Lab6

20307130340 杨孟颖 计算机

一、实验内容

1、 建立一个二级间接块索引，扩大可以存储文件的空间。

2、 向 xv6 添加符号链接。符号链接（或软链接）通过路径名引用链接的文件，可以跨磁盘设备。

二、实验原理

1、 inode 数据结构定义了两种类型的文件节点：一种是储存在硬盘上的数据结构，包含了 inode 类型、inode 指向的文件/文件夹大小、一个数据 blockno 的列表；另一种是存储在内存中的数据结构，拥有 on-disk inode 的拷贝以及其他 kernel 需要的 metadata。on-disk inode 以 struct dinode 的形式定义：

```
struct dinode {
  short type;           // File type
  short major;          // Major device number (T_DEVICE only)
  short minor;          // Minor device number (T_DEVICE only)
  short nlink;          // Number of links to inode in file system
  uint size;            // Size of file (bytes)
  uint addrs[NDIRECT+1];   // Data block addresses
};
```

type 指定了是文件、文件夹还是设备，type==0 表示这个 inode 处于空闲状态。nlink 表示连接到这个 inode 的 directory entry 的个数，用来判断这个 inode 应该何时被释放。size 记录了这个文件/文件夹的大小，addrs 记录了这个 inode 拥有的文件内容分布在的 disk block 的所有 blockno

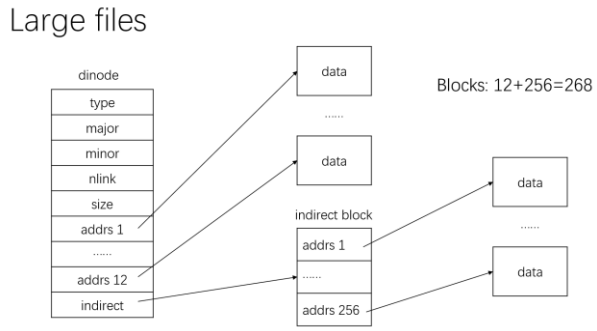
内存中的 inode 是 active inodes，用 struct inode 定义：

```
// in-memory copy of an inode
struct inode {
  uint dev;           // Device number
  uint inum;          // Inode number
  int ref;            // Reference count
  struct sleeplock lock; // protects everything below here
  int valid;          // inode has been read from disk?

  short type;         // copy of disk inode
  short major;
  short minor;
  short nlink;
  uint size;
  uint addrs[NDIRECT+1];
};
```

内存中有 C 指针指向了这个 inode，ref 是指向这个 inode 的指针数量，当 ref==0 时 kernel 将把这个 inode 从内存中剔除。iget 函数和 iput 函数实现对 inode pointer 的获取和释放。on-disk inode 结构 dinode 包括了一个 size 和 NDIRECT+1 个 addr，前 NDIRECT 个 addr 叫做 direct blocks，最后一个 addr

给出了 indirect block 的地址，因此一个文件的前 12kB（NDIRECT x BSIZE）可以从 inode 中的 direct block addr 直接读取，后 256kB（NINDIRECT x BSIZE）可以通过 indirect block addr 翻译得到。因此 xv6 支持的最大的文件大小为 268kB。



bmap()负责获取 inode 中的第 n 块 data block 的地址。当 bn<NDIRECT 时直接返回 ip->addrs[bn]，如果没有这个地址就调用 balloc 分配一个 data block。当 NDIRECT<bn<NINDIRECT 时先 bread(ip->dev, ip->addrs[NDIRECT])，然后获取 bp->data[bn-NDIRECT]。

在实现实验要求时，需要改变 inode 结构，使 inode 可以实现两级映射，并修改映射函数 bmap 和释放空间的函数 itrunc 使操作系统可以控制两级文件目录的地址。仿照 bmap() 中一级索引的查找方式写出二级索引的查找代码，bn / NINDIRECT 是第一层的编号，bn % NINDIRECT 是第二层的编号。

2、sys_link 传入一个参数 old 和一个参数 new，new 是需要链接到 old 的路径。sys_link 增加 ip->nlink，然后调用 nameiparent 查找到 new 的父文件夹，调用 dirlink 在父文件夹下创建一个名称为 new 的 directory entry，链接到 old 所代表的 inode。如果中间出现了错误，需要跳转到 bad 来减去 ip->nlink。参照 sys_link 来写 sys_sym_link。

sys_open 有两种模式，当 O_CREATEflag 置 1 时调用了 create，当置 0 时调用了 namei 来找到这个 inode，然后调用 filealloc 和 fdalloc 来分配 struct file 和 file descriptor。软链接本身也是一个文件，但是指向另一个文件。当打开一个软链接文件时，如果打开模式没有设置 O_NOFOLLOW，就会打开链接到的文件——如果链接到的文件也是一个软链接，则继续往下找。对于 sys_open()，当取到的文件时软链接并且没有设置 O_NOFOLLOW 时，就沿着链接一直往下找。和 writei() 对应的，用 readi() 读取文件的数据块即可得到链接地址。

三、实现过程

1、实现二级目录

修改 dinode 和 inode 的定义：

```
struct dinode {
  short type;           // File type
  short major;          // Major device number (T_DEVICE only)
  short minor;          // Minor device number (T_DEVICE only)
  short nlink;          // Number of links to inode in file system
  uint size;            // Size of file (bytes)
  uint addrs[NDIRECT+2];   // Data block addresses
```

```
};
struct inode {
  uint dev;           // Device number
  uint inum;          // Inode number
  int ref;            // Reference count
  struct sleeplock lock; // protects everything below here
  int valid;          // inode has been read from disk?

  short type;         // copy of disk inode
  short major;
  short minor;
  short nlink;
  uint size;
  uint addrs[NDIRECT+2];
};
```

Kernel.fs.c 中在 bmap 函数中加入二级目录中查找文件的映射的部分：

```
static uint
bmap(struct inode *ip, uint bn)
{
  uint addr, *a;
  struct buf *bp;

  if(bn < NDIRECT){
    if((addr = ip->addrs[bn]) == 0){
      addr = balloc(ip->dev);
      if(addr == 0)
        return 0;
      ip->addrs[bn] = addr;
    }
    return addr;
  }
  bn -= NDIRECT;

  if(bn < NINDIRECT){
    // Load indirect block, allocating if necessary.
    if((addr = ip->addrs[NDIRECT]) == 0){
      addr = balloc(ip->dev);
      if(addr == 0)
        return 0;
      ip->addrs[NDIRECT] = addr;
    }
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    if((addr = a[bn]) == 0){
```

```
      addr = balloc(ip->dev);
      if(addr){
        a[bn] = addr;
        log_write(bp);
      }
    }
    brelse(bp);
    return addr;
  }

  bn -= NINDIRECT;
  if(bn < NINDIRECT2){
      if((addr = ip->addrs[NDIRECT+1]) == 0)
      ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    if((addr = a[bn / NINDIRECT]) == 0){
      a[bn / NINDIRECT] = addr = balloc(ip->dev);
      log_write(bp);
    }
    brelse(bp);
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    if((addr = a[bn % NINDIRECT]) == 0){
      a[bn % NINDIRECT] = addr = balloc(ip->dev);
      log_write(bp);
    }
    brelse(bp);
    return addr;
  }

  panic("bmap: out of range");
}
```

Kernel.fs.c 中 itrunc 函数中加入释放二级目录中文件空间的部分。

```
void
itrunc(struct inode *ip)
{
  int i, j;
  struct buf *bp;
  uint *a;

  for(i = 0; i < NDIRECT; i++){
    if(ip->addrs[i]){
```

```
      bfree(ip->dev, ip->addrs[i]);
      ip->addrs[i] = 0;
    }
  }

  if(ip->addrs[NDIRECT]){
    bp = bread(ip->dev, ip->addrs[NDIRECT]);
    a = (uint*)bp->data;
    for(j = 0; j < NINDIRECT; j++){
      if(a[j])
        bfree(ip->dev, a[j]);
    }
    brelse(bp);
    bfree(ip->dev, ip->addrs[NDIRECT]);
    ip->addrs[NDIRECT] = 0;
  }
  if(ip->addrs[NDIRECT+1]){
    bp = bread(ip->dev, ip->addrs[NDIRECT+1]);
    a = (uint*)bp->data;
    for(j = 0; j < NINDIRECT; j++){
      if(a[j]){
        struct buf *bp2 = bread(ip->dev, a[j]);
        uint *a2 = (uint*)bp2->data;
        for(int k = 0; k < NINDIRECT; k++){
          if(a2[k])
            bfree(ip->dev, a2[k]);
        }
        brelse(bp2);
        bfree(ip->dev, a[j]);
        a[j] = 0;
      }
    }
    brelse(bp);
    bfree(ip->dev, ip->addrs[NDIRECT+1]);
    ip->addrs[NDIRECT+1] = 0;
  }

  ip->size = 0;
  iupdate(ip);
}
```

1、实现软连接

首先定义软连接模式 O_NOFOLLOW

Kernel/fcntl.h

```
#define O_NOFOLLOW 0x010
```

定义软链接类型

Kernel/stat.h

```
#define T_SYMLINK 4   // Symbolic link
```

参照 syslink 函数和软链接的特点实现 syslink 函数：

```
uint64
sys_symlink(void)
{
  char target[MAXPATH], path[MAXPATH];
  if(argstr(0, target, MAXPATH) < 0 || argstr(1, path, MAXPATH) < 0)
    return -1;

  begin_op();
  struct inode *ip;
  ip = create(path, T_SYMLINK, 0, 0);
  if(ip == 0){
    end_op();
    return -1;
  }
  if(writei(ip, 0, (uint64)target, 0, MAXPATH) != MAXPATH){
    end_op();
    return -1;
  }
  iupdate(ip);
  iunlockput(ip);
  end_op();
  return 0;
}
```

在 sysopen 中加入软链接形式的文件的查找：

```
uint64
sys_open(void)
{
  char path[MAXPATH];
  int fd, omode;
  struct file *f;
  struct inode *ip;
  int n;

  argint(1, &omode);
  if((n = argstr(0, path, MAXPATH)) < 0)
    return -1;

  begin_op();

  if(omode & O_CREATE){
```

```
  ip = create(path, T_FILE, 0, 0);
  if(ip == 0){
    end_op();
    return -1;
  }
} else {
  if((ip = namei(path)) == 0){
    end_op();
    return -1;
  }
  ilock(ip);
  if(ip->type == T_DIR && omode != O_RDONLY){
    iunlockput(ip);
    end_op();
    return -1;
  }
}

int cnt = 0;
while(ip->type == T_SYMLINK && !(omode & O_NOFOLLOW)){
  if((n = readi(ip, 0, (uint64)path, 0, MAXPATH)) != MAXPATH){
    iunlockput(ip);
    end_op();
    return -1;
  }
  iunlockput(ip);
  if((ip = namei(path)) == 0 || ++cnt > 10){
    end_op();
    return -1;
  }
  ilock(ip);
}

if(ip->type == T_DEVICE && (ip->major < 0 || ip->major >= NDEV)){
  iunlockput(ip);
  end_op();
  return -1;
}

if((f = filealloc()) == 0 || (fd = fdalloc(f)) < 0){
  if(f)
    fileclose(f);
  iunlockput(ip);
  end_op();
```

```
    return -1;
  }

  if(ip->type == T_DEVICE){
    f->type = FD_DEVICE;
    f->major = ip->major;
  } else {
    f->type = FD_INODE;
    f->off = 0;
  }
  f->ip = ip;
  f->readable = !(omode & O_WRONLY);
  f->writable = (omode & O_WRONLY) || (omode & O_RDWR);

  if((omode & O_TRUNC) && ip->type == T_FILE){
    itrunc(ip);
  }

  iunlock(ip);
  end_op();

  return fd;
}
```

实验结果：

```
$ bigfile
..........................................................
..........................................................
..........................................................
..........................................................
..........................................................
..........................................................
..........................................................
..........................................................
..........................................................
..........................................................
..........................................................
..................
wrote 65803 blocks
bigfile done; ok
```

```
$ symlinktest
Start: test symlinks
test symlinks: ok
Start: test concurrent symlinks
test concurrent symlinks: ok
```

```
== Test running bigfile ==
$ make qemu-gdb
running bigfile: OK (196.1s)
== Test running symlinktest ==
$ make qemu-gdb
(1.1s)
== Test   symlinktest: symlinks ==
  symlinktest: symlinks: OK
== Test   symlinktest: concurrent symlinks ==
  symlinktest: concurrent symlinks: OK
== Test usertests ==
$ make qemu-gdb
usertests: OK (289.5s)
== Test time ==
time: OK
```

四、遇到的问题与解决

1、定义的 syslink 函数在同一文档中的 create 函数之前，出现了函数未定义的问题。

```
kernel/sysfile.c: In function ‘sys_symlink’:
kernel/sysfile.c:63:12: error: implicit declaration of function ‘create’; did you mean ‘uvmcreate’? [-Werror=implicit-function-declaration]
   63 |   if((ip = create(path, T_SYMLINK, 0, 0)) == 0){
      |            ^~~~~~
      |            uvmcreate
kernel/sysfile.c:63:25: error: ‘T_SYMLINK’ undeclared (first use in this function)
   63 |   if((ip = create(path, T_SYMLINK, 0, 0)) == 0){
      |                         ^~~~~~~~~
kernel/sysfile.c:63:25: note: each undeclared identifier is reported only once for each function it appears in
kernel/sysfile.c: At top level:
kernel/sysfile.c:269:1: error: conflicting types for ‘create’; have ‘struct inode *(char *, short int,  short int,  short int)’
  269 | create(char *path, short type, short major, short minor)
      | ^~~~~~
kernel/sysfile.c:63:12: note: previous implicit declaration of ‘create’ with type ‘int()’
   63 |   if((ip = create(path, T_SYMLINK, 0, 0)) == 0){
      |            ^~~~~~
kernel/sysfile.c: In function ‘sys_open’:
kernel/sysfile.c:362:21: error: ‘T_SYMLINK’ undeclared (first use in this function)
  362 |   while(ip->type == T_SYMLINK && !(omode & O_NOFOLLOW)){
      |                     ^~~~~~~~~
kernel/sysfile.c:362:44: error: ‘O_NOFOLLOW’ undeclared (first use in this function)
  362 |   while(ip->type == T_SYMLINK && !(omode & O_NOFOLLOW)){
      |                                            ^~~~~~~~~~
cc1: all warnings being treated as errors
```

2、在实验过程中一直出现 BSIZE 大小不符合要求的问题，经检查发现没有按照 PPT 上的要求将 NDIRECT 的大小改为 11.改正后问题解决。

```
== Test running bigfile ==
$ make qemu-gdb
Failed to connect to QEMU; output:
mkfs: mkfs/mkfs.c:85: main: Assertion `(BSIZE % sizeof(struct dinode)) == 0' failed.
make[1]: *** [Makefile:265: fs.img] Aborted (core dumped)

Failed to shutdown QEMU.  You might need to 'killall qemu' or
'killall qemu.real'.

make: *** [Makefile:338: grade] Error 1
```

```
#define NDIRECT 11
#define NINDIRECT (BSIZE / sizeof(uint))
#define NINDIRECT2 (NINDIRECT*NINDIRECT)
#define MAXFILE (NDIRECT + NINDIRECT + NINDIRECT2)
```

五、总结

本次实验中我对 xv6 系统的文件管理的理解更加深刻了，了解了文件系统分配空间的方式和如何扩展。并且创建软链接使文件系统可以