Sistemas Operacionais

Prof. Robson de Souza

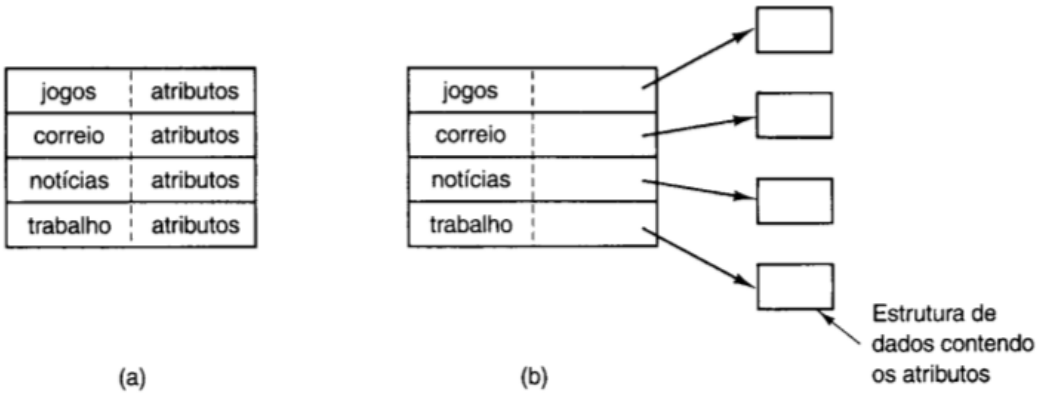
Aulas 35 e 36

Conteúdo: Sistemas de arquivos

Diretórios

Para organizar os arquivos, os sistemas de arquivos normalmente têm **diretórios** que, em muitos sistemas também são arquivos. Um diretório geralmente contém um certo número de entradas, uma por arquivo. Uma possibilidade de organização é fazer com que cada entrada tenha o nome do arquivo, os atributos do arquivo e os endereços de disco onde os dados estão armazenados.

Outra possibilidade é fazer com que a entrada armazene o nome do arquivo e um ponteiro para outra estrutura de dados onde os atributos e os endereços de disco estão localizados. Em ambos os casos, quando um arquivo é aberto, o sistema operacional pesquisa seu diretório até encontrar o nome do arquivo a ser aberto, então ele extrai os atributos e os endereços de disco e coloca-os em uma tabela na memória principal. Todas as subsequentes referencias ao arquivo utilizam as informações da memória principal.



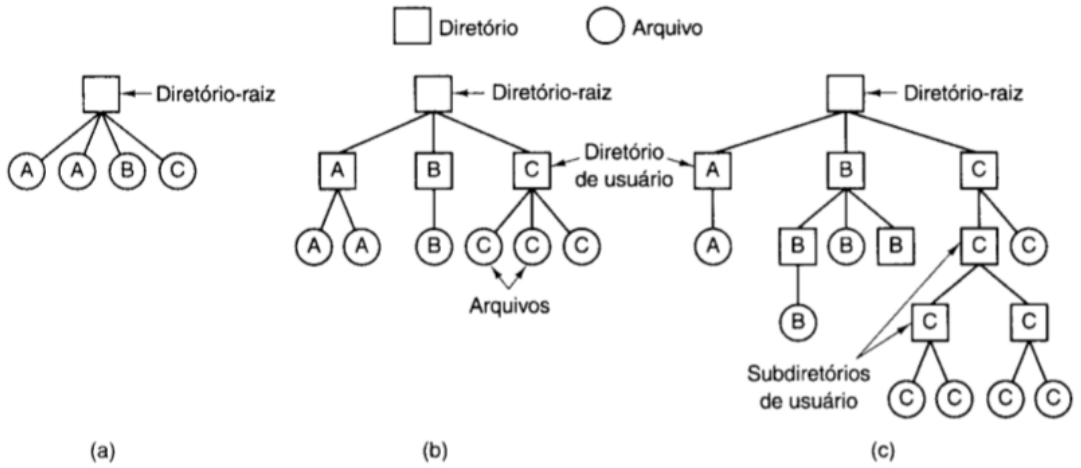
(a) Atributos na entrada de diretório. (b) Atributos em outra parte.

O modelo mais simples de diretórios é manter um único diretório com todos os arquivos dentro dele, só que isso gera uma série de desvantagens. Umas desvantagem que fica muito clara é no caso de existir vários usuários utilizando o mesmo sistema e alguns deles precisam salvar arquivos com o mesmo nome, fica claro que rapidamente isso gerará confusão e o sistema pode se tornar inoperante. Esse tipo de organização não é mais utilizado nos sistemas operacionais atuais.

Um aprimoramento na ideia de se ter um único diretório é ter um diretório para cada usuário. Nesse caso, o conflito gerado entre os usuários é resolvido, mas ainda assim não resolve o problema caso um usuário tenha muitos arquivos e em algum momento precise usar um nome de arquivo que já utilizou.

O que é necessário nesses casos é ter uma hierarquia geral (uma árvore de diretórios). Com essa abordagem, cada usuário pode ter tantos diretórios quantos forem necessários, de modo que os arquivos podem ser agrupados de maneira natural. Nesse caso, o sistema não fica limitado a um único diretório, um usuário pode criar um novo diretório ou mais dentro da raiz, podendo criar outros dentro dos que foram criados e assim por diante.

Isso dá maior liberdade e níveis de organização para um usuário, principalmente se ele tiver muitos arquivos a serem armazenados.



Três projetos de sistema de arquivos. (a) Único diretório compartilhado por todos usuários. (b) Um diretório por usuário. (c) Árvore arbitrária por usuário. As letras indicam o diretório ou o proprietário do arquivo.

Nomes de caminho

Quando o sistema de arquivos é organizado como uma árvore de diretórios, é necessário algum meio de especificar nomes de arquivo. Dois métodos diferentes são comumente utilizados. No primeiro método, a cada arquivo é dado um **nome de caminho absoluto** que consiste no caminho do diretório raiz até o arquivo. Por exemplo, o caminho **usr/ast/mailbox** significa que o diretório raiz contém um subdiretório usr que por sua vez possui um subdiretório ast que contém o arquivo mailbox. Nomes absolutos de caminho sempre iniciam no diretório raiz e são únicos.

Outro tipo de nome é o **nome de caminho relativo.** Esse é utilizado em conjunção com o conceito de **diretório de trabalho** (também chamado de diretório atual). Nesse caso, todos os nomes de caminho que não começam no diretório raiz são interpretados em relação ao diretório de trabalho. Por exemplo, se o diretório atual de trabalho for ***usr/ast***, então o arquivo cujo caminho absoluto foi exemplificado acima, pode ser referenciado simplesmente como **mailbox**. Ou seja, os dois comandos são equivalente desde que o diretório de trabalho seja usr/ast. A forma relativa é frequentemente mais conveniente e faz a mesma coisa que a forma absoluta.

Existem casos em que os programas precisam acessar um arquivo específico sem considerar qual é o diretório de trabalho. Nesse caso, eles sempre devem utilizar nomes absolutos de caminho. Por exemplo, um corretor ortográfico talvez precise ler **usr/lib/dictionary** para fazer seu trabalho. Ele deve utilizar o nome de caminho absoluto nesse caso, pois não saberá qual é o diretório de trabalho atual quando for chamado.

Operações com diretórios

As chamadas de sistema para diretórios geralmente permitem as mesmas operações básicas permitidas em arquivos com algumas exceções. Porém, as chamadas de sistema permitidas para gerenciar diretórios apresentam mais variação de sistema para sistema do que as chamadas de sistema para arquivos.

Exemplos de algumas chamadas para gerenciar diretórios no UNIX:

- CREATE → Um diretório é criado vazio.
- DELETE → Um diretório é excluído.
- OPENDIR → Os diretórios podem ser lidos. Por exemplo, para listar todos os arquivos em um diretório, um programa de listagem abre o diretório para ler o nome de todos os arquivos que ele contém.
- CLOSEDIR → Quando um diretório for lido, ele deve ser fechado para liberar espaço interno da tabela.
- LINK → Vinculação é uma técnica que permite que um arquivo apareça em mais de um diretório.

Essa chamada especifica um arquivo existente e um nome de caminho e cria um vínculo do arquivo existente para o nome especificado pelo caminho, assim, o mesmo arquivo pode aparecer em múltiplos diretórios.

Implementação do sistema de arquivos e busca dos dados

Até aqui, o foco foi no ponto de vista do usuário, porém, os implementadores estão interessados em como arquivos e diretórios são armazenados, em como o espaço em disco é gerenciado e em como fazer tudo funcionar eficiente e confiavelmente.

O modo como os arquivos e diretórios são organizados para serem inseridos e buscados no disco e na memória varia de sistema para sistema. Existem alguns algoritmos de busca em memória principal e memória secundária que podem ser utilizados para retornar os arquivos desejados pelo usuário.

*Busca sequencial

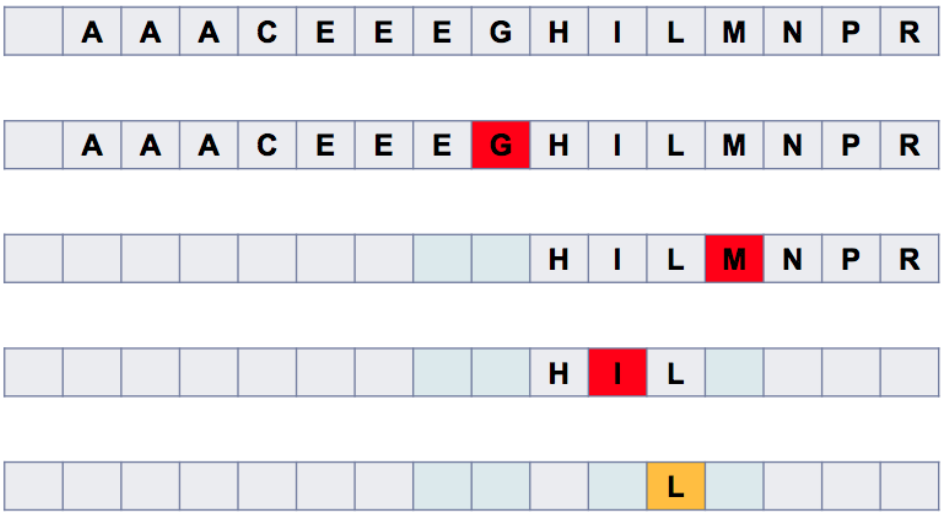
É um método de pesquisa mais simples: a partir do primeiro registro, pesquise sequencialmente até encontrar a chave procurada; então pare. O armazenamento é feito por meio de um conjunto de registros estruturado. Cada registro contém um campo chave que identifica o registro. Além da chave, podem existir outros componentes em um registro, os quais não têm influência nos algoritmos. O método de pesquisa retorna o índice do registro que contém a chave passada como parâmetro.

Essa busca é muito simples de ser implementada e pode ser utilizada se o sistema operacional for buscar na memória principal os arquivos, porém o custo computacional da busca sequencial é muito elevado.

*Busca binária

Esse método de pesquisa reduz o tempo de busca aplicando o paradigma dividir para conquistar. Basicamente, divide o vetor em duas partes, verifica em qual das partes o item com a chave se localiza e concentra-se apenas naquela parte até encontrar o valor desejado. Essa busca é bem mais eficiente que a sequencial, porém existem algumas restrições e desvantagens. Uma restrição é a necessidade de que as chaves precisam estar ordenadas e manter chaves ordenadas na inserção pode levar a comportamento quadrático. Outra desvantagem é o fato de que a trocas de posições podem reduzir a eficiência.

Exemplo: Pesquisa pela chave L

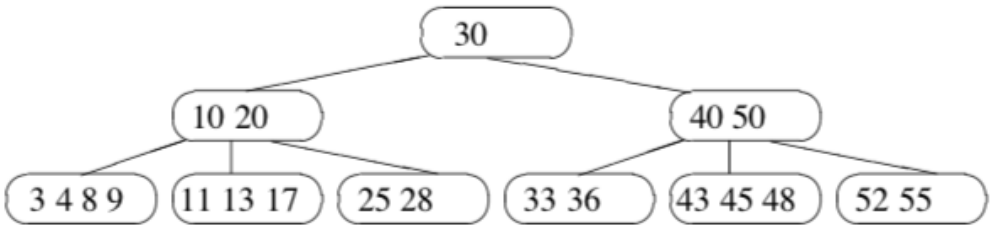


https://homepages.dcc.ufmg.br/~cunha/teaching/20121/aeds2/searching.pdf

Árvore binária de pesquisa

Esse mecanismo de busca pode ser utilizado tanto para memória principal quanto para memória secundária. As árvores de pesquisa mantêm uma ordem entre seus elementos. De modo que a raiz é maior que os elementos na árvore à esquerda e menor que os elementos na árvore à direita.

Um tipo de árvore muito utilizada para busca na memória secundária são as árvores B. Nesse caso, cada nó da árvore contém uma página da memória e para cada elemento nas páginas, existe um ponteiro que aponta para outro nó (página). Essas árvores seguem a mesma lógica de organização das árvores binárias.



http://www2.dcc.ufmg.br/livros/algoritmos/cap6/slides/c/completo1/cap6.pdf

Referências bibliográficas:

TANENBAUM, Andrew. 2ª ed. **Sistemas Operacionais Modernos,** Editora Pearson, 2003.

SILBERSCHATZ, Abraham. **Sistemas Operacionais com JAVA,** 6ª ed. Editora Campus

MACHADO, Francis B. **Arquitetura de Sistemas Operacionais,** 4ª ed, LTC, 2007.