

Sistemas de Arquivos

Sistemas Operacionais

Prof. Pedro Ramos
pramos.costar@gmail.com

SISTEMAS DE ARQUIVOS

Lembre-se: visão alto nível do SO é uma tradução da abstração de usuário para a realidade em hardware.

ABSTRAÇÃO USUÁRIO		RECURSO (HARDWARE)
Processos/Threads	$\leq \text{SO} \geq$	CPU
Espaço de Endereços	$\leq \text{SO} \geq$	Memória
Arquivos	$\leq \text{SO} \geq$	Disco/SSD

SISTEMAS DE ARQUIVOS

APLICAÇÕES

DAEMONS

SERVIDORES

SHELLS

Interface do Programador

Open()

Close()

Read()

Write()

Link()

Rename()

Interface Independente do Dispositivo

Setores

Trilhas

Seek()

ReadBlock()

WriteBlock()

Interface do Dispositivo

HARDWARE DISCO SSD

REQUISITOS DO USUÁRIO

- **Persistência** - Dados devem persistir entre processos, ciclos de energia, *crashes*
- **Velocidade** - Dados devem ser acessados de maneira rápida
- **Tamanho** - Pode haver terabytes, petabytes de Dados
- **Compartilhamento & Proteção** - Usuários podem compartilhar dados quando for apropriado, e mantê-los privados quando for necessário
- **Facilidade de uso** - Encontrar, examinar, modificar dados com facilidade.

SISTEMAS DE ARQUIVOS

Hardware fornece:

Persistência: Discos tem memória não volátil.

Velocidade: Ganho de velocidade através do acesso aleatório.

Tamanho: Discos crescem até hoje (disco típico em um PC = 500GB - 1TB).

O SO fornece:

Persistência: Redundância permite recuperação de algumas falhas adicionais.

Compartilhamento/Proteção: Permissões de leitura, escrita e execução para arquivos.

Facilidade de uso:

Associação de nomes a blocos de dados (*arquivos*).

Organização de grandes coleções de arquivos em diretórios.

Mapeamento transparente do conceito de arquivos e diretórios do usuário para localizações nos discos.

Ferramenta de busca nos sistemas de arquivos (como o Spotlight no macOS).

ARQUIVOS

Arquivo: Unidade lógica de armazenamento em um dispositivo de armazenamento.

Exemplos: reader.cc, a.out.

Os arquivos podem **conter**:

Programas: código-fonte, binário.

Dados: qualquer tipo de informação.

Os arquivos podem **ser**:

Estruturados: organizados em registros ou objetos (como nos mainframes da IBM) O SO conhece a localização exata do arquivo.

Não estruturados: uma sequência contínua de bytes (como implementado no Unix). A localização do arquivo existe no sistema de arquivos.

Atributos de um arquivo:

Nome, tipo, localização, tamanho, proteção, tempo de criação, ... metadados

INTERFACE DO USUÁRIO COM O SISTEMA DE ARQUIVOS

Operações comuns com arquivos (chamadas de sistema):

Operações de dados:

Create(): Cria um novo arquivo.

Open(): Abre um arquivo existente.

Read(): Lê dados de um arquivo.

Write(): Escreve dados em um arquivo.

Delete(): Remove um arquivo.

Close(): Fecha um arquivo aberto.

Seek(): Move o ponteiro para uma posição específica no arquivo.

Operações de nomeação e atributos:

HardLink(): Cria um link físico para um arquivo.

SoftLink(): Cria um link simbólico para um arquivo.

Rename(): Renomeia um arquivo.

SetAttribute(): Define atributos (proprietário, permissões, etc.).

GetAttribute(): Recupera atributos de um arquivo.

ESTRUTURAS QUE O SO PROVÊ

Tabela de Arquivos Abertos (compartilhada por todos os processos):

- Contador de aberturas.
- Atributos do arquivo (proprietário, permissões, tempos de acesso, etc.).
- Localização no disco.
- Ponteiros para localização do arquivo na memória.

Tabela de Arquivos por Processo - PARA CADA ARQUIVO que o processo acessa,

- Ponteiro para entrada na tabela de arquivos abertos.
- Posição atual no arquivo (deslocamento).
- Modo de acesso (leitura, escrita, leitura/escrita).
- Ponteiros para o buffer do arquivo.

OPERAÇÕES - CRIANDO UM ARQUIVO

- **Create(nome):**

- Aloca espaço no disco (verifica cotas, permissões, etc.). (**Tem espaço suficiente?**)
- Cria um “arquivo descritor” com nome, localização no disco e atributos.
- Adiciona esse arquivo descritor ao diretório onde o arquivo se encontra.
- Atributo opcional: tipo do arquivo (Word, executável, etc.).

OPERAÇÕES - CRIANDO UM ARQUIVO

- **Create(nome):**

- Aloca espaço no disco (verifica cotas, permissões, etc.).
- Cria um “arquivo descritor” com nome, localização no disco e atributos.
- Adiciona esse arquivo descritor ao diretório onde o arquivo se encontra.
- Atributo opcional: tipo do arquivo (Word, executável, etc.).
 - Vantagens:
 - Melhor detecção de erros.
 - Operações padrão especializadas (abrir com o app correto).
 - Otimizações no layout de armazenamento.
 - Desvantagens:
 - Sistema de arquivos e SO mais complexos.
 - Menos flexível para o usuário.
 - Escolhas de SO:
 - Unix: simplicidade (sem tipos de arquivo).
 - Mac/Windows: foco na facilidade de uso.

OPERAÇÕES - DELETANDO UM ARQUIVO

- **Delete(nome):**
 - Localiza o diretório que contém o arquivo.
 - Libera os blocos de disco utilizados pelo arquivo.
 - Remove o descritor de arquivo do diretório.
 - Considerar contadores de referência e *hardlinks*, se existirem.

OPERAÇÕES - ABRIR E FECHAR

- **fileId = Open(nome, modo):**

- Verifica se o arquivo já está aberto por outro processo. Se não:
 - Localiza o arquivo.
 - Copia o descriptor para a tabela global de arquivos abertos.
- Verifica as permissões em relação ao modo solicitado. *Se inválido, aborta.*
- Incrementa o contador de aberturas.
- Cria uma entrada na tabela de arquivos do processo apontando para a entrada na tabela global. Inicializa o ponteiro do arquivo para o início.

- **Close(fileId):**

- Remove a entrada do arquivo na tabela do processo.
- Decrementa o contador de aberturas na tabela global.
- Se o contador de aberturas for 0, remove a entrada da tabela global.

OPERAÇÕES - LER

- **Read(fileID, from, size, bufAddress)** (*acesso aleatório*):

- O S0 lê **size** bytes a partir da posição **from** no arquivo e copia para **bufAddress**.

Exemplo simplificado:

```
for (i = from; i < from + size; i++)  
    bufAddress[i - from] = file[i];
```

- **Read(fileID, size, bufAddress)** (*acesso sequencial*):

- O S0 lê **size** bytes a partir da posição atual **fp** e copia para **bufAddress**.

Atualiza a posição atual no arquivo:

```
for (i = 0; i < size; i++)  
    bufAddress[i] = file[fp + i];  
fp += size;
```

OPERAÇÕES - LER

- **Read(fileID, from, size, bufAddress)** (*acesso aleatório*):

- O S0 lê **size** bytes a partir da posição **from** no arquivo e copia para **bufAddress**.

Exemplo simplificado:


```
for (i = from; i < from + size; i++)  
    bufAddress[i - from] = file[i];
```

- **Read(fileID, size, bufAddress)** (*acesso sequencial*):

- O S0 lê **size** bytes a partir da posição atual **fp** e copia para **bufAddress**.

Atualiza a posição atual no arquivo:

```
for (i = 0; i < size; i++)  
    bufAddress[i] = file[fp + i];  
fp += size;
```



*FILE pointer, não é o
mesmo \$fp (frame pointer)
das threads/processos*

OPERAÇÕES

- **Escrever (Write):** Semelhante à leitura, mas copia dados do buffer para o arquivo.
- **Seek:** Atualiza apenas o ponteiro de arquivo (fp).
- **Mapeamento de Memória de um Arquivo:**
 - Mapeia uma parte do espaço de endereços virtuais para um arquivo.
 - Leitura/escrita na memória implica leitura/escrita correspondente no arquivo.
 - Simplifica o acesso a arquivos (não é necessário chamar `read/write`).

MÉTODOS DE ACESSO

- **Padrões comuns do ponto de vista do programador:**
 - **Sequencial:** Processa os dados em ordem, byte ou registro por vez.
 - Exemplo: compilador lendo um arquivo fonte.
 - **Por chave (direto):** Acessa um bloco com base em um valor de chave.
 - Exemplo: busca em banco de dados, tabela hash, dicionário.
- **Padrões comuns do ponto de vista do S0:**
 - **Sequencial:** Mantém um ponteiro para o próximo byte, atualizando-o a cada leitura/escrita.
 - **Aleatório:** Acessa diretamente qualquer bloco no arquivo, dado seu deslocamento.

NOMES E DIRETÓRIOS

- É necessário um método para acessar arquivos armazenados no disco.
- O SO usa números para identificar arquivos, mas os usuários preferem nomes textuais.
- **Diretório:** Estrutura de dados do SO para mapear nomes para descritores de arquivos.

Estratégias de nomeação:

- **Diretório de Nível Único:** Um único espaço de nomes para todo o disco, onde cada nome é único.
 1. Usa uma área especial do disco para armazenar o diretório.
 2. O diretório contém pares <nome, índice>.
 3. Se um usuário usa um nome, ninguém mais pode usá-lo.
 4. Usado por computadores antigos e PCs pessoais com discos pequenos.
- **Diretório de 2 Níveis:** Cada usuário tem um diretório separado, mas os arquivos de cada usuário devem ter nomes únicos.

NOMES E DIRETÓRIOS

- **Diretórios Multinível:** Espaço de nomes estruturado em árvore
(*Unix e outros SOs modernos*)
 1. Diretórios são armazenados no disco como arquivos, com um bit especial no arquivo descritor.
 2. Programas de usuário leem diretórios como qualquer outro arquivo, mas apenas chamadas de sistema especiais podem escrevê-los.
 3. Cada diretório contém pares <nome, descritor> em qualquer ordem. O arquivo referenciado por **um nome pode ser outro diretório**.
 4. Existe um diretório **raiz** especial.
 - Exemplo de pesquisa (*lookup*): **/usr/local/bin/netcape**

LINKS (NOMEAÇÃO REFERENCIAL)

- **hardlinks**: Links rígidos (Unix: comando **ln**)
 - Um link rígido adiciona uma segunda conexão para um arquivo.
 - Exemplo: criando um link rígido de B para A:
Inicialmente: **A → arquivo #100**
Após "**ln A B**":

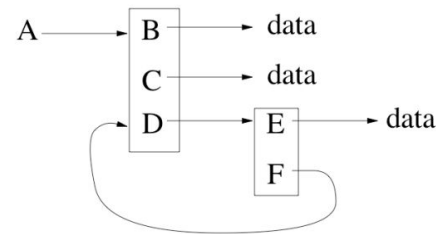
A → arquivo #100
B → arquivo #100
 - O SO mantém contagens de referência, de modo que o arquivo só será excluído após o último link ser deletado.

LINKS (NOMEAÇÃO REFERENCIAL)

- **hardlinks**: Links rígidos (Unix: comando **ln**)
 - Um link rígido adiciona uma segunda conexão para um arquivo.
 - Exemplo: criando um link rígido de B para A:
Inicialmente: **A → arquivo #100**
Após "**ln A B**":

A → arquivo #100

B → arquivo #100



- O SO mantém contagens de referência, de modo que o arquivo só será excluído após o último link ser deletado.
- **Problema**: o usuário pode criar links circulares com diretórios, e o SO nunca conseguiria liberar o espaço em disco.
- **Solução**: Não permitir links rígidos para diretórios.

SOFTLINKS

- **softlinks**: Links simbólicos (Unix: comando `ln -s`)
 - Um link simbólico cria apenas um ponteiro simbólico de um arquivo para outro.
 - Exemplo: criando um link simbólico de B para A:
Inicialmente: **A → arquivo #100**
Após "`ln -s A B`":

A → arquivo #100
B → A
 - Remover B não afeta A.
 - Remover A deixa o nome B no diretório, mas seu conteúdo não existe mais.
 - **Problema:** links circulares podem causar loops infinitos (exemplo: ao listar arquivos em diretórios e subdiretórios).
 - **Solução:** limitar o número de links percorridos.

OPERAÇÕES EM DIRETÓRIOS

- **Buscar um arquivo:** localizar uma entrada para um arquivo.
- **Criar um arquivo:** adicionar uma entrada no diretório.
- **Excluir um arquivo:** remover a entrada do diretório.
- **Listar um diretório:** listar todos os arquivos (comando `ls` no UNIX).
- **Renomear um arquivo.**
- **Percorrer o sistema de arquivos.**

OPERAÇÕES EM DIRETÓRIOS

- **Buscar um arquivo:** localizar uma entrada para um arquivo.
- **Criar um arquivo:** adicionar uma entrada no diretório.
- **Excluir um arquivo:** remover a entrada do diretório.
- **Listar um diretório:** listar todos os arquivos (comando `ls` no UNIX).
- **Renomear um arquivo.**
- **Percorrer o sistema de arquivos.**

**QUAL LIMITE ASSINTÓTICO
SUPERIOR PARA OPERAÇÕES
EM DIRETÓRIOS? (Número de
arquivos/diretórios únicos: N)**

- a) $O(N)$
- b) $O(\log N)$
- c) $O(N \cdot \log N)$

PROTEÇÃO

- O SO deve permitir que os usuários controlem o compartilhamento de seus arquivos => **controle de acesso aos arquivos.**
- Conceder ou negar acesso às operações de arquivos dependendo das informações de proteção.
- **Listas de Acesso e Grupos (Windows NT)**
 - Manter uma lista de acesso para cada arquivo com nome de usuário e tipo de acesso.
 - As listas podem se tornar grandes e difíceis de manter em um sistema com muitos usuários.
- **Bits de Controle de Acesso (UNIX)**
 - Três categorias de usuários (proprietário, grupo, público) (owner, group, world).
 - Três tipos de privilégios de acesso (leitura, escrita, execução).
 - Manter um bit para cada combinação (111101000 = **rwxr-x---**).

PERGUNTAS?

REFERÊNCIAS

- **TANENBAUM, Andrew.** Sistemas operacionais modernos.
- **SILBERSCHATZ, Abraham et al.** Fundamentos de sistemas operacionais: princípios básicos.
- **MACHADO, Francis; MAIA, Luiz Paulo.** Arquitetura de Sistemas Operacionais.
- **CARISSIMI, Alexandre et al.** Sistemas operacionais.