bigger files for xv6

- 姓名：陈姝宇
- 班级：软工3班
- 学号：3017218119
- 日期：10月1日

1 实验内容

增加xv6文件的最大大小。

> 当前，xv6文件限制为140个扇区或71,680字节。此限制来自以下事实：xv6索引节点包含12个“直接”块编号和一个“单间接”块编号，这是指最多容纳128个以上块编号的块，总共12 + 128 = 140 。

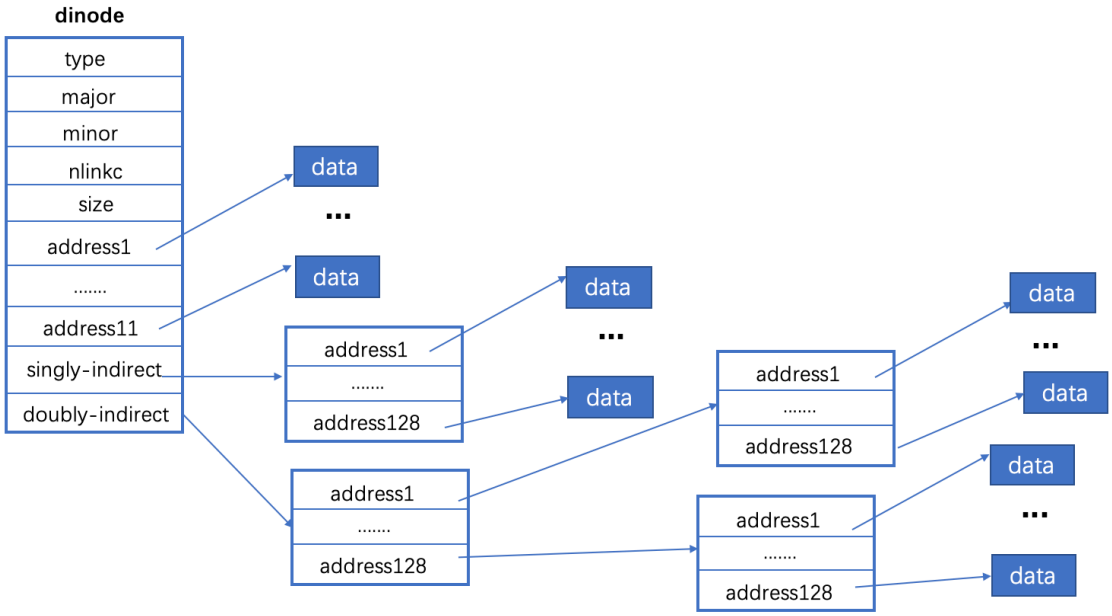
更改xv6文件系统代码，以在每个inode中支持“双重间接”块，其中包含128个单间接块地址，每个间接块最多可以包含128个数据块地址。结果将是一个文件最多可以包含16523个扇区（约8.5兆字节）。

2. 实验分析

经过计算可知要使文件最多可以有16523个扇区，直接指向数据块的地址个数应当为11个（no.1-no.11），single-indirect为1个（no.12），doubly-indirect为1个（no.13）。

```
128*128=16384
16523-16384=139
139-128=11
```

文件结构如下图所示：



3 实验步骤

3.1 初始准备

1. 链接ubuntu虚拟机，进入xv6-public项目中

```
✘ ⚙ chenshuyu@chenshuyudeMacBook-Pro-2 ~/AndroidProject ssh 172.16.85.131
chenshuyu@172.16.85.131's password:
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-62-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

156 packages can be updated.
61 updates are security updates.

New release '18.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Fri Oct 11 19:54:39 2019 from 172.16.85.1
➜  ~ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos  examples.desk
top  test  xv6-public
➜  ~ cd xv6-public
➜  xv6-public git:(master) ls
BUGS        _zombie          cat.sym        entryother.o  grep.c       ioapic.o    lr
.asm        mkdir.d    printf.o     show1         string.o   traps.h       vectors.o
LICENSE     asm.h            console.c      exec.c        grep.d       kalloc.c    lr
```

2. 修改Makefile

① 使用vim打开Makefile

```
➜  xv6-public git:(master) sudo vim Makefile
[sudo] password for chenshuyu:
```

② 修改Makefile的`CPUS`定义，使其显示为CPUS = 1，

```
219 ifndef CPUS
220 CPUS := 1
221 endif
```

③ 在QEMUOPTS之前加QEMUEXTRA = -snapshot

```
222 QEMUEXTRA = -snapshot
223 QEMUOPTS = -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,inde
    x=0,media=disk,format=raw -smp $(CPUS) -m 512 $(QEMUEXTRA)
```

3. `mkfs`初始化文件系统时，其可用数据块少于1000个，而可用数据块太少而无法展示您将要进行的更改。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）

```
 11 #define LOGSIZE      (MAXOPBLOCKS*3)  // max data blocks in on-disk log
 12 #define NBUF         (MAXOPBLOCKS*3)  // size of disk block cache
 13 #define FSSIZE       20000  // size of file system in blocks
 14
~
```

4. 创建扇区大小

① 下载big.c，上传到虚拟机中的XV6目录下，并检查是否上传成功。

```
✘ ⚙ chenshuyu@chenshuyudeMacBook-Pro-2  ~  sftp 172.16.85.131
chenshuyu@172.16.85.131's password:
Connected to 172.16.85.131.
sftp> put /Users/chenshuyu/Downloads/big.c xv6-public
Uploading /Users/chenshuyu/Downloads/big.c to /home/chenshuyu/xv6-public/big.c
/Users/chenshuyu/Downloads/big.c              100%  985    616.2KB/s   00:00
```

```
➜  xv6-public git:(master) ✗ find bi
big.c  bio.c  bio.d  bio.o
```

② 将其添加到列表UPROGS，启动XV6，并运行 big。它创建的文件与xv6允许的一样大，并报告生成的大小。应该说140个扇区。

```
168 UPROGS=\
169         _cat\
170         _echo\
171         _forktest\
172         _grep\
173         _init\
174         _kill\
175         _ln\
176         _ls\
177         _mkdir\
178         _rm\
179         _sh\
180         _stressfs\
181         _usertests\
182         _wc\
183         _zombie\
184         _big\
```

```
QEMU - Press Ctrl-Alt to exit mouse grab
$ big
.
wrote 140 sectors
done; ok
```

3.2 阅读源代码

> 磁盘上的inode的格式由定义结构dinode在fs.h文件。您对NDIRECT， NINDIRECT，MAXFILE 和 `struct dinode` 的addrs [] 元素特别感兴趣。看看 这里为标准XV6的inode的示意
>
> 磁盘上的文件的数据的代码在 `BMAP ()` 在 `fs.c` 。读写文件时都会调用 `bmap ()` 。写入时， `bmap ()` 会根据需要分配新的块来保存文件内容，并在需要时分配一个间接块来保存块地址。
>
> bmap（）处理两种块号。BN 参数是一个“逻辑块” -相对于所述文件的开始块号。ip-> addrs [] 中的块号以及Bread() 的参数是磁盘块号。您可以将 bmap()视为将文件的逻辑块号映射到磁盘块号。

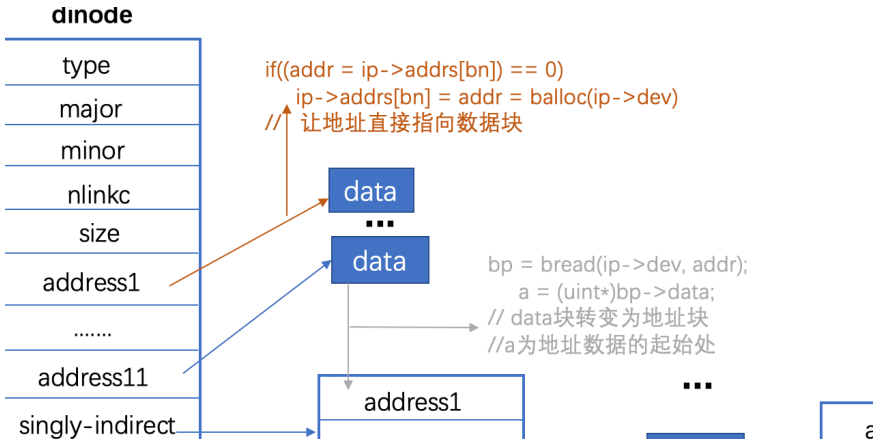
```
28 // On-disk inode structure
29 struct dinode {
30   short type;           // File type
31   short major;          // Major device number (T_DEV only)
32   short minor;          // Minor device number (T_DEV only)
33   short nlink;          // Number of links to inode in file system
34   uint size;            // Size of file (bytes)
35   uint addrs[NDIRECT+1];   // Data block addresses
36 };
```

```
24 #define NDIRECT 12
25 #define NINDIRECT (BSIZE / sizeof(uint))
26 #define MAXFILE (NDIRECT + NINDIRECT)
```

- **NDIRECT** 是直接指向数据块的地址个数，
- **NINDIRECT** = (BSIZE/sizeof(uint)) 是直接指向的数据块存储地址，再由地址指向数据块，存有多少个地址，即间接指向多少个数据块
- **MAXFILE** 文件最大扇区数

```
374 bmap(struct inode *ip, uint bn)
375 {
376   uint addr, *a; // 一级数据块
377   struct buf *bp; // 暂存
378 
379   if(bn < NDIRECT){ // 1-12区间，直接指向data
380     if((addr = ip->addrs[bn]) == 0)
381       ip->addrs[bn] = addr = balloc(ip->dev); // 直接指向data块
382     return addr; //返回地址
383   }
384   bn -= NDIRECT; // 大于直接指向数据块的最大大小，先减去这部分
385 
386   if(bn < NINDIRECT){ // 一级指向
387     // Load indirect block, allocating if necessary.
388     if((addr = ip->addrs[NDIRECT]) == 0)
389       ip->addrs[NDIRECT] = addr = balloc(ip->dev); // 先分配一级数据块
390     bp = bread(ip->dev, addr);
391     a = (uint*)bp->data; // 一级数据块的内容
392     if((addr = a[bn]) == 0){ // 由一级地址指向data
393       a[bn] = addr = balloc(ip->dev);
394       log_write(bp); //写入bp
395     }
396     brelse(bp); // 释放bp缓存
397     return addr; //返回地址
398   }
399 
400   panic("bmap:out of range") // 超过最大范围
401}
// 第一个if是地址直接指向数据块；
// 第二个if是地址先在第12块分配指向的数据块，再从这个数据块指向另外一个数据块
```

虽然不知道代码中用的bread、log_write 函数具体有什么作用，但是可以根据文件大小创建原理，大胆猜测每一行代码的作用，代码内容的图片解释图如下：



3.3 实际操作

① 修改全局变量值：NDIRECT 应改为11，同时第12个为原来indirect的模式，原来indirect模式的地址改为doubly-indirect。MAXFILE也应换成新的最大值。

```
24 #define NDIRECT 11
25 #define NINDIRECT (BSIZE / sizeof(uint))
26 #define MAXFILE (NDIRECT + NINDIRECT + NINDIRECT * NINDIRECT)
27
```

```
28 // On-disk inode structure
29 struct dinode {
30   short type;           // File type
31   short major;          // Major device number (T_DEV only)
32   short minor;          // Minor device number (T_DEV only)
33   short nlink;          // Number of links to inode in file system
34   uint size;            // Size of file (bytes)
35   uint addrs[NDIRECT+2];   // Data block addresses
36 };
```

② bmap函数中增加第三种if情况，需要有二级间接指向：

先是为第13块指向一级data，再通过 bn/NINDIRECT 找到一级data中哪个位置的地址应当指向所求数据块对应的二级data，注意数组是从0开始，所以除法运算出来的数正好为地址在数组中的位置。

然后再取余数，找到第二块data中的哪一个地址指向传入参数对应的的data。

```
401   if(bn < NINDIRECT*NINDIRECT){
402     // Load doubly-indirect block, allocating if necessary.
403     if((addr = ip->addrs[NDIRECT+1]) == 0)
404       ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);
405     bp = bread(ip->dev, addr);
406     a = (uint*)bp->data;
407     first = bn / NINDIRECT; // caculate how many one which point to two
408     if((addr = a[first]) == 0){
409       a[first] = addr = balloc(ip->dev);
410       log_write(bp);
411     }
412     bpsecond = bread(ip->dev, addr);
413     seconda = (unit*) bpsecond->data;
414     second = bn % NINDIRECT;
415     if((addr = seconda[second]) == 0){
416       addr = seconda[seond] = balloc(ip->dev);
417       log_write(bpsecond)
418     }
419     brelse(bptemp);
420     brelse(bp);
421     return addr;
422   }
423
```

运行之后有语法错误：

```
.o fs.c
fs.c: In function ‘bmap’:
fs.c:413:16: error: ‘unit’ undeclared (first use in this function)
     seconda = (unit*) bpsecond->data;
               ^
fs.c:413:16: note: each undeclared identifier is reported only once for each fun
ction it appears in
fs.c:413:21: error: expected expression before ‘)’ token
     seconda = (unit*) bpsecond->data;
                     ^
fs.c:416:22: error: ‘seond’ undeclared (first use in this function)
       addr = seconda[seond] = balloc(ip->dev);
                      ^
fs.c:418:5: error: expected ‘;’ before ‘}’ token
     }
     ^
fs.c:419:12: error: ‘bptemp’ undeclared (first use in this function)
     brelse(bptemp);
            ^
<builtin>: recipe for target 'fs.o' failed
make: *** [fs.o] Error 1
```

根据错误提示修改代码：

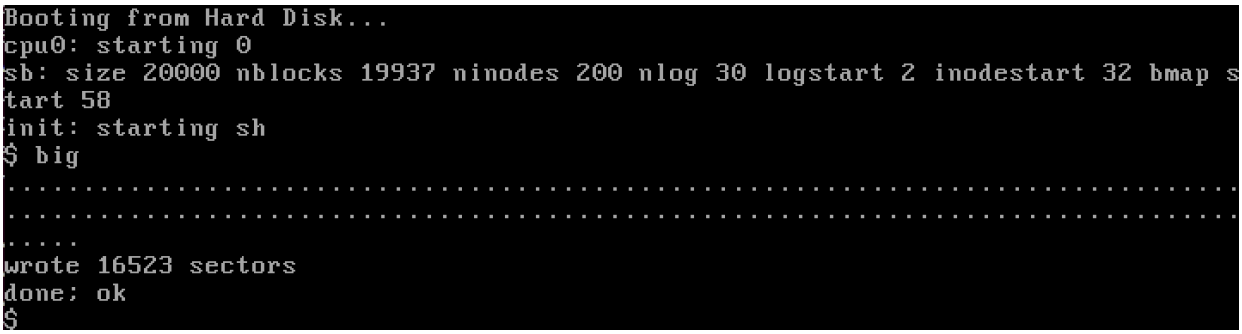
```
412     bpsecond = bread(ip->dev, addr);
413     seconda = (uint*) bpsecond->data;
414     second = bn % NINDIRECT;
415     if((addr = seconda[second]) == 0){
416       addr = seconda[second] = balloc(ip->dev);
417       log_write(bpsecond);
418     }
419     brelse(bpsecond);
420     brelse(bp);
421     return addr;
```

```
// 修改后的bmap函数
374 bmap(struct inode *ip, uint bn)
375 {
376   uint addr, *a, *seconda, first, second;
377   struct buf *bp, *bpsecond;
378
379   if(bn < NDIRECT){
380     if((addr = ip->addrs[bn]) == 0)
381       ip->addrs[bn] = addr = balloc(ip->dev);
382     return addr;
383   }
384   bn -= NDIRECT;
385
386   if(bn < NINDIRECT){
387     // Load indirect block, allocating if necessary.
388     if((addr = ip->addrs[NDIRECT]) == 0)
389       ip->addrs[NDIRECT] = addr = balloc(ip->dev);
390     bp = bread(ip->dev, addr);
391     a = (uint*)bp->data;
```

```
392     if((addr = a[bn]) == 0){
393       a[bn] = addr = balloc(ip->dev);
394       log_write(bp);
395     }
396     brelse(bp);
397     return addr;
398   }
399   bn -= NINDIRECT;
400 
401   if(bn < NINDIRECT*NINDIRECT){
402     // Load doubly-indirect block, allocating if necessary.
403     if((addr = ip->addrs[NDIRECT+1]) == 0)
404       ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);
405     bp = bread(ip->dev, addr);
406     a = (uint*)bp->data;
407     first = bn / NINDIRECT; // caculate how many one which point to two
408     if((addr = a[first]) == 0){
409       a[first] = addr = balloc(ip->dev);
410       log_write(bp);
411     }
412     bpsecond = bread(ip->dev, addr);
413     seconda = (uint*) bpsecond->data;
414     second = bn % NINDIRECT;
415     if((addr = seconda[second]) == 0){
416       addr = seconda[second] = balloc(ip->dev);
417       log_write(bpsecond);
418     }
419     brelse(bpsecond);
420     brelse(bp);
421     return addr;
422   }
423 
424   panic("bmap: out of range");
425 }
```

4 测试结果

进入xv6-public，执行make qemu，再执行big。出现如下图所示，创建16523个sectors，成功增加xv6文件的最大大小。



5 实验总结

① 将big.c加入makefile中，一开始是直接在最后一行加入big.c\，发现不能运行，然后根据列表UPROGS的特点，仿照前面参数的格式，改成_big\就能成功执行big。

② 使用vim编写c语言代码，没有IDE智能，特别容易打错参数名或者函数名，也就是代码有语法错误。

③ 即使不知道bmap中调用的函数源码，但是也可以通过函数名和文件大小创建原理来推测其作用，在创建二索引块的时候，依照之前的形式编写代码，注意参数名不能混淆。

④ 我的ubuntu系统上运行qemu，鼠标会被吞掉，使用快捷键也不行。如果想退出qemu，输入命令是不能成功退出的，只能直接关掉虚拟机再重启。