日曜研究室

技術的な観点から日常を綴ります

[xv6 #69] Chapter 5 – File system – Exercises

テキストの77ページ

本文

1. どういう場合にballocでpanicが起きるか？ それを回復することはできるか？

2. どういう場合にiallocでpanicが起きるか？ それを回復することはできるか？

3. inode世代番号。

4. fileがない場合になぜfileallocはpanicを起こさないのか？ なぜこれがより一般的でそれゆえ良いハンドリングの方法なのか？

5. sys_linkがiunlock(ip)とdirlinkを呼び出す間に、そのipが、他のプロセスによってアンリンクされたipと一致すると仮定する。
リンクは正しく作成されるだろうか？
なぜ正しく作成されるのか？もしくはなぜ正しく作成されないのか？

6. createは正常に完了するために4つの関数呼び出し（一つはialloc、他の三つはdirlink）を必要とする。
そうでない場合、createはpanicを呼ぶ。
なぜこれは許容できるのか？
なぜ、それらの4つの関数呼び出しは失敗し得ないのだろうか？

7. sys_chdirはiput(cp->cwd)の前にiunlock(ip)を呼び、iput(cp->cwd)はcp->cwdをロックしようとする。
iputの後のiunlock(ip)はデッドロックを引き起こさないが、それは何故か？

作業

1. について

見ての通り、ballocは空いてるブロックを探して返すので、空きブロックがなければpanicまで到達します。
panicじゃなくて呼び出し元に例外を通知するような仕組みにするとpanicは回避出来るかもしれませんが、どちらにしろ無いものを魔法の力で捻出することは出来ないので回復は無理だと思います。

fs.cのballoc関数

```
01 // Allocate a zeroed disk block.
02 static uint
03 balloc(uint dev)
04 {
05   int b, bi, m;
06   struct buf *bp;
07   struct superblock sb;
08
09   bp = 0;
10   readsb(dev, &sb);
11   for(b = 0; b < sb.size; b += BPB){
12     bp = bread(dev, BBLOCK(b, sb.ninodes));
13     for(bi = 0; bi < BPB && bi < (sb.size - b); bi++){
14       m = 1 << (bi % 8);
15       if((bp->data[bi/8] & m) == 0){  // Is block free?
16         bp->data[bi/8] |= m;  // Mark block in use on disk.
17         log_write(bp);
18         brelse(bp);
19         bzero(dev, b + bi);
20         return b + bi;
21       }
22     }
23     brelse(bp);
24   }
25   panic("balloc: out of blocks");
26 }
```

2. について

これも1と同じだと思います。

fs.cのialloc関数

```
01 // Allocate a new inode with the given type on device dev.
02 struct inode*
03 ialloc(uint dev, short type)
04 {
05   int inum;
06   struct buf *bp;
07   struct dinode *dip;
08   struct superblock sb;
09
10   readsb(dev, &sb);
11   for(inum = 1; inum < sb.ninodes; inum++){  // loop over inode
```

```
    blocks
12      bp = bread(dev, IBLOCK(inum));
13      dip = (struct dinode*)bp->data + inum%IPB;
14      if(dip->type == 0){  // a free inode
15        memset(dip, 0, sizeof(*dip));
16        dip->type = type;
17        log_write(bp);   // mark it allocated on the disk
18        brelse(bp);
19        return iget(dev, inum);
20      }
21      brelse(bp);
22    }
23    panic("ialloc: no inodes");
24  }
```

3. について

元の文もinode generation numbers.だけなのでちょっと課題の意図が分かりませんが、inode世代番号を実装するにはどうしたら良いか？という事だと解釈して考えてみます。
まずinode世代番号というのは、あるinodeが指し示すデータが変更されたタイミングでインクリメントされる値です。（多分…）
これはinodeの変更状況を追跡するのに使えて、その値を元に割り当てのアルゴリズムなんかも改良出来たりするんじゃないかななんて思います。（想像）
inode世代番号なんて言葉正直今日知りました。
で、inode世代番号の実装方法ですが、まずinode構造体とdinode構造体に世代番号のフィールドを追加する必要があるでしょう。
dinodeのほうには追加せずに、例えばスーパーブロックとかにまとめて記録するのもありかもしれません。
そしたら、fs.cのwriteiの関数でその世代番号をインクリメントしてあげればいいと思いますが、ジャーナリングのために実際はwriteiではなくそこから呼ばれるlog_writeで書き込みが行われるので、そちらで実装する必要があると思います。
実際の実装レベルであまり細かいところまで考えるとロックとかでドツボにハマリそうなのでこの程度で勘弁していただきたく…

4. について

ballocやiallocは、ブロックやinode単位の割り当てであり、そのレベルで失敗してもユーザプログラムは正直何も出来ないはず（何か出来るということはカーネルがやってることより難しいことをしなきゃならないわけで…）なので、panicせずユーザプログラムに事後処理の責任を負わせるのは現実的じゃないと思います。
一方、filealloc が失敗するのは、システム全体で同時に開けるファイルの最大数（xv6の場合はたったの100）に達したときであって、前者の失敗の深刻度と比べたら、かなりヌルい失敗と言えます。
ユーザプログラムも、filealloc が失敗したら、必要なファイルが開けなかったとして単に終了するとか、開けるようになるまで、それまで開いてて必要のないファイルを閉じるとかの事後処理が行え、その実装はプログラムの内容にもよりますが基本的にはそう難しくないはずです。

なので、このレベルの失敗でいちいちpanicするよりは、ユーザプログラムにその後の処理を任せた方が現実的かなと思います。

file.cのfilealloc関数

```
01 // Allocate a file structure.
02 struct file*
03 filealloc(void)
04 {
05   struct file *f;
06
07   acquire(&ftable.lock);
08   for(f = ftable.file; f < ftable.file + NFILE; f++){
09     if(f->ref == 0){
10       f->ref = 1;
11       release(&ftable.lock);
12       return f;
13     }
14   }
15   release(&ftable.lock);
16   return 0;
17 }
```

5. について

質問の答えとしては、「正しく作成されない。なぜならdirlinkが失敗するので、badラベル以下の行が実行され巻き戻されるので。」ということになるかと思います。
トランザクションを使ってるので、中途半端な状態で終わることはないので問題ないはずです。

sysfile.cのsys_link関数

```
01 // Create the path new as a link to the same inode as old.
02 int
03 sys_link(void)
04 {
05   char name[DIRSIZ], *new, *old;
06   struct inode *dp, *ip;
07
08   if(argstr(0, &old) < 0 || argstr(1, &new) < 0)
09     return -1;
10   if((ip = namei(old)) == 0)
11     return -1;
12
13   begin_trans();
14
15   ilock(ip);
16   if(ip->type == T_DIR){
17     iunlockput(ip);
18     commit_trans();
19     return -1;
20   }
21
22   ip->nlink++;
23   iupdate(ip);
24   iunlock(ip);
```

```
25
26    if((dp = nameiparent(new, name)) == 0)
27      goto bad;
28    ilock(dp);
29    if(dp->dev != ip->dev || dirlink(dp, name, ip->inum) < 0){
30      iunlockput(dp);
31      goto bad;
32    }
33    iunlockput(dp);
34    iput(ip);
35
36    commit_trans();
37
38    return 0;
39
40  bad:
41    ilock(ip);
42    ip->nlink--;
43    iupdate(ip);
44    iunlockput(ip);
45    commit_trans();
46    return -1;
47  }
```

6. について

まず、dirlinkが失敗（panicではなく）するパターンというのは、対象のディレクトリにすでに同じ名前が登録されているときだけです。
3つのdirlinkのうち2つは、新しく作成されたディレクトリに対して”.”と”..”を作成する処理であり、上記の理由から失敗しようがないはずです。
また3つめのdirlinkは、新しく作成したディレクトリ、もしくはファイルを親ディレクトリに登録するためのものですが、すでに親ディレクトリに同じ名前が登録されている場合は、前半のif((ip = dirlookup(dp, name, &off)) != 0)の部分によって除外されるので、3つめのdirlinkは失敗しないことになります。
当然、一連の処理中はdp（親ディレクトリのinode）がロックされているので、途中で他のプロセスで変更されることを考慮する必要もありません。

sysfile.cのcreate関数

```
01  static struct inode*
02  create(char *path, short type, short major, short minor)
03  {
04    uint off;
05    struct inode *ip, *dp;
06    char name[DIRSIZ];
07
08    if((dp = nameiparent(path, name)) == 0)
09      return 0;
10    ilock(dp);
11
12    if((ip = dirlookup(dp, name, &off)) != 0){
13      iunlockput(dp);
14      ilock(ip);
15      if(type == T_FILE && ip->type == T_FILE)
```

```
16        return ip;
17      iunlockput(ip);
18      return 0;
19    }
20
21    if((ip = ialloc(dp->dev, type)) == 0)
22      panic("create: ialloc");
23
24    ilock(ip);
25    ip->major = major;
26    ip->minor = minor;
27    ip->nlink = 1;
28    iupdate(ip);
29
30    if(type == T_DIR){  // Create . and .. entries.
31      dp->nlink++;  // for ".."
32      iupdate(dp);
33      // No ip->nlink++ for ".": avoid cyclic ref count.
34      if(dirlink(ip, ".", ip->inum) < 0 || dirlink(ip, "..", dp->inum)
< 0)
35        panic("create dots");
36    }
37
38    if(dirlink(dp, name, ip->inum) < 0)
39      panic("create: dirlink");
40
41    iunlockput(dp);
42
43    return ip;
44  }
```

7. について

本文でははiput(cp->cwd)となってますが、ソースではiput(proc->cwd)となっています。
また、ソースではiputでiunlockは呼ばれません。
ソースのログを辿ったところ、この課題は今のソースではなく古いソースに対するもののようです。
古いソースでは、iputはロック中のinodeを要求するようになってました。（現在は未ロックのinodeを要求する。）
iputだけでなく関連の関数の実装も今と大きく違うみたいなので、この課題はパスします。

sysfile.cのsys_chdir関数

```
01  int
02  sys_chdir(void)
03  {
04    char *path;
05    struct inode *ip;
06
07    if(argstr(0, &path) < 0 || (ip = namei(path)) == 0)
08      return -1;
09    ilock(ip);
10    if(ip->type != T_DIR){
11      iunlockput(ip);
12      return -1;
13    }
14    iunlock(ip);
```

```
15    iput(proc->cwd);
16    proc->cwd = ip;
17    return 0;
18  }
```

感想

今までのExercisesの回の中では一番まともに対応できたんじゃないかと思います。
まぁ思い込みで嘘書いてる可能性も大ですが…

一応今回で本文は終わりです。
長いようで短かったです。
興味深く読めたので、ここまで続けれたんだと思います。

あとは、付録A, Bが残ってますのでぼちぼちそちらも読んでいきたいと思います。

カテゴリー: 技術 | タグ: xv6 | 投稿日: 2012/4/29 日曜日 [http://peta.okechan.net/blog/archives/1680] |