Homework: bigger files for xv6

刘恒星 2022229044

April 14, 2023

1 实验过程

在这个作业中，需要增加 xv6 系统允许的文件大小。当前的 xv6 文件大小被限制为 140 扇区 (sectors)，因为 xv6 的索引节点 (innode) 包含了 12 个“直接的”块 (block) 的序号和一个“单独间接 (singly-indirect)”块序号，这个序号指向一个最多可以存储 128 个块号的块。为了增加允许的文件大小，需要修改 xv6 代码在每个索引节点中支持“双间接 (doubly-indirect)”块，包含 128 个单间接块的地址，每个块又包含最多 128 个数据块的地址。最终一个文件将会能够包含 11 + 128 + 128 * 128 = 16523 个扇区。

准备步骤

修改 xv6 Makefile 中 CPUS 的定义:

CPUS := 1

在 QEMUOPTS 之前添加

QEMUEXTRA = -snapshot

当 xv6 创建大文件时，以上两个步骤能够极大地加速 qemu。

mkfs 初始化文件系统时，定义其可用数据块少于 1000 个，而可用数据块太少而无法支持我们将要进行的更改。修改 param.h 以将 FSSIZE 设置为：

#define FSSIZE 20000 //文件系统的大小（以块为单位）

实验分析和步骤

将 big.c 添加到 xv6 目录下，将它添加到 UPROGS 列表中，启动 xv6，执行 big 命令，将会在 xv6 中创建一个 xv6 允许情况下最大的文件并报告大小

innode 定义在 fs.h 的 struct dinnode 中，关注其中的 NDIRECT, NINDIRECT, MAXFILE, addrs[]。

在磁盘上查找文件数据的代码在 fs.c 的 bmap() 中，在读取和写入文件时，bmap() 都会被调用。

bmap() 处理两种块序号，bn 是“逻辑块号 (logic numbers)”，一个与文件开头部分相关的块号，在 ip->addrs[] 中的块号是“磁盘块号 (disk block numbers)”

修改 bmap()，使它除了实现单间接块和直接块之外还实现了双间接块。单间接块的数目将会由 12 变为 11，多出的一个块将会成为双间接块。ip->addrs[] 的起始 11 个元素应该是直接块，第 12 个元素应该是单间接块，第 13 个元素应该为新的双间接块。

Hints

1. 记得修改 NDIRECT 时, 也要修改 file.h 中 inode->addrs[] 及 fs.h 中 dinode->addrs[] 的大小 (打开文件后会将硬盘上的索引节点复制到内存中, 因此要保证二者一致)。

2. 修改 NDIRECT 的定义后, 确保创建一个新的 fs.img, 因为 mkfs 会使用 NDIRECT 来构建初始的文件系统, 如果删除了 fs.img, 在 Unix(而非 xv6) 下 make 会为你构建一个新的。

3. 对每个调用 bread() 读取的块都要记得使用 brelse()。

修改代码如下

```
// fs.h
#define NDIRECT 11
#define NINDIRECT (BSIZE / sizeof(uint))
#define MAXFILE (NDIRECT + NINDIRECT * 129)

// 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
};

//file.h
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];
};

//fs.c

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

  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);//Allocate a zeroed disk block.
    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 < MAXFILE-NINDIRECT-NDIRECT){
    // Load indirect block, allocating if necessary.
    if((addr = ip->addrs[NDIRECT+1]) == 0)
      ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);//Allocate a zeroed disk block.
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    if((addr = a[bn/NINDIRECT]) == 0){
      a[bn/NINDIRECT] = addr = balloc(ip->dev);
      log_write(bp);
    }
    dp=bread(ip->dev, addr);
    a = (uint*)dp->data;
    if((addr = a[bn%NINDIRECT]) == 0){
      a[bn%NINDIRECT] = addr = balloc(ip->dev);
      log_write(dp);
    }
    brelse(dp);
    brelse(bp);
    return addr;
  }

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

效果如下：

```
Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 20000 nblocks 19937 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ big
.....................................................................................................
wrote 16523 sectors
done; ok
```