar com múltiplos sistemas de arquivos não é visível para os usuários ou processos.

No entanto, a presença de múltiplos sistemas de arquivos é definitivamente visível à implementação, a maioria dos sistemas UNIX usou o conceito de um VFS para tentar integrar múltiplos sistemas de arquivos em uma estrutura ordeira. A ideia fundamental é abstrair a parte do sistema de arquivos que é comum a todos os sistemas de arquivos e colocar aquele código em uma camada separada que chama os sistemas de arquivos subjacentes para realmente gerenciar os dados.

O VFS também tem uma interface “inferior” para os sistemas de arquivos reais, que são rotulados de interface do VFS. Essa interface consiste em várias dúzias de chamadas de funções que os VFS podem fazer para cada sistema de arquivos para realizar o trabalho. Assim, para criar um novo sistema de arquivos que funcione com o VFS, os projetistas do novo sistema de arquivos devem certificar-se de que ele proporcione as chamadas de funções que o VFS exige.

Para compreender como o VFS funciona, vamos repassar um exemplo cronologicamente. Quando o sistema é inicializado, o sistema de arquivos raiz é registrado com o VFS. Além disso, quando outros sistemas de arquivos são montados, seja no momento da inicialização ou durante a operação, também devem registrar-se com o VFS. Quando um sistema de arquivos se registra, o que ele basicamente faz é fornecer uma lista de endereços das funções que o VFS exige, seja como um longo vetor de chamada (tabela) ou como vários deles, um por objeto de VFS, como demanda o VFS. Então, assim que um sistema de arquivos tenha se registrado com o VFS, este sabe como, digamos, ler um bloco a partir dele e modo similar, o VFS então também sabe como realizar todas as funções que o sistema de arquivos real deve fornecer: ele apenas chama a função cujo endereço foi fornecido quando o sistema de arquivos registrou.

Após um sistema de arquivos ter sido montado, ele pode ser usado. Por exemplo, se um sistema de arquivos foi montado em /usr e um processo fizer a chamada durante a análise do caminho, o VFS vê que um novo sistema de arquivos foi montado em /usr e localiza seu superbloco pesquisando a lista de superblocos de sistemas de arquivos montados. Tendo feito isso, ele pode encontrar o diretório-raiz do sistema de arquivos montado e examinar o caminho include/unistd.h ali. O VFS então cria um v-node e faz uma chamada para o sistema de arquivos real para retornar todas as informações no i-node do arquivo. Essa informação é copiada para o v-node (em RAM), junto com outras informações, e, o mais importante, cria o ponteiro para a tabela de funções para chamar operações em v-nodes, como read, write, close e assim por diante.

Mais tarde, quando o processo realiza um read usando o descritor de arquivos, o VFS localiza o v-node do processo e das tabelas de descritores de arquivos e segue o ponteiro até a tabela de funções, na qual estão os endereços dentro do sistema de arquivos real, no qual reside o arquivo solicitado. A função responsável pelo read é chamada agora e o código dentro do sistema de arquivos real vai e busca o bloco solicitado. O VFS não faz ideia se os dados estão vindo do disco local, um sistema de arquivos remoto através da rede, um pen-drive ou algo diferente. As estruturas de dados envolvidas são mostradas na Figura 4.19. Começando com o número do processo chamador e o descritor do arquivo, então o v-node, o ponteiro da função de leitura e a função de acesso dentro do sistema de arquivos real são localizados.

4.4 Gerenciamento e otimização de sistemas de arquivos

4.4.1 Gerenciamento de espaço em disco

Arquivos costumam ser armazenados em disco, portanto o gerenciamento de espaço de disco é uma preocupação importante para os projetistas de sistemas de arquivos. Duas estratégias gerais são possíveis para armazenar um arquivo de n bytes: ou são alocados n bytes consecutivos de espaço, ou o arquivo é dividido em uma série de blocos contíguos.

Como vimos, armazenar um arquivo como uma sequência contígua de bytes tem o problema óbvio de que se um arquivo crescer, ele talvez tenha de ser movido dentro do disco. O mesmo problema ocorre para segmentos na memória, por essa razão, quase todos os sistemas de arquivos os dividem em blocos de tamanho fixo que não precisam ser adjacentes.

Tamanho do bloco

Uma vez que tenha sido feita a opção de armazenar arquivos em blocos de tamanho fixo, a questão que surge é qual tamanho o bloco deve ter. Dado o modo como os discos são organizados, o setor, a trilha e o cilindro são candidatos óbvios para a unidade de alocação. Em um sistema de paginação, o tamanho da página também é um argumento importante.

Ter um tamanho de bloco grande significa que todos os arquivos, mesmo um de 1 byte, ocuparão um cilindro inteiro. Também significa que arquivos pequenos desperdiçam uma grande quantidade de espaço de disco. Por outro lado, um tamanho de bloco pequeno significa que a maioria dos arquivos ocupará múltiplos blocos e, desse modo, precisará de múltiplas buscas e atrasos rotacionais para lê-los, reduzindo o desempenho.

Tanenbaum et al. estudaram a distribuição do tamanho do arquivo no Departamento de Ciências de Computação de uma grande universidade de pesquisa em 1984 e então novamente em 2005, assim como em um servidor da web comercial hospedando um site de política. Os resultados foram que, para cada grupo, a porcentagem de todos os arquivos menores ou iguais ao tamanho foi listada.

Por exemplo, em 2005, 59,13% de todos os arquivos na Universidade de Vrije tinham 4 KB ou menos e 90,84% de todos eles, 64 KB ou menos. O tamanho de arquivo médio era de 2.475 bytes. Por um lado, com um tamanho de bloco de 1 KB, apenas em torno de 30-50% de todos os arquivos cabem em um único bloco, enquanto com um bloco de 4 KB, a percentagem de arquivos que cabem em um bloco sobe para a faixa de 60-70%. Outros dados no estudo mostram que com um bloco de 4 KB, 93% dos blocos do disco são usados por 10% dos maiores arquivos. Isso significa que o desperdício de espaço ao fim de cada pequeno arquivo é insignificante. Por outro lado, utilizar um pequeno bloco significa que cada arquivo consistirá em muitos blocos. Ler cada bloco exige uma busca e um atraso rotacional, então a leitura de um arquivo consistindo em muitos blocos pequenos será lenta.

Imagine uma curva tracejada que mostra a taxa de dados para um disco desses como uma função do tamanho do bloco e outra curva sólida que mostra eficiência de espaço como uma função do tamanho do bloco. Para calcular a eficiência de espaço, precisamos fazer uma suposição a respeito do tamanho médio do arquivo. Para simplificar, vamos presumir que todos os arquivos tenham 4 KB.

O tempo de acesso para um bloco é completamente dominado pelo tempo de busca e atraso rotacional, então levando-se em consideração que serão necessários 9 ms para acessar um bloco, quanto mais dados forem buscados, melhor. Assim, a taxa de dados cresce quase linearmente com o tamanho do bloco.

Agora considere a eficiência de espaço. Com arquivos de 4 KB e blocos de 1 KB, 2 KB ou 4 KB, os arquivos usam 4, 2 e 1 bloco, respectivamente, sem desperdício. Com um bloco de 8 KB e arquivos de 4 KB, a eficiência de espaço cai para 50%. Na realidade, poucos arquivos são um múltiplo exato do tamanho do bloco do disco, então algum espaço sempre é desperdiçado no último bloco de um arquivo.

O que as curvas mostram, no entanto, é que o desempenho e a utilização de espaço estão inerentemente em conflito. Pequenos blocos são ruins para o desempenho, mas bons para a utilização do espaço do disco. Para esses dados, não há equilíbrio que seja razoável. Historicamente, sistemas de arquivos escolheram tamanhos na faixa de 1 KB a 4 KB, mas com discos agora excedendo 1 TB, pode ser melhor aumentar o tamanho do bloco para 64 KB e aceitar o espaço de disco desperdiçado.

Em um experimento para ver se o uso de arquivos do Windows NT era apreciavelmente diferente do uso de arquivos do UNIX, Vogels tomou medidas nos arquivos na Universidade de Cornell. Ele observou que o uso de arquivos no NT é mais complicado que no UNIX. Vogels observou um tamanho médio (ponderado pelo uso) de arquivos apenas lidos como 1KB, arquivos apenas escritos como 2,3 KB e arquivos lidos e escritos como 4,2 KB. Considerando as diferentes técnicas de mensuração de conjuntos de dados, e o ano, esses resultados são certamente compatíveis com os da Universidade de Vrije.

Monitoramento dos blocos livres

Como monitorar os blocos livres: dois métodos são amplamente usados. O primeiro consiste em usar uma lista encadeada de blocos de disco, a outra técnica de gerenciamento de espaço livre é o mapa de bits. Um disco com n blocos exige um mapa de bits com n bits. Blocos livres são representados por 1s no mapa, blocos alocados por 0s (ou vice-versa).

Se os blocos livres tenderem a vir em longos conjuntos de blocos consecutivos, o sistema da lista de blocos livres pode ser modificado para controlar conjuntos de blocos em vez de blocos individuais. apenas um bloco de ponteiros precisa ser mantido na memória principal. Quando um arquivo é criado, os blocos necessários são tomados do bloco de ponteiros. Quando ele se esgota, um novo bloco de ponteiros é lido do disco. De modo similar, quando um arquivo é removido, seus blocos são liberados e adicionados ao bloco de ponteiros na memória principal. Quando esse bloco completa, ele é escrito no disco.

Cotas de disco

Para evitar que as pessoas exagerem no uso do espaço de disco, sistemas operacionais de múltiplos usuários muitas vezes fornecem um mecanismo para impor cotas de disco. A ideia é que o administrador do sistema designe a cada usuário uma cota máxima de arquivos e blocos, e o sistema operacional se certifique de que os usuários não excedam essa cota.

Quando um usuário tenta entrar no sistema, este examina o arquivo de cotas para ver se ele excedeu o limite flexível para o número de arquivos ou o número de blocos de disco. Se qualquer um dos limites foi violado, um aviso é exibido, e o contador de avisos restantes é reduzido para um. Se o contador chegar a zero, o usuário ignorou o aviso vezes demais, e não tem permissão para entrar.

Esse método tem a propriedade de que os usuários podem ir além de seus limites flexíveis durante uma sessão de uso, desde que removam o excesso antes de se desconectarem. Os limites estritos jamais podem ser excedidos.

4.4.2 Backups (cópias de segurança) do sistema de arquivos

Apesar de o sistema de arquivos não conseguir oferecer qualquer proteção contra a destruição física dos equipamentos e da mídia, ele pode ajudar a proteger as informações. Assim, a solução é fazer cópias de segurança (backups).

Backups para fita são geralmente feitos para lidar com um de dois problemas potenciais:

1. Recuperação em caso de um desastre.

2. Recuperação de uma bobagem feita.

O primeiro problema diz respeito a fazer o computador funcionar novamente após uma quebra de disco, fogo, enchente ou outra catástrofe natural. A segunda razão é que os usuários muitas vezes removem acidentalmente arquivos de que precisam mais tarde outra vez. Backups levam esse princípio mais longe ainda e permitem que arquivos que foram removidos há dias, mesmo semanas, sejam restaurados de velhas fitas de backup.

Fazer backup leva um longo tempo e ocupa um espaço significativo, portanto é importante fazê-lo de maneira eficiente e conveniente. Normalmente é desejável fazer o backup apenas de diretórios específicos e tudo neles em vez de todo o sistema de arquivos. Segundo, é um desperdício fazer o backup de arquivos que não mudaram desde o último backup, o que leva à ideia de cópias incrementais. A forma mais simples de cópia incremental é realizar uma cópia (backup) completa periodicamente, digamos por semana ou por mês, e realizar uma cópia diária somente daqueles arquivos que foram modificados desde a última cópia completa.

Terceiro, visto que quantidades imensas de dados geralmente são copiadas, pode ser desejável comprimir os dados antes de escrevê-los na fita. No entanto, com muitos algoritmos de compressão, um único defeito na fita de backup pode estragar o algoritmo e tornar um arquivo inteiro ou mesmo uma fita inteira ilegível. Desse modo, a decisão de comprimir os dados de backup deve ser cuidadosamente considerada.

Quarto, é difícil realizar um backup em um sistema de arquivos ativo. Se os arquivos e diretórios estão sendo adicionados, removidos e modificados durante o processo de cópia, a cópia resultante pode ficar inconsistente.

Quinto e último, fazer backups introduz muitos problemas não técnicos na organização.

Uma segunda preocupação é copiar blocos defeituosos. É quase impossível manufaturar discos grandes sem quaisquer defeitos. Alguns blocos defeituosos estão sempre presentes.

No entanto, às vezes os blocos passam a apresentar defeitos após a formatação, caso em que o sistema operacional eventualmente vai detectá-los.

Uma cópia lógica começa em um ou mais diretórios especificados e recursivamente copia todos os arquivos e diretórios encontrados ali que foram modificados desde uma determinada data de base. Assim, em uma cópia lógica, o disco da cópia recebe uma série de diretórios e arquivos cuidadosamente identificados, o que torna fácil restaurar um arquivo ou diretório específico mediante pedido.

4.4.3 Consistência do sistema de arquivos

Outra área na qual a confiabilidade é um problema é a consistência do sistema de arquivos. Muitos sistemas de arquivos leem blocos, modificam-nos e só depois os escrevem. Se o sistema cair antes de todos os blocos modificados terem sido escritos, o sistema de arquivos pode ser deixado em um estado inconsistente.

Para lidar com sistemas de arquivos inconsistentes, a maioria dos programas tem um programa utilitário que confere a consistência do sistema de arquivos. Esse utilitário pode ser executado sempre que o sistema é iniciado, especialmente após uma queda.

Dois tipos de verificações de consistência podem ser feitos: blocos e arquivos. Para conferir a consistência do bloco, o programa constrói duas tabelas, cada uma contendo um contador para cada bloco, inicialmente contendo 0. Os contadores na primeira tabela monitoram quantas vezes cada bloco está presente em um arquivo; os contadores na segunda tabela registram quantas vezes cada bloco está presente na lista de livres (ou o mapa de bits de blocos livres).

Além de conferir para ver se cada bloco está contabilizado corretamente, o verificador do sistema de arquivos também confere o sistema de diretórios. Ele, também, usa uma tabela de contadores, mas esses são por arquivo, em vez de por bloco. Ele começa no diretório-raiz e recursivamente percorre a árvore, inspecionando cada diretório no sistema de arquivos.

Essas duas operações, conferir os blocos e conferir os diretórios, muitas vezes são integradas por razões de eficiência.

4.4.4 Desempenho do sistema de arquivos

O acesso ao disco é muito mais lento do que o acesso à memória. Como consequência dessa diferença em tempo de acesso, muitos sistemas de arquivos foram projetados com várias otimizações para melhorar o desempenho. Nesta seção cobriremos três delas.

Cache de blocos

A técnica mais comum usada para reduzir os acessos ao disco é a cache de blocos ou cache de buffer. Nesse contexto, uma cache é uma coleção de blocos que logicamente pertencem ao disco, mas estão sendo mantidas na memória por razões de segurança.

Vários algoritmos podem ser usados para gerenciar a cache, mas um comum é conferir todas as solicitações para ver se o bloco necessário está na cache. Se estiver, o pedido de leitura pode ser satisfeito sem acesso ao disco. Se o bloco não estiver, primeiro ele é lido na cache e então copiado para onde quer que seja necessário. Solicitações subsequentes para o mesmo bloco podem ser satisfeitas a partir da cache.

Quando um bloco tem de ser carregado em uma cache cheia, alguns blocos têm de ser removidos.

É indesejável manter blocos de dados na cache por tempo demais antes de serem escritos. Considere o drama de alguém que está usando um computador pessoal para escrever um livro. Mesmo que o nosso escritor periodicamente diga ao editor para escrever para o disco o arquivo que está sendo editado, há uma boa chance de que tudo ainda esteja na cache e nada no disco. Se o sistema cair, a estrutura do sistema de arquivos não será corrompida, mas um dia inteiro de trabalho será perdido.

Sistemas adotam duas abordagens para lidar com isso. A maneira UNIX é ter uma chamada de sistema, sync, que força todos os blocos modificados para o disco imediatamente. Quando o sistema é inicializado, um programa, normalmente chamado update, é inicializado no segundo plano para adentrar um laço infinito que emite chamadas sync, dormindo por 30 s entre chamadas. Como consequência, não mais do que 30 s de trabalho são perdidos pela quebra. O Windows tem uma chamada de sistema equivalente a sync, chamada FlushFileBuffers.

Alguns sistemas operacionais integram a cache de buffer com a cache de páginas.

Leitura antecipada de blocos

Uma segunda técnica para melhorar o desempenho percebido do sistema de arquivos é tentar transferir blocos para a cache antes que eles sejam necessários para aumentar a taxa de acertos.

Para ver se a leitura antecipada vale a pena ser feita, o sistema de arquivos pode monitorar os padrões de acesso para cada arquivo aberto. Por exemplo, um bit associado com cada arquivo pode monitorar se o arquivo está em “modo de acesso sequencial” ou “modo de acesso aleatório”.

Redução do movimento do braço do disco

A cache e a leitura antecipada não são as únicas maneiras de incrementar o desempenho do sistema de arquivos. Outra técnica importante é reduzir o montante de movimento do braço do disco colocando blocos que têm mais chance de serem acessados em sequência próximos uns dos outros, de preferência no mesmo cilindro.

É claro, o movimento do braço do disco e o tempo de rotação são relevantes somente se o disco os tem. Mais e mais computadores vêm equipados com discos de estado sólido (SSDs — Solid State Disks) que não têm parte móvel alguma.

4.4.5 Desfragmentação de disco

Quando o sistema operacional é inicialmente instalado, os programas e os arquivos que ele precisa são instalados de modo consecutivo começando no início do disco, cada um seguindo diretamente o anterior. Todo o espaço de disco livre está em uma única unidade contígua seguindo os arquivos instalados. No entanto, à medida que o tempo passa, arquivos são criados e removidos, e o disco prejudica-se com a fragmentação, com arquivos e espaços vazios por toda parte. Em consequência, quando um novo arquivo é criado, os blocos usados para isso podem estar espalhados por todo o disco, resultando em um desempenho ruim. O desempenho pode ser restaurado movendo os arquivos a fim de deixá-los contíguos e colocando todo (ou pelo menos a maior parte) o espaço livre em uma ou mais regiões contíguas no disco.

A desfragmentação funciona melhor em sistemas de arquivos que têm bastante espaço livre em uma região contígua ao fim da partição.

Os sistemas de arquivos Linux (especialmente ext2 e ext3) geralmente sofrem menos com a desfragmentação do que os sistemas Windows pela maneira que os blocos de discos são selecionados, então a desfragmentação manual raramente é exigida. Também, os SSDs não sofrem de maneira alguma com a fragmentação. Na realidade, desfragmentar um SSD é contra produtivo. Não apenas não há ganho em desempenho, como os SSDs se desgastam; portanto, desfragmentá-los apenas encurta sua vida.

Referências

TANENBAUM, A. S. Sistemas Operacionais Modernos. [s.l: s.n.].