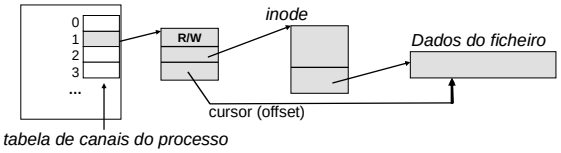
Sistema de Ficheiros: I/O

- Canais de I/O: cada processo tem um certo numero de canais virtuais que se podem ligar a ficheiros
- *File Descriptor*: o número dessa entrada na Tabela, usado nas restantes operações
- Canal I/O → uma entrada numa Tabela Global de Canais Abertos



Abertura de ficheiro: open()

`int open(char *filename, int flags)`

- Percorre as directorias para verificar permissões e encontrar o ficheiro
- Reserva uma entrada na tab. canais, com um apontador para uma tabela global de ficheiros abertos
- Na tabela global cada entrada tem:
  - O valor do cursor (offset); o modo de abertura do canal (leitura, escrita ou leitura e escrita); um apontador para a tabela de Inodes em memória
- Na tabela de Inodes fica o do ficheiro aberto

Retorno das chamadas ao SO

- As chamadas ao sistema retornam inteiros:
  - o valor devolvido pela operação:

```
f = open( "fich1", O_RDONLY);
```

  - indicação de erro: -1
    - neste caso a variável global `errno` tem o número do erro

```
f = open("fich1",O_RDONLY);
if (f == -1)  perror("fich1");
else …
```

    - o manual descreve os erros possíveis

Criação de ficheiro: creat/open

```
int creat(char *filename, int mode)
```

ou

```
int open(char *filename, int flags, int mode)
```

onde `flags` inclui `O_CREAT`

```
f = creat("novo", S_IRUSR|S_IWUSR);
```

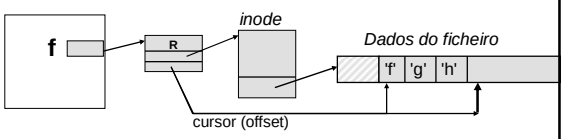
é equivalente a:

```
f = open("novo", O_WRONLY|O_CREAT|O_TRUNC,
                              S_IRUSR|S_IWUSR);
```


Leitura

`int read(int fd, void *buf, int count)`

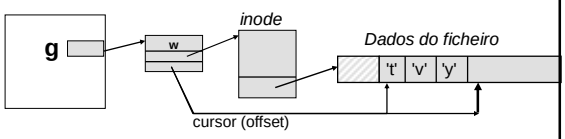
```
char b[3];
… f = open(…,O_RDONLY);
n = read( f, b, 3);
…
```



Escrita

`int write(int fd, void *buf, int count)`

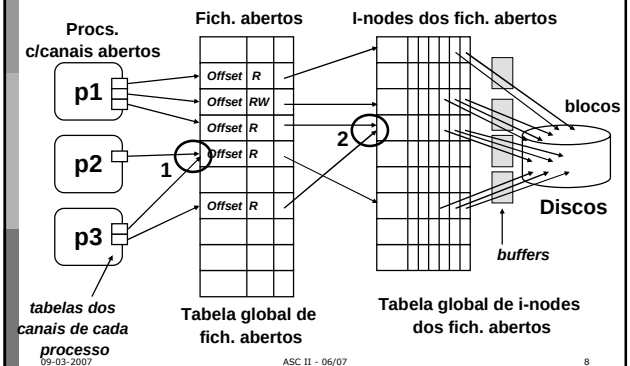
```
char b[3] = {'t','v','y'};
… g = open(…,O_WRONLY);
n = write(g, b, 3);
…
```



Ajuste do cursor

- O cursor pode ser diferente
  - na abertura indicamos outro valor inicial:
    `open("abc", O_WRONLY|O_APPEND)`
  - alteramos após a abertura:
    `lseek(int fd, int offset, int whence)`
    onde `whence` pode ser:
    `SEEK_SET, SEEK_CUR, SEEK_END`

Tabelas em memória do SO



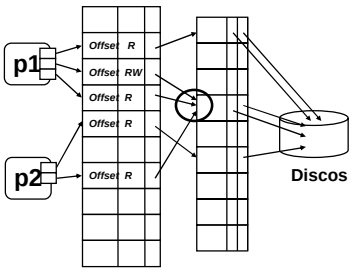
Exemplo da situação 2

- **Processo p1**

fd1 = open("f", O_RDONLY);
fd2 = open("g", O_RDWR);
fd3 = open("g", O_RDONLY);

- **Processo p2**

fd1 = open("h", O_RDONLY);
fd2 = open("g", O_RDONLY);



Partilha de informação no SO

- Todas as entidades geridas pelo SO devem ter apenas uma representação
  - essa representação é partilhada entre todos os que necessitem
  - vantagens:
    - poupa memória
    - mais fácil de gerir (p.ex. quando de uma alteração, basta actualizar uma única representação)

Abrir um ficheiro em disco (open)

- Reserva entrada na Tab.Inodes em memória: lê o I-node de disco para essa entrada
- Abrir um ficheiro já aberto: usa-se a mesma entrada na Tab.Inodes em memória
  - incrementa contador de uso dessa entrada
- Uma nova entrada na Tab. Global Ficheiros Abertos *(inicia: offset, modo, I-node)*
  (a partilha desta entrada fica para mais tarde…)
- Uma nova entrada na Tab. Canais do processo (que passa a apontar a entrada na Tab. Global)

Fechar um ficheiro: close()

`int close(int fd)`

- Fecha-se o respectivo canal de I/O!
- Liberta a entrada respectiva na tab. canais do processo
- Se foi escrito, indica ao sistema que todas as alterações terminaram
- Liberta entrada na tabela global de ficheiros abertos?
- Liberta entrada na tabela de Inodes de ficheiros abertos?

Libertação das entradas

- As entradas nas duas tabelas só são libertadas quando os seus contadores de uso ficam nulos
- Na operação close(fd):
  - liberta a entrada fd na tabela de canais do processo
  - decrementa contador da entrada na tab. global antes referida por este descritor fd
  - se chegou a zero:
    - liberta esta entrada (este canal deixa de existir)
    - decrementa a respectiva entrada na tab. Inodes
    - se esta chega a zero:
      - liberta esta entrada na tabela de Inodes (o ficheiro deixa estar em uso)

Directorias

- São como ficheiros especiais
- São alterados por chamadas próprias
  - exemplo: `creat` ou `open` com `O_CREAT`
  - outras …
- São lidas por funções próprias:
  - `opendir`, `readdir`, `closedir`
- As permissões associadas têm um significado ligeiramente diferente…

Formato das directorias

- O formato das directorias depende da versão do sistema Unix e do sistema de ficheiros no disco:
  - nas 1as versões (v7) cada entrada 16 bytes (14 – nome-ficheiro; 2-numero-inode)
  - noutras versões: nomes maiores, entradas com tamanho variável
- norma POSIX.1 (Portable Operating System Interface) define funções que uniformizam o acesso a directorias

Permissões de acesso a directorias

- Read: pode abrir e ler a lista de nomes de ficheiros na directoria
- Write: pode criar e remover ficheiros na directoria
- eXecute: pode "entrar" na directoria: pode aceder a um nome nessa directoria e pode mudar a directoria corrente para essa directoria (chamada '`chdir`')
  - Para abrir ficheiro/executar programa deve ter permissão Execute em todas as directorias do caminho absoluto do ficheiro

Funções de acesso a directorias

- `DIR *opendir(char *dname);`
- `struct dirent *readdir(DIR *dp);`
  - ler uma entrada (de cada vez)
  - devolve NULL quando atinge o fim
- `int closedir(DIR *dp);`
- `void rewinddir(DIR *dp);`

Acesso a directorias (2)

- A estrutura DIR é interna à interface e mantém informação sobre a directoria, como uma lista de entradas 'dirent'
- Cada entrada é uma estrutura:

```
struct dirent {
        inot_t d_ino;        /*i-node*/
        char d_name[NAME_MAX+1];
}
```