

→ influencia a gestão do disco

Algumas características dos "discos"

- Muito dependente de vários fatores:
 - Buffers e controladores IO
 - Controlador interno ao disco
 - Características das memórias (SSD) e do disco (p.e. RPM)

Nota: SSDs são mais rápidos que HDDs.

Device	Random		Sequential	
	Reads (MB/s)	Writes (MB/s)	Reads (MB/s)	Writes (MB/s)
Toshiba SSD 740 (SSD)	180	280	470	380
Seagate 8TB HDD	180	180	420	270
Seagate 8TB HDD (7200 RPM)	180	180	420	270

Figura 16.4: SSDs And Hard Drives: Performance Comparison

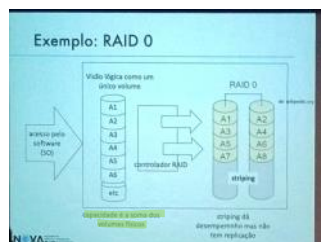
→ HDD e SSD são mt diferentes
 → mt ≠ entre ler e escrever
 → mt ≠ entre ler/escrever sequencialmente e aleatoriamente

→ SSD quanto + escritos + se gastam
 → HDD convém desfragmentar por causa do acesso

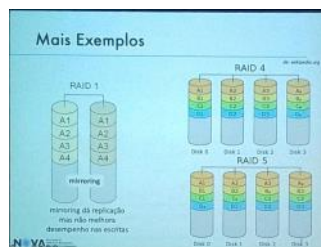
Agregando discos num volume

- RAID - Redundant Array of Inexpensive/Independent Disks
- Objetivos:
 - Redundância - tolerar falhas dos discos
 - Desempenho - latência e taxa de transferência
- Como:
 - RAID 0: duplica os dados em dois discos
 - operando em paralelo - sobre vários discos em simultâneo
- Várias organizações dos discos: RAID 0, 1, ..., 6
- Idealmente implementado nos controladores ou em disk arrays
- se por software (driver no SO) tem fraco desempenho

→ Para ter melhor desempenho, se se escrever e ler em simultâneo diminui o tempo de execução das tarefas



→ Permite dividir o dado por 2 discos
 → permite que haja uma leitura em simultâneo/paralelo
 → A_p, B_p → quando os discos de paridade, vai ter todas as alterações realizadas nos discos numa linha



→ RAID 5 é o + comum
 → RAID 1 permite que haja leituras diferentes nos mesmos dados ao mesmo tempo

Volumes

- Para o SO existem dispositivos (devices) que são volumes de dados onde guardar informação:
 - Um volume pode ser um disco ou parte de um disco partilhado
 - Um volume pode consistir numa agregação de discos em RAID
 - Um volume pode ser uma parte de RAID
- Gestão de ficheiros usa uma interface genérica para ler e escrever blocos nos volumes, suportada pelos device-drivers:
 - Um bloco = vários setores contíguos
- Para guardar ficheiros no disco, o SO tem de gravar neles as suas próprias estruturas de dados:
 - Formatar: criar novos blocos, num formato pré-definido, as estruturas de dados "memórias" → "o sistema de ficheiros em disco"
 - Ou formatação para mais organização
 - Format/diskpart (windows), mkfs (linux/macos)
- Um disco pode começar por um boot block, código de arranque do SO (os 14 sectores)

→ Bloco = conjunto de setores
 → vantagem de ter blocos grandes → melhor desempenho
 → desvantagem de ter blocos grandes → gestão do espaço (se o tamanho for mais pequeno que o espaço disponível haverá fragmentação)
 → um volume formatado para swap não é utilizado para gestão de ficheiros?
 → há discos que tem algo que quando o pc arranca vai à procura?

Vários discos lógicos separados

- Cada volume tem o seu Sistema de Ficheiros
- Em MS-Windows, normalmente, cada um corresponde a uma "drive"

→ A
→ D
→ C

Sistema de ficheiros integrado

- Em Unix, há um sistema de ficheiros integrado
- O root file system está presente desde o início (boot)
- Qualquer outro volume tem de ser integrado na hierarquia já existente, para ser acessível aos processos
- O SF no volume é colocado (mounted) numa diretoria que funciona como mount point.
- É possível que cada volume tenha um sistema de ficheiros diferente.

SF corrente /dev/sda1

* Existem 2 interfaces (a do bloco e a)

UNIX file systems

- Os SF no Unix incluem normalmente:
 - Superblock: descrição do volume e da formatação
 - id do SF, sua dimensão, nº de inodes e de blocos de dados, dim dos blocos, metadados, etc.
 - Índice de inodes e bitmap de blocos de dados: que inodes e blocos de dados estão livres ou em uso
 - Os inodes: informação de todos os nds (ficheiros, diretórias, etc).
 - Blocos de Dados: os blocos com o conteúdo das diretórias e dos ficheiros

→ seja um descrição do tipo de ficheiros que estão lá formatados, o volume, os inodes (pode dizer onde estes começam) ⇒ 1º bloco é o superblock
→ A formatação do disco é feita de acordo com o objetivo de utilização do disco
→ a seguir ao superblock costumam estar os blocks associados à gestão

Visão do utilizador: espaço de nomes

- Espaço de nomes hierárquico
- Cada nome (nd) pode ser uma diretoria, um ficheiro em disco ou noutro dispositivo...
- Cada diretoria contém uma lista de outras diretórias e/ou ficheiros

Estrutura de diretórias e ficheiros

- Um ficheiro/diretória é um i-node e pelo menos um nome

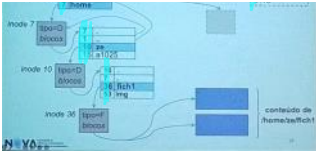
→ leitura do x dado envolve várias leituras

Caminho /home/ze/fich1

Caminho /home/ze/fich1

Caminho /home/ze/fich1

PERCURSO até chegar a /home/ze/fich1



/home/ze/fich1

Obtendo informação (stat)

```
int stat(char *path, struct stat *info)
int fstat(int fd, struct stat *info)
```

Coloca em info informação obtida tipicamente do inode:

```
struct stat {
    dev_t st_dev; /* ID of device */
    ino_t st_ino; /* inode number */
    mode_t st_mode; /* protection and type */
    uid_t st_uid; /* user ID of owner */
    off_t st_size; /* total size, in bytes */
    time_t st_mtime; /* time of last modification */
    nlink_t st_nlink; /* number of hard links */
    ...
}
```

a quem o ficheiro pertence
influencia as permissões que cada utilizador tem

Nomes e Identificadores

- Nome do ficheiro pode ser quase qualquer coisa (não pode incluir "/")
 - associado nas entradas do SF ao inode do ficheiro
- Cada utilizador tem um UID e cada grupo de utilizadores tem um GID
 - Seu nome associado ao UID e GID do utilizador e grupo
 - Cada processo tem o UID e GID do processo pai
- Cada processo tem também um UID e GID de root
 - Quando o processo é executado, o UID e GID são alterados para os do processo pai
- UID e GID são usados para verificar o acesso aos ficheiros
- Chamadas ao sistema que obtêm os identificadores para o processo:
 - int getuid(void) / int geteuid(void)
 - int getgid(void) / int getegid(void)
- O "superuser" (root) - uid 0 tem normalmente todos os privilégios

Permissões de acesso a ficheiros

- Cada nó do SF tem um uid do dono ("owner") e do grupo ("group")
 - do processo que o criou ou explicitamente alterado por chmod
- Cada nó do SF define as permissões por classes de utilizadores:
 - owner: cujo UID foi associado ao ficheiro na sua criação ou explicitamente indicado
 - group: grupo (GID) a cujo UID pertence ou explicitamente indicado para o ficheiro
 - other: nome dono nem grupo
- Permissões nos modos de acesso (a quando de open ou execute):
 - r: read -> leitura (leite) -> executar (leitura)
 - w: write -> escrita (leite) -> executar (leitura)
 - x: execute -> execução (leite) -> executar (leitura)
- Exemplo (letras representativas):

owner	group	other	teste: ls -l
111	101	100	(binkrio)
7	5	4	(base octal)

permissões de ficheiro
permissão a uma mão em programação

Máscara de criação de ficheiros

```
mode_t umask(mode_t newmask)
```

- Cada processo tem um "umask" associado
- Sempre que cria um ficheiro ou uma diretoria, os bits que estiverem a 1 em "umask" forçam a 0 os bits correspondentes no mode das permissões
- Exemplo: oldmask = umask(022);


```
define: 000 010 010
```
- fd = creat("f", 0666);
 - em vez ficar: 110 110 110 (0666 = rw-rw-rw-)
 - fica: 110 100 100 (0644 = rw-r--r--)

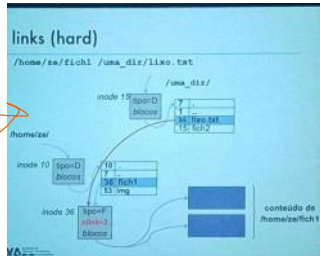
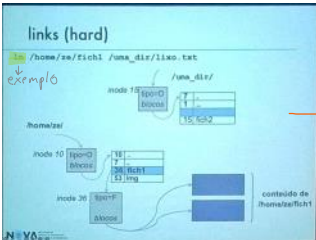
nde permissões

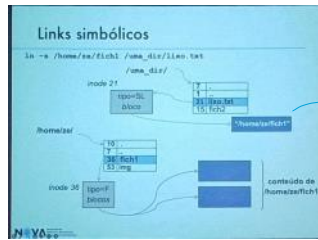
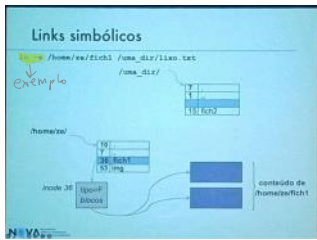
não quero dar direito de escrita ao grupo nem a outros

Nomes: link e unlink

- Podemos ter mais nomes para o mesmo ficheiro
 - se nome com open/create
 - sempre que nome a diretoria referindo-se ao mesmo inode (não podem ser diretoria)
- link: link (char *name, char *newname)
 - links simbólicos ou atalhos (não conta como link)
 - link: link (char *name, char *newname)
- Um ficheiro é apagado quando não tem mais nomes (links)
 - link: unlink (char *name)
- Para diretoria: mkdir e rmdir
 - chamadas complicadas... O que o SO precisa fazer?
- No linha de comandos temos: ls, rm, mkdir, rmdir

/home/ze/fich1
/home/paulo/xpto
→ após fazer link o xpto aponta também para o fich1
como o fich1 for apagado ele não deixa de existir, apenas para a ter o nome xpto





pode ser criado um link simbólico para algo que não exista e que vai ser criado ou para algo que não tenha acesso mas terá

Renomear/mover: rename

```
int rename(char *old, char *new)
```

- Muda o nome de ficheiros e de diretórias
 - permite mudar o nome e mover de diretoria
- Para as diretórias mantém a sua consistência
 - (i.e.)
- garante que o ficheiro não desaparece
 - Com link e unlink poderíamos ter o mesmo efeito mas sujeitos a perder o ficheiro ou duplicar

NIVEL