Universidade de Aveiro

Departamento de Electrónica, Telecomunicações e Informática

Sistemas de Operação

(Ano Letivo de 2016/17)

sofs16 setembro de 2016

Nota prévia

O *sofs16* é um sistema de ficheiros simples e limitado, baseado no sistema de ficheiros ext2 do Linux, que foi concebido com propósitos meramente didáticos e é destinado a ser desenvolvido nas aulas práticas de Sistemas de Operação no ano letivo de 2016/2017. O suporte físico considerado é um ficheiro regular do sistema de ficheiros da plataforma hardware que vai ser usada no seu desenvolvimento.

1 Introdução

Quase todas os programas, durante a sua execução, produzem, consultam e/ou alteram quantidades variáveis de informação que é armazenada de um modo mais ou menos permanente em dispositivos físicos agrupados sob o nome genérico de memória de massa. Cabem nesta categoria os discos magnéticos, os discos ópticos e os SSD, entre vários outros.

Independentemente do suporte físico, estruturalmente verifica-se que:

- os dispositivos de armazenamento de massa são normalmente vistos de um modo lógico como um array de blocos, cada bloco contendo entre 256 e 8K bytes;
- os blocos são numerados (modelo LBA) e o acesso a um bloco particular, para leitura ou escrita, é feito na sua globalidade, fornecendo o seu número identificador.

A manipulação da informação contida diretamente no dispositivo físico não pode ser deixada à responsabilidade do programador de aplicações. A complexidade inerente à sua estrutura interna e a necessidade de garantir critérios de qualidade, relacionados com a eficiência no acesso e a integridade e a partilha, exigem a criação de um modelo uniforme de interação.

O conceito de **ficheiro** surge, assim, como a unidade lógica de armazenamento em memória de massa e de acesso à informação. Quer com isto dizer-se que a leitura e a escrita de dados em memória de massa se faz sempre no âmbito estrito de um ficheiro.

Sob o ponto de vista do programador de aplicações, um ficheiro pode ser visto como um tipo de dados específico, caracterizado por um conjunto de atributos e por um conjunto de operações. O papel do sistema de operação é implementar este tipo de dados, fornecendo um conjunto de chamadas ao sistema que estabelecem uma interface simples e segura de comunicação com a memória de massa. A parte do sistema de operação que se dedica a esta tarefa, designa-se por **sistema de ficheiros**. Diferentes implementações conduzem a diferentes tipos de sistemas de ficheiros. Os sistemas de operação atuais suportam diferentes sistemas de ficheiros, associados, quer com dispositivos físicos distintos, quer com o mesmo dispositivo.

1.1 Ficheiro como um tipo de dados

Os atributos de um ficheiro são variados e dependem da implementação. Destaca-se aqui um conjunto mínimo que está habitualmente presente:

Nome — o ficheiro passa a ter uma identidade própria com a atribuição de um **nome**, tornando-se independente do processo e do utilizador que o criaram, e mesmo do sistema de operação em que foi criado.

Identificador interno — o nome, enquanto elemento individualizador de um ficheiro, é adequado para uma referência externa (tipicamente de origem humana), mas é pouco prático em termos de organização interna; o **identificador interno** constitui, por isso, o elemento de discriminação que vai permitir referenciar os outros atributos do ficheiro e, através deles, a informação propriamente dita.

tamanho — o **tamanho** representa o comprimento do conteúdo informativo, normalmente em número de bytes.

pertença — de forma a permitir controlo de acesso, o ficheiro tem indicação de a quem pertence (normalmente quem o criou).

proteção — conjugado com a pertença, este atributo permite o controlo de acesso, especificando quem pode ler o seu conteúdo, escrever nele ou realizar operações de execução.

monitorização de acesso — datas e tempos dos instantes de criação, de última modificação e de último acesso, por exemplo.

localização do conteúdo informativo — sendo o acesso à memória de massa realizado bloco a bloco, é necessário manter uma lista ordenada com a identificação dos blocos que contêm o seu conteúdo informativo.

tipo — os sistemas de ficheiros admitem ficheiros de tipos diversos. Consideram-se aqui 3 tipos:

ficheiros regulares — são os ficheiros convencionais; folhas de cálculo, documentos e programas são exemplos de ficheiros regulares.

diretórios — são ficheiros internos, com um formato pré-definido, que permitem definir a estrutura hierárquica com que os ficheiros regulares são tipicamente acedidos.

atalhos — são ficheiros internos, com formato pré-definido, que descrevem a localização de um outro ficheiro através da especificação de um caminho, absoluto ou relativo, na estrutura hierárquica pré-existente

As operações que se podem realizar sobre um ficheiro são variadas e dependem do sistema de operação. Há, porém, um núcleo base que de algum modo está sempre presente. Essas operações são disponibilizadas através de chamadas ao sistema. Apresentam-se a seguir listas, não exaustivas, de chamadas ao sistema disponibilizadas pelo Linux, para os 3 tipos de ficheiros acima considerados.

- comuns aos 3 tipos: `open`, `close`, `chmod`, `chown`, `utime`, `stat`, `rename`.
- ficheiros regulares: `mknod`, `read`, `write`, `truncate`, `lseek`, `link`, `unlink`.
- diretórios: `mkdir`, `rmdir`, `getdents`.
- atalhos: `symlink`, `readlink`, `link`, `unlink`.

1.2 FUSE

Em termos gerais, a introdução de um novo sistema de ficheiros no sistema de operação implica a realização de duas tarefas bem definidas. A primeira consiste na integração do código associado à implementação do tipo de dados ficheiro no núcleo, kernel, do sistema de operação e, a segunda, na sua instanciação sobre um ou mais dispositivos de memória de massa do sistema computacional.

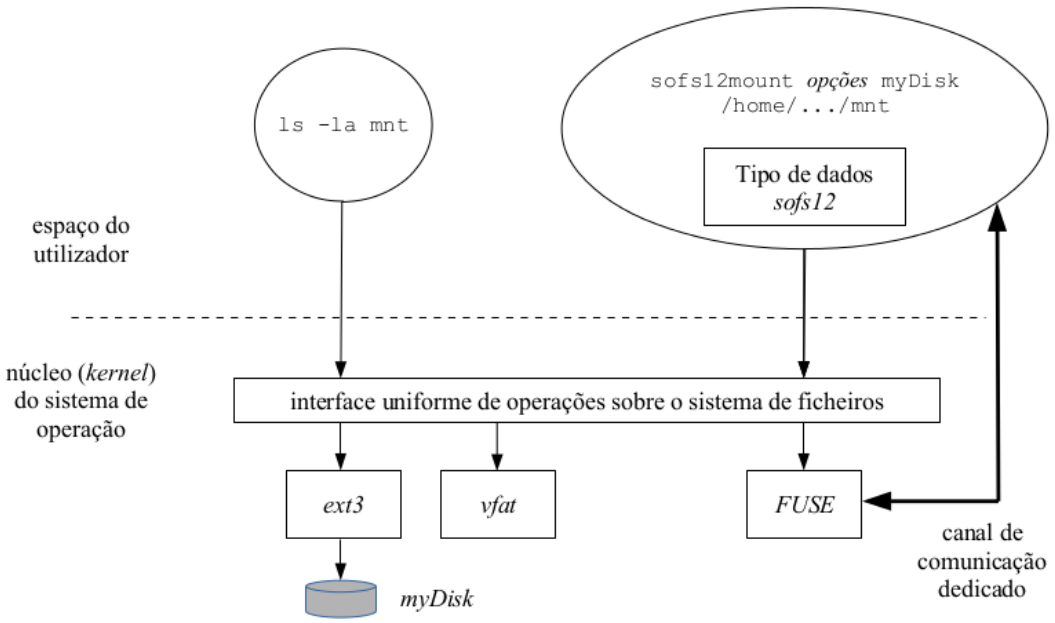
Em situações de kernel monolítico, a integração do código traduz-se na criação de um novo kernel através da compilação e linkagem dos diferentes ficheiros fonte descritivos do sistema de operação. Em situações de kernel modular, o módulo associado é compilado e linkado separadamente, sendo acoplado a um kernel pré-existente em run time. Qualquer que seja o caso, porém, trata-se de uma tarefa muito complexa e especializada e que exige um conhecimento profundo das estruturas de dados internas e da funcionalidade apresentada pelo sistema de operação-alvo, estando, por isso, só ao alcance de programadores de sistemas experientes.

O FUSE (File system in User SpacE) constitui uma solução alternativa muito engenhosa que visa a construção de sistemas de ficheiros no espaço do utilizador, como se se tratasse de meras aplicações, impedindo que eventuais inconsistências e erros que surjam na sua implementação, sejam transmitidos diretamente ao kernel e conduzam à sua inoperacionalidade.

A infraestrutura oferecida pelo FUSE é formada por duas partes principais:

- módulo de interface com o sistema de ficheiros — funciona como um mediador de comunicações entre o interface uniforme de operações sobre ficheiros fornecido pelo kernel e a respetiva implementação em espaço do utilizador;
- biblioteca de implementação — fornece as estruturas de dados de comunicação e o protótipo das operações que têm que ser desenvolvidas para garantir a compatibilidade operacional do tipo de dados definido pelo utilizador com o modelo subjacente; fornece, além disso, um conjunto de funcionalidades destinadas a instanciar o tipo de dados sobre um dispositivo de memória secundária e à sua integração no sistema de operação.

O diagrama abaixo ilustra o tipo de organização que resulta quando o sistema de ficheiros *sofs16*, instanciado sobre o dispositivo `myDisk`, representado aqui por um ficheiro do sistema de ficheiros ext3 de Linux, é integrado no sistema de operação usando o FUSE.



2 Arquitetura do *sofs16*

Há 4 entidades fulcrais na definição do sistema de ficheiros *sofs16*: superbloco, nó-i, cluster, diretório.

- O **superbloco** é uma estrutura de dados, armazenada no bloco número 0, que contém atributos globais relativos ao disco como um todo ou às restantes estruturas de dados.
- O **nó-i** (*inode* em inglês, de nó identificador) é uma estrutura de dados que contém todos os atributos de um ficheiro, com a exceção do nome. Há uma região contígua do disco, designada **tabela de nós-i**, reservada para alojamento de todos os nós-i. Isto significa que a identificação de um nó-i pode ser feita pelo índice que representa a sua posição relativa na tabela.
- Os blocos do disco reservados para o armazenamento da informação dos ficheiros estão organizados em grupos de blocos contíguos, designados **clusters**. Um campo do superbloco indica o tamanho do cluster em número de blocos. A identificação de um cluster é feita pelo índice que representa a sua posição relativa na zona de clusters.
- Os diretórios, como referido anteriormente, são ficheiros internos que permitem definir a estrutura hierárquica de acesso aos ficheiros. Um **diretório** é uma estrutura de dados constituída por um conjunto de **entradas de diretório**, cada uma associando um nome ao identificador de um nó-i. Sendo dados de um ficheiro, as entradas de um diretório são armazenadas em clusters. Assume-se que o diretório raiz ocupa o nó-i número 0.

Numa visão muito geral, os N blocos de um disco *sofs16* estão divididos em 3 região, tal como mostra a figura seguinte.

2.1 Lista de nós-i livres

O número de nós-i de um disco *sofs16* é fixo. Num dado momento, haverá nós-i que estão a ser usados por ficheiros alojados no disco e haverá outros que estão disponíveis. Quando se pretende criar um novo ficheiro é necessário atribuir-lhe um nó-i que esteja livre. É por isso necessário:

- definir uma política de utilização dos nós-i livres que permita decidir qual usar no momento em que um seja necessário.
- definir e manter no disco uma estrutura de dados que permita implementar a política anterior;

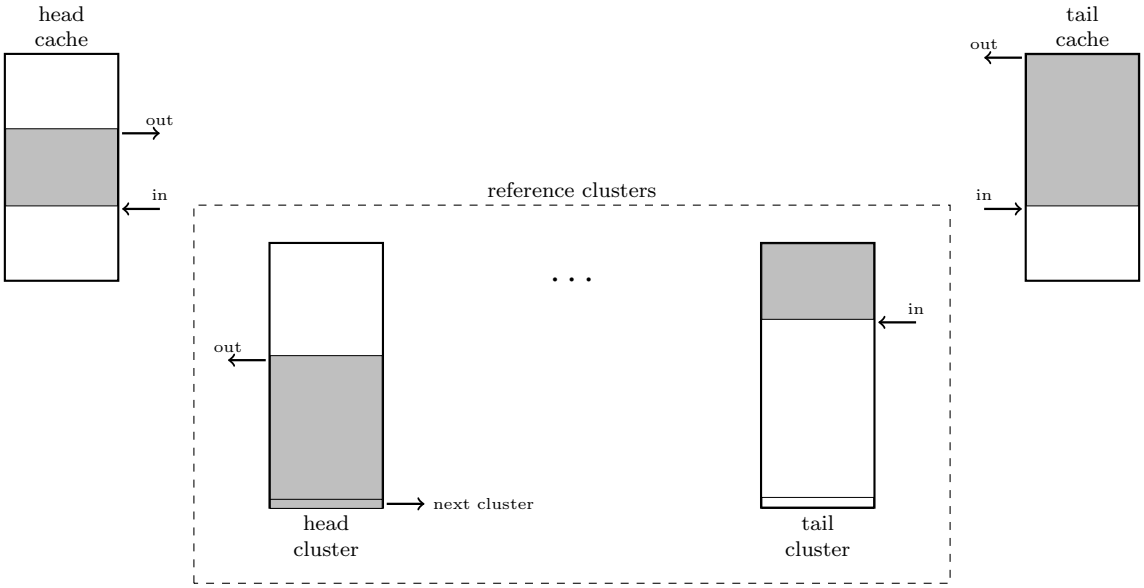
No *sofs16*, usa-se uma política de utilização tipo FIFO, determinando que o primero nó-i livre a ser utilizado é aquele que está há mais tempo livre. A implementação baseia-se numa lista ligada de nós-i livres. A lista é construída usando os próprios nós-i. Um campo da estrutura de dados do nó-i, quando este está livre, contém o índice do próximo nó-i livre. Dois campos do superbloco contêm os índices do primeiro e do último nós-i livres.

2.2 Lista de clusters livres

O número de clusters de um disco *sofs16* é também fixo. Semelhantemente ao que foi dito para os nós-i, num dado momento, haverá clusters que estão a ser usados por ficheiros alojados no disco e outros que estão disponíveis. Também semelhantemente, será necessário:

- definir uma política de utilização dos clusters livres que permita decidir qual usar no momento em que um seja necessário.
- definir e manter no disco uma estrutura de dados que permita implementar a política anterior;

No *sofs16*, para a gestão dos clusters livres, também se usa uma política tipo FIFO, determinando que o primero cluster livre a ser utilizado deve ser aquele que está a mais tempo livre. A implementação da lista, neste caso, é, no entanto, bastante diferente da usada para os nós-i. O FIFO é implementado de forma indireta, usando o superbloco e os clusters para armazenar a sequência de referências de clusters livres. A figura seguinte ilustra essa implementação.



Os primeiros e os últimos elementos da lista de clusters livres estão armazenados no superbloco, em duas regiões designadas, respetivamente, por `head cache` e `tail cache`. Funcionam como buffers circulares. Os restantes elementos da lista estão armazenados em clusters. A última entrada de cada um destes clusters é usada para indicar qual o próximo cluster com referências. O primeiro e o último clusters contendo referências de clusters livres são designados, respetivamente, por `head cluster` e `tail cluster`. Campos do superbloco contém a referência do `head cluster` e o índice da sua primeira posição ocupada, assim como a referência do `tail cluster` e o índice da sua primeira posição vazia.

2.3 Lista das referências dos clusters de dados de um ficheiro

Os clusters de dados de um ficheiro são de uso exclusivo, ou seja, cada cluster apenas pode pertencer a um ficheiro. O número de clusters necessários para albergar a informação de um ficheiro é, por isso, dado por

$$N_c = \texttt{roundup}(\frac{\texttt{size}}{\texttt{bpc} * \texttt{BLOCK\_SIZE}})$$

onde `size`, `bpc` e `BLOCK_SIZE` representam respetivamente o tamanho do ficheiro em bytes, o número de blocos por cluster e o tamanho do bloco em bytes.

Na atividade habitual de um disco é impraticável que todos os clusters de dados de um ficheiro ocupem uma zona contígua do disco. É, portanto, necessário definir uma estrutura de dados para definir a sequência de clusters usados por cada ficheiro.

N_c pode ser um número muito elevado. Considerando, por exemplo, que o bloco mede 512 bytes (o habitual) e que cada cluster corresponde a 2 blocos, um ficheiro de 1 GByte ocupa 1 milhão de clusters. Mas, N_c também pode ser muito pequeno. Na realidade, para um ficheiro de 0 bytes, N_c é igual a 0. Faz, por isso, todo o sentido que a estrutura de dados seja flexível, podendo crescer à medida das necessidades.

O acesso aos dados de um ficheiro não é em geral sequencial, mas sim aleatório. Imagine, por exemplo, que num dado momento é necessário aceder ao byte índice j de um dado ficheiro. Qual é o cluster do disco que contém esse byte? Dividindo j pelo tamanho do cluster em bytes, obtém-se o índice do cluster no contexto do ficheiro que contém esse byte. A estrutura de dados deve permitir identificar de forma eficiente qual o número do cluster pretendido.

A estrutura de dados usada foi o array dinâmico. No *sofs16*, cada nó-i em uso possui um array, designado `d`, que identifica os clusters si usados para armazenamento dos seus dados. Sendo `BPC` o número de bytes por cluster, `d[0]` indica o número do cluster do disco que contém os primeiros `BPC` bytes do ficheiro, `d[1]` os seguintes `BPC` bytes, e assim sucessivamente.

Os primeiros `5` elementos do array `d` são armazenados diretamente no nó-i, no campo precisamente designado `d`. Os elementos seguintes, quando existem, são armazenados de forma indireta e duplamente indireta. O campo `i1` do nó-i representa um array de `2` posições, cada uma indicando o número de um cluster usado para estender o array `d`. Na realidade, sendo `rpc` o número de referências a clusters que se podem armazenar num cluster, `i1[0]` é o número do cluster que contém os elementos `5` a `rpc+4` do array `d` e `i1[1]` é o número do cluster que contém os elementos `rpc+5` a `2*rpc+4`. O campo `i2` do nó-i é usado para estender o array `i1`. Na realidade, `i2`, quando existe, indica o número de um cluster que contém os elementos `2` a `rpc+1` do array `i1`.

O padrão `NULL_REFERENCE` é usado para representar uma referência não existente. Por exemplo: se `d[1]` for igual a `NULL_REFERENCE`, o ficheiro não possui o cluster de dados índice `1`; se `i1[0]` for igual a `NULL_REFERENCE`, `d[5]` a `d[rpc+4]` são iguais a `NULL_REFERENCE`.

2.4 Diretórios

Como referido anteriormente, um diretório pode ser visto como um array de entrada de diretório. Cada entrada de diretório é uma estrutura de dados contendo um array de bytes de tamanho fixo para armazenamento de um nome e um número indicando o nó-i que lhe está associado. Há duas entradas preenchidas com nomes especiais, "." e "..". A entrada "." indica o nó-i do próprio diretório. A entrada ".." indica o nó-i do diretório-pai.

No *sofs16*, o tamanho de um diretório deve ser igual ao tamanho em bytes que as suas entradas ocupam. Como as entradas têm tamanho fixo, isso significa que o número de entradas num diretório pode ser obtido dividindo o seu tamanho em bytes pelo tamanho de uma entrada.

Implica também que, quando uma entrada é libertada (em consequência do apagamento de um ficheiro), o diretório pode ter de ser rescrito de modo a eliminar o buraco resultante. Uma forma simples de o fazer corresponde a transferir a última entrada para essa posição apagada.

3 Formatação

A operação de formatação é responsável por transformar os blocos de um dispositivo de armazenamento num disco *sofs16* vazio. É a primeira operação a ser realizada antes de se poder usar o disco.

Um disco vazio (acabado de formatar) possui diretório raiz, que, por sua vez, possui duas entradas ocupadas, com os nomes `"."` e `".."`. Por convenção, no caso da raiz, a entrada `".."` aponta para a própria raiz.

A operação de formatação deve:

- Escolher o valor adequado para o número de nós-i do disco, tendo em consideração o valor pedido pelo utilizador e o número total de blocos do disco.
- Preencher a tabela de nós-i, assumindo que o nó-i número `0` está ocupado pela raiz e que os restantes estão livres. A lista de nós-i livres deve começar no nó-i número 1 e ir sequencialmente até ao último.
- Preenchar o diretório raiz, assumindo que os seus dados ocupam o cluster `0`.
- Preencher a lista de clusters livres, assumindo que após formatação:
  - as duas caches estão vazias;
  - os dados do diretório raiz estão armazenados no cluster `0`;
  - os clusters usados para implementar a lista são tomados sequencialmente a partir do `1`.
- Preencher todos os campos do superbloco, tendo em consideração os pontos anteriores.
- Preencher, caso essa ordem tenha sido dado, todos os clusters livres com zeros.