File System

目录：

Large files

实验目的

实验目的：

> 这一个实验是要使XV6支持更大的文件。原始XV6中的文件块号`dinode.addr`是使用一个大小为12的直接块表以及一个大小为256的一级块表，即文件最大为`12+256`块。本次实验的目标为修改文件系统结构和分配磁盘块代码以支持更大的文件。以支持每个inode中可包含256个一级间接块地址的"二级间接"块，每个一级间接块最多可以包含256个数据块地址。结果将是一个文件将能够包含多达65803个块，或256*256+256+11个块（11而不是12，因为我们将为二级间接块牺牲一个直接块号）

实验提示：

> - 确保您理解`bmap()`。写出`ip->addrs[]`、间接块、二级间接块和它所指向的一级间接块以及数据块之间的关系图。确保您理解为什么添加二级间接块会将最大文件大小增加256*256个块（实际上要-1，因为您必须将直接块的数量减少一个）。
> - 考虑如何使用逻辑块号索引二级间接块及其指向的间接块。
> - 如果更改`NDIRECT`的定义，则可能必须更改**file.h**文件中`struct inode`中`addrs[]`的声明。确保`struct inode`和`struct dinode`在其`addrs[]`数组中具有相同数量的元素。
> - 如果更改`NDIRECT`的定义，请确保创建一个新的**fs.img**，因为`mkfs`使用`NDIRECT`构建文件系统。
> - 如果您的文件系统进入坏状态，可能是由于崩溃，请删除**fs.img**（从Unix而不是xv6执行此操作）。`make`将为您构建一个新的干净文件系统映像。
> - 别忘了把你`bread()`的每一个块都`brelse()`。
> - 您应该仅根据需要分配间接块和二级间接块，就像原始的`bmap()`。
> - 确保`itrunc`释放文件的所有块，包括二级间接块。

实验过程

1. 首先我们在`fs.h`中添加相应的宏定义

```
/*
 *修改NDIRECT为11，前11块为数据磁盘块
 *新增二级目录总大小NNINDIRECT
 *修改最大文件大小
 *addrs[]大小改为 NDIRECT+2，仍是13
 */
#define NDIRECT 11
#define NINDIRECT (BSIZE / sizeof(uint))
#define NDINDIRECT ((BSIZE / sizeof(uint)) * (BSIZE / sizeof(uint)))
#define MAXFILE (NDIRECT + NINDIRECT + NDINDIRECT)
#define NADDR_PER_BLOCK (BSIZE / sizeof(uint))  // 一个块中的地址数量
```

2. 然后由于`NDIRECT`定义改变，其中一个直接块变为了二级间接块，所以我们需要修改`inode`和`dnode`中结构体中`addrs`元素数量

```
//fs.h
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,改变addr的大小。
};

//file.h
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];//改变addr的大小
};
```

3. 然后我们修改`bmap`，使其支持二级索引（其实就是重复一次块表的查询过程）。

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

  /*
  当块号小于NDIRECT时我们直接获取地址然后返回
```

```
  如果地址为则说明未分配磁盘块，则分配一个
  */
  if (bn < NDIRECT)
  {
    ....
  }
  bn -= NDIRECT;
  /*
  当NDIRECT <= bn < NINDIRECT先从对应的磁盘块读取目录检查一级目录是否存在/分配磁盘块在一级
目录中寻找
  存在返回，不存在分配并用log_wirte()写调用了bread(),需要调用brelease()
  */
  if(bn < NINDIRECT){
    .....
    }

    bn -= NINDIRECT;

    // 二级间接块的情况
    if (bn < NDINDIRECT)
    {
      int level2_idx = bn / NADDR_PER_BLOCK; // 要查找的块号位于二级间接块中的位置
      int level1_idx = bn % NADDR_PER_BLOCK; // 要查找的块号位于一级间接块中的位置
      // 读出二级间接块
      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[level2_idx]) == 0)
      {
        a[level2_idx] = addr = balloc(ip->dev);
        // 更改了当前块的内容，标记以供后续写回磁盘
        log_write(bp);
      }
      brelse(bp);

      bp = bread(ip->dev, addr);
      a = (uint *)bp->data;
      if ((addr = a[level1_idx]) == 0)
      {
        a[level1_idx] = addr = balloc(ip->dev);
        log_write(bp);
      }
      brelse(bp);
      return addr;
    }

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

4. 最后我们仿照一级块表的处理修改 `itrunc` 函数，使其能够释放二级块表对应的块。

```
void
itrunc(struct inode *ip)
{
  int i, j;
```

```
  struct buf *bp;
  uint *a;

  for(i = 0; i < NDIRECT; i++){
    .....
  }

  if(ip->addrs[NDIRECT]){
   .....
  }

  struct buf *bp1;
  uint *a1;
  if (ip->addrs[NDIRECT + 1])
  {
    bp = bread(ip->dev, ip->addrs[NDIRECT + 1]);
    a = (uint *)bp->data;
    for (i = 0; i < NADDR_PER_BLOCK; i++)
    {
      // 每个一级间接块的操作都类似于上面的
      // if(ip->addrs[NDIRECT])中的内容
      if (a[i])
      {
        bp1 = bread(ip->dev, a[i]);
        a1 = (uint *)bp1->data;
        for (j = 0; j < NADDR_PER_BLOCK; j++)
        {
          if (a1[j])
            bfree(ip->dev, a1[j]);
        }
        brelse(bp1);
        bfree(ip->dev, a[i]);
      }
    }
    brelse(bp);
    bfree(ip->dev, ip->addrs[NDIRECT + 1]);
    ip->addrs[NDIRECT + 1] = 0;
  }

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

实验结果

```
xv6 kernel is booting

init: starting sh
$ bigfile
```

```
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
...................
wrote 65803 blocks
bigfile done; ok
$ usertests
usertests starting
test manywrites:
OK
test execout: OK
test copyin: OK
test copyout: OK
test copyinstr1: OK
test copyinstr2: OK
test copyinstr3: OK
test rwsbrk: OK
test truncate1: OK
test truncate2: OK
test truncate3: OK
test reparent2: OK
test pgbug: OK
test sbrkbugs: usertrap(): unexpected scause 0x000000000000000c pid=3242
            sepc=0x00000000000056a6 stval=0x00000000000056a6
usertrap(): unexpected scause 0x000000000000000c pid=3243
            sepc=0x00000000000056a6 stval=0x00000000000056a6
OK
test badarg: OK
test reparent: OK
test twochildren: OK
test forkfork: OK
test forkforkfork: OK
test argptest: OK
test createdelete: OK
test linkunlink: OK
test linktest: OK
test unlinkread: OK
test concreate: OK
test subdir: OK
test fourfiles: OK
test sharedfd: OK
test dirtest: OK
test exectest: OK
test bigargtest: OK
test bigwrite: OK
test bsstest: OK
test sbrkbasic: OK
test sbrkmuch: OK
test kernmem: usertrap(): unexpected scause 0x000000000000000d pid=6169
            sepc=0x000000000000215e stval=0x0000000080000000
usertrap(): unexpected scause 0x000000000000000d pid=6170
            sepc=0x000000000000215e stval=0x000000008000c350
usertrap(): unexpected scause 0x000000000000000d pid=6171
```

```
            sepc=0x000000000000215e stval=0x00000000800186a0
usertrap(): unexpected scause 0x000000000000000d pid=6172
            sepc=0x000000000000215e stval=0x00000000800249f0
usertrap(): unexpected scause 0x000000000000000d pid=6173
            sepc=0x000000000000215e stval=0x0000000080030d40
usertrap(): unexpected scause 0x000000000000000d pid=6174
            sepc=0x000000000000215e stval=0x0000000080003d090
usertrap(): unexpected scause 0x000000000000000d pid=6175
            sepc=0x000000000000215e stval=0x00000000800493e0
usertrap(): unexpected scause 0x000000000000000d pid=6176
            sepc=0x000000000000215e stval=0x0000000080055730
usertrap(): unexpected scause 0x000000000000000d pid=6177
            sepc=0x000000000000215e stval=0x0000000080061a80
usertrap(): unexpected scause 0x000000000000000d pid=6178
            sepc=0x000000000000215e stval=0x0000000080006ddd0
usertrap(): unexpected scause 0x000000000000000d pid=6179
            sepc=0x000000000000215e stval=0x0000000080007a120
usertrap(): unexpected scause 0x000000000000000d pid=6180
            sepc=0x000000000000215e stval=0x0000000080086470
usertrap(): unexpected scause 0x000000000000000d pid=6181
            sepc=0x000000000000215e stval=0x00000000800927c0
usertrap(): unexpected scause 0x000000000000000d pid=6182
            sepc=0x000000000000215e stval=0x0000000080009eb10
usertrap(): unexpected scause 0x000000000000000d pid=6183
            sepc=0x000000000000215e stval=0x00000000800aae60
usertrap(): unexpected scause 0x000000000000000d pid=6184
            sepc=0x000000000000215e stval=0x00000000800b71b0
usertrap(): unexpected scause 0x000000000000000d pid=6185
            sepc=0x000000000000215e stval=0x00000000800c3500
usertrap(): unexpected scause 0x000000000000000d pid=6186
            sepc=0x000000000000215e stval=0x00000000800cf850
usertrap(): unexpected scause 0x000000000000000d pid=6187
            sepc=0x000000000000215e stval=0x00000000800dbba0
usertrap(): unexpected scause 0x000000000000000d pid=6188
            sepc=0x000000000000215e stval=0x00000000800e7ef0
usertrap(): unexpected scause 0x000000000000000d pid=6189
            sepc=0x000000000000215e stval=0x00000000800f4240
usertrap(): unexpected scause 0x000000000000000d pid=6190
            sepc=0x000000000000215e stval=0x0000000080100590
usertrap(): unexpected scause 0x000000000000000d pid=6191
            sepc=0x000000000000215e stval=0x0000000080010c8e0
usertrap(): unexpected scause 0x000000000000000d pid=6192
            sepc=0x000000000000215e stval=0x0000000080118c30
usertrap(): unexpected scause 0x000000000000000d pid=6193
            sepc=0x000000000000215e stval=0x0000000080124f80
usertrap(): unexpected scause 0x000000000000000d pid=6194
            sepc=0x000000000000215e stval=0x00000000801312d0
usertrap(): unexpected scause 0x000000000000000d pid=6195
            sepc=0x000000000000215e stval=0x0000000080013d620
usertrap(): unexpected scause 0x000000000000000d pid=6196
            sepc=0x000000000000215e stval=0x0000000080149970
usertrap(): unexpected scause 0x000000000000000d pid=6197
            sepc=0x000000000000215e stval=0x0000000080155cc0
usertrap(): unexpected scause 0x000000000000000d pid=6198
            sepc=0x000000000000215e stval=0x0000000080162010
usertrap(): unexpected scause 0x000000000000000d pid=6199
            sepc=0x000000000000215e stval=0x0000000080016e360
usertrap(): unexpected scause 0x000000000000000d pid=6200
```

```
            sepc=0x000000000000215e stval=0x000000008017a6b0
usertrap(): unexpected scause 0x000000000000000d pid=6201
            sepc=0x000000000000215e stval=0x0000000080186a00
usertrap(): unexpected scause 0x000000000000000d pid=6202
            sepc=0x000000000000215e stval=0x0000000080192d50
usertrap(): unexpected scause 0x000000000000000d pid=6203
            sepc=0x000000000000215e stval=0x000000008019f0a0
usertrap(): unexpected scause 0x000000000000000d pid=6204
            sepc=0x000000000000215e stval=0x00000000801ab3f0
usertrap(): unexpected scause 0x000000000000000d pid=6205
            sepc=0x000000000000215e stval=0x00000000801b7740
usertrap(): unexpected scause 0x000000000000000d pid=6206
            sepc=0x000000000000215e stval=0x00000000801c3a90
usertrap(): unexpected scause 0x000000000000000d pid=6207
            sepc=0x000000000000215e stval=0x00000000801cfde0
usertrap(): unexpected scause 0x000000000000000d pid=6208
            sepc=0x000000000000215e stval=0x00000000801dc130
OK
test sbrkfail: usertrap(): unexpected scause 0x000000000000000d pid=6216
            sepc=0x00000000000041fe stval=0x0000000000012000
OK
test sbrkarg: OK
test validatetest: OK
test stacktest: usertrap(): unexpected scause 0x000000000000000d pid=6220
            sepc=0x00000000000022ce stval=0x0000000000000fb90
OK
test opentest: OK
test writetest: OK
test writebig: OK
test createtest: OK
test openiput: OK
test exitiput: OK
test iput: OK
test mem: OK
test pipe1: OK
test preempt: kill... wait... OK
test exitwait: OK
test rmdot: OK
test fourteen: OK
test bigfile: OK
test dirfile: OK
test iref: OK
test forktest: OK
test bigdir: OK
ALL TESTS PASSED
$
```

Symbolic links

实验目的

实验目的：

> 这一个实验是要实现系统调用 *symlink()*符号链接，即软链接功能。软链接就是在文件中保存指向文件的路径名，在打开文件的时候根据保存的路径名再去查找实际文件。与软链接相反的就是硬链接，硬链接是将文件的`inode`号指向目标文件的`inode`，并将引用计数加一。

实验提示：

> - 首先，为`symlink`创建一个新的系统调用号，在**user/usys.pl**、**user/user.h**中添加一个条目，并在**kernel/sysfile.c**中实现一个空的`sys_symlink`。
> - 向**kernel/stat.h**添加新的文件类型（`T_SYMLINK`）以表示符号链接。
> - 在**kernel/fcntl.h**中添加一个新标志（`O_NOFOLLOW`），该标志可用于`open`系统调用。请注意，传递给`open`的标志使用按位或运算符组合，因此新标志不应与任何现有标志重叠。一旦将**user/symlinktest.c**添加到**Makefile**中，您就可以编译它。
> - 实现`symlink(target, path)`系统调用，以在`path`处创建一个新的指向`target`的符号链接。请注意，系统调用的成功不需要`target`已经存在。您需要选择存储符号链接目标路径的位置，例如在`inode`的数据块中。`symlink`应返回一个表示成功（0）或失败（-1）的整数，类似于`link`和`unlink`。
> - 修改`open`系统调用以处理路径指向符号链接的情况。如果文件不存在，则打开必须失败。当进程向`open`传递`O_NOFOLLOW`标志时，`open`应打开符号链接（而不是跟随符号链接）。
> - 如果链接文件也是符号链接，则必须递归地跟随它，直到到达非链接文件为止。如果链接形成循环，则必须返回错误代码。你可以通过以下方式估算存在循环：通过在链接深度达到某个阈值（例如10）时返回错误代码。
> - 其他系统调用（如`link`和`unlink`）不得跟随符号链接；这些系统调用对符号链接本身进行操作。
> - 您不必处理指向此实验的目录的符号链接。

实验过程

1. 首先我们需要添加一些系统调用的相关配置操作
    1. 先修改`Makefile`
    2. 修改`user/usys.pl`和`user/user.h`，增加一个system call

```
//Makefile
UPROGS=\
    $U/_symlinktest\

//user/usys.pl
entry("symlink");

//syscall.h
#define SYS_symlink 22

//syscall.c
extern uint64 sys_symlink(void);
static uint64 (*syscalls[])(void) = {
    .......
    [SYS_symlink] sys_symlink,
};
```

2. 添加提示中的相关定义，`T_SYMLINK`以及`O_NOFOLLOW`

```
// fcntl.h
#define O_NOFOLLOW 0x004
// stat.h
#define T_SYMLINK 4
```

3. 在**kernel/sysfile.c**中实现`sys_symlink`

```
//添加新的sys_symlink调用
uint64
sys_symlink(void) {
  char target[MAXPATH], path[MAXPATH];
  struct inode* ip_path;

  if(argstr(0, target, MAXPATH) < 0 || argstr(1, path, MAXPATH) < 0) {
    return -1;
  }

  begin_op();
  // 分配一个inode结点，create返回锁定的inode
  ip_path = create(path, T_SYMLINK, 0, 0);
  if(ip_path == 0) {
    end_op();
    return -1;
  }
  // 向inode数据块中写入target路径
  if(writei(ip_path, 0, (uint64)target, 0, MAXPATH) < MAXPATH) {
    iunlockput(ip_path);
    end_op();
    return -1;
  }

  iunlockput(ip_path);
  end_op();
  return 0;
}
```

4. 最后在`sys_open`中添加对符号链接的处理就行了，当模式不是`O_NOFOLLOW`的时候就对符号链接进行循环处理，直到找到真正的文件，如果循环超过了一定的次数（10次），就说明可能发生了循环链接，就返回-1。但是要注意`namei`函数不会对`ip`上锁，需要使用`ilock`来上锁，而`create`则会上锁。

```
uint64
sys_open(void)
{
  ...

  if(ip->type == T_DEVICE && (ip->major < 0 || ip->major >= NDEV)){
    ...
  }

  // 处理符号链接
  if(ip->type == T_SYMLINK && !(omode & O_NOFOLLOW)) {
    // 若符号链接指向的仍然是符号链接，则递归的跟随它
    // 直到找到真正指向的文件
    // 但深度不能超过MAX_SYMLINK_DEPTH
```

```
    for(int i = 0; i < MAX_SYMLINK_DEPTH; ++i) {
      // 读出符号链接指向的路径
      if(readi(ip, 0, (uint64)path, 0, MAXPATH) != MAXPATH) {
        iunlockput(ip);
        end_op();
        return -1;
      }
      iunlockput(ip);
      ip = namei(path);
      if(ip == 0) {
        end_op();
        return -1;
      }
      ilock(ip);
      if(ip->type != T_SYMLINK)
        break;
    }
    // 超过最大允许深度后仍然为符号链接，则返回错误
    if(ip->type == T_SYMLINK) {
      iunlockput(ip);
      end_op();
      return -1;
    }
  }

  if((f = filealloc()) == 0 || (fd = fdalloc(f)) < 0){
    ...
  }

  ...
  return fd;
}
```

实验结果

```
$ symlinktest
Start: test symlinks
test symlinks: ok
Start: test concurrent symlinks
test concurrent symlinks: ok
$ usertests
usertests starting
test manywrites: OK
test execout: OK
test copyin: OK
test copyout: OK
test copyinstr1: OK
test copyinstr2: OK
test copyinstr3: OK
test rwsbrk: OK
test truncate1: OK
test truncate2: OK
test truncate3: OK
test reparent2: OK
test pgbug: OK
```

```
test sbrkbugs: usertrap(): unexpected scause 0x000000000000000c pid=9594
            sepc=0x00000000000056a6 stval=0x00000000000056a6
usertrap(): unexpected scause 0x000000000000000c pid=9595
            sepc=0x00000000000056a6 stval=0x00000000000056a6
OK
test badarg: OK
test reparent: OK
test twochildren: OK
test forkfork: OK
test forkforkfork: OK
test argptest: OK
test createdelete: OK
test linkunlink: OK
test linktest: OK
test unlinkread: OK
test concreate: OK
test subdir: OK
test fourfiles: OK
test sharedfd: OK
test dirtest: OK
test exectest: OK
test bigargtest: OK
test bigwrite: OK
test bsstest: OK
test sbrkbasic: OK
test sbrkmuch: OK
test kernmem: usertrap(): unexpected scause 0x000000000000000d pid=12521
            sepc=0x000000000000215e stval=0x0000000080000000
usertrap(): unexpected scause 0x000000000000000d pid=12522
            sepc=0x000000000000215e stval=0x000000008000c350
usertrap(): unexpected scause 0x000000000000000d pid=12523
            sepc=0x000000000000215e stval=0x00000000800186a0
usertrap(): unexpected scause 0x000000000000000d pid=12524
            sepc=0x000000000000215e stval=0x00000000800249f0
usertrap(): unexpected scause 0x000000000000000d pid=12525
            sepc=0x000000000000215e stval=0x0000000080030d40
usertrap(): unexpected scause 0x000000000000000d pid=12526
            sepc=0x000000000000215e stval=0x000000008003d090
usertrap(): unexpected scause 0x000000000000000d pid=12527
            sepc=0x000000000000215e stval=0x00000000800493e0
usertrap(): unexpected scause 0x000000000000000d pid=12528
            sepc=0x000000000000215e stval=0x0000000080055730
usertrap(): unexpected scause 0x000000000000000d pid=12529
            sepc=0x000000000000215e stval=0x0000000080061a80
usertrap(): unexpected scause 0x000000000000000d pid=12530
            sepc=0x000000000000215e stval=0x000000008006ddd0
usertrap(): unexpected scause 0x000000000000000d pid=12531
            sepc=0x000000000000215e stval=0x000000008007a120
usertrap(): unexpected scause 0x000000000000000d pid=12532
            sepc=0x000000000000215e stval=0x0000000080086470
usertrap(): unexpected scause 0x000000000000000d pid=12533
            sepc=0x000000000000215e stval=0x00000000800927c0
usertrap(): unexpected scause 0x000000000000000d pid=12534
            sepc=0x000000000000215e stval=0x000000008009eb10
usertrap(): unexpected scause 0x000000000000000d pid=12535
            sepc=0x000000000000215e stval=0x00000000800aae60
usertrap(): unexpected scause 0x000000000000000d pid=12536
            sepc=0x000000000000215e stval=0x00000000800b71b0
```

```
usertrap(): unexpected scause 0x000000000000000d pid=12537
            sepc=0x000000000000215e stval=0x00000000800c3500
usertrap(): unexpected scause 0x000000000000000d pid=12538
            sepc=0x000000000000215e stval=0x00000000800cf850
usertrap(): unexpected scause 0x000000000000000d pid=12539
            sepc=0x000000000000215e stval=0x00000000800dbba0
usertrap(): unexpected scause 0x000000000000000d pid=12540
            sepc=0x000000000000215e stval=0x00000000800e7ef0
usertrap(): unexpected scause 0x000000000000000d pid=12541
            sepc=0x000000000000215e stval=0x00000000800f4240
usertrap(): unexpected scause 0x000000000000000d pid=12542
            sepc=0x000000000000215e stval=0x0000000080100590
usertrap(): unexpected scause 0x000000000000000d pid=12543
            sepc=0x000000000000215e stval=0x000000008010c8e0
usertrap(): unexpected scause 0x000000000000000d pid=12544
            sepc=0x000000000000215e stval=0x0000000080118c30
usertrap(): unexpected scause 0x000000000000000d pid=12545
            sepc=0x000000000000215e stval=0x0000000080124f80
usertrap(): unexpected scause 0x000000000000000d pid=12546
            sepc=0x000000000000215e stval=0x00000000801312d0
usertrap(): unexpected scause 0x000000000000000d pid=12547
            sepc=0x000000000000215e stval=0x000000008013d620
usertrap(): unexpected scause 0x000000000000000d pid=12548
            sepc=0x000000000000215e stval=0x0000000080149970
usertrap(): unexpected scause 0x000000000000000d pid=12549
            sepc=0x000000000000215e stval=0x0000000080155cc0
usertrap(): unexpected scause 0x000000000000000d pid=12550
            sepc=0x000000000000215e stval=0x0000000080162010
usertrap(): unexpected scause 0x000000000000000d pid=12551
            sepc=0x000000000000215e stval=0x000000008016e360
usertrap(): unexpected scause 0x000000000000000d pid=12552
            sepc=0x000000000000215e stval=0x000000008017a6b0
usertrap(): unexpected scause 0x000000000000000d pid=12553
            sepc=0x000000000000215e stval=0x0000000080186a00
usertrap(): unexpected scause 0x000000000000000d pid=12554
            sepc=0x000000000000215e stval=0x0000000080192d50
usertrap(): unexpected scause 0x000000000000000d pid=12555
            sepc=0x000000000000215e stval=0x000000008019f0a0
usertrap(): unexpected scause 0x000000000000000d pid=12556
            sepc=0x000000000000215e stval=0x00000000801ab3f0
usertrap(): unexpected scause 0x000000000000000d pid=12557
            sepc=0x000000000000215e stval=0x00000000801b7740
usertrap(): unexpected scause 0x000000000000000d pid=12558
            sepc=0x000000000000215e stval=0x00000000801c3a90
usertrap(): unexpected scause 0x000000000000000d pid=12559
            sepc=0x000000000000215e stval=0x00000000801cfde0
usertrap(): unexpected scause 0x000000000000000d pid=12560
            sepc=0x000000000000215e stval=0x00000000801dc130
OK
test sbrkfail: usertrap(): unexpected scause 0x000000000000000d pid=12568
            sepc=0x00000000000041fe stval=0x0000000000012000
OK
test sbrkarg: OK
test validatetest: OK
test stacktest: usertrap(): unexpected scause 0x000000000000000d pid=12572
            sepc=0x00000000000022ce stval=0x000000000000fb90
OK
test opentest: OK
```

```
test writetest: OK
test writebig: OK
test createtest: OK
test openiput: OK
test exitiput: OK
test iput: OK
test mem: OK
test pipe1: OK
test preempt: kill... wait... OK
test exitwait: OK
test rmdot: OK
test fourteen: OK
test bigfile: OK
test dirfile: OK
test iref: OK
test forktest: OK
test bigdir: OK
ALL TESTS PASSED
```

最终结果

```
== Test running bigfile ==
$ make qemu-gdb
running bigfile: OK (326.9s)
== Test running symlinktest ==
$ make qemu-gdb
(1.5s)
== Test   symlinktest: symlinks ==
  symlinktest: symlinks: OK
== Test   symlinktest: concurrent symlinks ==
  symlinktest: concurrent symlinks: OK
== Test usertests ==
$ make qemu-gdb
usertests: OK (457.0s)
    (Old xv6.out.usertests failure log removed)
== Test time ==
time: OK
Score: 100/100
```

```
== Test running bigfile ==
$ make qemu-gdb
running bigfile: OK (326.9s)
== Test running symlinktest ==
$ make qemu-gdb
(1.5s)
== Test   symlinktest: symlinks ==
  symlinktest: symlinks: OK
== Test   symlinktest: concurrent symlinks ==
  symlinktest: concurrent symlinks: OK
== Test usertests ==
$ make qemu-gdb
usertests: OK (457.0s)
    (Old xv6.out.usertests failure log removed)
== Test time ==
time: OK
Score: 100/100
```

实验感悟

经过这次实验，让我对Xv6的文件系统有了一个更清晰的认识。xv6 文件系统的实现分为七层，如下图所示。磁盘层在 virtio 磁盘上读写块。Buffer 缓存层缓存磁盘块，并同步访问它们，确保一个块只能同时被内核中的一个进程访问。 日志层允许上层通过事务更新多个磁盘块，并确保在崩溃时，磁盘块是原子更新的（即 全部更新或不更新）。inode 层将一个文件都表示为一个 inode，每个文件包含一个唯一 的 i-number 和一些存放文件数据的块。目录层将实现了一种特殊的 inode，被称为目 录，其包含一个目录项序列，每个目录项由文件名称和 i-number 组成。路径名层提供 65 了层次化的路径名，如/usr/rtm/xv6/fs.c，可以用递归查找解析他们。文件描述符层用文件系统接口抽象了许多 Unix 资源（如管道、设备、文件等），使得程序员的生产力得到 大大的提高。

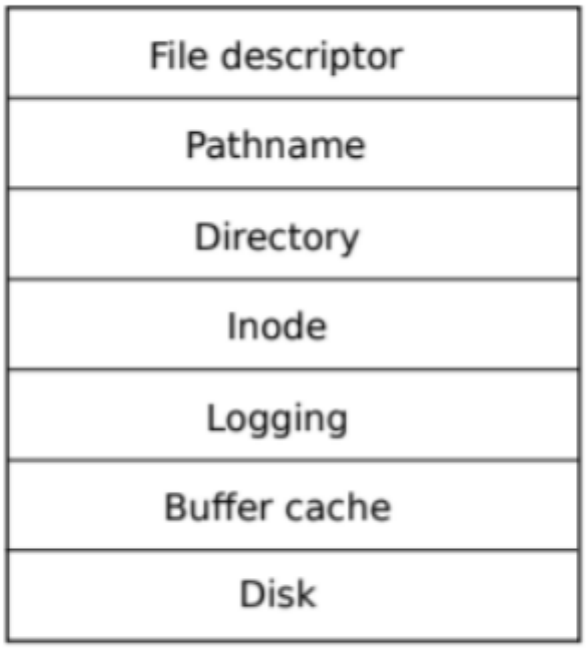

Figure 8.1: Layers of the xv6 file system.

通过第二个实验让我对软链接和硬链接部分也有了初步的了解，不仅学会了在Ubantu上创建链接，通过对代码的阅读也弄清楚了软链接和硬连接的原理。软链接就是在文件中保存指向文件的路径名，在打开文件的时候根据保存的路径名再去查找实际文件。与软链接相反的就是硬链接，硬链接是将文件的`inode`号指向目标文件的`inode`，并将引用计数加一。有很大的收获。