Operating System
Project #5: File System

Global Economics / Computer Science
2014311577 Dongmin Kim

Fixing Bmap Function

1. Fixing struct types

First of all, to support doubly-indirect block for the file system, we need to fix the inode structures for both on-disk and on memory.

On-disk inode is defined in fs.h, as follows.

```
#define NDIRECT 11
#define NINDIRECT (BSIZE / sizeof(uint))
#define NDINDIRECT (BSIZE / sizeof(uint)) * (BSIZE / sizeof(uint))
#define MAXFILE (NDIRECT + NINDIRECT + NDINDIRECT)

// On-disk inode structure
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+2];   // Data block addresses
};
```

Here, number of direct block(NDIRECT) is 11, and indirect block(NINDIRECT) is (BSIZE / sizeof(uint)), which is 128 (512/4). Similarly, it is possible to define doubly-indirect block. Since doubly indirect block contains pointer to the block of pointers, so it can cover up to (BSIZE/sizeof(uint)) * (BSIZE/sizeof(uint)), which is 16384. So in total, this system can cover up to 16523 sectors.
So the difference is the structure of diode->addrs, since it allows 11 direct/ 1 indirect/ 1 double-indirect pointers. Therefore the size is NDIRECT + 2.

similarly, in-memory copy can be fixed as follows, and this is defined in file.h

```
// in-memory copy of an inode
struct inode {
  uint dev;           // Device number
  uint inum;          // Inode number
  int ref;            // Reference count
  struct sleeplock lock; // protects everything below here
  int valid;          // inode has been read from disk?

  short type;         // copy of disk inode
  short major;
  short minor;
  short nlink;
  uint size;
  uint addrs[NDIRECT+2];
};
```

2. bmap function

bmap function is modified as follows:

```
static uint
bmap(struct inode *ip, uint bn)
{
  uint addr, *a, *a2;
  struct buf *bp, *bp2;
  uint idx1, idx2;

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

  if(bn < NINDIRECT){
    // Load indirect block, allocating if necessary.
    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;
  }

  bn -= NINDIRECT;

  idx1 = (bn) / (BSIZE/sizeof(uint));
  idx2 = (bn) % (BSIZE/sizeof(uint));

  if(bn < NDINDIRECT){
    // Load double indirect block, allocating if necessary.

    if((addr = ip->addrs[NDIRECT+1]) == 0)
      ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;

    if((addr = a[idx1]) == 0){
      a[idx1] = addr = balloc(ip->dev);
      log_write(bp);
    }
    brelse(bp);

    bp2 = bread(ip->dev, addr);
    a2 = (uint*)bp2->data;

    if((addr = a2[idx2]) == 0){
      a2[idx2] = addr = balloc(ip->dev);
      log_write(bp2);
    }

    brelse(bp2);
    return addr;
  }

  panic("bmap: out of range");
}
```

First two parts allocates Direct and Indirect blocks.

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

  if(bn < NINDIRECT){
    // Load indirect block, allocating if necessary.
    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;
  }
```

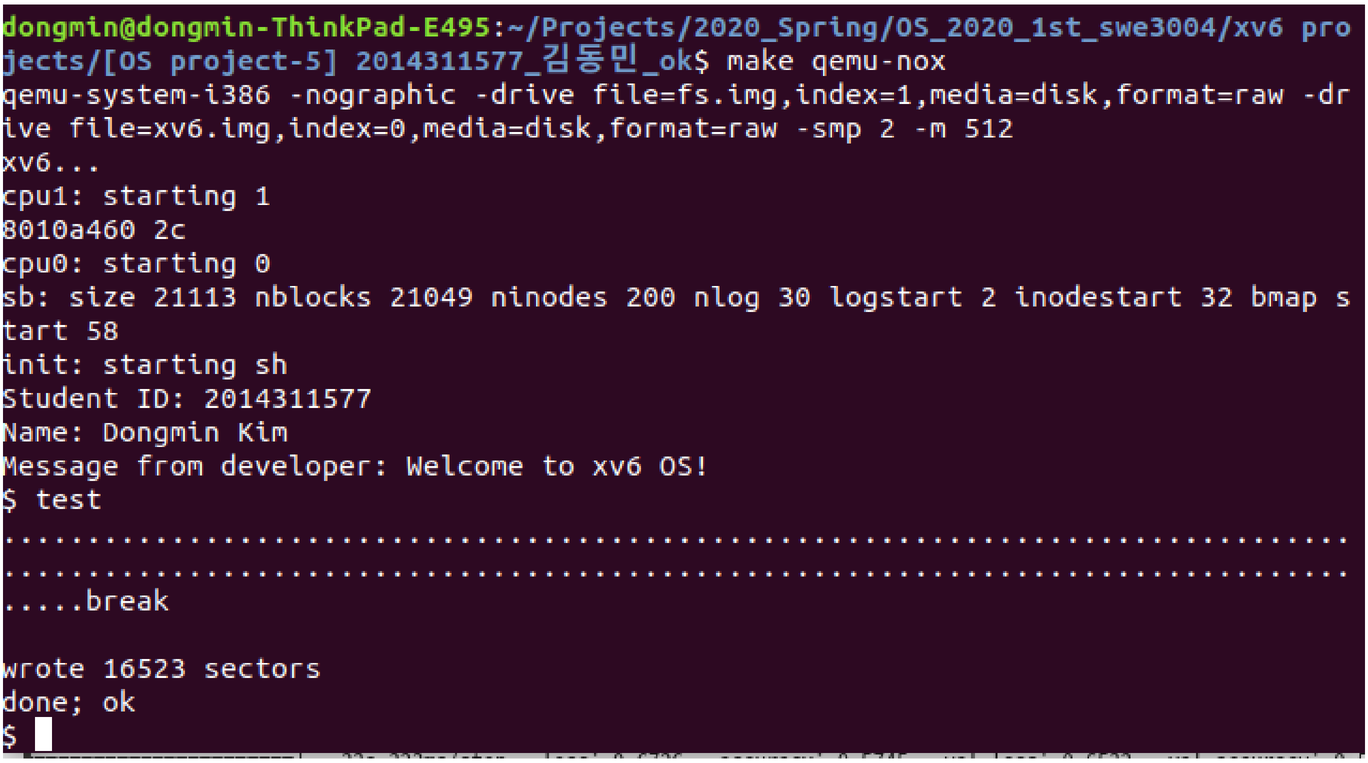
Similar to those processes above, allocation for doubly indirect block can be done as follows:

```
bn -= NINDIRECT;

idx1 = (bn) / (BSIZE/sizeof(uint));
idx2 = (bn) % (BSIZE/sizeof(uint));

if(bn < NDINDIRECT){
  // Load double indirect block, allocating if necessary.

  if((addr = ip->addrs[NDIRECT+1]) == 0)
    ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);

  bp = bread(ip->dev, addr);
  a = (uint*)bp->data;

  if((addr = a[idx1]) == 0){
    a[idx1] = addr = balloc(ip->dev);
    log_write(bp);
  }
  brelse(bp);

  bp2 = bread(ip->dev, addr);
  a2 = (uint*)bp2->data;

  if((addr = a2[idx2]) == 0){
    a2[idx2] = addr = balloc(ip->dev);
    log_write(bp2);
  }

  brelse(bp2);
  return addr;
}

panic("bmap: out of range");
```

Firstly, last index of the pointer array in dinode is referred, and allocate a block if necessary.
And two reference numbers, which are idx1, idx2 indicates their own location in first/ and second pointer blocks. Therefore, by calling a[idx1], it is possible to locate and allocate the blocks that contains pointer blocks. Similarly, by calling a2[idx2], it is possible to locate and allocate the blocks that contains actual blocks.

3. Testing

For the testing, FSSIZE were set to 21113.



```
dongmin@dongmin-ThinkPad-E495:~/Projects/2020_Spring/OS_2020_1st_swe3004/xv6 pro
jects/[OS project-5] 2014311577_김동민_ok$ make qemu-nox
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=raw -dr
ive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu1: starting 1
8010a460 2c
cpu0: starting 0
sb: size 21113 nblocks 21049 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap s
tart 58
init: starting sh
Student ID: 2014311577
Name: Dongmin Kim
Message from developer: Welcome to xv6 OS!
$ test
................................................................................
................................................................................
.....break

wrote 16523 sectors
done; ok
$ 
```