

操作系统原理

实 验 报 告

学 院 智能与计算学部
年 级 2019 级
班 级 留学生班
学 号 6319000359
姓 名 张明君

2020 年 12 月 5 日

天津大学

操作系统原理实验报告

题目: Bigger files for xv6

学院名称	智能与计算学部
专 业	计算机科学与技术
学生姓名	张明君
学 号	6319000359
年 级	2019 级
班 级	留学生班
时 间	2020. 12. 05

目 录

实验名称	1
实验目的	1
实验内容	1
实验步骤与分析	3
实验结论及心得体会	11

Homework: bigger files for xv6

1. 实验目的

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 don't have to modify xv6 to handle deletion of files with doubly-indirect blocks.

If all goes well, `big` will now report that it can write 16523 sectors. It will take `big` a few dozen seconds to finish.

2. 实验内容

Preliminaries

Modify your Makefile's `CPUS` definition so that it reads:

```
CPUS := 1
```

Add

```
QEMUEXTRA = -snapshot
```

right before `QEMUOPTS`

The above two steps speed up qemu tremendously when xv6 creates large files.

`mkfs` initializes the file system to have fewer than 1000 free data blocks, too few to show off the changes you'll make. Modify `param.h` to set `FSSIZE` to:

```
#define FSSIZE      20000 // size of file system in blocks
```

Download [big.c](#) into your xv6 directory, add it to the `UPROGS` list, start up xv6, and run `big`. It creates as big a file as xv6 will let it, and reports the resulting size. It should say 140 sectors.

What to Look At

The format of an on-disk inode is defined by `struct dinode` in `fs.h`. You're particularly interested in `NDIRECT`, `NINDIRECT`, `MAXFILE`, and the `addrs[]` element of `struct dinode`. Look [here](#) for a diagram of the standard xv6 inode.

The code that finds a file's data on disk is in `bmap()` in `fs.c`. Have a look at it and make sure you understand what it's doing. `bmap()` is called both when reading and writing

a file. When writing, `bmap()` allocates new blocks as needed to hold file content, as well as allocating an indirect block if needed to hold block addresses. `bmap()` deals with two kinds of block numbers. The `bn` argument is a "logical block" -- a block number relative to the start of the file. The block numbers in `ip->addrs[]`, and the argument to `bread()`, are disk block numbers. You can view `bmap()` as mapping a file's logical block numbers into disk block numbers.

Your Job

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 don't have to modify xv6 to handle deletion of files with doubly-indirect blocks. If all goes well, `big` will now report that it can write 16523 sectors. It will take `big` a few dozen seconds to finish.

Hints

Make sure you understand `bmap()`. Write out a diagram of the relationships between `ip->addrs[]`, the indirect block, the doubly-indirect block and the singly-indirect blocks it points to, and data blocks. Make sure you understand why adding a doubly-indirect block increases the maximum file size by 16,384 blocks (really 16383, since you have to decrease the number of direct blocks by one).

Think about how you'll index the doubly-indirect block, and the indirect blocks it points to, with the logical block number.

If you change the definition of `NDIRECT`, you'll probably have to change the size of `addrs[]` in `struct inode` in `file.h`. Make sure that `struct inode` and `struct dinode` have the same number of elements in their `addrs[]` arrays.

If you change the definition of `NDIRECT`, make sure to create a new `fs.img`, since `mkfs` uses `NDIRECT` too to build the initial file systems. If you delete `fs.img`, `make` on Unix (not xv6) will build a new one for you.

If your file system gets into a bad state, perhaps by crashing, delete `fs.img` (do this from Unix, not xv6). `make` will build a new clean file system image for you.

Don't forget to `brelse()` each block that you `bread()`.

You should allocate indirect blocks and doubly-indirect blocks only as needed, like the original `bmap()`.

3. 实验步骤与分析（要细化如何实现的思路或流程图）

In this assignment you'll increase the maximum size of an xv6 file. Currently xv6 files are limited to 140 sectors, or 71,680 bytes. This limit comes from the fact that an xv6 inode contains 12 "direct" block numbers and one "singly-indirect" block number, which refers to a block that holds up to 128 more block numbers, for a total of $12+128=140$. You'll change the xv6 file system code to support a "doubly-indirect" block in each inode, containing 128 addresses of singly-indirect blocks, each of which can contain up to 128 addresses of data blocks. The result will be that a file will be able to consist of up to 16523 sectors (or about 8.5 megabytes).

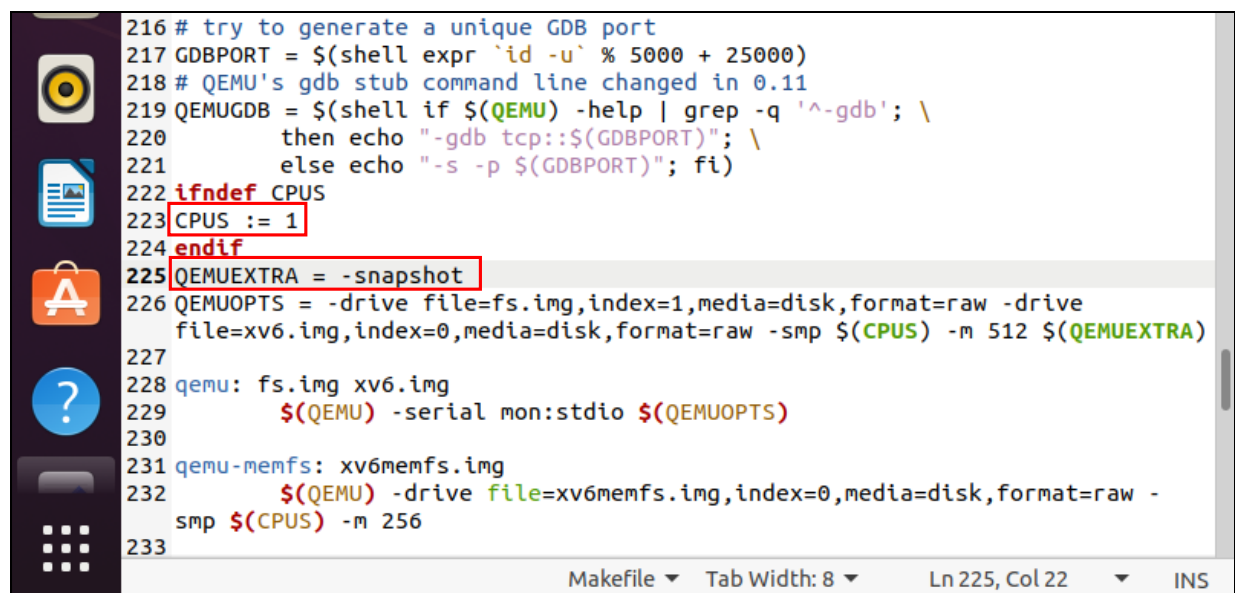
Preliminaries

Modify your Makefile's CPUS definition so that it reads:

```
CPUS := 1
```

And then we have to add

QEMUEXTRA = -snapshot right before QEMUOPTS and then don't forget to add `_big\` in Makefile



```
216 # try to generate a unique GDB port
217 GDBPORT = $(shell expr `id -u` % 5000 + 25000)
218 # QEMU's gdb stub command line changed in 0.11
219 QEMUGDB = $(shell if $(QEMU) -help | grep -q '^-gdb'; \
220     then echo "-gdb tcp::$(GDBPORT)"; \
221     else echo "-s -p $(GDBPORT)"; fi)
222 ifndef CPUS
223 CPUS := 1
224 endif
225 QEMUEXTRA = -snapshot
226 QEMUOPTS = -drive file=fs.img,index=1,media=disk,format=raw -drive
    file=xv6.img,index=0,media=disk,format=raw -smp $(CPUS) -m 512 $(QEMUEXTRA)
227
228 qemu: fs.img xv6.img
229     $(QEMU) -serial mon:stdio $(QEMUOPTS)
230
231 qemu-memfs: xv6memfs.img
232     $(QEMU) -drive file=xv6memfs.img,index=0,media=disk,format=raw -
    smp $(CPUS) -m 256
233
```

Figure 1: Use only one CPU

Modify `param.h` to set FSSIZE to:

```
#define FSSIZE      20000 // size of file system in blocks
```

What to Look At

The format of an on-disk inode is defined by `struct dinode` in `fs.h`. You're particularly interested in `NDIRECT`, `NINDIRECT`, `MAXFILE`, and the `addrs[]` element of `struct dinode`. Look for diagram

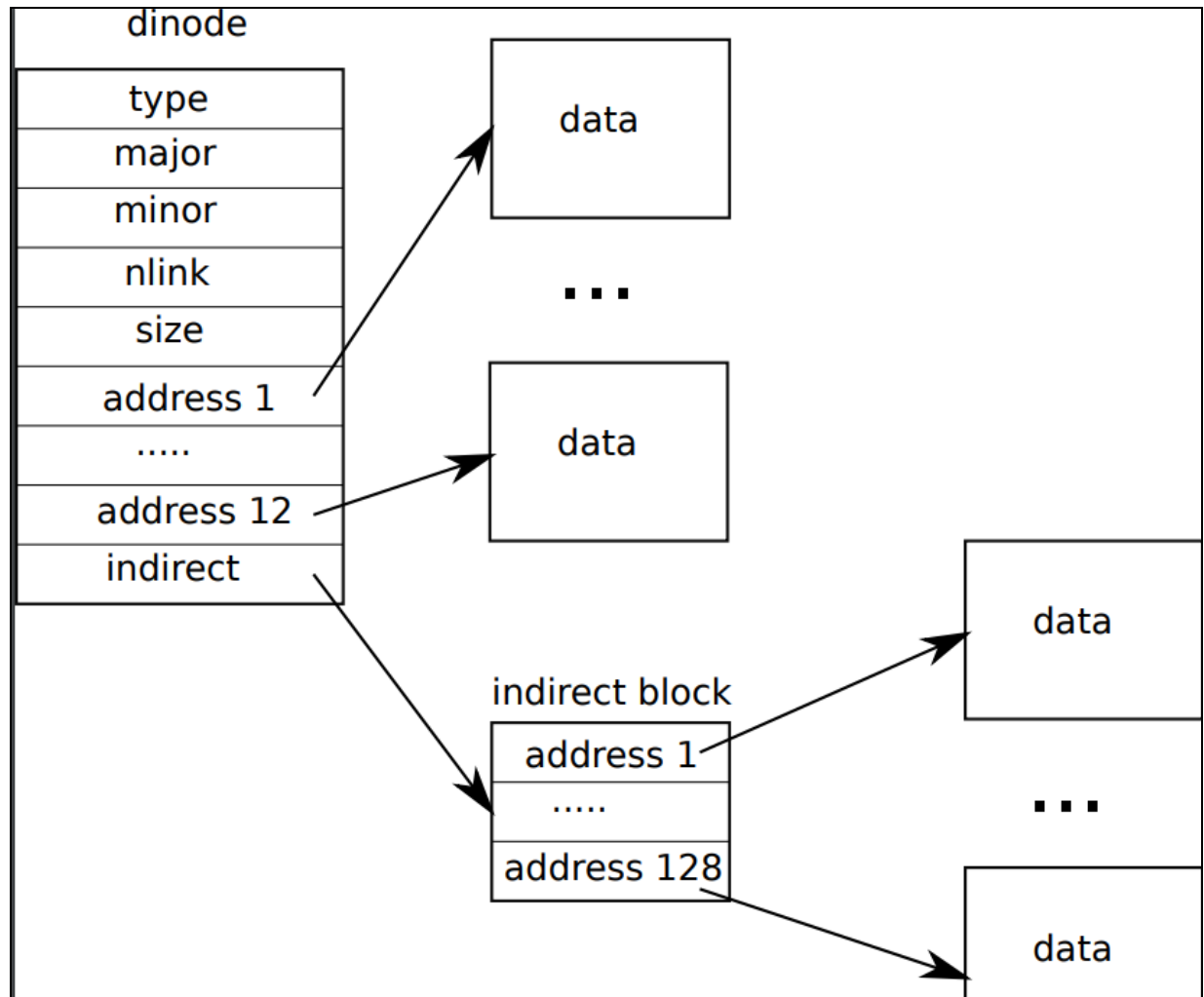


Figure 2:Struct Dinode

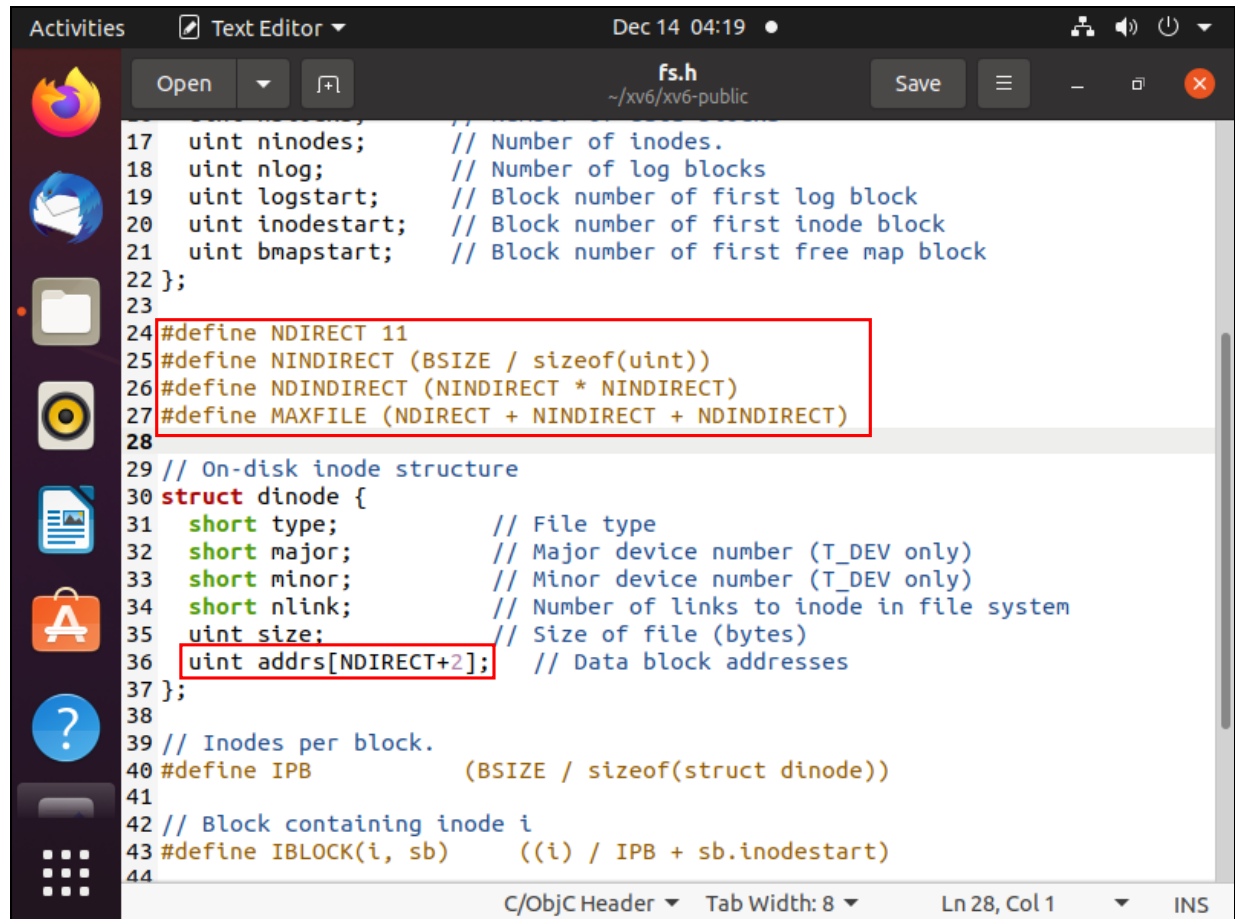
The code that finds a file's data on disk is in `bmap()` in `fs.c`. Have a look at it and make sure you understand what it's doing. `bmap()` is called both when reading and writing a file. When writing, `bmap()` allocates new blocks as needed to hold file content, as well as allocating an indirect block if needed to hold block addresses.

`bmap()` deals with two kinds of block numbers. The `bn` argument is a "logical block" -- a block number relative to the start of the file. The block numbers in `ip->addrs[]`, and the argument to `bread()`, are disk block numbers. You can view `bmap()` as mapping a file's logical block numbers into disk block numbers.

Your Job

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.

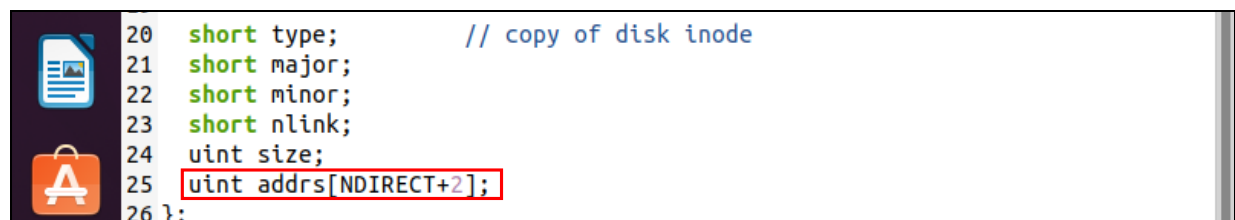
1/First step, We have to add some codes in the file `fs.h` like below:



```
17  uint ninodes;           // Number of inodes.
18  uint nlog;              // Number of log blocks
19  uint logstart;          // Block number of first log block
20  uint inodestart;        // Block number of first inode block
21  uint bmapstart;         // Block number of first free map block
22 };
23
24 #define NDIRECT 11
25 #define NINDIRECT (BSIZE / sizeof(uint))
26 #define NDINDIRECT (NINDIRECT * NINDIRECT)
27 #define MAXFILE (NDIRECT + NINDIRECT + NDINDIRECT)
28
29 // On-disk inode structure
30 struct dinode {
31     short type;           // File type
32     short major;          // Major device number (T_DEV only)
33     short minor;          // Minor device number (T_DEV only)
34     short nlink;          // Number of links to inode in file system
35     uint size;            // Size of file (bytes)
36     uint addrs[NDIRECT+2]; // Data block addresses
37 };
38
39 // Inodes per block.
40 #define IPB (BSIZE / sizeof(struct dinode))
41
42 // Block containing inode i
43 #define IBLOCK(i, sb) ((i) / IPB + sb.inodestart)
44
```

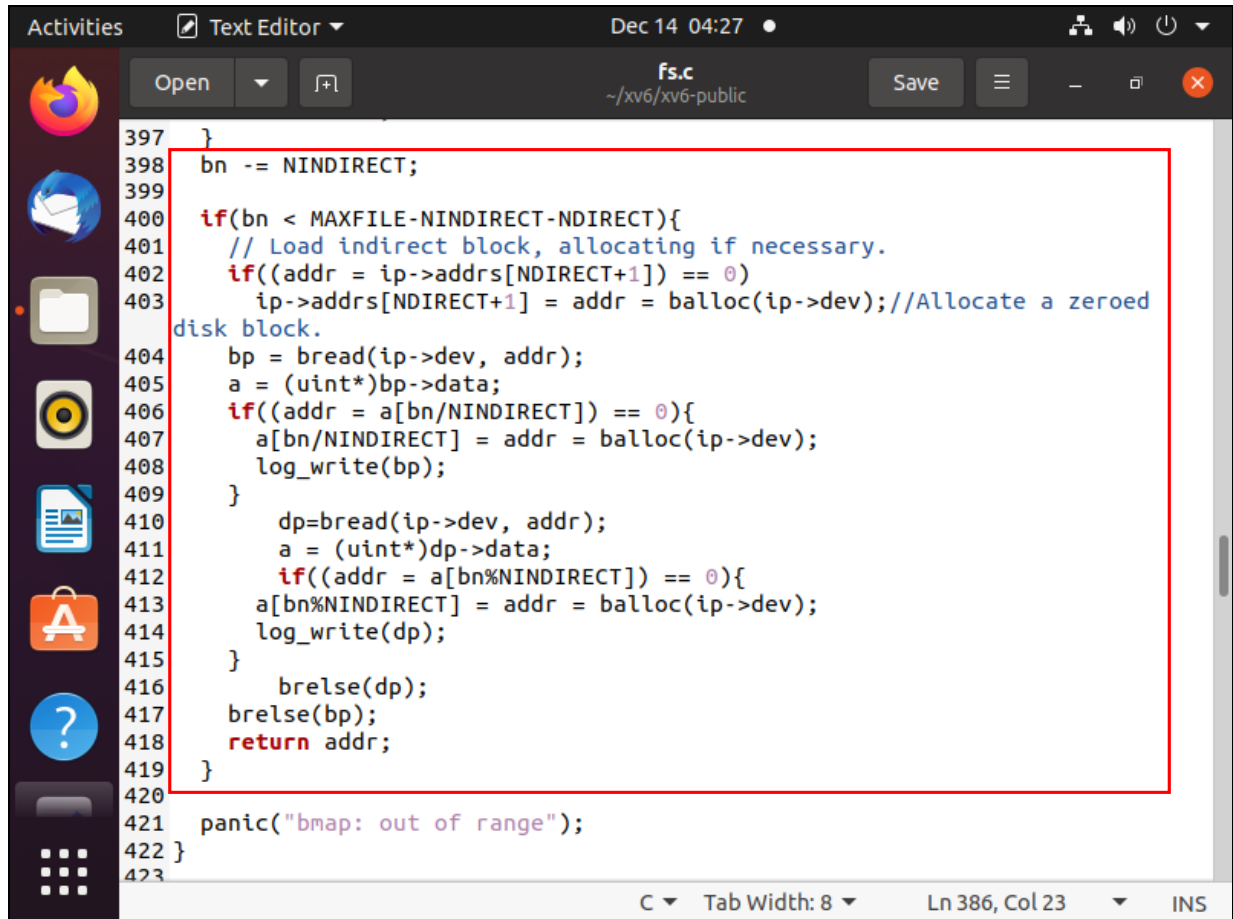
Figure 3: Define in fs.h file

2/We also have to change some code in `file.h` in struct inode section:



```
20  short type;             // copy of disk inode
21  short major;
22  short minor;
23  short nlink;
24  uint size;
25  uint addrs[NDIRECT+2];
26
```


3/Now we have to go to **fs.c** to add some codes there. We have to add this code in **to static unit bmap()** section.



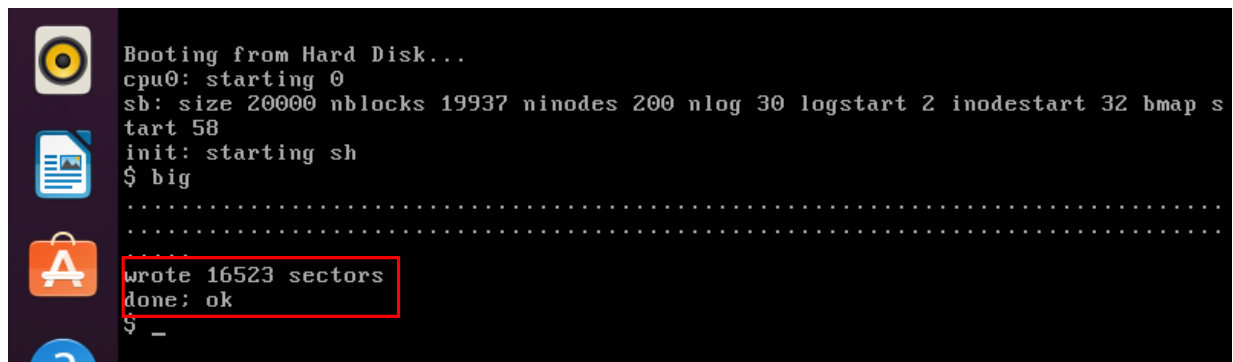
```

397 }
398 bn -= NINDIRECT;
399
400 if(bn < MAXFILE-NINDIRECT-NDIRECT){
401     // Load indirect block, allocating if necessary.
402     if((addr = ip->addrs[NDIRECT+1]) == 0)
403         ip->addrs[NDIRECT+1] = addr = balloc(ip->dev); //Allocate a zeroed
disk block.
404     bp = bread(ip->dev, addr);
405     a = (uint*)bp->data;
406     if((addr = a[bn/NINDIRECT]) == 0){
407         a[bn/NINDIRECT] = addr = balloc(ip->dev);
408         log_write(bp);
409     }
410     dp=bread(ip->dev, addr);
411     a = (uint*)dp->data;
412     if((addr = a[bn%NINDIRECT]) == 0){
413         a[bn%NINDIRECT] = addr = balloc(ip->dev);
414         log_write(dp);
415     }
416     brelse(dp);
417     brelse(bp);
418     return addr;
419 }
420
421 panic("bmap: out of range");
422 }
423

```

Figure 4:Add code in fs.c file

4/The last step we have to delete the **fs.img** file. Make sure you delete **fs.img** make on Unix (not xv6). If your file system gets into a bad state, perhaps by crashing, delete **fs.img** (do this from Unix, not xv6). **make** will build a new clean file system image for you. After that We have to make **gemu** in terminal and then write command **big** to see the result below:



```

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

```

Figure 5:Final answer

4. 实验结论与心得体会

This experiment is good and fun though because the hint and tip that they gave you is just too much and you can easily find the answer of the experiment just like in the first part we have to change CPU to 1 and then just follow the tip it'll be completed easily. And the hardest part is that we have to add some code in the fs.c file to 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. Finally, We got the correct answer and this experiment is done.