Aula 10 - Sistema de Arquivos (Parte 2)

Sistemas Operacionais
Ciência da Computação
IFB - Campus Taguatinga

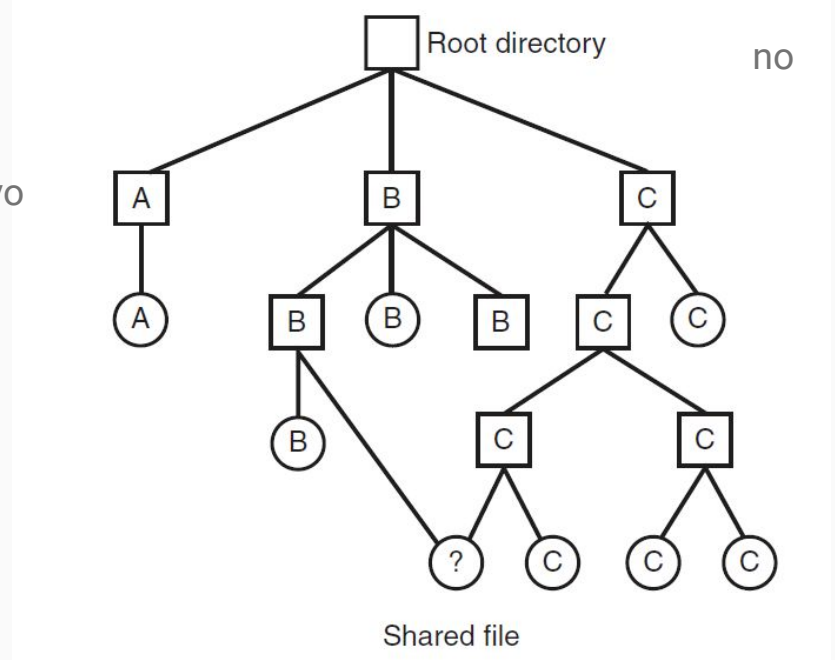
Professor João Victor de A. Oliveira

Hoje

- Arquivos Compartilhados
- Estruturas de Sistemas de Arquivos
  - Sistemas de arquivos estruturados em diário (log)
  - Sistemas de Arquivos *journailing*
  - Sistemas de arquivos virtuais

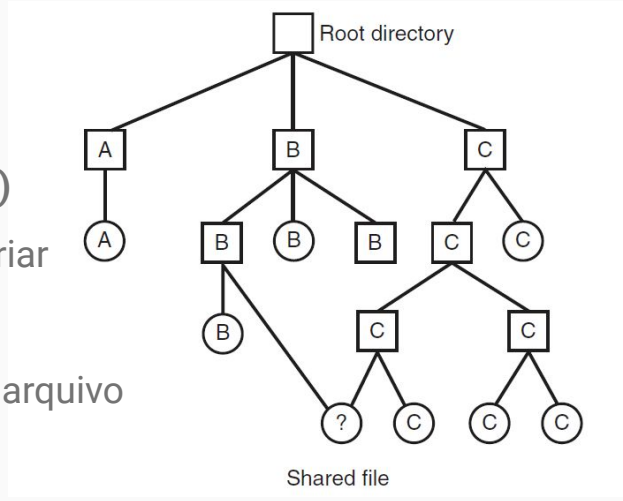
Arquivos compartilhados

- **Vários usuários podem precisar compartilhar arquivos**
  - Pode ser conveniente que um arquivo compartilhado apareça simultaneamente em diretórios diferentes pertencendo a usuários distintos
  - Seja o **arquivo do usuário C** presente também no diretório do **usuário B**
  - A conexão entre o diretório do usuário B e o arquivo compartilhado é chamada **ligação**
  - O sistema de arquivos agora é um grafo acíclico orientado - DAG (de difícil manutenção)



Arquivos compartilhados

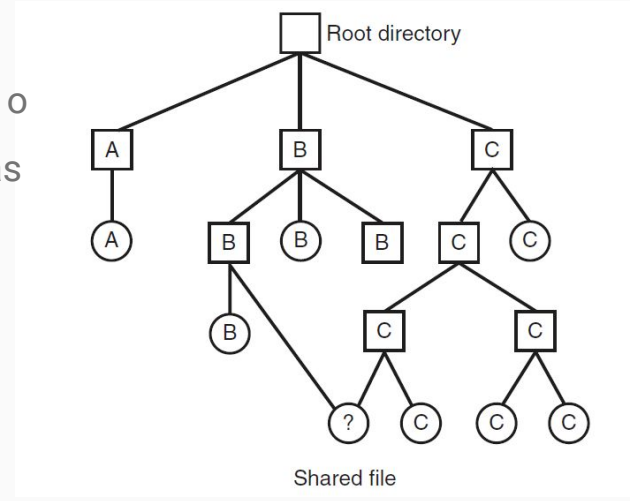
- **Problema em compartilhar arquivos**
  - Se os diretórios realmente contiverem endereços de disco
    - Uma cópia desses endereços terá de ser feita no diretório do usuário B quando o arquivo for ligado
    - Se B ou C adicionarem blocos no arquivo, os novos blocos serão listados somente no diretório do usuário que estiver realizando a adição
    - **Mudanças não serão visíveis a outros usuários!!!!!**
- **Solução:**
  - **Blocos do disco não serem listados em diretórios (i-node)**
  - B se liga a um dos arquivos de C, obrigando o sistema a criar um novo arquivo do tipo **LINK** e a inserí-lo no diretório B
  - **LINK** é um arquivo que só contém o nome do caminho do arquivo para o qual ele está ligado



Arquivos compartilhados

- Quando B lê do arquivo ligado, o SO vê que o arquivo sendo lido é do tipo LINK, verifica o seu nome e então o lê
- Essa abordagem é chamada **ligação simbólica**

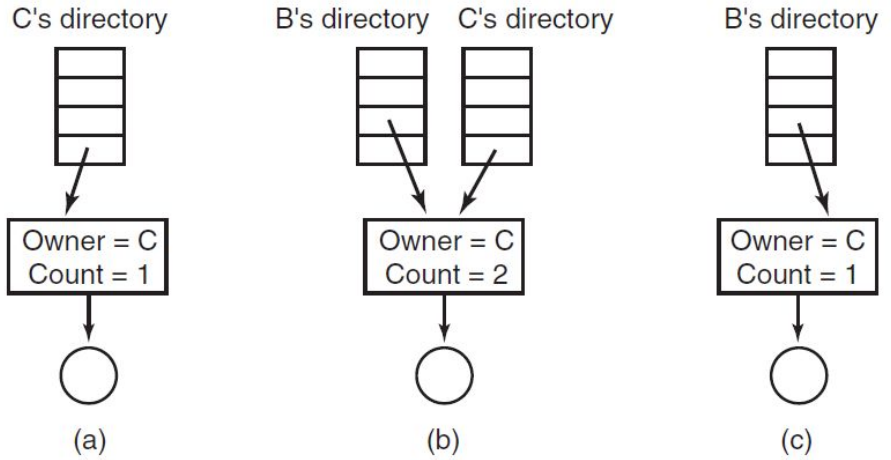
- **Cada um desses dois métodos apresenta problemas**
  - **Blocos do disco não serem listados em diretórios (i-node)**
    - B se liga com o arquivo compartilhado, o i-node grava o proprietário do arquivo como C
    - Criar uma ligação não muda a propriedade, mas aumenta o contador de ligações no i-node, ou seja, o SO sabe quantas entradas de diretório apontam para o arquivo



Shared file

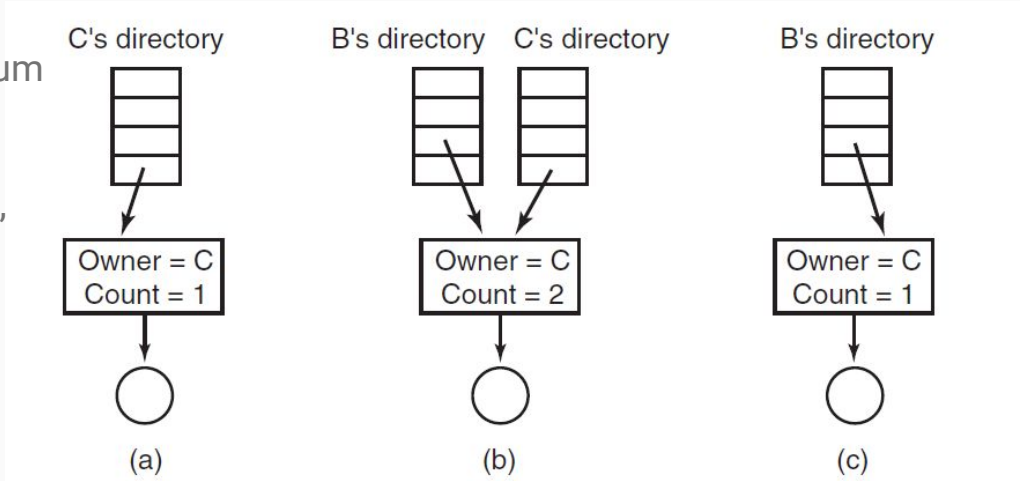
Arquivos compartilhados

- **Blocos do disco não são listados em diretórios (i-node)**
  - Se C, tentar remover o arquivo, o sistema se vê em um dilema
    - Se remover o arquivo em C e limpar o i-node, B terá uma entrada de diretório apontando para um i-node inválido
    - Se o i-node for transferido mais tarde para outro arquivo, a ligação de B apontará para o arquivo errado
  - O sistema pode até avaliar que o arquivo ainda está em uso, **mas não há maneira fácil de encontrar todas as entradas de diretório para o arquivo a fim de removê-las**



Arquivos compartilhados

- **Blocos do disco não são listados em diretórios (i-node)**
  - Única coisa a se fazer: remover a entrada de diretório de C, mas deixar o i-node intacto, com o contador em 1
    - Agora temos a situação na qual B é o único usuário com uma entrada de diretório para um arquivo cujo o proprietário é C
- **Ligações simbólicas**
  - Somente o verdadeiro proprietário tem um ponteiro para o i-node
  - Quando o proprietário remove o arquivo, ele é destruído



Estruturas de Sistemas de Arquivos

Sistemas de arquivos estruturados em diário (log)

- **Mudanças na tecnologia**
  - CPU mais rápido
  - Memórias crescem exponencialmente de tamanho
- **Gargalo de desempenho surge nos sistemas de arquivos**
  - Para minimizar este problema, pesquisadores em Berkeley propuseram um tipo novo de sistemas de arquivos, o **LFS (Log-structure File System - Sistema de arquivos estruturado em diário)**

Sistemas de arquivos estruturados em diário (log)

- **LFS**
  - A medida que as CPU's ficam mais rápidas e as memórias RAMs ficam maiores, **caches em disco** também estão aumentando rapidamente
    - Conseguem satisfazer uma fração substancial de todas as solicitações de leitura diretamente da cache do sistema de arquivos (sem a necessidade de acesso ao disco)
  - **Problema:** no futuro, a maior parte dos acessos ao disco será para escrita
    - Mecanismo de leitura antecipada não trará tantos benefícios assim
  - **Outro problema:** Operações de escrita são feitas em pedaços muito pequenos (ineficiente)
    - Uma escrita de disco de 50 µs muitas vezes é precedida por uma busca de 10 ms e um atraso rotacional de 4 ms
    - Com esses parâmetros a eficiência dos discos cai para uma fração de 1%

Sistemas de arquivos estruturados em diário (log)

- Operações de escrita são feitas em pedaços muito pequenos
  - Considere criar um arquivo novo em um sistema UNIX
    - Para escrever esse arquivo, o i-node para o diretório, o bloco do diretório, o i-node para o arquivo e o próprio arquivo **devem ser todos escritos**
  - Observe que poderíamos postergar essas operações
    - Fazê-lo expõe o sistema de arquivos a sérios problemas de consistência se uma queda no sistema ocorrer antes que as escritas tenham sido concluídas
    - Por isso, as escritas de i-node são, em geral feitas imediatamente

Sistemas de arquivos estruturados em diário (log)

- Projetistas do LFS decidiram reimplementar o sistema de arquivos UNIX de forma a usar a **largura total de banda do disco**
  - Mesmo diante uma carga de trabalho consistindo de pequenas operações de escritas aleatórias
- **Idéia básica:** Estruturar o disco inteiro como um grande **diário (log)**
  - De modo periódico (quando houver necessidade), todas as operações de escrita pendentes armazenadas na memória **são agrupadas em um único segmento**
    - Estas, então, são escritas para o disco como um único segmento contíguo ao fim do diário
    - Desse modo, um único segmento pode conter i-nodes, blocos de diretório e blocos de dados, tudo misturado

Sistemas de arquivos estruturados em diário (log)

- No começo de cada segmento há um **resumo do segmento**
  - Diz o que pode ser encontrado nesse segmento
- 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
- Encontrar i-nodes nessa abordagem não é tão fácil
  - Para contornar isso, é mantido um mapa do i-node, indexado pelo i-número
  - Mapa é armazenado em disco e também mantido em cache
- **Como abrir um arquivo no LFS?**
  - Usar o mapa para localizar o i-node para o arquivo
  - Uma vez localizado, os endereço dos blocos podem ser encontrados a partir dele

Sistemas de arquivos estruturados em diário (log)

- **Problema:** Discos reais são finitos, logo o diário um dia ocupará o disco inteiro
  - Felizmente muitos segmentos existentes podem ter blocos que não são mais necessários
  - Solução: *thread* **limpador**
    - Passa o seu tempo escaneando o diário circularmente para compactá-lo
  - O disco passa a ser um grande **buffer circular**, com um *thread* de escrita adicionando novos segmentos ao início e um thread limpador removendo os antigos do final
- LFS supera o UNIX em desempenho por uma ordem de magnitude em escritas pequenas
  - Tão bom ou melhor para leituras e escritas grandes

Sistemas de arquivos *journaling*

- LFS é uma boa ideia, mas altamente incompatível com os sistemas de arquivos existentes
  - Podemos usar ideias da LFS nos sistemas de arquivos existentes, como a **robustez diante falhas**
- **Idéia básica:** Manter um diário do que o sistema de arquivos vai fazer, antes que ele o faça
  - Se o sistema falhar antes que ele possa fazer seu trabalho planejado, ao ser reinicializado, ele pode procurar no diário para ver o que acontecia no momento da falha e concluir o trabalho
  - Esses tipos de sistemas de arquivos são chamados de **sistemas de arquivos journaling**
    - **Utilizados em sistemas de arquivos NTFS (Windows), Linux ext3**
    - **OS X oferece sistemas de arquivos journaling também...**

Sistemas de arquivos *journaling*

- Considere uma operação corriqueira simples: remover um arquivo
  - **Essa operação exige 3 passos:**
    1. Remover o arquivo do seu diretório
    2. Liberar o i-node para o conjunto de i-nodes livres
    3. Retornar todos os blocos de disco para o conjunto de blocos livres
  - Na ausência de falhas no sistema, a **ordem** na qual esses passos são executados não importa
  - Na presença de falhas a ordem importa!
    - Ex.: Suponha que o primeiro passo tenha sido concluído e ocorra uma falha no sistema
      - i-node e os blocos do arquivo não serão acessíveis a partir de arquivo algum
      - Também não serão acessíveis para realocação
      - Estarão em um limbo, diminuindo os recursos disponíveis
    - Se a falha ocorrer no segundo passo
      - Blocos de discos serão perdidos

Sistemas de arquivos *journaling*

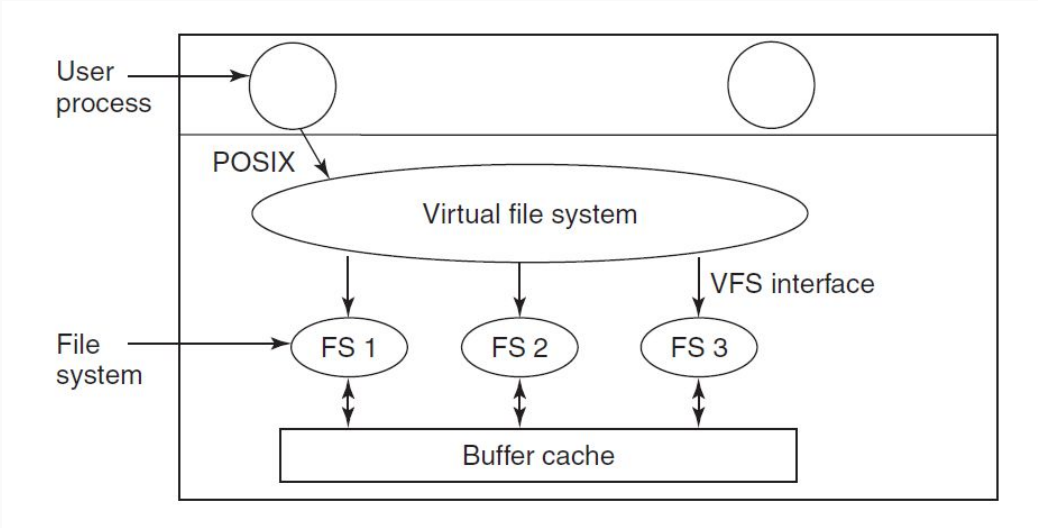
- Na presença de falhas a ordem importa!
  - E caso a ordem seja alterada?
    - Se o i-node for liberado primeiro e depois falhar?
    - Se os blocos forem liberados primeiro e depois falhar?
- O que faz o sistema de arquivos journaling?
  - Primeiro escreve uma entrada no diário, listando as três ações a serem concluídas
  - A entrada no diário é então escrita para o disco (de maneira **previdente**, ou seja, verificando se foi escrita corretamente)
  - Só após a entrada no diário ter sido escrita é que começam as várias operações
    - Só após as operações terem sido concluídas e bem sucedidas essa entrada no diário é apagada

Sistemas de Arquivos virtuais

- Muitos sistemas de arquivos diferentes estão em uso - muitas vezes no mesmo computador - ainda que dentro de um mesmo sistema operacional
  - **Windows** pode ter um sistema de arquivos NTFS principal, mas também uma antiga unidade ou partição FAT-32 e, de tempos em tempos um flash drive, um antigo CD-ROM ou um DVD podem ser necessários
    - Windows lida com esses sistemas díspares identificando cada um com uma letra de unidade diferente, como em C:, D:, etc...
  - **Sistemas UNIX** fazem uma tentativa séria de integrar múltiplos sistemas de arquivos em uma única estrutura
    - Em um sistema LINUX, pode ter o ext2 como um diretório raiz, com a partição ext3 montada em /usr e um segundo disco rígido com o sistema de arquivos ReiserFS montado em /home, assim como um CD-ROM ISO 9660 temporariamente montado em /usr
    - **Do ponto de vista de um usuário, só existe uma única hierarquia de sistema de arquivos**

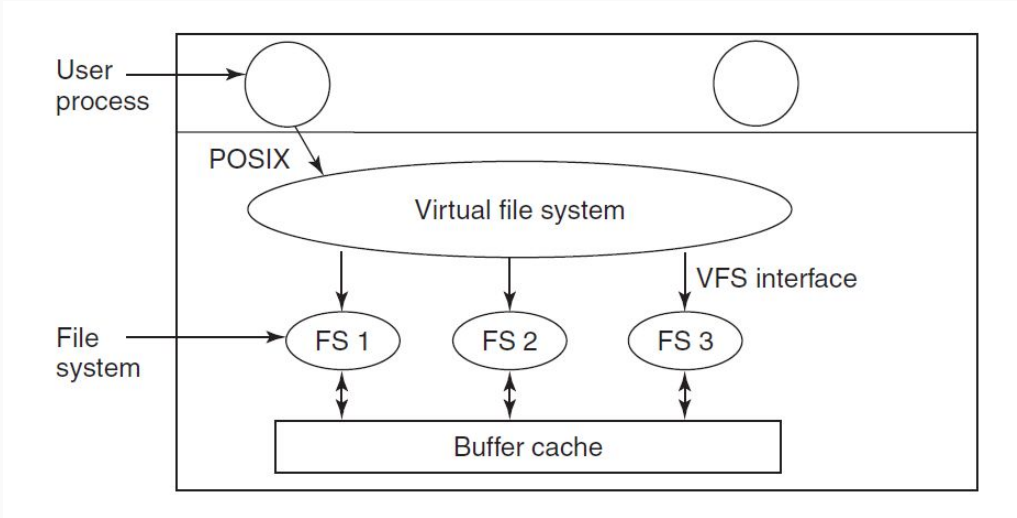
Sistemas de Arquivos virtuais

- Para integrar múltiplos sistemas de arquivos, a maioria dos sistemas UNIX usou o conceito de um **VFS (*Virtual File System* - Sistema de Arquivos Virtuais)**
  - **Idéia:** 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



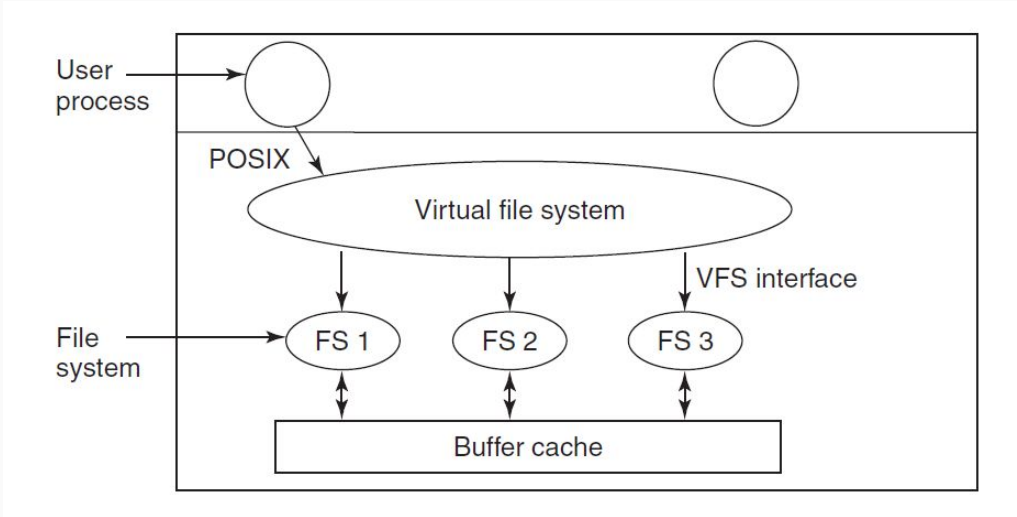
Sistemas de Arquivos virtuais

- VFS
  - Todas as chamadas de sistemas relativas a arquivos são direcionadas ao VFS para o processamento inicial
    - **Essas chamadas são chamadas de POSIX padrão**, como *open, read, write, lseek*
  - Desse modo o VFS tem uma interface “superior” para os processos de usuário e é conhecida como **interface POSIX**



Sistemas de Arquivos virtuais

- VFS
  - O VFS também possui uma interface “inferior” para os sistemas de arquivos reais, chamados de **interface** do VFS
    - Fazem chamadas para cada tipo de sistema de arquivos, de forma a realizarem seu trabalho



Sistemas de Arquivos virtuais

- A motivação original para produzir o VFS era dar suporte a sistemas de arquivos remotos
  - Usando o protocolo **NFS (*Network File System* -** Sistemas de arquivos de rede*)*
  - O VFS foi feito de tal forma que enquanto o sistema de arquivo real fornecer as funções que o VFS exigir, o VFS não sabe ou se preocupa onde estão armazenados os dados ou como são os sistemas de arquivos subjacentes.

Próxima Aula

- Princípio de Softwares de Entrada e Saída

Referências

Tanenbaum, A. S. e Bos, H.. Sistemas Operacionais Modernos. 4.ed. Pearson/Prentice-Hall. 2016.