日曜研究室

技術的な観点から日常を綴ります

[xv6 #61] Chapter 5 – File system – Code: Block allocator

テキストの70ページ

本文

inodeが指すブロックは、割り当てられていなければならない。
xv6のブロックアロケータは、ディスク上の空きビットマップ（1ビットが1ブロックに対応する）を維持する。
0ビットは、対応するブロックが空いているということを示す。
1ビットは、対応するブロックが利用中であるということを示す。
ブートセクタやスーパーブロックやinodeブロックやビットマップブロックに対応するビット群は、常に1がセットされている。

ブロックアロケータは、2つの機能を提供する。
balloc関数は、新たなディスク上のブロックを割り当て、bfree関数はブロックを解放する。
balloc関数は、readsb関数の呼び出しから開始し、ディスク（もしくはバッファキャッシュ）からスーパーブロックを読み込み、変数sbに格納する。
balloc関数は、（BBLOCKマクロを使って）ブートセクタやスーパーブロックやinodeブロックによって消費されているブロックがいくつあるか計算する事によって、どのブロックがそのデータブロックの空きビットマップを保持するか決定する。
ループでは、ブロック0からsb.sizeまで各ブロックを走査する。
ビットマップのビットが0の部分に対応するブロック、つまり空きであると示されているブロックを探す。
balloc関数がそのようなブロックを見つけた場合、ビットマップを更新し、そのブロックを返す。
効率のため、ループは2つの部分からなる。
外側のループは、各ビットマップブロックのビットを読み取る。
内側のループは、ひとつのビットマップブロックにあるBPBビットを全て調べる。
2つのプロセスが同時にブロックを割り当てようとして起こる可能性がある競合は、バッファキャッシュが一度に一つのプロセスだけを通すという事実によって防がれる。

fs.cのballoc, bfree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 }
27
28 // Free a disk block.
29 static void
30 bfree(int dev, uint b)
31 {
32   struct buf *bp;
33   struct superblock sb;
34   int bi, m;
35
36   readsb(dev, &sb);
37   bp = bread(dev, BBLOCK(b, sb.ninodes));
38   bi = b % BPB;
39   m = 1 << (bi % 8);
40   if((bp->data[bi/8] & m) == 0)
41     panic("freeing free block");
42   bp->data[bi/8] &= ~m;  // Mark block free on disk.
43   log_write(bp);
44   brelse(bp);
45 }
```

bfree関数は、対応するビットマップブロックを探し、対応するビットをクリアする。
こちらもまた、bread関数やbrelse関数によって暗黙的に排他処理されるので、bfree関数自体には排他のためのロックは必要ない。

感想

厳密にはまだinodeレベルの話ではなく、ブロックレベルの話のようです。
ですがinodeの割り当てなどで直接このあたりを使うんでしょう。

ちょっと前回の投稿から間が空きました。
さくらのVPS 512を2つ契約してるんですが、その1つをさくらのVPS 1Gに移行してました。
さくらのVPS 1Gいいですよね。安くて、性能もなかなか。

元々CentOS 5.5だったのを、新しいサーバではFedora 16を使うようにしたんですが、新しいsystemdの使い方を調べたり、NetworkManagerの不具合っぽい症状に悩まされたり、使ってたフレームワークのメジャーバージョンを上げたら互換性がなくてアプリの修正が必要になったり、元々気になってたアプリ性能を改善するためにgroongaを入れてそれを使うようにしたりと、移行のついでに思い切ってやりすぎたおかげで時間がかかってしまいました。
でもかなり勉強になったのでよしとします。

特にsystemdは慣れると、いろんなアプリケーションのサービス化が楽だし、groongaは正直まだ使いづらい（特にPythonからは。今のところrubyの方がライブラリが整備されてるみたい。）けど性能は素晴らしいの一言に尽きます。

カテゴリー: 技術 | タグ: xv6 | 投稿日: 2012/4/8 日曜日 [http://peta.okechan.net/blog/archives/1630] |