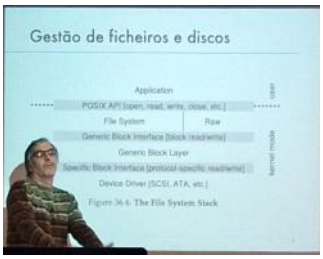
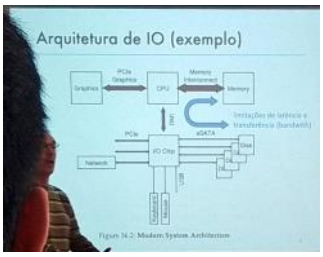
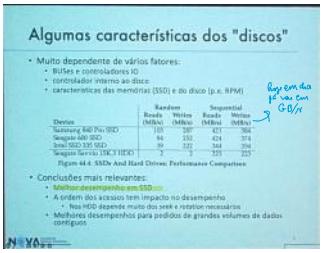
Aula16 - Gestão de Ficheiros

9 de novembro de 2023 11:41



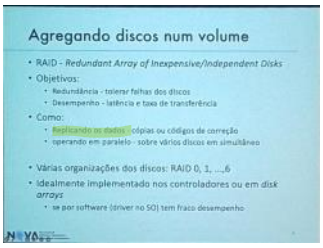


→ influencia a gestão do disco

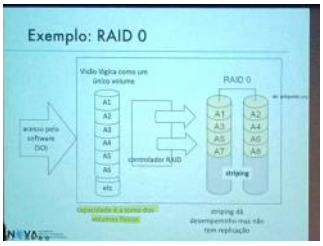


→ HDD e SSD são mt diferentes
→ mt ≠ entre ler e escrever
→ mt ≠ entre ler/escrever sequencialmente e aleatóriamente

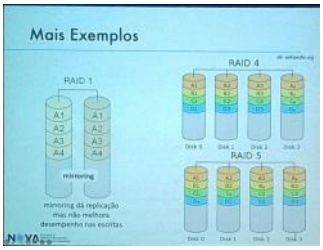
→ SSD quanto + escritos + se gastam
→ HDD convém desfragmentar por causa dos acessos



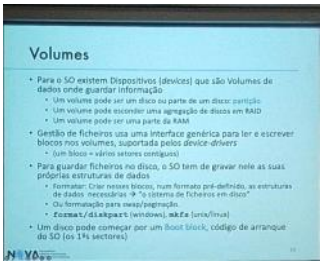
→ Para ter melhor desempenho, se se escrever e ler em simultâneo diminui o tempo de execução das tarefas



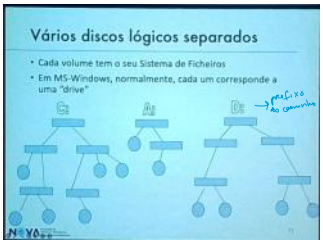
→ permite dividir os dados por 2 discos
→ permite que haja uma leitura em simultâneo/paralelo
→ A_p, B_p → guarda os discos de paridade, vou ter todas as alterações realizadas nos discos na mesma linha



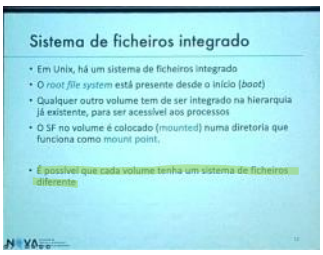
→ RAID 5 é o + comum
→ Raid 1 permite que haja leituras diferentes nos mesmos dados ao mesmo tempo

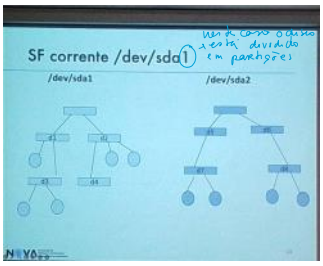


→ Bloco = Conjunto de setores
→ vantagem de ter blocos grandes → melhor desempenho
→ desvantagem de ter blocos grandes → gestão de espaço (se o tamanho for mais pequeno que o espaço disponível haverá fragmentação)
→ um volume formatado para swap não é utilizado para gestão de ficheiros ??
→ há discos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

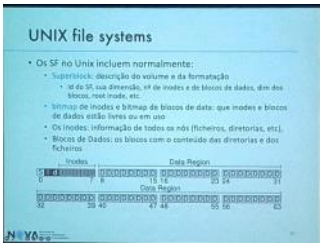


→A
→D
→C

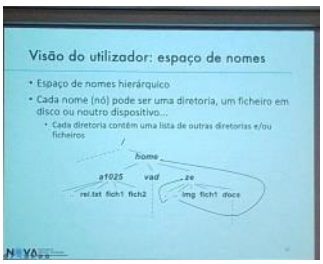


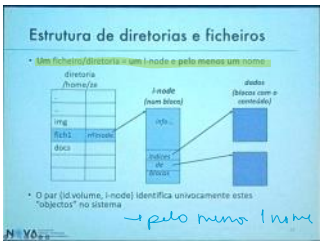


* Existem 2 interfaces (a do bloco e a)

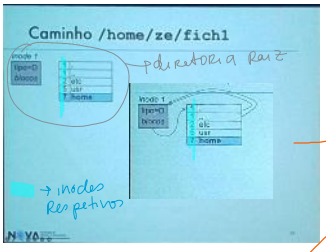


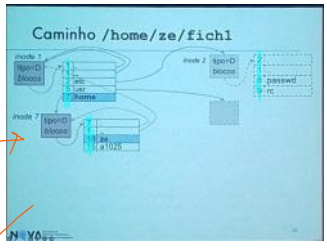
→ Seja um descritor do tipo de ficheiros que estão lá formatados, o volume, os inodes (pode dizer onde estes começam) ⇒ 1º bloco é o superblock
→ A formatação do disco é feita de acordo com o objetivo de utilização do disco
→ A seguir ao superblock costumam estar os blocks associados à gestão

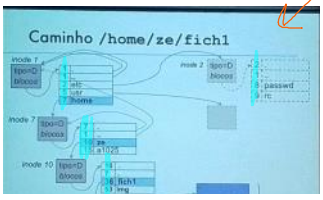




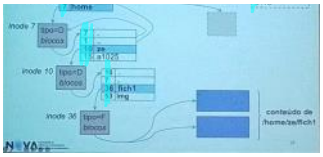
→ Leitura de x dados envolve várias leituras







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 modific.*/
    nlink_t st_nlink;/* number of hard links */
    ...
};
```

a quem o ficheiro pertence

influencia as permissões que cada utilizador tem

Nomes e Identificadores

- Nome de ficheiro pode ser quase qualquer coisa (não pode incluir "/")
  - associado nas diretorias ao nº inode do ficheiro
- Cada utilizador tem um User-ID (UID) e pertence a um grupo de utilizadores (GID)
  - Têm nomes associados nas BDs de utilizadores e grupos
  - Cada processo herda o UID e GID do processo pai
- Cada processo tem também *Effective* UID e *Effective* GID que são normalmente iguais aos UID e GID
  - Podem ser diferentes se o programa executado tiver permissão para executar como outro utilizador (set uid ou set gid)
  - EUID e EGID são usados para validar 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

do root

Permissões de acesso a ficheiros

- Cada nó do SF tem tem o uid do dono ('owner') e do grupo ('group')
  - do processo que os 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: nem dono nem grupo
- Permissões nos modos de acesso (a quando de open ou execve):
  - ler (Read) – escrever (Write) – executar (eXecute)
- Exemplo (várias representações):

```
owner  group  other
 rwx    r-x    r--      (texto: ls -l)
 111    101    100      (binário)
   7      5      4      (base octal)
S_IRWXU              S_IRWXO  (defines em sys/stat.h)
      S_IRGRP|S_IXGRP
```

permissões de ficheiros

permissões a uma mas 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
                            --- -W- -W-
fd = creat("f", 0666);
  • em vez ficar:       110 110 110 (0666 = rw-rw-rw-)
  • fica:               110 100 100 (0644 = rw-r--r--)
```

de permissões

não quero dar direitos de escrita nem ao grupo nem a outros

Nomes: link e unlink

- Podemos ter mais nomes para o mesmo ficheiro
  - 1º nome com open/create
  - acrescentar nome a diretoria referindo-se ao mesmo inode (não podem ser diretorias)

```
int link(char *name, char *newname)
```

  - links simbólicos ou atalhos (não conta como link)

```
int symlink(char *name, char *newname)
```

- Um ficheiro é apagado quando não tem mais nomes (links):

```
int unlink (char *name)
```

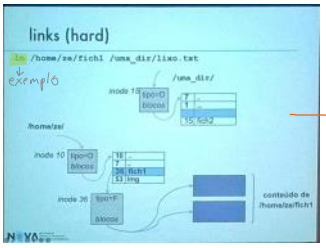
- Para diretorias: `mkdir` e `rmdir`
  - chamadas complicadas... O que o SO precisa fazer?
- Na linha de comados temos: `ls`, `rm`, `mkdir`, `rmdir`

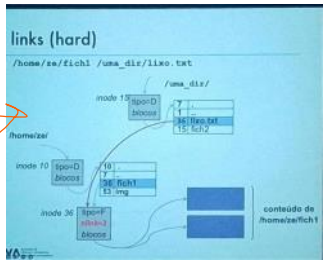
Remove

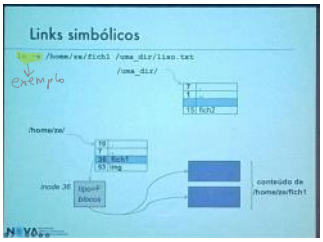
/home/ze/fich1
/home/paulo/xpto

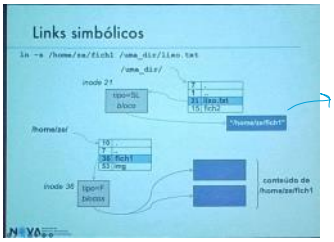
após fazer link o xpto aponta também para o fich1

caso 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 tenho acesso mas terei