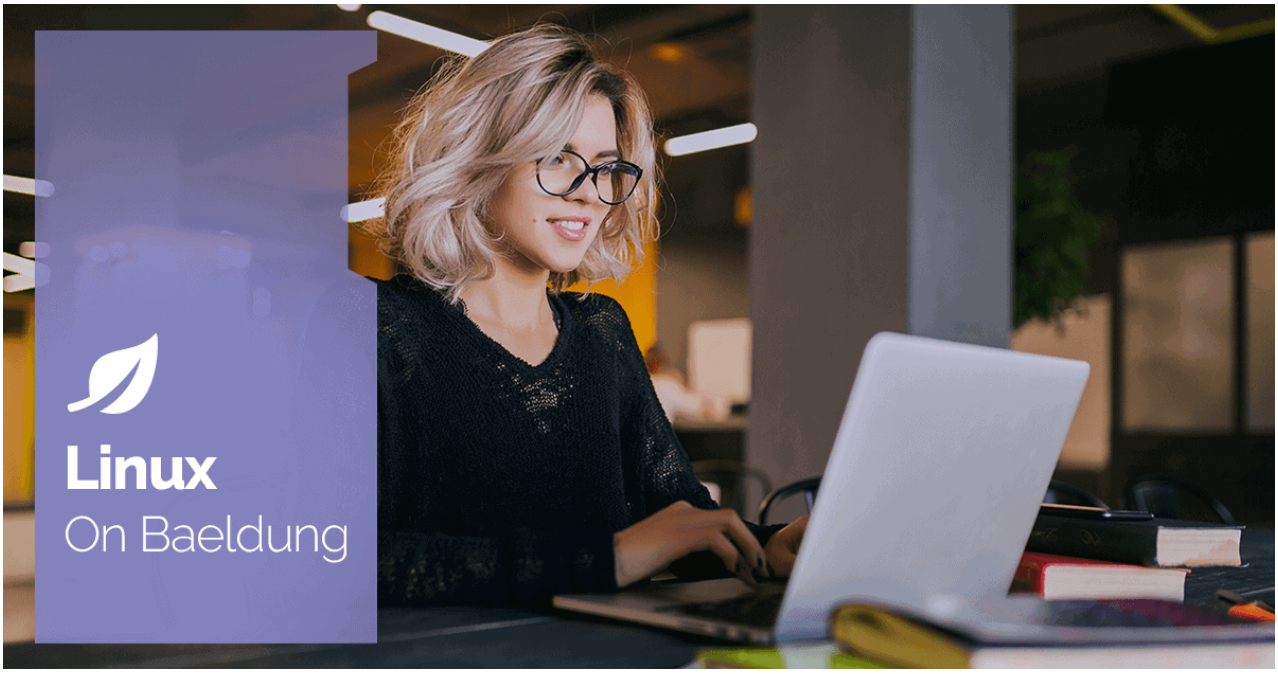
inodes Linux



1. Introdução

Muitas vezes pensamos em arquivos como objetos com conteúdo. No entanto, o que mais lidamos diariamente são apenas links. **Abaixo, os objetos dos arquivos são, na verdade, inodes** .

Neste tutorial, tratamos dos inodes do Linux – o que são, bem como por que e quando os usamos. Começamos com o conceito de armazenamento. Depois disso, discutiremos como os sistemas de arquivos se aplicam aos dispositivos de armazenamento e verificaremos seu funcionamento interno. A seguir, os inodes ganham destaque para uma discussão abrangente. Finalmente, exploramos alguns objetos do sistema de arquivos baseados em inodes e observamos algumas considerações especiais.

Testamos o código neste tutorial no Debian 11 (Bullseye) com GNU Bash 5.1.4. É compatível com POSIX e deve funcionar em qualquer ambiente desse tipo.

2. Armazenamento

A maioria das máquinas trabalha com pelo menos memória principal e secundária. Na maioria das vezes chamamos a memória principal de memória de acesso aleatório (RAM), enquanto a memória ou *armazenamento* secundário consiste em discos rígidos, unidades de estado sólido e dispositivos semelhantes.

Independentemente do tipo, os dispositivos de armazenamento geralmente possuem um controlador. É responsável por expor uma interface do respectivo módulo de memória à máquina. Os controladores apresentam o dispositivo como blocos do mesmo tamanho.

Devemos observar que um *setor* ou bloco mínimo de memória secundária geralmente tem 512 bytes . Isto será importante mais tarde.

Para aproveitar melhor os dispositivos de armazenamento, um sistema operacional (SO) possui drivers para se comunicar com o controlador. No entanto, um driver ainda apresenta uma estrutura de memória muito crua. Essa estrutura, por sua vez, é ainda mais refinada para uso por meio de outra abstração.

3. Sistema de arquivos

Embora os drivers sejam a forma como os sistemas operacionais veem e controlam o armazenamento, os sistemas de arquivos são a forma como eles o ordenam. Para fazer isso, **um sistema de arquivos normalmente usa um mecanismo antigo que podemos encontrar em livros – um índice** .

3.1. Organização de metadados

Considere como as seções de um livro são listadas com atributos importantes, como números de páginas e títulos. Todas essas informações estão dentro do índice. Por outro lado, o próprio índice está bem embalado em uma extremidade e ocupa apenas uma fração do livro, mas permite uma navegação rápida e fácil.

Mesmo sem todas as páginas em mãos, utilizando o índice, podemos ter uma noção de:

- quanto tempo dura o livro
- quais seções são curtas e quais são longas
- como as seções são ordenadas
- quais seções são subseções de quais outras seções
- características básicas das seções (como o título)

Agora imagine que algumas seções do livro estejam vazias. A estrutura resultante geralmente é a ideia mais básica de um sistema de arquivos.

Em particular, o índice representa os *metadados* associados a um sistema de arquivos. **As seções, livres ou alocadas em arquivos, também são chamadas de *blocos* e representam partes físicas da memória** . O índice do sistema de arquivos os organiza para uma navegação rápida e fácil, assim como as páginas de um livro.

3.2. Exemplos

Na verdade, a maioria dos sistemas de arquivos possui algum tipo de organização de suporte, que aponta para os blocos:

- *FAT* (File Allocation Table) possui uma Tabela de Alocação de Arquivos (daí o nome *FAT*)
- *exFAT* (Extended File Allocation Table) basicamente estende as tabelas de alocação de arquivos
- *NTFS* (New Technology File System) usa uma tabela de arquivos mestre muito mais avançada

O Linux suporta muitos sistemas de arquivos, mas apenas alguns deles são nativos . Um exemplo notável são os sistemas da família *ext* (Extended Filesystem): ext2, ext3 e ext4. Outro bom exemplo é o *XFS* (sistema de arquivos X). De agora em diante, discutiremos apenas como os sistemas de arquivos nativos do Linux funcionam devido ao seu uso generalizado e a um conceito que eles compartilham – a chave para seus metadados.

Vamos explorar esse conceito.

4. *inodos*

O termo *inode* vem de *index node* . Como já falamos sobre as organizações de sistemas de arquivos serem como índices nos livros, podemos ver como os inodes se encaixam perfeitamente nessa ideia.

Em particular, podemos igualar um inode ao número da linha de uma entrada de índice em um livro . **Dessa forma, os inodes nos permitem indexar o índice principal** . Os sistemas operacionais atribuem a cada inode um número inteiro exclusivo, para que possam ser usados como chaves para as estruturas internas do sistema de arquivos que já discutimos.

Para obter a maior parte das informações que um inode aponta para um determinado arquivo, podemos usar o comando *stat* , que executa internamente a *chamada do sistema* stat :

```
$ stat file.ext
File: file.ext
Size: 166 Blocks: 8 IO Block: 4096 regular file
Device: 810h/2064d Inode: 666 Links: 1
Access: (0777/-rwxrwxrwx) Uid: ( 1000/ x) Gid: ( 1000/ x)
Access: 2021-11-11 10:00:00.101020201 +0200
Modify: 2021-11-11 10:00:00.101020201 +0200
Change: 2021-11-11 10:00:00.101020201 +0200
Birth: 2021-11-11 10:00:00.101020201 +0200
```

Vamos dividir essas informações em seus constituintes, de cima para baixo, da esquerda para a direita . A saída começa com o nome e tamanho do arquivo. Este último está em bytes (166) e blocos (8) de um determinado tamanho (8*512 = 4096 bytes). Como já mencionamos, o bloco físico geralmente tem 512 bytes, mas podemos

especificar tamanhos de bloco IO ao formatar um dispositivo ou partição com um determinado sistema de arquivos. A ligação não é direta, mas existe. No final da segunda linha, vemos que usamos *stat* em um *arquivo normal* .

A seguir, temos a identificação do dispositivo em hexadecimal (810) e decimal (2064). Claro, vemos o número do inode (666). O último na quarta linha é o número de links para este inode. Discutiremos links na próxima seção.

Há uma linha inteira dedicada aos direitos de propriedade e acesso. Por fim, vemos quatro linhas dedicadas à data de acesso, modificação, alteração e criação (nascimento). Observe que alterar um arquivo significa modificar seus metadados.

Na verdade, muitos objetos do sistema de arquivos possuem seu próprio inode exclusivo. Na verdade, independentemente da sua natureza específica, todos são arquivos.

5. Objetos do sistema de arquivos

Até agora, falamos sobre metadados do sistema de arquivos. No entanto, todos os sistemas de arquivos existem para organizar os dados reais que armazenamos com eles. Para nós, os metadados estão quase todos ocultos. O que é visível são os objetos do sistema de arquivos.

O principal objeto na maioria dos sistemas de arquivos é o arquivo. **Em sistemas de arquivos Linux nativos, o termo *arquivo* significa qualquer objeto que tenha um inode associado** . Eles podem ser de muitos tipos diferentes:

- Regular
- Diretório
- Bloco especial
- Personagem especial
- Pipe nomeado (FIFO)
- Link simbólico
- Soquete

Vamos explorar os mais comuns.

5.1. Arquivo normal

Um arquivo normal é a coleção de dados que normalmente associamos à palavra *arquivo* . Além dos metadados de seu inode, **os arquivos regulares possuem conteúdos que não são do sistema, que lemos e escrevemos por meio de editores e outras ferramentas** .

É importante ressaltar que podemos ver como o conteúdo de um arquivo é espalhado via *debugfs* (Filesystem Debugger):

```
$ debugfs /dev/sda
debugfs 1.46.2 (20-Nov-2021)
debugfs: inode_dump -b /file.ext
0000 0af3 0100 0400 0000 0000 0000 0000 0000 ................
0020 0100 0000 0034 1100 0000 0000 0000 0000 .....4..........
0040 0000 0000 0000 0000 0000 0000 0000 0000 ................
*
```

Primeiro, iniciamos *o debugfs* com um dispositivo de bloco do sistema de arquivos como argumento. Depois disso, usamos o comando *inode_dump* com o sinalizador *-b* para mostrar em quais blocos, por seus identificadores, um arquivo consiste.

Geralmente armazenamos a maioria dos arquivos dentro de diretórios.

5.2. Diretório

Diretórios são contêineres em um sistema de arquivos – eles podem armazenar a maioria dos objetos, incluindo outros diretórios . Os diretórios do Linux são uma lista de nomes de arquivos para mapeamentos de números de inodes. Os usuários veem essa estrutura na forma de uma árvore.

Na verdade, podemos usar o comando *tree* para confirmar isso:

```
$ tree -L 2 -d /etc/systemd/
/etc/systemd/
|-- network
|-- system
| |-- default.target.wants
| |-- getty.target.wants
| |-- multi-user.target.wants
| |-- network-online.target.wants
| |-- remote-fs.target.wants
| |-- sockets.target.wants
| |-- sysinit.target.wants
| `-- timers.target.wants
`-- user
```

O sinalizador *-d* serve para listar apenas diretórios, enquanto o número após *-L* indica o nível de profundidade abaixo do caminho especificado (último argumento).

Em sistemas de arquivos Linux nativos, cada diretório possui as seguintes informações exclusivas associadas:

- . (referência do próprio inode)
- .. (referência de inode de diretório de nível superior)
- lista de objetos contidos

Observe que o último ponto é o que os blocos de um diretório contêm: uma tabela com nomes de arquivos e seus números de inode associados . Na verdade, este é o único lugar onde os sistemas de arquivos armazenam nomes de arquivos. Os inodes de arquivos regulares não armazenam seus nomes, apenas seus outros metadados:

```
$ debugfs /dev/sda
debugfs 1.46.2 (20-Nov-2021)
debugfs: stat /file.ext
Inode: 666 Type: regular Mode: 0644 Flags: 0x80000
Generation: 890607921 Version: 0x00000000:00000001
User: 1000 Group: 0 Project: 0 Size: 166
File ACL: 0
Links: 1 Blockcount: 8
Fragment: Address: 0 Number: 0 Size: 0
ctime: 0x61a50568:29f85c40 -- Mon Nov 29 18:52:56 2021
atime: 0x61a5055f:8491b840 -- Mon Nov 29 18:52:47 2021
mtime: 0x61a5055f:8491b840 -- Mon Nov 29 18:52:47 2021
crtime: 0x61a5055f:8491b840 -- Mon Nov 29 18:52:47 2021
Size of extra inode fields: 32
Inode checksum: 0x6af32853
EXTENTS:
(0):1064843
```

Como podemos confirmar acima, o nome do arquivo não faz parte dos dados do inode, embora o comando *stat* o adicione para maior clareza.

Podemos criar estruturas de árvores mais complexas por meio de links.

5.3. Link

Às vezes, podemos querer que um objeto de sistema de arquivos exista em mais de um lugar ao mesmo tempo, sem realmente duplicar os dados. Nestes casos, usamos links .

Na verdade, podemos pensar nos links como atalhos para outro objeto. Existem dois tipos de links: hard e soft (simbólicos).

Enquanto os links virtuais apontam diretamente para o objeto original por seu nome atual, os links físicos estão conectados ao seu inode e são apenas outro nome para o mesmo arquivo . Esta é uma exceção à nossa declaração anterior sobre onde existem nomes de arquivos – links simbólicos também os mantêm. Na prática, os objetos desaparecem quando todos os seus links físicos são removidos, pois os links simbólicos sempre apontam para um link físico. Curiosamente, os links simbólicos têm seus próprios inodes especiais.

Além disso, podemos não apenas criar links de arquivos, mas também vincular a um diretório. Fazer isso pode criar estruturas circulares complexas a partir de árvores de diretórios.

Além disso, o mecanismo inode pode resultar em algum comportamento específico, que discutiremos a seguir.

6. Especificações do sistema de arquivos *inode*

Ao usar um sistema de arquivos baseado em inodes, às vezes temos que considerar seu funcionamento interno. Vejamos alguns casos especiais.

6.1. Máximo *de inodes*

É possível que um sistema de arquivos fique sem inodes, apesar de haver espaço livre disponível . Isso geralmente acontece quando muitos arquivos pequenos não ocupam muito armazenamento, mas são deduzidos da contagem total de inodes.

Na prática, podemos ver informações de contagem total de inodes através do comando *df* (Disk Free):

```
$ df -ih
Filesystem Inodes IUsed IFree IUse% Mounted on
/dev/sda 16M 106K 16M 1% /
```

A saída acima afirma que usamos 106 mil dos 16 milhões de inodes possíveis. Claro, podemos liberar inodes para ter mais disponíveis.

6.2. Realocação

Quando excluímos um inode, o sistema de arquivos pode alocá-lo para outro arquivo no futuro. No entanto, **excluir um arquivo pelo nome não é o mesmo que excluir um inode** . O primeiro remove apenas um ponteiro (hard link ou soft link), enquanto o último destrói metadados sobre o arquivo na tabela de inodes. Pense assim:

- arquivos são apenas nomes vinculados a inodes
- inodes consistem em descrições de arquivos e seus ponteiros de bloco de conteúdo
- blocos retêm o conteúdo do arquivo

Conseqüentemente, podemos verificar a tabela de inodes para restaurar um arquivo excluído, mas não podemos fazer o mesmo com os metadados de um inode em circunstâncias normais. No entanto, mesmo a perda de ponteiros de bloco pode ser reversível por meio de varreduras e algum algoritmo de detecção antes que os dados sejam reescritos permanentemente.

6.3. Arquivos embutidos

Outra função útil às vezes disponível em sistemas de arquivos são os arquivos embutidos. **Eles estarão disponíveis se o tamanho dos metadados do inode obrigatório for menor que o tamanho do inode** :

```
$ debugfs /dev/sda
debugfs 1.46.2 (20-Nov-2021)
debugfs: stat /file.ext
Inode: 666 Type: regular Mode: 0644 Flags: 0x80000
Generation: 890607921 Version: 0x00000000:00000001
User: 1000 Group: 0 Project: 0 Size: 166
File ACL: 0
Links: 1 Blockcount: 8
Fragment: Address: 0 Number: 0 Size: 0
ctime: 0x61a50568:29f85c40 -- Mon Nov 29 18:52:56 2021
atime: 0x61a5055f:8491b840 -- Mon Nov 29 18:52:47 2021
mtime: 0x61a5055f:8491b840 -- Mon Nov 29 18:52:47 2021
crtime: 0x61a5055f:8491b840 -- Mon Nov 29 18:52:47 2021
Size of extra inode fields: 32
Inode checksum: 0x6af32853
EXTENTS:
(0):1064843
```

O espaço extra, 32 bytes no exemplo acima, pode conter dados e, para arquivos muito pequenos, isso significa que não precisamos de espaço adicional em disco para armazenar dados e metadados. O recurso é recente (maio de 2016) e pode precisar ser habilitado explicitamente por meio de *e2fsprogs* .

6.4. *estatísticas* e *debugfs estatísticas*

Embora a maioria das informações de *stat* em *debugfs* sejam as mesmas do comando *stat* , existem algumas diferenças .

Um deles é o número da geração. Ele pode distinguir números de inodes vistos por dois sistemas operacionais diferentes, por exemplo, cliente e servidor. Usamos isso em cenários específicos onde essas diferenças fazem sentido.

Outro grupo são os campos de fragmentos. Eles mostram quando os blocos usam o agora obsoleto recurso de fragmentação de bloco.

O último campo que vemos é EXTENTS. Está lá quando habilitamos um recurso que permite ao sistema de arquivos armazenar blocos contíguos apenas com os identificadores do primeiro e do último bloco na sequência.

7. Resumo

Neste artigo, discutimos inodes e seu papel principal nos sistemas de arquivos. Para tanto, analisamos vários níveis onde as informações em um inode são essenciais.

Concluindo, **os inodes são a ligação entre os arquivos que o usuário vê e o que o sistema realmente armazena sobre esses arquivos** .