日曜研究室

技術的な観点から日常を綴ります

[xv6 #64] Chapter 5 – File system – Code: directory layer

テキストの71〜73ページ

本文

ディレクトリの層はシンプルである。
なぜならディレクトリは、今まで以上に特殊な種類のファイル、ではないからである。
ディレクトリのinodeはT_DIRというタイプを持ち、そのデータはディレクトリエントリの連なりである。
各エントリは、dirent構造体であり、名前とinode番号を含む。
名前は、最大でDIRSIZ(14)文字までであり、もしそれに満たなかったら、NUL(0)で終端される。
inode番号ゼロのディレクトリエントリは、空きを示す。

fs.h

```
01 // On-disk file system format.
02 // Both the kernel and user programs use this header file.
03
04 // Block 0 is unused.  Block 1 is super block.
05 // Inodes start at block 2.
06
07 #define ROOTINO 1  // root i-number
08 #define BSIZE 512  // block size
09
10 // File system super block
11 struct superblock {
12   uint size;         // Size of file system image (blocks)
13   uint nblocks;      // Number of data blocks
14   uint ninodes;      // Number of inodes.
15   uint nlog;         // Number of log blocks
16 };
17
18 #define NDIRECT 12
19 #define NINDIRECT (BSIZE / sizeof(uint))
20 #define MAXFILE (NDIRECT + NINDIRECT)
21
22 // On-disk inode structure
23 struct dinode {
```

```
  short type;           // File type
  short major;          // Major device number (T_DEV only)
  short minor;          // Minor device number (T_DEV only)
  short nlink;          // Number of links to inode in file system
  uint size;            // Size of file (bytes)
  uint addrs[NDIRECT+1];   // Data block addresses
};

// Inodes per block.
#define IPB           (BSIZE / sizeof(struct dinode))

// Block containing inode i
#define IBLOCK(i)     ((i) / IPB + 2)

// Bitmap bits per block
#define BPB           (BSIZE*8)

// Block containing bit for block b
#define BBLOCK(b, ninodes) (b/BPB + (ninodes)/IPB + 3)

// Directory is a file containing a sequence of dirent structures.
#define DIRSIZ 14

struct dirent {
  ushort inum;
  char name[DIRSIZ];
};
```

dirlookup関数は、引数で渡された名前に合致するディレクトリエントリをディレクトリの中から探す。
ディレクトリエントリが見つかった場合、それに対応するinodeのポインタ（ロックされていない）を返し、そして、呼び出し側がその見つかったディレクトリエントリを編集したい場合に備えて、*poffにそのディレクトリエントリのオフセットを格納する。
dirlookup関数が名前に対応するディレクトリエントリを見つけた場合、*poffを更新し、そのブロックを解放し、そしてiget関数から取得したロックされていないinodeを返す。（ここ、一つ前の文と内容が被ってる上に、ブロックを解放とか、ソースと一致してない部分があるため、もしかしたら昔のソースに対する説明の消し忘れかもしれない。）
iget関数は、ロックされていないinodeを返すということがその理由である。
呼び出し側がロック済みのdpを持っていて、カレントディレクトリの別名である「.」に対してdirlookup関数を使った場合、呼び出し元に戻るまえにそのinodeをロックしようとして、つまりdpを再度ロックしようとしてデッドロックを引き起こす可能性がある。（複数のプロセスや親ディレクトリの別名「..」がからんでデッドロックを引き起こすようなもっと複雑な場合もたくさんある。問題が起きるのは「.」のときだけではないのである。）
呼び出し側は、dpをアンロックしそしてそれからipをロックすることで、一度に一つのロックだけを保持する事を保証出来る。
（と、説明されてますが、dirlookupのコードパスを追ってみたところ、inode単位でロックを取得してるところ（ilock関数を呼んでるところ）は見当たりませんでした。何なんでしょうコレ。）

fs.cのnamecmp, dirlookup, dirlink関数

```
// Directories

```

```
03 int
04 namecmp(const char *s, const char *t)
05 {
06   return strncmp(s, t, DIRSIZ);
07 }
08
09 // Look for a directory entry in a directory.
10 // If found, set *poff to byte offset of entry.
11 // Caller must have already locked dp.
12 struct inode*
13 dirlookup(struct inode *dp, char *name, uint *poff)
14 {
15   uint off, inum;
16   struct dirent de;
17
18   if(dp->type != T_DIR)
19     panic("dirlookup not DIR");
20
21   for(off = 0; off < dp->size; off += sizeof(de)){
22     if(readi(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
23       panic("dirlink read");
24     if(de.inum == 0)
25       continue;
26     if(namecmp(name, de.name) == 0){
27       // entry matches path element
28       if(poff)
29         *poff = off;
30       inum = de.inum;
31       return iget(dp->dev, inum);
32     }
33   }
34
35   return 0;
36 }
37
38 // Write a new directory entry (name, inum) into the directory dp.
39 int
40 dirlink(struct inode *dp, char *name, uint inum)
41 {
42   int off;
43   struct dirent de;
44   struct inode *ip;
45
46   // Check that name is not present.
47   if((ip = dirlookup(dp, name, 0)) != 0){
48     iput(ip);
49     return -1;
50   }
51
52   // Look for an empty dirent.
53   for(off = 0; off < dp->size; off += sizeof(de)){
54     if(readi(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
55       panic("dirlink read");
56     if(de.inum == 0)
57       break;
58   }
59
60   strncpy(de.name, name, DIRSIZ);
61   de.inum = inum;
62   if(writei(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
63     panic("dirlink");
64
65   return 0;
66 }
```

dirlink関数は、引数で与えられた名前とinode番号を元に、新しいディレクトリエントリをディレクトリdpに書き込む。
もし名前が既に存在した場合、dirlink関数はエラーを返す。
メインのループでは、未割り当てのエントリを探すためにディレクトリエントリ群を走査する。
未割り当てのエントリが見つかれば、ループを終了する。
その時のoffにはそのエントリのオフセットが格納されている。
未割り当てのエントリが見つからない場合、offにdp->sizeが格納された状態でループを終える。
どちらにしろ、dirlink関数はそれからオフセットoffに書き込むことで新しいエントリをディレクトリに追加する。

感想

ディレクトリの実装についてです。
inodeはすでに自身の種類を表すビットを持つことが出来るようになってるので、inodeの層から見るとすでに「ディレクトリレディ」な状態と言えます。

そして、通常のファイルの場合はデータ部分に単にその内容を持つわけですが、ディレクトリの場合はファイル名とそれに対応するファイルのinodeをデータとして持つわけです。

本文にも括弧付きで書きましたが、ちょっとソースと合致してない部分が見受けられます。
「ブロックの解放」については、昔のソースをみたら確かにブロックを確保して、最後にbrelseを呼ぶようになってる時期があったのは確認しました。
デッドロックの方については、昔のソースをざっと見ただけだとまだ該当するようなところは見つけてないので、もしかしたら、あの説明が合ってて、私の認識が間違ってる可能性もおおいにあります。

カテゴリー: 技術 | タグ: xv6 | 投稿日: 2012/4/14 土曜日 [http://peta.okechan.net/blog/archives/1659] |
