Lab: file system

In this lab you will add large files and symbolic links to the xv6 file system.

Before writing code, you should read "Chapter 8: File system" from the xv6 book and study the corresponding code.

Fetch the xv6 source for the lab and check out the `util` branch:

```
$ git fetch
$ git checkout fs
$ make clean
```

Large files (moderate)

In this assignment you'll increase the maximum size of an xv6 file. Currently xv6 files are limited to 268 blocks, or 268*BSIZE bytes (BSIZE is 1024 in xv6). 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 256 more block numbers, for a total of 12+256=268 blocks.

The `bigfile` command creates the longest file it can, and reports that size:

```
$ bigfile
..
wrote 268 blocks
bigfile: file is too small
$
```

The test fails because `bigfile` expects to be able to create a file with 65803 blocks, but unmodified xv6 limits files to 268 blocks.

You'll change the xv6 file system code to support a "doubly-indirect" block in each inode, containing 256 addresses of singly-indirect blocks, each of which can contain up to 256 addresses of data blocks. The result will be that a file will be able to consist of up to 65803 blocks, or 256*256+256+11 blocks (11 instead of 12, because we will sacrifice one of the direct block numbers for the double-indirect block).

Preliminaries

The `mkfs` program creates the xv6 file system disk image and determines how many total blocks the file system has; this size is controlled by `FSSIZE` in `kernel/param.h`. You'll see that `FSSIZE` in the repository for this lab is set to 200,000 blocks. You should see the following output from `mkfs/mkfs` in the make output:

```
nmeta 70 (boot, super, log blocks 30 inode blocks 13, bitmap blocks 25) blocks 199930 total 200000
```

This line describes the file system that `mkfs/mkfs` built: it has 70 meta-data blocks (blocks used to describe the file system) and 199,930 data blocks, totaling 200,000 blocks.
If at any point during the lab you find yourself having to rebuild the file system from scratch, you can run `make clean` which forces make to rebuild fs.img.

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 at Figure 8.3 in the xv6 text 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 number" -- a block number within the file, 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 are done with this exercise when `bigfile` writes 65803 blocks and `usertests -q` runs successfully:

```
$ bigfile
..................................................................................................................................
wrote 65803 blocks
done; ok
$ usertests -q
...
ALL TESTS PASSED
$
```

`bigfile` will take at least a minute and a half to run.

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 256*256 blocks (really -1, 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 declaration 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.