Fs

Design

0x1 doubly-indirect block

> Modify `bmap()` so that it implements a doubly-indirect block, in addition to direct blocks and a singly-indirect block. You'll have to have only 11 direct blocks, rather than 12, to make room for your new doubly-indirect block; you're not allowed to change the size of an on-disk inode. The first 11 elements of `ip->addrs[]` should be direct blocks; the 12th should be a singly-indirect block (just like the current one); the 13th should be your new doubly-indirect block. You are done with this exercise when `bigfile` writes 65803 blocks and `usertests` runs successfully:

Xv6 file system uses ext-like 12 direct blocks and 1single-indirect block in its inodes.

my job is to add a doubly-indirct block index in xv6 inode

1. modify inode structure:

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

// On-disk inode structure
struct dinode {
  short type;           // File type
  short major;          // Major device number (T_DEVICE only)
  short minor;          // Minor device number (T_DEVICE only)
  short nlink;          // Number of links to inode in file system
  uint size;            // Size of file (bytes)
  uint addrs[NDIRECT+2];   // Data block addresses
};
```

2. modify `bmap()`

   `bmap()` return the disk block address of the nth block in inode ip. my job is to add the search in doubly-indirect index.

```
static uint
bmap(struct inode *ip, uint bn)
{
  //printf("handle %d\n",bn);
  uint addr, *a;
```

```
    struct buf *bp;
    //if the data block in direct block
    ...
    //if data block in indirect block
    ...
    //if data block in double indirect block
    bn-= NINDIRECT;
    if(bn < NINDIRECT *NINDIRECT ) {
      //printf("alloc in double block\n");
      //locate double indirect block
      uint l1 = bn / NINDIRECT;
      uint l2 = bn % NINDIRECT;
      //load l0 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;
      uint addr1;
      if ((addr1 = a[l1]) == 0){
        a[l1] = addr1 = balloc(ip->dev);// save l1 indirect block in addr1
        log_write(bp);
      }
      brelse(bp);
      //load l1 indirect block, allocating if necessary
      bp = bread(ip->dev,addr1);
      a = (uint*)bp->data;
      if((addr = a[l2]) == 0){
        a[l2] = addr = balloc(ip->dev);
        log_write(bp);
      }
      brelse(bp);
      return addr;
    }
    panic("bmap: out of range");
}
```

0x2 symbolic link

> You will implement the `symlink(char *target, char *path)` system call, which creates a new symbolic link at path that refers to file named by target. For further information, see the man page symlink. To test, add symlinktest to the Makefile and run it. Your solution is complete when the tests produce the following output (including usertests succeeding).

1. write a new type "symlink" record in inode

```
uint64 sys_symlink(void){

```

```
  char name[DIRSIZ], new[MAXPATH], old[MAXPATH];
  struct inode *dp, *ip;

  if(argstr(0, old, MAXPATH) < 0 || argstr(1, new, MAXPATH) < 0)
    return -1;
  begin_op();
  //printf("from %s to %s\n", old,new);
  ip = namei(old);
  dp = namei(new);
  if ( ip !=0 && ip->type != T_DIR ){
    // add a link to the old path
    ilock(ip);
    ip->nlink++;
    iupdate(ip);
    iunlockput(ip);
  }
  if (dp==0){
    //fsprintf("dp is null\n");
    if (nameiparent(new,name)==0){
      printf("a null path under a null parent\n");
      goto bad;
    }
    if((dp= create(new,T_SYMLINK,0,0)) == 0){
      printf("error in create symlink block in \n");
      goto bad;
    }
    iunlock(dp);
  }
  ilock(dp);
  // dp holds the new path inode
  if(writei(dp,0,(uint64)old,dp->size,MAXPATH) != MAXPATH){
    printf("write error in symlink\n");
    iunlock(dp);
    goto bad;
  }
  //printf("%d -> %d\n", dp->inum, );
  dp->type=T_SYMLINK;
  iunlockput(dp);
  end_op();

  return 0;
//must release all inode lock before going to bad
bad:
  end_op();
  return -1;

}

```

2. recursively follow the link to find to node

```
struct inode * fetchSym(char *path, int depth){
  if (depth > 8){
    return 0;
  }
  struct inode *ip;
  if((ip = namei(path)) == 0){
    return 0;
  }
  ilock(ip);
  if(ip->type==T_SYMLINK ){
    char next[MAXPATH];
    if(readi(ip,0,(uint64)next,ip->size-MAXPATH ,MAXPATH) == 0 ) {
        iunlock(ip);
        return 0;
    }
    iunlock(ip);
    return  fetchSym(next, depth+1 );
  }
  iunlock(ip);
  return ip;
}
```

Result

Pass all test

```
make[1]: Leaving directory '/home/vielo/code/xv6lab'
== Test running bigfile ==
$ make qemu-gdb
running bigfile: OK (120.7s)
== Test running symlinktest ==
$ make qemu-gdb
(1.0s)
== Test   symlinktest: symlinks ==
  symlinktest: symlinks: OK
== Test   symlinktest: concurrent symlinks ==
  symlinktest: concurrent symlinks: OK
== Test usertests ==
$ make qemu-gdb
usertests: OK (221.5s)
== Test time ==
time: OK
Score: 100/100
```