The inode Layer II

xv6-rev7

Carmi Merimovich

CS School, Tel Aviv Academic College

January 2,2017

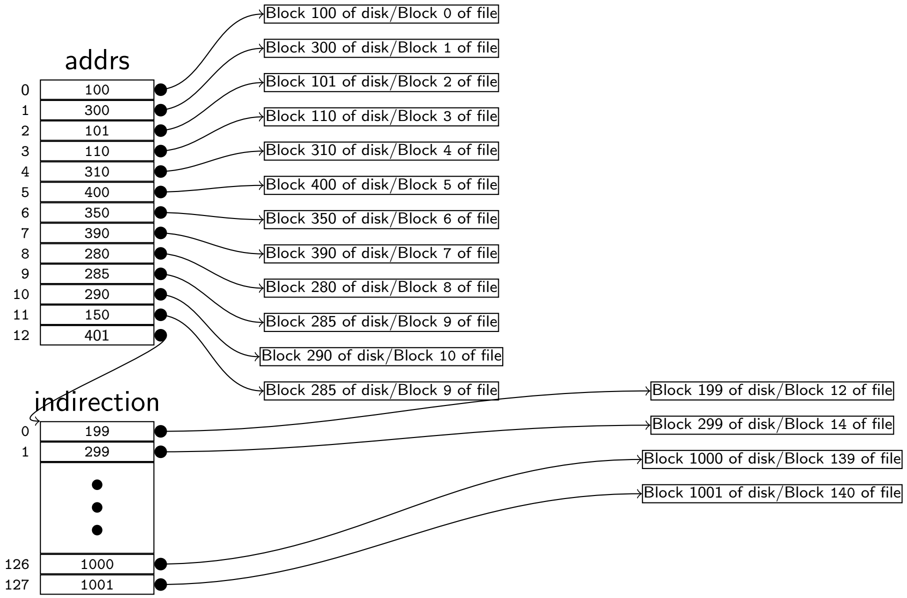
(memory) inode

```
#define NDIRECT 12

 struct inode {
  uint dev; // Device number
  uint inum; // Inode number
  int ref; // Reference count
  int flags; // I_BUSY, I_VALID

  short type; // copy of disk inode
  short major;
  short minor;
  short nlink;
  uint size;
  uint addrs[NDIRECT+1];
 };
#define I_BUSY 0x1
#define I_VALID 0x2
```

We aim at readi.

Beforehand we need bmap for which we need addrs.

addrs



addrs
0 100
1 300
2 101
3 110
4 310
5 400
6 350
7 390
8 280
9 285
10 290
11 150
12 401
Block 100 of disk/Block 0 of file
Block 300 of disk/Block 1 of file
Block 101 of disk/Block 2 of file
Block 110 of disk/Block 3 of file
Block 310 of disk/Block 4 of file
Block 400 of disk/Block 5 of file
Block 350 of disk/Block 6 of file
Block 390 of disk/Block 7 of file
Block 280 of disk/Block 8 of file
Block 285 of disk/Block 9 of file
Block 290 of disk/Block 10 of file
Block 285 of disk/Block 9 of file
indirection
0 199
1 299
126 1000
127 1001
Block 199 of disk/Block 12 of file
Block 299 of disk/Block 14 of file
Block 1000 of disk/Block 139 of file
Block 1001 of disk/Block 140 of file

bmap(ip,bn)

- Return the physical block # which block #bn of the file ip uses.
- If asked for a non existent block, allocates it.
- Allocation uses the buffer layer function balloc().
- If the inode is updated it is the caller responsibility to call iupdate.

bmap() (1)

```
bmap(struct inode *ip, uint bn) {
 uint addr, *a;
 struct buf *bp;

 if (bn < NDIRECT) {
  if ((addr = ip->addrs[bn]) == 0)
   ip->addrs[bn] = addr = balloc(ip->dev);
  return addr;
 }
```

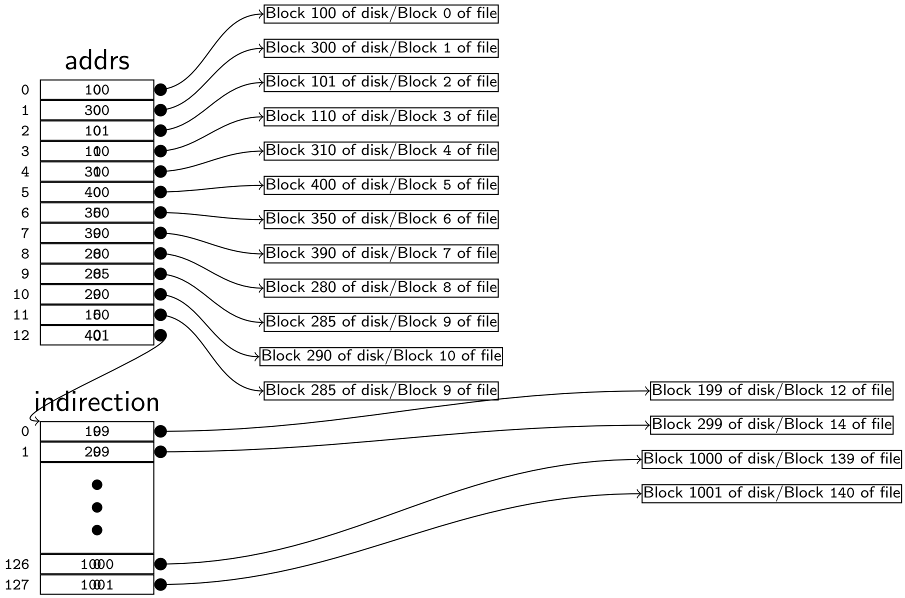
bmap() (2)

```
bn -= NDIRECT;

if (bn < NINDIRECT) {
 if ((addr = ip->addrs[NDIRECT]) == 0)
  ip->addrs[NDIRECT] = addr = balloc(ip->dev);
 bp = bread(ip->dev, addr);
 a = (uint*)bp->data;
 if ((addr = a[bn]) == 0) {
  a[bn] = addr = balloc(ip>dev);
  log_write(bp);
 }
 brelse(bp);
 return addr;
}
panic("bmap: out of range");
}
```

readi

```
readi(struct inode *ip, char *dst, uint off, uint n) {
 uint tot, m;
 struct buf *bp;
 if (ip->type == T_DEV) {
  if (ip->major < 0 || ip->major >= NDEV ||
                           !devsw[ip->major].read)
   return -1;
  return devsw[ip->major].read(ip, dst, n);
 }
 if (off > ip->size || off + n < off) return -1;
 if (off + n > ip->size)
  n = ip->size - off;
 for (tot=0; tot<n; tot+=m, off+=m, dst+=m) {
  bp = bread(ip->dev, bmap(ip, off/BSIZE));
  m = min(n - tot, BSIZE - off%BSIZE);
  memmove(dst, bp->data + off%BSIZE, m);
  brelse(bp);
 }
 return n;
```

Filling in addrs



addrs
0
100
1
300
2
101
3
110
4
310
5
400
6
350
7
390
8
280
9
285
10
290
11
150
12
401
indirection
0
199
1
299
126
1000
127
1001
Block 100 of disk/Block 0 of file
Block 300 of disk/Block 1 of file
Block 101 of disk/Block 2 of file
Block 110 of disk/Block 3 of file
Block 310 of disk/Block 4 of file
Block 400 of disk/Block 5 of file
Block 350 of disk/Block 6 of file
Block 390 of disk/Block 7 of file
Block 280 of disk/Block 8 of file
Block 285 of disk/Block 9 of file
Block 290 of disk/Block 10 of file
Block 285 of disk/Block 9 of file
Block 199 of disk/Block 12 of file
Block 299 of disk/Block 14 of file
Block 1000 of disk/Block 139 of file
Block 1001 of disk/Block 140 of file

writei (1)

```
int writei(struct inode *ip,char*src,uint off,uint n
 uint tot, m;
 struct buf *bp;

 if (ip->type == T_DEV) {
  if (ip->major < 0 || ip->major >= NDEV || !devsw[i
   return -1;
  return devsw[ip->major].write(ip, src, n);
 }

 if (off > ip->size || off + n < off) return -1;
 if (off + n > MAXFILE*BSIZE) return -1;
```

writei (2)

```
for (tot=0; tot<n; tot+=m, off+=m, src+=m) {
  bp = bread(ip->dev, bmap(ip, off/BSIZE));
  m = min(n - tot, BSIZE      off%BSIZE);
  memmove(bp->data + off%BSIZE, src, m);
  log_write(bp);
  brelse(bp);
}

if (n > 0 && off > ip->size) {
  ip->size = off;
  iupdate(ip);
}
return n;
}
```

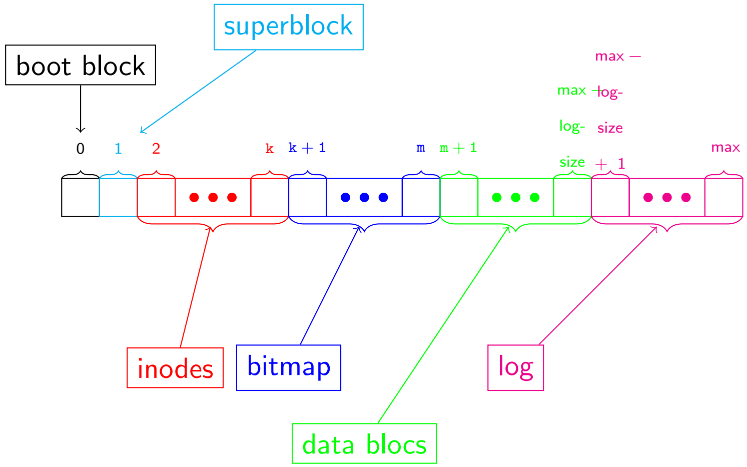
itrunc (1)

```
static void itrunc(struct inode *ip) {
 int i, j;
 struct buf *bp;
 uint *a;

 for (i = 0; i < NDIRECT; i++){
  if (ip->addrs[i]){
   bfree(ip->dev, ip->addrs[i]);
   ip->addrs[i] = 0;
  }
 }
```

itrunc (2)

```
  if (ip->addrs[NDIRECT]) {
    bp = bread(ip->dev, ip->addrs[NDIRECT]);
    a = (uint*)bp->data;
    for(j = 0; j < NINDIRECT; j++) {
      if(a[j])
        bfree(ip->dev, a[j]);
    }
    brelse(bp);
    bfree(ip->dev, ip->addrs[NDIRECT]);
    ip->addrs[NDIRECT] = 0;
  }

  ip->size = 0;
  iupdate(ip);
}
```

XV6 on disk structure: The big picture


superblock
boot block
max −
max + log-
log- size
0
1
2
k
k + 1
m
m + 1
size
+ 1
max
inodes
bitmap
log
data blocs