Operating Systems Lab (XV6)

Assignment Week

Objective

In this week's exercise you will be implementing support for double indirection in inode structure. By default, xv6 inode has pointers to 12 direct block pointers and one indirect pointer, this pointer point to a block, which can hold 128 direct pointers.

Hence, currently xv6 can support 128 + 12 blocks to be referenced through an inode. Assuming each block corresponds to a sector, an inode can only refer to 140 sectors (approx 70 kB).

Your job is to make changes to the inode structure so that it can support one double indirection pointer. This pointer would point to a block of indirect pointers, i.e. 128 blocks of indirect pointers, each of which refers to a block of direct pointers thus referencing 128*128 sectors, by a single double indirect pointer.

Background

First you should get familiar with the structure of an inode, which can be found in xv6 references.

Also, analyse the code of bmap function in fs.c to figure out what it is doing and how it is doing it.

Download the big.c file provided along with this assignment and run it on xv6.

Before attempting to do these tasks, it would be worthwhile to make some modifications to Qemu so that writing sectors does not take much time.

Make the following changes to the Makefile :

```
CPUS := 1
```

and

add -snapshot to QEMUOPTS

The files that need to be modified are :

- mkfs.c
- param.h
- fs.h
- fs.c

Modify mkfs so that we can accomodate writing large files in xv6. Change the values to look like following :

```
int nblocks = 20985;
int nlog = LOGSIZE;
int ninodes = 200;
int size = 21029;
```

Change param.h so that FSSIZE equals 20000.

<u>fs.h :</u>

Change the value of NDIRECT to 11.
Add NDINDIRECT to represent ((NINDIRECT)*NINDIRECT)
Change MAXFILE to be able to accomodate NDIRECT+NINDIRECT+NDINDIRECT entries.

Also, you should change the size of addrs array in inode structure to accomodate NDIRECT+2 entries in total.

<u>fs.c</u>

We will be making changes to bmap function to look beyond the one indirect pointer to check if a double indirect entry exists, and if it doesn't then create one.

As you will see that this indirection is just like using double indexes to get to the actual data.

Add the following code to bmap function just before :

“panic("bmap: out of range");”

```
  bn -= NINDIRECT;

  if(bn < NDINDIRECT) {
        if((addr = ip->addrs[NDIRECT+1]) == 0) {
                ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);
  }
        bp = bread(ip->dev, addr);
        a = (uint *) bp->data;

        uint block_index = bn/NINDIRECT;
        if((addr = a[block_index]) == 0) {
                a[block_index] = addr = balloc(ip->dev);
        }
        brelse(bp);

        uint offset = bn%NINDIRECT;
        struct buf * bp2;

        bp2 = bread(ip->dev, addr);
        a = (uint *)bp2->data;

        if((addr = a[offset]) == 0) {
                a[offset] = addr = balloc(ip->dev);
        }
        brelse(bp2);
        return addr;
}
```

The above code calculates the actual address from the given block number by :

bn -= NINDIRECT;

If the address is in valid range, then we check whether there exists an entry in for a block of disk that contains indirect pointers, and create if necessary.

```
  if(bn < NDINDIRECT) {
        if((addr = ip->addrs[NDIRECT+1]) == 0) {
                ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);
  }
        bp = bread(ip->dev, addr);
        a = (uint *) bp->data;
```

Each block contains NINDIRECT entries, to find out which entry number is referenced by the provided block number (bn) we divide it accordingly, and see if it is allocated.

```
        uint block_index = bn/NINDIRECT;
        if((addr = a[block_index]) == 0) {
                a[block_index] = addr = balloc(ip->dev);
        }
        brelse(bp);
```

The block pointed by that entry wil contain the final address that this “bn” is tring to reference.
This is calculated by :

```
        uint offset = bn%NINDIRECT;
        struct buf * bp2;

        bp2 = bread(ip->dev, addr);
        a = (uint *)bp2->data;

        if((addr = a[offset]) == 0) {
                a[offset] = addr = balloc(ip->dev);
        }

        brelse(bp2);
```

Finally the address is returned.

Assignment

Write a report detailing all the steps performed above. Apart from that, you should also mention the need of the following variables/functions :

- bmap()
- addr[]
- brelse()
- bread()
- balloc()

References :

- https://www.khoury.northeastern.edu/~pjd/cs7680/homework/xv6-big-files.html
- https://github.com/aswinzz/XV6-OS/blob/master/Lab2/fs.c
- https://www.cs.columbia.edu/~junfeng/12sp-w4118/lectures/l23-fs-xv6.pdf