日曜研究室

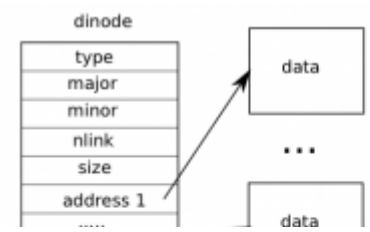
技術的な観点から日常を綴ります

[xv6 #63] Chapter 5 – File system – Code: Inode contents

テキストの71〜73ページ

本文

ディスク上のinodeの構造を表すdinode構造体は、サイズとブロック番号の配列を含んでいる（図5-4参照）。
inodeのデータは、dinode構造体のaddrs配列に記録されているブロック群から見つける事ができる。
データの最初のNDIRECT個のブロックは、そのaddrs配列の最初のNDIRECT個のエントリに記録されている。
そのようなブロック群はダイレクトブロック（direct blocks）と呼ばれる。
データの次のNINDIRECT個のブロックは、そのinodeには直接記録されておらず、データブロックにある。
そのようなデータブロックはインダイレクトブロック（indirect block）と呼ばれる。
addrs配列の最後のエントリは、インダイレクトブロックのアドレス用である。
上で述べたように、ファイルの最初の6kB（NDIRECT * BSIZE）分は、inodeにリストアップされているブロック群から読み込むことが出来るが、次の64kB（NINDIRECT * BSIZE）分は、インダイレクトブロックを読み込んだ後にしか読み込むことは出来ない。
これはディスク上での良い表現法だが、この構造を利用するコードにとっては複雑である。
bmap関数は、readi関数やwritei関数（すぐ後で見ることになるだろう）などのより高い層のルーチンのために、この構造を管理する。
bmap関数は、inodeであるipを受け取り、そのinodeのbn番目のデータブロックのブロック番号を返す。
もしipがそのようなブロックをまだ持っていない場合は、bmap関数がそれを割り当てる。



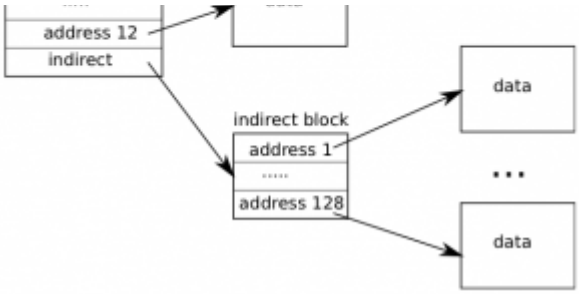


図5-4 ディスク上におけるファイルの表現

fs.cのbmap関数

```
01 // Inode contents
02 //
03 // The contents (data) associated with each inode is stored
04 // in a sequence of blocks on the disk.  The first NDIRECT blocks
05 // are listed in ip->addrs[].  The next NINDIRECT blocks are
06 // listed in the block ip->addrs[NDIRECT].
07
08 // Return the disk block address of the nth block in inode ip.
09 // If there is no such block, bmap allocates one.
10 static uint
11 bmap(struct inode *ip, uint bn)
12 {
13   uint addr, *a;
14   struct buf *bp;
15
16   if(bn < NDIRECT){
17     if((addr = ip->addrs[bn]) == 0)
18       ip->addrs[bn] = addr = balloc(ip->dev);
19     return addr;
20   }
21   bn -= NDIRECT;
22
23   if(bn < NINDIRECT){
24     // Load indirect block, allocating if necessary.
25     if((addr = ip->addrs[NDIRECT]) == 0)
26       ip->addrs[NDIRECT] = addr = balloc(ip->dev);
27     bp = bread(ip->dev, addr);
28     a = (uint*)bp->data;
29     if((addr = a[bn]) == 0){
30       a[bn] = addr = balloc(ip->dev);
31       log_write(bp);
32     }
33     brelse(bp);
34     return addr;
35   }
36
37   panic("bmap: out of range");
38 }
```

bmap関数は、簡単な仕事から始める。
最初のNDIRECT個のブロックは、inodeそれ自身にリストアップされている。
次のNINDIRECT個のブロックは、ip->addrs[NDIRECT]で示されるインダイレクトブロックにリストアップされている。

bmap関数は、そのインダイレクトブロックを読み込み、そしてそのインダイレクトブロックの中の正しい位置からブロック番号を読み出す。
もしそのブロック番号が、NDIRECT + NINDIRECTを越えている場合、bmap関数はpanic関数を呼ぶ。
呼び出し側は、範囲外のブロック番号について問い合わせないようにする責任がある。

bmap関数は、必要に応じてブロックを割り当てる。
未割り当てのブロックは、ブロック番号0として示される。
よってbmap関数は0に遭遇したら、必要に応じて割り当てられた新しいブロックの番号でそれを置き換える。

bmap関数は、そのinodeが成長するときに必要に応じてブロックを割り当てる。
itrunc関数は、それらを解放し、inodeのサイズを0にリセットする。
itrunc関数は、ダイレクトブロックを解放することから初め、そしてそれからインダイレクトブロックの内容を解放し、そして最後にインダイレクトブロックそのものを解放する。

fs.cのitrunc関数

```
01 // Truncate inode (discard contents).
02 // Only called after the last dirent referring
03 // to this inode has been erased on disk.
04 static void
05 itrunc(struct inode *ip)
06 {
07   int i, j;
08   struct buf *bp;
09   uint *a;
10
11   for(i = 0; i < NDIRECT; i++){
12     if(ip->addrs[i]){
13       bfree(ip->dev, ip->addrs[i]);
14       ip->addrs[i] = 0;
15     }
16   }
17
18   if(ip->addrs[NDIRECT]){
19     bp = bread(ip->dev, ip->addrs[NDIRECT]);
20     a = (uint*)bp->data;
21     for(j = 0; j < NINDIRECT; j++){
22       if(a[j])
23         bfree(ip->dev, a[j]);
24     }
25     brelse(bp);
26     bfree(ip->dev, ip->addrs[NDIRECT]);
27     ip->addrs[NDIRECT] = 0;
28   }
29
30   ip->size = 0;
31   iupdate(ip);
32 }
```

bmap関数は、readi関数やwritei関数のような、inodeのデータストリームへアクセスする関数を書きやすくする。

readi関数は、inodeからデータを読み込む。
readi関数は、指定されたオフセットとカウントが、ファイルの終わりを超えないかどうかチェックする事から始める。
ファイルの終わりを超えたところから始まるような読み込みにはエラーを返す。
ファイルの終わりまたはファイルの終わりをまたぐような指示があった場合は、指定されたよりも少ないデータを返す。
メインのループは、ファイルの各ブロックを処理し、バッファからデータをdstに格納する。
writei関数は、3つの例外を除いてreadi関数ととてもよく似ている。
ファイルの終わりから、もしくはファイルの終をまたぐような書き込みは、ファイルの最大サイズを上限としてファイルを大きくする。
メインのループでは、データをバッファから取り出すのではなく、バッファに格納する。
そして、書き込みがファイルを拡張した場合、writei関数はそのサイズを更新しなければならない。

fs.cのreadi, writei関数

```
01 // Read data from inode.
02 int
03 readi(struct inode *ip, char *dst, uint off, uint n)
04 {
05   uint tot, m;
06   struct buf *bp;
07
08   if(ip->type == T_DEV){
09     if(ip->major < 0 || ip->major >= NDEV || !devsw[ip->major].read)
10       return -1;
11     return devsw[ip->major].read(ip, dst, n);
12   }
13
14   if(off > ip->size || off + n < off)
15     return -1;
16   if(off + n > ip->size)
17     n = ip->size - off;
18
19   for(tot=0; tot<n; tot+=m, off+=m, dst+=m){
20     bp = bread(ip->dev, bmap(ip, off/BSIZE));
21     m = min(n - tot, BSIZE - off%BSIZE);
22     memmove(dst, bp->data + off%BSIZE, m);
23     brelse(bp);
24   }
25   return n;
26 }
27
28 // PAGEBREAK!
29 // Write data to inode.
30 int
31 writei(struct inode *ip, char *src, uint off, uint n)
32 {
33   uint tot, m;
34   struct buf *bp;
35
36   if(ip->type == T_DEV){
37     if(ip->major < 0 || ip->major >= NDEV ||
!devsw[ip->major].write)
38       return -1;
39     return devsw[ip->major].write(ip, src, n);
40   }
```

```
41
42    if(off > ip->size || off + n < off)
43      return -1;
44    if(off + n > MAXFILE*BSIZE)
45      return -1;
46
47    for(tot=0; tot<n; tot+=m, off+=m, src+=m){
48      bp = bread(ip->dev, bmap(ip, off/BSIZE));
49      m = min(n - tot, BSIZE - off%BSIZE);
50      memmove(bp->data + off%BSIZE, src, m);
51      log_write(bp);
52      brelse(bp);
53    }
54
55    if(n > 0 && off > ip->size){
56      ip->size = off;
57      iupdate(ip);
58    }
59    return n;
60  }
```

readi関数とwritei関数の両方共、ip->type == T_DEVのチェックから始める。
これは、ファイルシステム上にデータがない、特殊なデバイスを制御するためにある。
その場合については、ファイルディスクリプタの層の節で説明するだろう。

stati関数は、inodeのメタデータをstat構造体にコピーする。
この構造体は、statシステムコール経由でユーザプログラムにそのまま提供される。

fs.cのstati関数

```
01  // Copy stat information from inode.
02  void
03  stati(struct inode *ip, struct stat *st)
04  {
05    st->dev = ip->dev;
06    st->ino = ip->inum;
07    st->type = ip->type;
08    st->nlink = ip->nlink;
09    st->size = ip->size;
10  }
```

感想

ファイルがディスク上でどのように表現されているかと、その周辺の（といってもどれも必要不可欠）関数についての説明です。

bmapの実装からして1ファイル70kB（NDIRECT * BSIZE + NINDIRECT * BSIZE）までですしょうか。
writeiを見ても、MAXFILE * BSIZEまでとなっていて、MAXFILEはNDIRECT + NINDIRECTと定義されてるので多分そうでしょう。
これを拡張するには、インダイレクトブロックでもダイレクトブロックのように、addrs配列の最後に次のインダイレクトブロックのブロック番号を含めるようにすればいいのかな。

カテゴリー: 技術 | タグ: xv6 | 投稿日: 2012/4/12 木曜日 [http://peta.okechan.net/blog/archives/1651] |