Linux文件空洞与稀疏文件

http://www.ilinuxkernel.com

目 录

- Linux文件空洞与稀疏文件
- 文件系统数据存储
- 文件系统调试

Linux文件空洞与稀疏文件

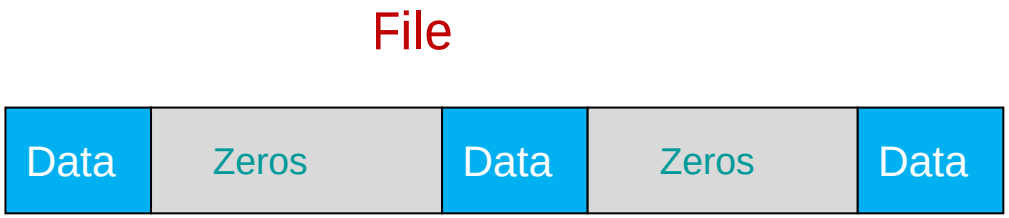
❑ 文件空洞

✓ 在UNIX文件操作中，**文件位移量可以大于文件的当前长度**

在这种情况下，对该文件的下一次写将延长该文件，并在文件中构成一个空洞。位于文件中但没有写过的字节都被设为 0。

✓ 如果 offset 比文件的当前长度更大，下一个写操作就会把文件“撑大（extend）”在文件里创造“空洞（hole）”。

没有被实际写入文件的所有字节由重复的 0 表示。空洞是否占用硬盘空间是由文件系统（file system）决定的。



Linux文件空洞与稀疏文件

❑ 稀疏文件（Sparse File）

稀疏文件与其他普通文件基本相同，区别在于文件中的部分数据是全0，且这部分数据不占用磁盘空间。

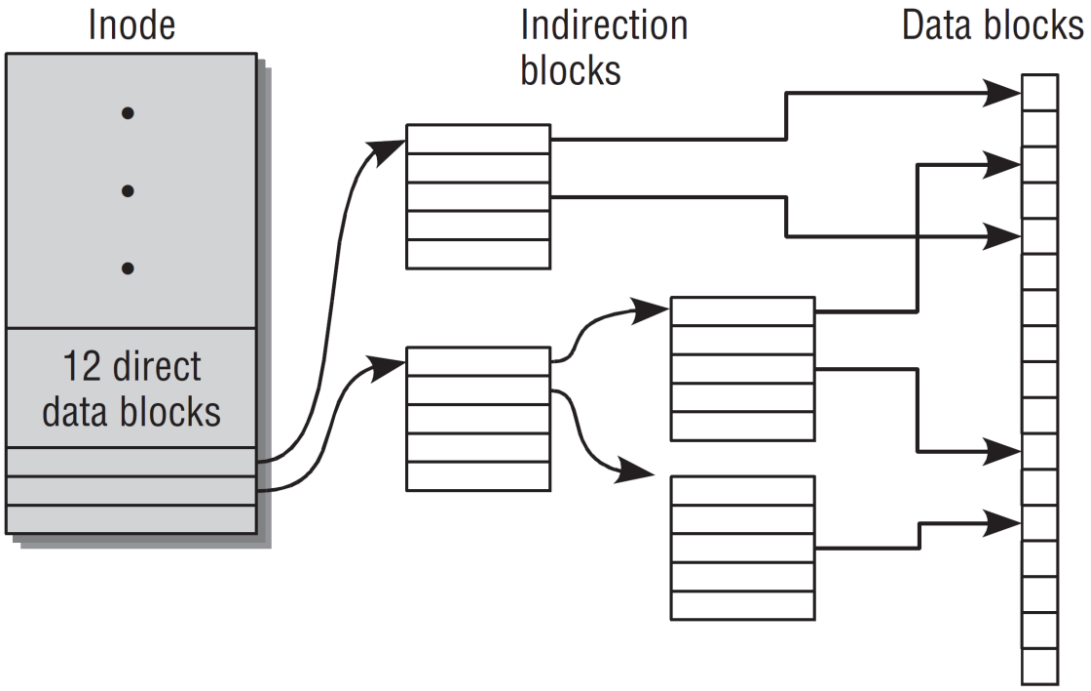
下面是稀疏文件的创建与查看

```
[root@localhost ~]# dd if=/dev/zero of=sparse-file bs=1 count=1 seek=1024k
[root@localhost ~]# ls -l sparse-file
-rw-r--r-- 1 root root 1048577 Oct 15 17:50 sparse-file
[root@localhost ~]# du -sh sparse-file
8.0K    sparse-file
[root@localhost ~]# cat anaconda-ks.cfg >> sparse-file
[root@localhost ~]# du -sh sparse-file
12Ksparse-file
[root@localhost ~]# du -sh anaconda-ks.cfg
12Kanaconda-ks.cfg
[root@localhost ~]#
```

Linux文件空洞与稀疏文件

❑ Linux文件系统inode数据块存储

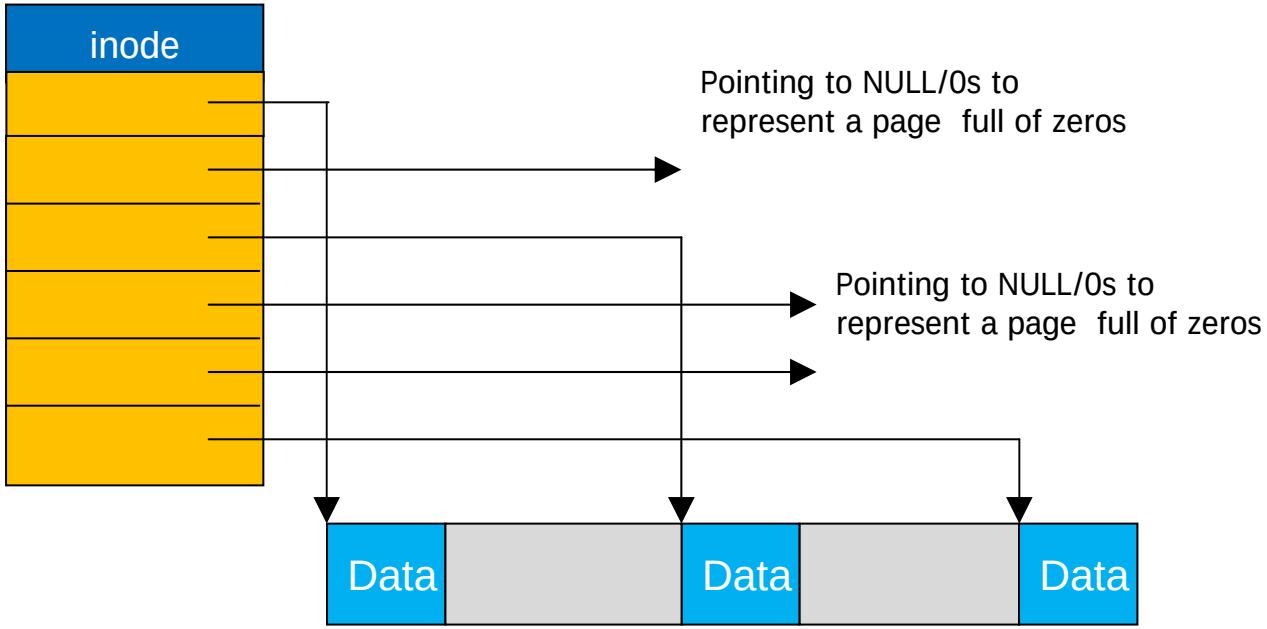
索引节点采用了多重索引结构，主要体现在直接指针和3个间接指针。直接指针包含12个直接指针块，它们直接指向包含文件数据的数据块，紧接在后面的3个间接指针是为了适应文件的大小变化而设计。



Linux文件空洞与稀疏文件

❑ Linux稀疏文件inode数据块存储

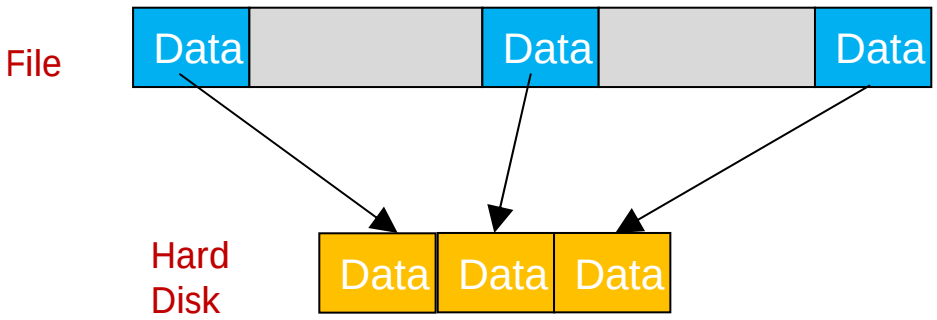
文件系统存储稀疏文件时，inode索引节点中，只给出实际占用磁盘空间的Block 号，数据全零且不占用磁盘空间的文件Block并没有物理磁盘Block号。



Linux文件空洞与稀疏文件

❑ Linux稀疏文件inode数据块存储

- ✓ 文件空洞部分不占用磁盘空间
- ✓ 文件所占用的磁盘空间仍然是连续的



实例：

[root@localhost mnt]# du -sh sparse-file
20K sparse-file
[root@localhost mnt]# ls -lh sparse-file
-rw-r--r-- 1 root root 1.1G Oct 15 10:36 sparse-file
[root@localhost mnt]#

debugfs: stat sparse-file
Inode: 49153 Type: regular Mode: 0644 Flags: 0x0 Generation: 3068382963
User: 0 Group: 0 Size: 1073742848
File ACL: 0 Directory ACL: 0
Links: 1 Blockcount: 40
Fragment: Address: 0 Number: 0 Size: 0
ctime: 0x507b76af -- Mon Oct 15 10:36:31 2012
atime: 0x507b765f -- Mon Oct 15 10:35:11 2012
mtime: 0x507b76af -- Mon Oct 15 10:36:31 2012
BLOCKS:
(IND):106496, (256):106497, (DIND):106504, (IND):106505, (262144):106506
TOTAL: 5

文件系统数据存储

❑ Linux文件系统数据块存储多重索引

✓Linux文件系统数据存放采用inode多重索引结构，有直接指针和3个间接指针。

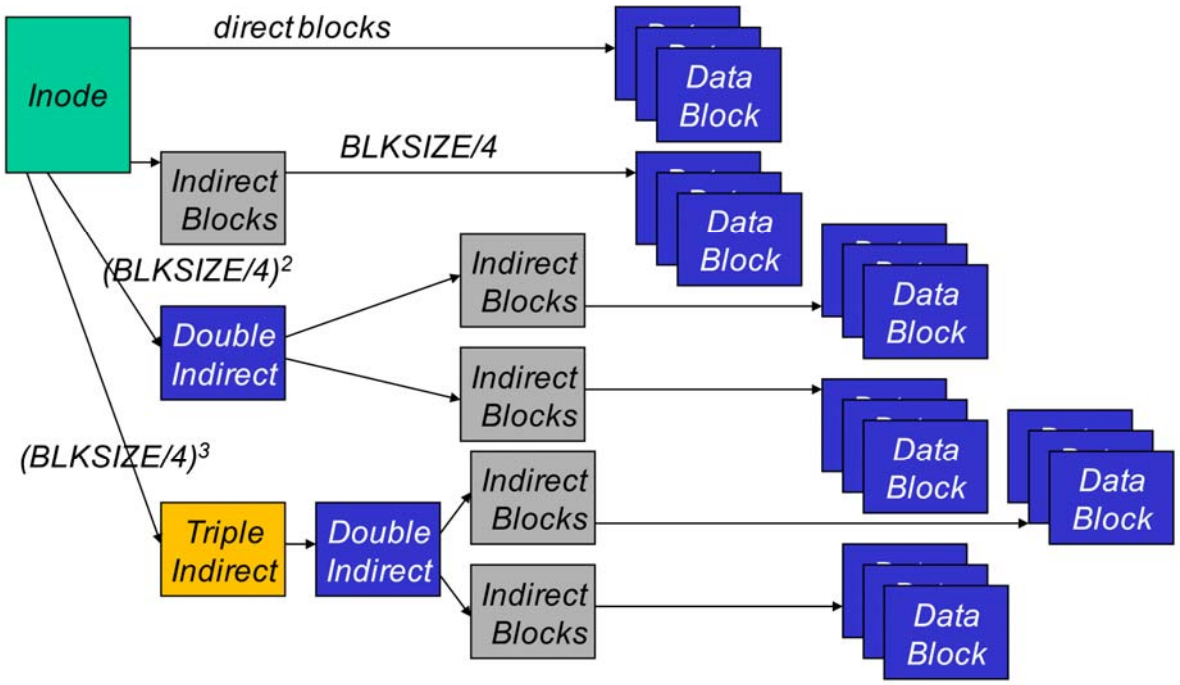
类似于编程中的变量定义：

```
unsigned long  blk;
unsigned long  *blk;
unsigned long  **blk;
unsigned long  ***blk;
```

✓直接指针直接指向保存数据的Block号。

✓一级指针指向一个Block，该Block中的数据是Block指针，指向真正保存数据的Block。

✓二级三级指针以此类推。



文件系统数据存储

（1）前12个直接指针，直接指向存储的数据区域

*如Blocks大小为4096，则前12个直接指针就可以保存**48KB**文件。*

（2）一级指针可存储文件大小计算

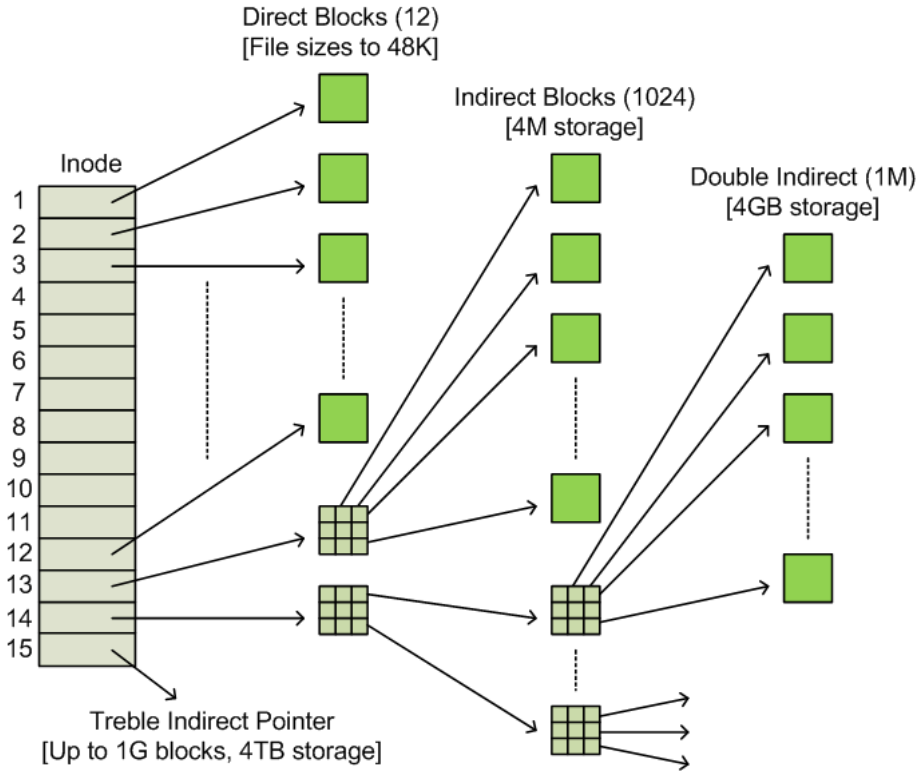
*假设每个指针占用4个字节，则一级指针指向的Block可保存4096/4个指针，可指向1024个Blocks。一级指针可存储文件数据大小为1024*4096 = **4MB**。*

(3)二级指针可存储文件大小计算

*同样按照Blocks大小为4096，则二级指针可保存的Block指针数量为(4096/4) * (4096/4) = 1024*1024。则二级指针可保存的文件数量大小为(1024*1024)*4096 = **4GB。***

(4)三级指针可存储文件大小计算

*以一级、二级指针计算方法类推，三级指针可存储的文件数据大小为(1024*1024*1024)*4096 = **4TB。***



文件系统调试

❑ Linux中如何读写硬盘（或Virtual Disk）上指定物理扇区

- 读指定物理扇区

  dd if=<源设备> of=<输出设备或文件> skip=<指定扇区值> bs=512 count=1

- 写指定物理扇区

  dd if=<输入设备或文件> of=<输出设备> seek=<指定扇区值> bs=512 count=1

❑ 如何读写文件系统上指定Block所存储的数据

步骤:

1、计算文件系统Block对应的物理扇区

*物理扇区号 = Block Num * Block Size / Hard Disk Sector Size*

2、使用dd命令读写指定物理扇区

读数据：dd if=<源设备> of=<输出设备或文件> skip=<物理扇区号> bs=512 count=Block Size/Sector Size

写数据： dd if=<输入设备或文件> of=<输出设备> seek=<物理扇区号> bs=512 count= Block Size/Sector Size

文件系统调试

❑ 文件占用物理磁盘空间查看

1、磁盘分区/dev/cciss/c0d1p1，文件系统为ext3，挂载在/data1目录下

```
[root@localhost data1]# df  -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/cciss/c0d0p1      95G  5.4G   85G   6% /
tmpfs                  32G     0   32G   0% /dev/shm
/dev/cciss/c0d2p1     1.1T  287G  758G  28% /data
/dev/cciss/c0d1p1     942M   67M  827M   8% /data1
[root@localhost data1]# ls -ls linux-2.6.18-194.el5.tar.bz2
50516 -rw-r--r-- 1 root root 51670183 Oct 18 19:23 linux-2.6.18-194.el5.tar.bz2
[root@localhost data1]#
```

2、通过debugfs来查看文件linux-2.6.18-194.el5.tar.bz2所占用的实际物理磁盘空间

```
[root@localhost ~]# debugfs /dev/cciss/c0d1p1
debugfs 1.39 (29-May-2006)
debugfs:  ls
 2  (12) .    2  (12) ..    11  (20) lost+found
 12  (4052) linux-2.6.18-194.el5.tar.bz2
debugfs:  stat linux-2.6.18-194.el5.tar.bz2
Inode: 12   Type: regular    Mode:  0644   Flags: 0x0   Generation: 4168353056
User:     0   Group:     0   Size: 51670183
File ACL: 0    Directory ACL: 0
Links: 1   Blockcount: 101032
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x507fe6cb -- Thu Oct 18 19:23:55 2012
atime: 0x507fe6cb -- Thu Oct 18 19:23:55 2012
mtime: 0x507fe6cb -- Thu Oct 18 19:23:55 2012
BLOCKS:
(0-11):24576-24587, (IND):24588, (12-1035):24589-25612, (DIND):25613,
(IND):25614, (1036-2059):25615-26638, (IND):26639, (2060-3083):26640-27663, (IND
):27664, (3084-4107):27665-28688, (IND):28689, (4108-5131):28690-29713,
(IND):29714, (5132-6155):29715-30738, (IND):30739, (6156-7179):30740-31763, (I
ND):31764, (7180-8182):31765-32767, (8183-8203):33312-33332,
(IND):33333, (8204-9227):33334-34357, (IND):34358, (9228-10251):34359-35382, (IND):35383,
 (10252-11275):35384-36407, (IND):36408, (11276-12299):36409-37432,
(IND):37433, (12300-12614):37434-37748
TOTAL: 12629
```

```
[root@localhost ~]# dumpe2fs /dev/cciss/c0d1p1 |grep "Block size"
dumpe2fs 1.39 (29-May-2006)
Block size:               4096
[root@localhost ~]
```

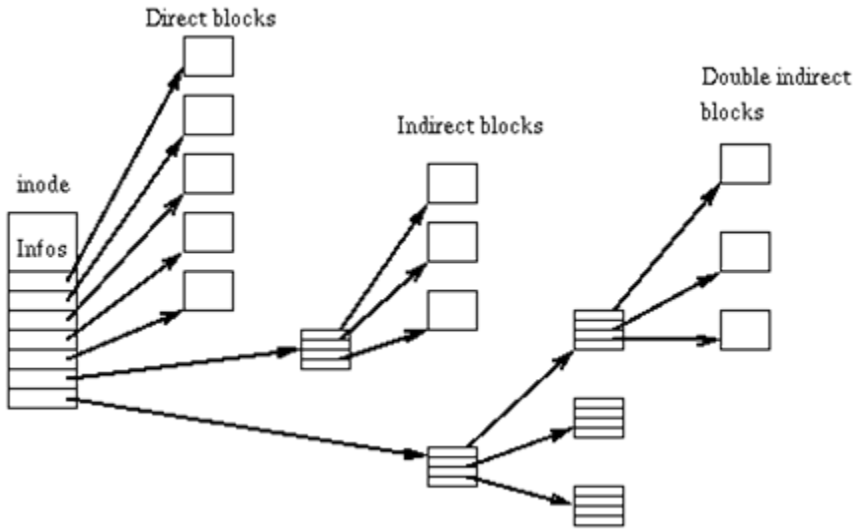
（1）linux-2.6.18-194.el5.tar.bz2文件大小为51670183，占用实际的物理磁盘空间为12629 个Blocks

（2）通过dumpe2fs工具可以查看当前文件系统块大小为4096字节。即该文件实际占用的物理磁盘空间为12629*4096 = 51728384 = 50516K

文件系统调试

Linux-2.6.18-194.el5.tar.bz2文件所占用的Blocks分布

(0-11):24576-24587, (IND):24588, (12-1035):24589-25612, (DIND):25613, (IND):25614, (1036-2059):25615-26638, (IND):26639, (2060-3083):26640-27663, (IND
):27664, (3084-4107):27665-28688, (IND):28689, (4108-5131):28690-29713, (IND):29714, (5132-6155):29715-30738, (IND):30739, (6156-7179):30740-31763, (I
ND):31764, (7180-8182):31765-32767, (8183-8203):33312-33332, (IND):33333, (8204-9227):33334-34357, (IND):34358, (9228-10251):34359-35382, (IND):35383,
(10252-11275):35384-36407, (IND):36408, (11276-12299):36409-37432, (IND):37433, (12300-12614):37434-37748
TOTAL: 12629



1、前12个直接指针，直接指向存储的数据区域，即（0-11）所占用的Blocks为24576-24587。

2、对于一个51MB的文件来说，需要使用一级指针和二级指针。（见前面分析，一级指针支持文件大小48KB+4MB，
使用二级指针支持文件大小48KB+4MB+4GB）

我们先分析一级指针：

一级指针指向的Block是24588，根据前面读取文件系统指定Block原始数据方法，得到该Block存储的数据为：

文件系统调试

Block 24588对应的硬盘起始物理扇区为 24588*4096/512 = 196704

```
[root@localhost ~]# dd if=/dev/cciss/c0d1p1 of=linux_kernel_code_inode.bin skip=196704 bs=512 count=8
8+0 records in
8+0 records out
4096 bytes (4.1 kB) copied, 0.011518 seconds, 356 kB/s
[root@localhost ~]# hexdump -C linux_kernel_code_inode.bin
00000000  0d 60 00 00 0e 60 00 00  0f 60 00 00 10 60 00 00  |.`...`...`...`..|
00000010  11 60 00 00 12 60 00 00  13 60 00 00 14 60 00 00  |.`...`...`...`..|
00000020  15 60 00 00 16 60 00 00  17 60 00 00 18 60 00 00  |.`...`...`...`..|
00000030  19 60 00 00 1a 60 00 00  1b 60 00 00 1c 60 00 00  |.`...`...`...`..|
00000040  1d 60 00 00 1e 60 00 00  1f 60 00 00 20 60 00 00  |.`...`...`.. `..|
00000050  21 60 00 00 22 60 00 00  23 60 00 00 24 60 00 00  |!`.."`..#`..$`..|
00000060  25 60 00 00 26 60 00 00  27 60 00 00 28 60 00 00  |%`..&`..'`..(`..|
  ... ...
00000fa0  f5 63 00 00 f6 63 00 00  f7 63 00 00 f8 63 00 00  |.c...c...c...c..|
00000fb0  f9 63 00 00 fa 63 00 00  fb 63 00 00 fc 63 00 00  |.c...c...c...c..|
00000fc0  fd 63 00 00 fe 63 00 00  ff 63 00 00 00 64 00 00  |.c...c...c...d..|
00000fd0  01 64 00 00 02 64 00 00  03 64 00 00 04 64 00 00  |.d...d...d...d..|
00000fe0  05 64 00 00 06 64 00 00  07 64 00 00 08 64 00 00  |.d...d...d...d..|
00000ff0  09 64 00 00 0a 64 00 00  0b 64 00 00 0c 64 00 00  |.d...d...d...d..|
00001000
```

一级指针指向的Block范围为0x600d~0x640c,对应的十进制为24569～25612，我们debugfs中看到的值确实如此。（注意：该Block号码段前面的12～1035，是值文件内的Block号。）

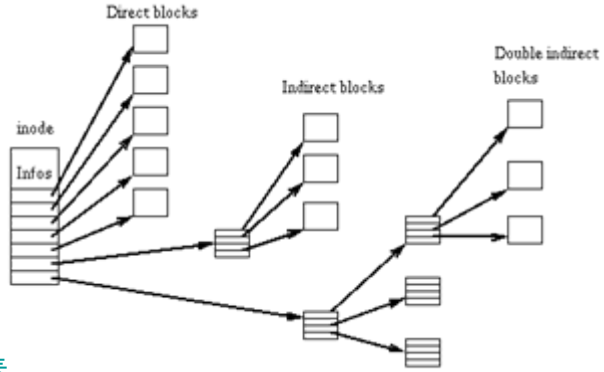
*(0-11):24576-24587, (IND):24588, (12-1035):**24589-25612**, (DIND):25613, (IND):25614, (1036-2059):25615-26638, (IND):26639, (2060-3083):26640-27663,*

文件系统调试

前面分析了一级指针，我们继续分析Linux-2.6.18-194.el5.tar.bz2文件二级指针所占用的Blocks分布

*(0-11):24576-24587, (IND):24588, (12-1035):24589-25612, (DIND):**25613**, (IND):25614, (1036-2059):25615-26638, (IND):26639, (2060-3083):26640-27663, (IND):27664, (3084-4107):27665-28688, (IND):28689, (4108-5131):28690-29713, (IND):29714, (5132-6155):29715-30738, (IND):30739, (6156-7179):30740-31763, (IND):31764, (7180-8182):31765-32767, (8183-8203):33312-33332, (IND):33333, (8204-9227):33334-34357, (IND):34358, (9228-10251):34359-35382, (IND):35383, (10252-11275):35384-36407, (IND):36408, (11276-12299):36409-37432, (IND):37433, (12300-12614):37434-37748*
TOTAL: 12629

二级指针指向的第一级指针Blocks号为25613
这里强调一下，block 25613中保存的数据仍然是指针，指针指向Block中，保存的是直接指针。



在分析二级指针前，我们先计算该文件所占用二级指针中第一级指针多少个元素。
（51670183 – 4194304 – 49152）/ 4194304 = 11.3，即要使用12个元素。

我们再来看二级指针指向的block 25613中的数据。
二级指针指向的一级指针所在Block分别为0x640e、0x680f、0x6c10、0x7011、0x7412、0x7813、0x7c14、0x8235、0x8636、0x8a37、0x8e38、0x9239，对应的十进制值分别为：***25614、26639、27664、28689、29714、30739、31764、33333、34358、35383、36408、37433***

```
[root@localhost ~]# dd if=/dev/cciss/c0d1p1 of=linux_kernel_code_inode_2.bin skip=204904 bs=512 count=8
[root@localhost ~]# hexdump -C linux_kernel_code_inode_2.bin
00000000  0e 64 00 00 0f 68 00 00  10 6c 00 00 11 70 00 00  |.d...h...l...p..|
00000010  12 74 00 00 13 78 00 00  14 7c 00 00 35 82 00 00  |.t...x...|..5...|
00000020  36 86 00 00 37 8a 00 00  38 8e 00 00 39 92 00 00  |6...7...8...9...|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001000
```

文件系统调试

Linux-2.6.18-194.el5.tar.bz2文件所占用的Blocks分布

(0-11):24576-24587, (IND):24588, (12-1035):24589-25612, (DIND):25613, (IND):25614, (1036-2059):25615-26638, (IND):26639, (2060-3083):26640-27663, (IND
):27664, (3084-4107):27665-28688, (IND):28689, (4108-5131):28690-29713, (IND):29714, (5132-6155):29715-30738, (IND):30739, (6156-7179):30740-31763, (I
ND):31764, (7180-8182):31765-32767, (8183-8203):33312-33332, (IND):33333, (8204-9227):33334-34357, (IND):34358, (9228-10251):34359-35382, (IND):35383,
(10252-11275):35384-36407, (IND):36408, (11276-12299):36409-37432, (IND):37433, (12300-12614):37434-37748
TOTAL: 12629

前面分析了Linux-2.6.18-194.el5.tar.bz2文件所使用直接指针、一级指针和二级指针。

现在根据上面打印信息，就可以确定**该文件占用的Blocks为**

文件Blocks号	对应的文件系统Block号
（0-11）	24576～24587
(12-1035)	24589～25612
（1036-2059）	25615～26638
（2060-3083）	26640～27663
（3084-4107）	27665～28688
（4108-5131）	28690～29713
（5132-6155）	29715～30738
（6156-7179）	30740～31763
（7180-8182）	31765～32767
（8183-8203）	33312～33332
(8204-9227)	33334～34357

文件Blocks号	对应的文件系统Block号
（9228-10251）	34359～35382
（10252-11275）	35384～36407
（11276-12299）	36409～37432
(12300-12614)	37434～37748

Thank you