Lab6: file system

- 姓名：吴欣怡
- 学号：PB21051111
- 虚拟机用户名：OS-PB21051111

Large files

实验分析

修改dinode等的定义以及bmap()内的操作，为混合索引机制添加二级索引页，来扩大能够支持的最大文件大小。

实验过程

修改fs.h中NDIRECT、MAXFILE、dinode的定义，腾出一个盘块号存储二级索引的盘块号。

```
// kernel/fs.h
#define NDIRECT 11 // 12 -> 11
#define NINDIRECT (BSIZE / sizeof(uint))
#define MAXFILE (NDIRECT + NINDIRECT + NINDIRECT * NINDIRECT)

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 (NDIRECT+1 -> NDIRECT+2)
};
```

修改file.h中的addrs

```
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]; // NDIRECT+1 -> NDIRECT+2
};
```

修改bmap，对于bn > NINDIRECT+NDIRECT的部分处理，依次找到一二级索引。
二级间接块的第一级是一个指针数组，其中每个指针指向一个一级间接块。
每个一级间接块都包含多个指针，每个指针又指向一个直接块，从而形成了二级间接块的两级结构。

```
static uint
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;
  }
  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;

  if(bn < NINDIRECT*NINDIRECT){
    // Load 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 bn_1=bn/NINDIRECT;
    if((addr = a[bn_1]) == 0){
      a[bn_1] = addr = balloc(ip->dev);
      log_write(bp);
    }
    brelse(bp);
    uint bn_2=bn%NINDIRECT;
    bp = bread(ip->dev, addr);
    a=(uint*)bp->data;
    if((addr = a[bn_2]) == 0){
      a[bn_2] = addr = balloc(ip->dev);
      log_write(bp);
    }
```

```
    brelse(bp);
    return addr;
  }

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

修改itrunc 函数,通过释放文件占用的直接块、一级间接块和二级间接块的存储空间，将文件的大小重置为零。

```
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;
    }
  }

  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;
  }

  if(ip->addrs[NDIRECT+1]){
    bp = bread(ip->dev, ip->addrs[NDIRECT+1]);
    a = (uint*)bp->data;
    for(j = 0; j < NINDIRECT; j++){
      if(a[j]){
        // 获取二级间接块
        struct buf *bp2=bread(ip->dev, a[j]);
        uint *a2=(uint*)bp2->data;
        // 释放二级间接块内的直接块

        for(int k=0;k<NINDIRECT;k++){
          if(a2[k])
            bfree(ip->dev,a2[k]);
        }
        brelse(bp2);
```

```
        bfree(ip->dev,a[j]);
      }
        //bfree(ip->dev, a[j]);
    }

    brelse(bp);
    bfree(ip->dev, ip->addrs[NDIRECT+1]);
    ip->addrs[NDIRECT+1] = 0;
  }
  ip->size = 0;
  iupdate(ip);
}
```

Symbolic links

实验分析

为 xv6 的文件系统添加符号链接支持。实现 symlink 系统调用，用于创建符号链接。

实验过程

修改syscall.c

```
...//
extern uint64 sys_uptime(void);
extern uint64 sys_symlink(void);

static uint64 (*syscalls[])(void) = {
...//
[SYS_close]   sys_close,
[SYS_symlink]   sys_symlink,
};
```

修改usys.pl

```
...//
entry("uptime");
entry("symlink");
```

修改syscall.h

```
...//
#define SYS_close  21
#define SYS_symlink 22
```

修改定义：

```
#define O_RDONLY   0x000
#define O_WRONLY   0x001
#define O_RDWR     0x002
#define O_CREATE   0x200
#define O_TRUNC    0x400
#define O_NOFOLLOW 0x800
```

编写sys_symlink函数：
调用 create 函数创建一个符号链接类型的 inode，该 inode 表示链接文件。使用 writei 函数将链接的目标路径写入 inode。调用 iunlockput 函数，该函数解锁 inode 并将其释放（因为这里不再需要它）。

```
uint64
sys_symlink(void){
  struct inode *ip;
  char target[MAXPATH],path[MAXPATH];
  if(argstr(0,target,MAXPATH)<0 || argstr(1,path,MAXPATH)<0)
    return -1;
  begin_op();

  ip=create(path,T_SYMLINK,0,0);
  if(ip==0){
    end_op();
    return -1;
  }
  if(writei(ip,0,(uint64)target,0,strlen(target))<0){
    end_op();
    return -1;
  }
  iunlockput(ip);
  end_op();
  return 0;
}
```

修改sys_open函数，遇到ip->type为T_SYMLINK的情况
如果打开模式中包含 O_NOFOLLOW 且目标文件是符号链接，则递归地跟随符号链接，最多跟随 10 层，否则认为有环。
通过 readi 函数读取符号链接的目标路径，并更新 path 变量。

```
uint64
sys_open(void)
{
  char path[MAXPATH];
  int fd, omode;
  struct file *f;
  struct inode *ip;
  int n;

  if((n = argstr(0, path, MAXPATH)) < 0 || argint(1, &omode) < 0)
    return -1;
```

```
  begin_op();

  if(omode & O_CREATE){
    ip = create(path, T_FILE, 0, 0);
    if(ip == 0){
      end_op();
      return -1;
    }
  } else {
    int symlink_depth = 0;
    while(1) { // recursively follow symlinks
      if((ip = namei(path)) == 0){
        end_op();
        return -1;
      }
      ilock(ip);
      if(ip->type == T_SYMLINK && (omode & O_NOFOLLOW) == 0) {
        if(++symlink_depth > 10) {
          // too many layer of symlinks, might be a loop
          iunlockput(ip);
          end_op();
          return -1;
        }
        if(readi(ip, 0, (uint64)path, 0, MAXPATH) < 0) {
          iunlockput(ip);
          end_op();
          return -1;
        }
        iunlockput(ip);
      } else {
        break;
      }
    }
    if(ip->type == T_DIR && omode != O_RDONLY){
      iunlockput(ip);
      end_op();
      return -1;
    }
  }

  // .......

  iunlock(ip);
  end_op();

  return fd;
}
```

实验评分

```
ubuntu@VM7782-OS-PB21051111:/home/ubuntu/桌面/xv6-labs-2020$ python3 grade-lab-f
s
make: “kernel/kernel”已是最新。
== Test running bigfile == running bigfile: OK (172.9s)
== Test running symlinktest == (4.7s)
== Test   symlinktest: symlinks ==
  symlinktest: symlinks: OK
== Test   symlinktest: concurrent symlinks ==
  symlinktest: concurrent symlinks: OK
== Test usertests == usertests: OK (259.7s)
== Test time ==
time: OK
Score: 100/100
```

实验总结

学习了文件系统的一些操作