Linux Extend FS e VFS

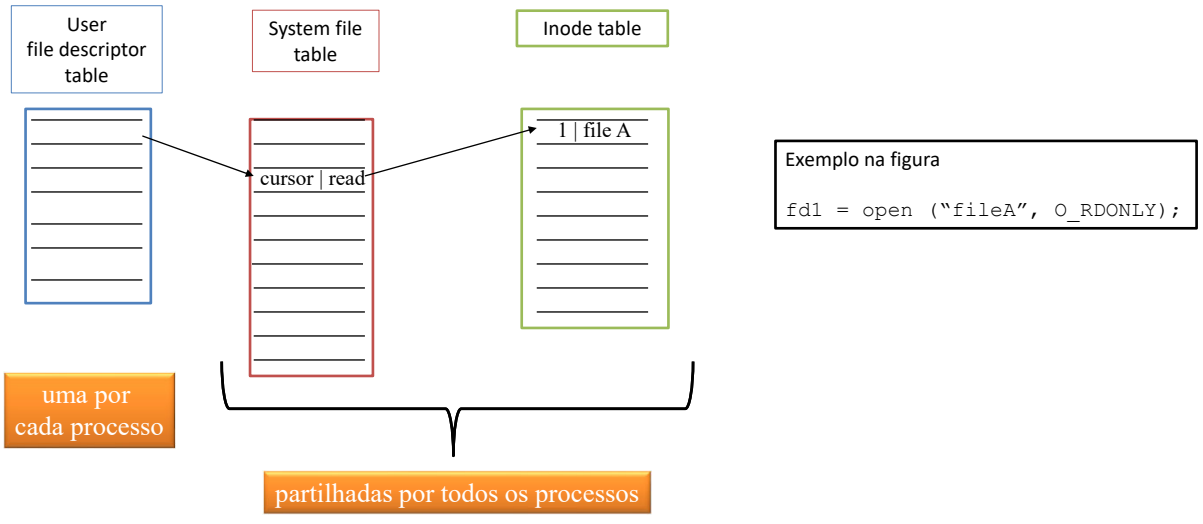
Estruturas de Suporte à Utilização dos Ficheiros

- Todos os sistemas de ficheiros definem um conjunto de estruturas em memória volátil para a gerir a informação persistente mantida em disco.
- Objetivos:
  - Criar e gerir os canais virtuais entre as aplicações e os ficheiros e diretórios
  - Aumentar o desempenho do sistema mantendo a informação em *caches*
  - Tolerar eventuais faltas
  - Isolar as aplicações da organização do sistema de ficheiros
  - Possibilitar a gestão de várias organizações de estruturas de ficheiros em simultâneo

Tabelas de Ficheiros Abertos do Unix

- Existem duas tabelas para referenciar os ficheiros abertos, mantidas no espaço de memória protegido pelo que só podem ser acedidas pelo núcleo
- **Tabela de ficheiros abertos do processo**
  - Contém um descritor para cada um dos ficheiros abertos que referencia a tabela global de ficheiros abertos, o índice nessa tabela é o *file descriptor* que é devolvido no `open()`
- **Tabela de ficheiros abertos global/sistema**
  - Contém informação relativa a um ficheiro aberto: cursor com a posição atual de leitura/escrita, modo como o ficheiro foi aberto

Tabelas de Ficheiros



Exemplo na figura

```
fd1 = open ("fileA", O_RDONLY);
```

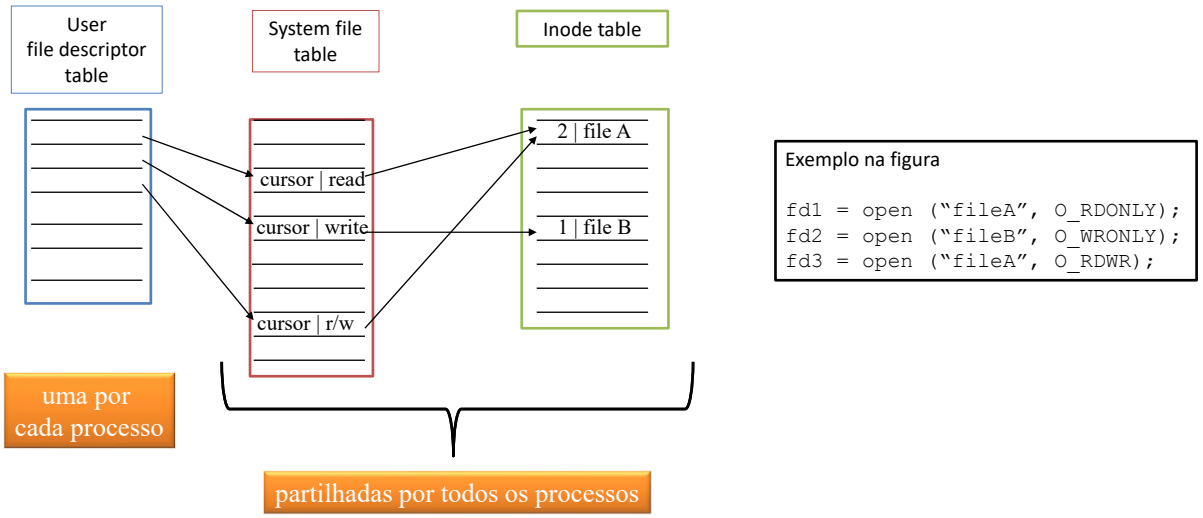
Estruturas de Suporte à Utilização dos Ficheiros

- A existência de duas tabelas justifica-se:
  - garantir o isolamento entre processos
  - permitir a partilha de ficheiros sempre que necessário (e.g. os cursores de escrita e leitura de um ficheiro entre dois ou mais processos)
  - Por exemplo um processo filho em Unix herda os ficheiros abertos do processo pai e os cursores

Objeto ficheiro

- Quando um processo chama `open()` é criado um objeto ficheiro e colocado na **primeira posição livre** da **tabela de descritores do processo** um ponteiro para esse objeto, sendo devolvido ao utilizador o índice dessa entrada na tabela (*file descriptor*)
- **Pode existir mais do que um objeto ficheiro para o mesmo ficheiro**. Se dois processos abrirem o mesmo ficheiro cada um deles fica com um ponteiro para um objeto ficheiro diferente, pois de outro modo partilhariam o mesmo cursor
- Acontece exatamente o mesmo **se um processo abrir um ficheiro duas vezes**. Por exemplo, se um processo abrir um ficheiro para leitura e abri-lo de novo para escrita fica com dois objetos ficheiro diferentes com dois cursores distintos, um para leitura e outro para escrita,

Tabelas de Ficheiros



Exemplo na figura

```
fd1 = open ("fileA", O_RDONLY);
fd2 = open ("fileB", O_WRONLY);
fd3 = open ("fileA", O_RDWR);
```


Exercício

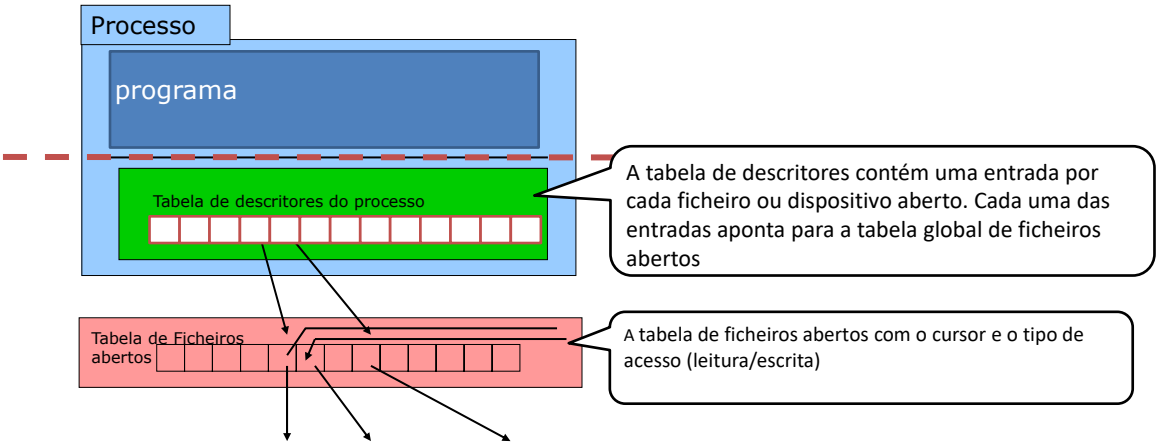
After each of the calls to write() in the following code, explain what the content of the output file would be, and why:

```
fd1 = open(file, O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
fd2 = dup(fd1);
fd3 = open(file, O_RDWR);
write(fd1, "Hello,", 6);
write(fd2, "world", 6);
lseek(fd2, 0, SEEK_SET);
write(fd1, "HELLO,", 6);
write(fd3, "Gidday", 6);
```

Duplica o *file de*scriptor apontando para a o mesmo objeto ficheiro

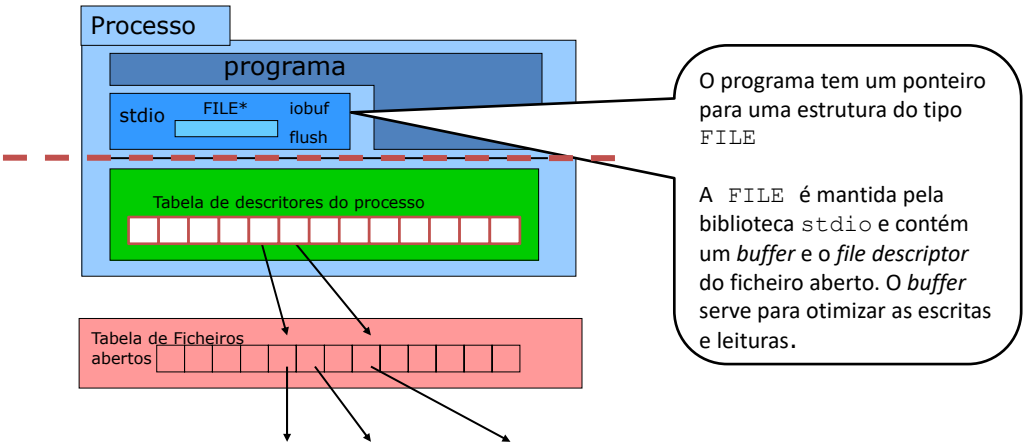
Linux programming interface

Visão Global



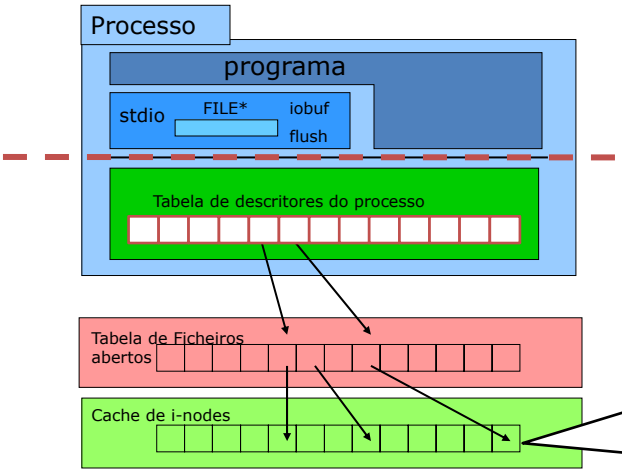
10

Visão Global – com stdio



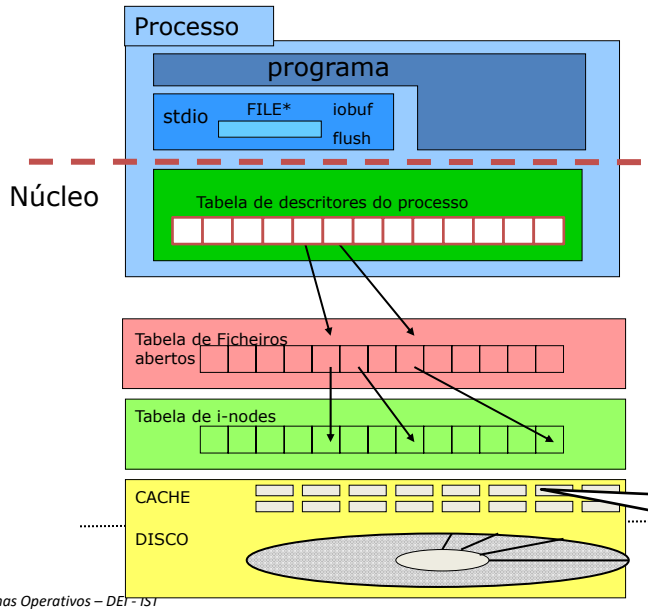
11

Visão Global



A tabela de *inodes* é uma cache dos *inodes* em disco.
Os índices no inode ou em blocos auxiliares referenciam os blocos do disco com a informação do ficheiro.

Visão Global

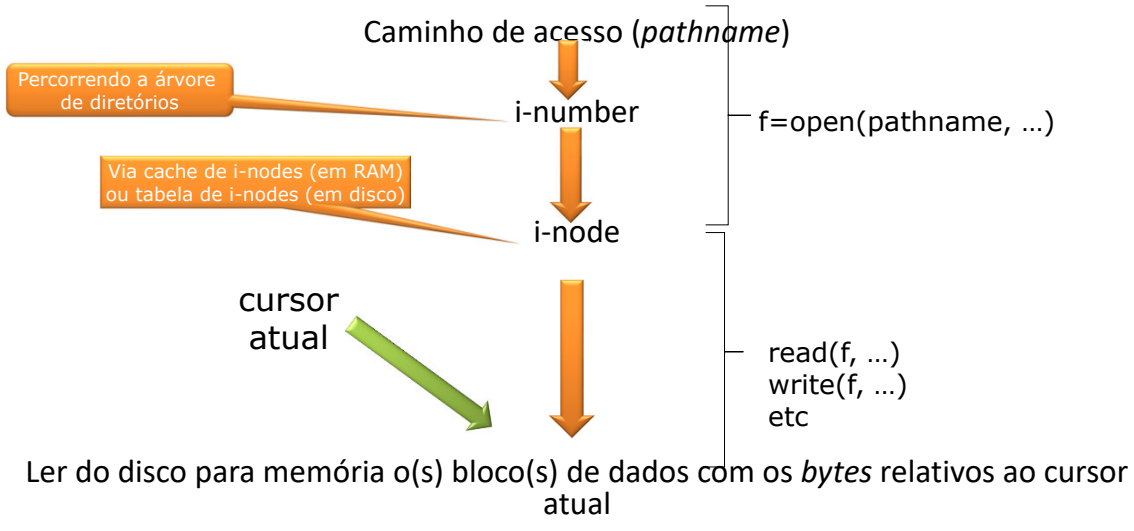


Cache de bloco em memória, fundamental para reduzir acessos ao disco

Ler um Ficheiro

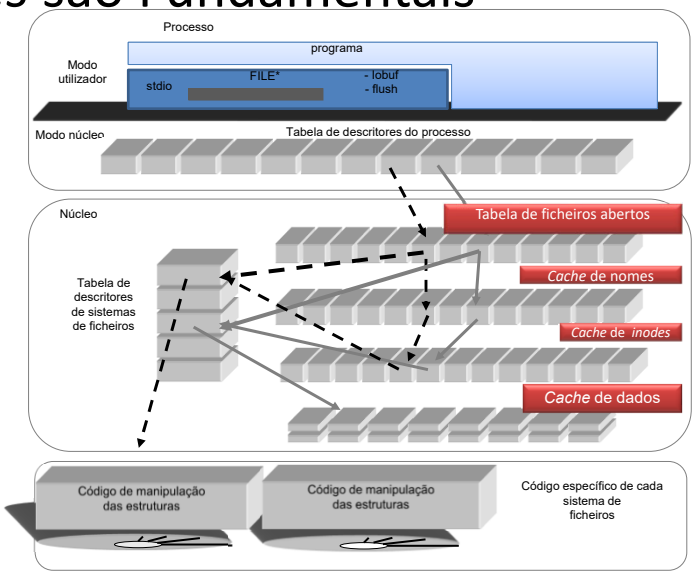
- Para tal é preciso traduzir um tuplo `<inode, cursor>` para um endereço de memória:
  - Localizar o bloco de disco onde está a informação a aceder: calcular qual a ordem do bloco a aceder para determinar onde ir buscar o índice respetivo
  - Obter o *inode* e os eventuais blocos de índices para chegar ao número do bloco
  - Ler o bloco para a memória principal
  - Calcular o endereço de memória dentro do bloco mapeado em memória que contém a informação a aceder
- Otimizações das operações:
  - dimensões dos blocos e dos índices são potências de 2
  - as operações de multiplicação e divisão são substituídas por operações de deslocamento de *bits* (instruções *shift* do processador), efetuadas de forma muito eficiente
  - A informação pode já estar em memória nas diversas *caches*

Sequência para aceder ao conteúdo de um ficheiro



As *Caches* são Fundamentais

- Falta ver com algum detalhe as caches fundamentais para otimizar o desempenho

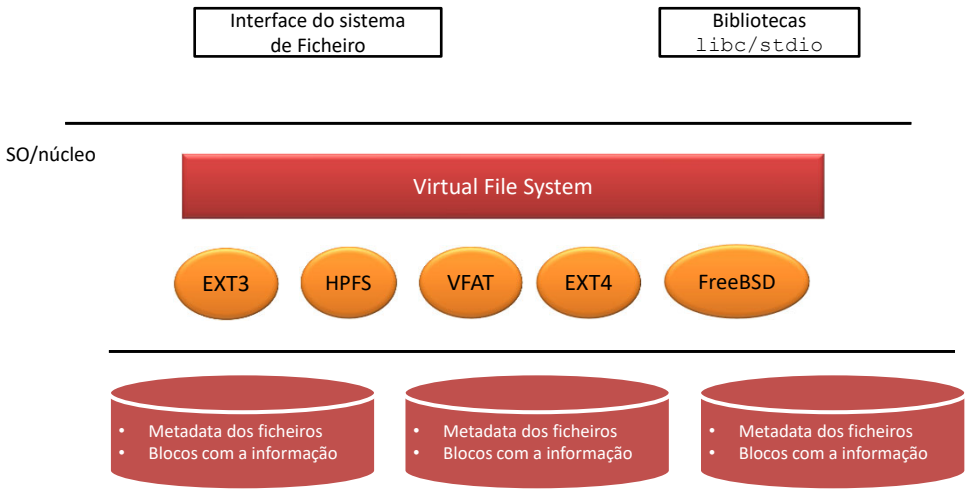


Virtual File System em Linux

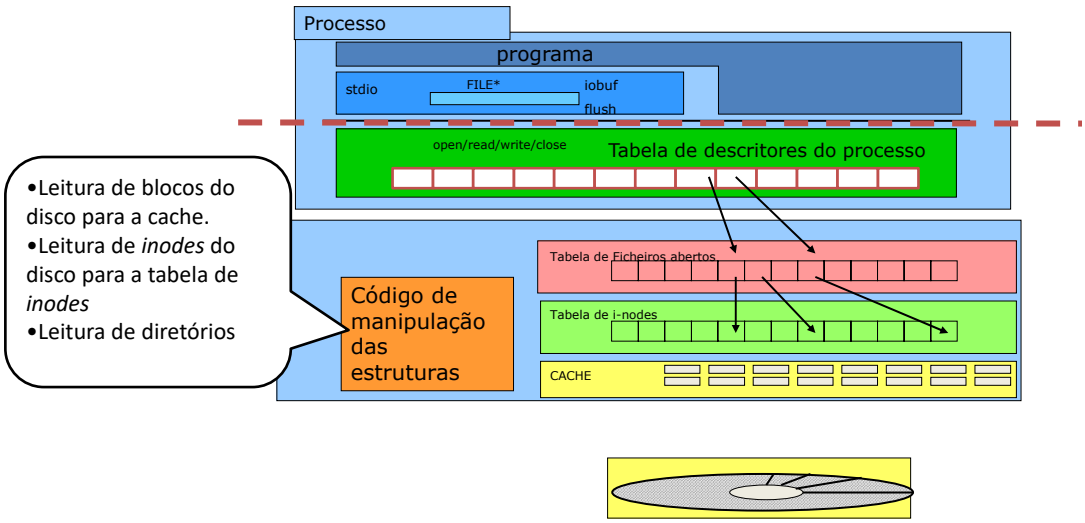
Estruturas em Memória - VFS

- O objectivo do VFS :
  - a criação de um sistema de ficheiros comum, virtual, que suporta vários sistemas de ficheiros nativos, em simultâneo.
- Neste modelo:
  - cada ficheiro é manipulado por um conjunto de operações (leitura, escrita, abertura, etc.) diferente,
  - dependendo do sistema de ficheiros nativo em que estão armazenados

Organização VFS



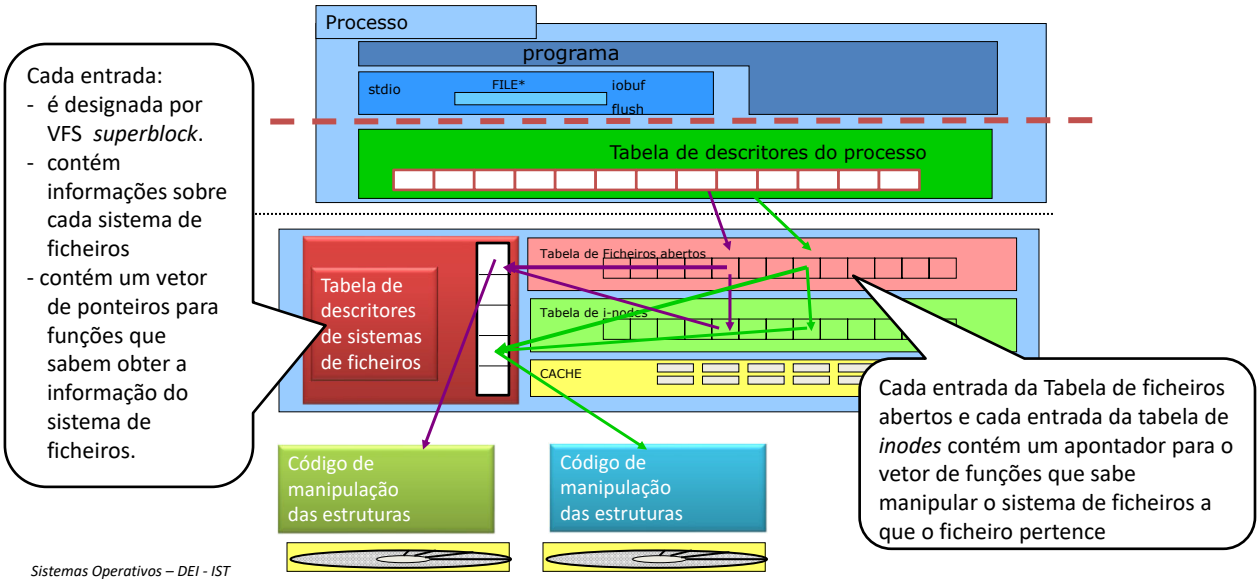
Um único Sistema de Ficheiros



- Leitura de blocos do disco para a cache.
- Leitura de *inodes* do disco para a tabela de *inodes*
- Leitura de diretórios

20

Mais do que um Sistema de ficheiros



Cada entrada:
- é designada por VFS *superblock*.
- contém informações sobre cada sistema de ficheiros
- contém um vetor de ponteiros para funções que sabem obter a informação do sistema de ficheiros.

Cada entrada da Tabela de ficheiros abertos e cada entrada da tabela de *inodes* contém um apontador para o vetor de funções que sabe manipular o sistema de ficheiros a que o ficheiro pertence

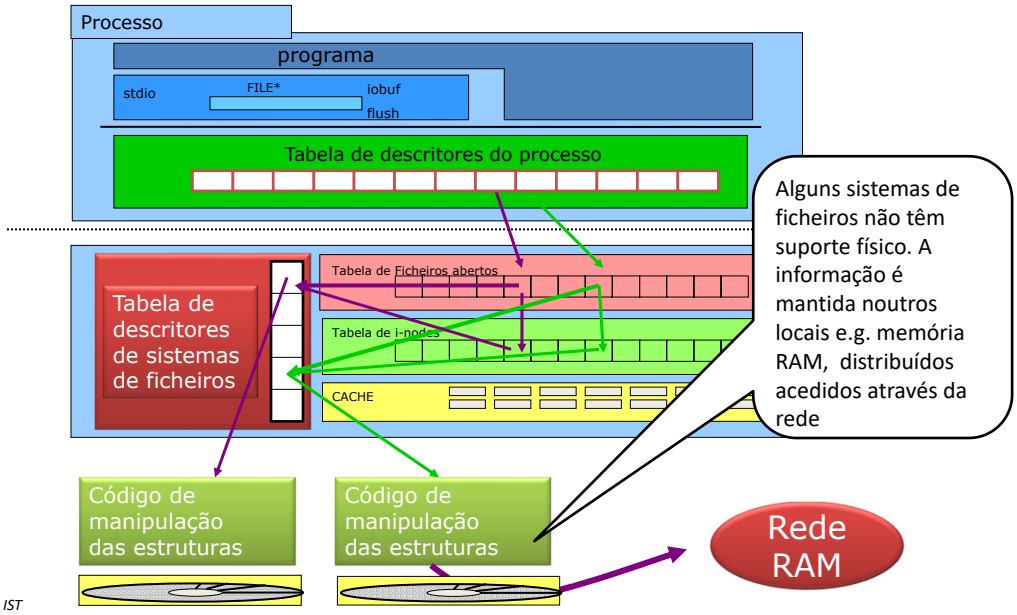
21

Constituição dos objetos ficheiro

```
struct file {
   struct list_headf_list;    // Ponteiro para o próximo elemento na lista
   struct dentry*f_dentry;    // Ponteiro para o objecto dentry associado
   struct vfsmount *f_vfsmnt; // Ponteiro para o sistema de ficheiros
   struct file_operations*f_op;// Ponteiro para a tabela de despacho
   atomic_tf_count;           // Número de utilizações do ficheiro
   unsigned int  f_flags;     // Flags especificadas na abertura do ficheiro
   mode_t  f_mode;            // Modo de acesso
   int   f_error;             // Código de erro para escrita em rede
   loff_t  f_pos;             // Posição actual de leitura ou escrita
   struct fown_struct f_owner;// Dados para notificação assíncrona
   unsigned int  f_uid, f_gid;// Id do dono e do grupo
   struct file_ra_statef_ra;  // Dados para a leitura em avanço
   unsigned long f_version;   // Versão incrementada em cada uso
   void  *f_security;         // Estrutura de segurança genérica (SELinux)
   void  *private_data;       // Necessário para os periféricos
   struct list_headf_ep_links;// Lista de eventos para manipulação assíncrona
   spinlock_t f_ep_lock;      // Lock para protecção da lista de eventos
   struct addressspace*f_mapping;// Ficheiro mapeado em memória
};
```

https://docs.huihoo.com/doxygen/linux/kernel/3.7/structfile.html

Sistemas de Ficheiros virtuais



Journaling

- O objetivo da camada do núcleo *Journal Block Device* (JBD) é impedir que o sistema de ficheiros fique num estado inconsistente.
  - A JBD é atualmente utilizada apenas pelo sistema de ficheiros EXT3 (terceira versão do sistema de ficheiros *extend*).
- O JBD evita que uma operação de escrita seja parcialmente realizada, sendo efetuada de forma atómica.
- O JBD:
  - escreve os blocos modificados no *journal*, e só depois de se ter assegurado que a informação já existe de forma persistente, é que a atualiza nos blocos do sistema de ficheiros.
  - se acontecer uma falha nesse período o sistema operativo no reinício pode recuperar e refazer a escrita pois ela estava registada no *journal*.

Journaling

- A dupla escrita, no *journal* e no sistema de ficheiros, introduz uma penalização no desempenho do sistema pelo que muitos sistemas de *journal* só garantem a consistência dos metadados do sistema de ficheiros.
- O JBD permite configurar o tipo de *journaling* que se pretende efectuar, na associação do sistema de ficheiros a um directório (*mount*).
- Para além das duas opções já referidas o JBD fornece ainda uma terceira opção (por omissão):
  - apenas os metadados são enviados para o *journal,*
  - mas os blocos de dados relativos a um bloco de metadados são escritos primeiro que os metadados

Conclusões

- As estruturas de dados em memória RAM são fundamentais para o funcionamento do sistema de ficheiros
- Em Linux, existem duas tabelas para os ficheiros abertos, uma pertence a cada processo e outra é global ao sistema, contendo esta o cursor e o modo de abertura
- Na tabela de ficheiros abertos globais a entrada (objeto ficheiro) aponta para a cache de *inodes* para poder aceder à metadata em particular os índices de blocos
- No Linux, é possível no VFS montar diferentes sistemas de ficheiros. O objeto ficheiro tem uma tabela que redireciona a execução para as funções do SF a que o ficheiro pertence