CSCI350 Project 5

Due: November 11 by 11:59pm

Objectives

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
```

Add

```
QEMUEXTRA = -snapshot
```

right before `QEMUOPTS`

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

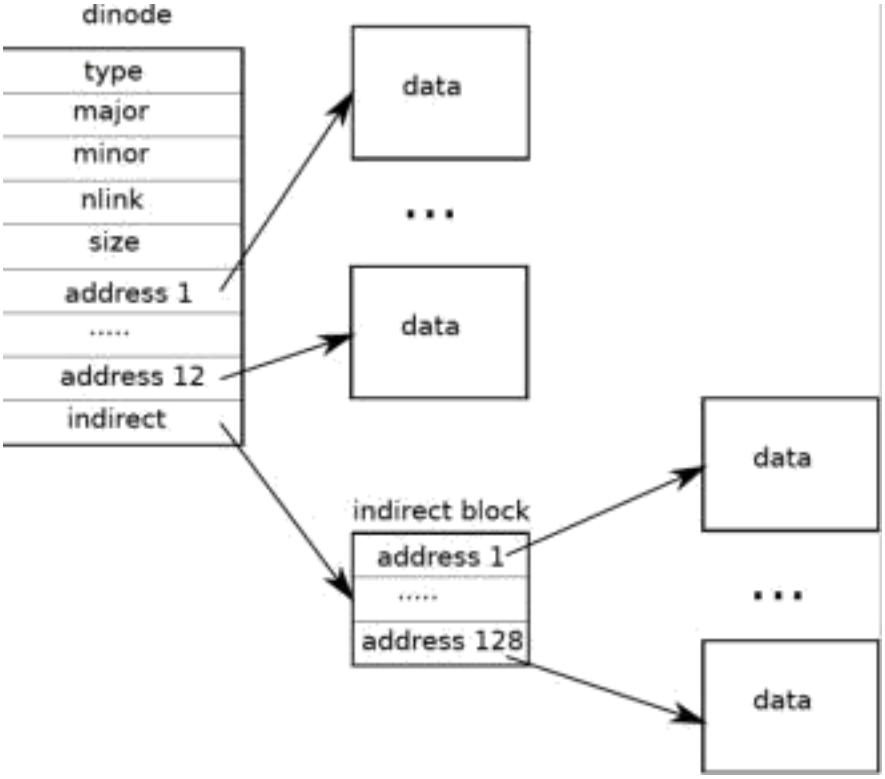
`mkfs` initializes the file system to have fewer than 1000 free data blocks, too few to show off the changes you'll make. Modify `param.h` to set `FSSIZE` to:

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

Download `testfs.c` (in Resources) into your xv6 directory, add it to the UPROGS list, start up xv6, and run `testfs`. 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`. This is 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 number" – a data 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 to disk block numbers.

Your Task

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, `testfs.c` will now report that it can write 16523 sectors. It will take `testfs.c` a few dozen seconds to finish.

Hints

Quick overview of some disk operation functions that may be relevant:

1. `balloc()` allocates a free on-disk block for use and returns the disk block number.

2. `bread()` returns an in-memory block pointer (pointer to a block buffer), given a disk block number. If the block is not already loaded from disk, it performs a disk load. Appropriate type-cast should be performed on the byte array `bp->data[]` in the block buffer in order to correctly read the block data.

3. `log_write()` given an in-memory block pointer, writes its content to its disk counterpart with logging enabled.

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.

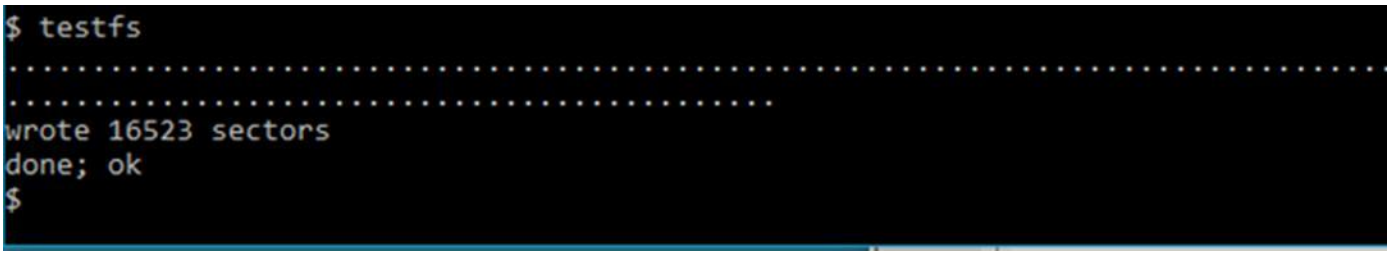
If you change the definition of `NDIRECT`, make sure to create a new `fs.img`, since `mkfs` uses `NDIRECT` too to build the initial file systems. If you delete `fs.img`, `make` on Unix (not xv6) will build a new one for you.

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()`.

Expected output:



Grading and Submission

The total number of the points for this project is 100.

Your report (see SubmissionIstructions document in Resources for details): 10 points.

- Describe your changes to the xv6 code
  - Name the files your modified and clearly describe the changes you made.
  - Please don't modify any files you don't need to! You'll make grading harder for us if you do.
- Your code must match the general code style of xv6

Correct implementation of the xv6 file system extension (correct output of `testfs.c`): 70 points.
Free Response Questions: 20 points.