实例分析 第四部分

小组：田雅茹 杜江霖 刘艺颖 杨涵显

Q1：请逐行介绍sys_mkdir, sys_chdir和sys_mknod函数的功能。

```
sys_mkdir(void)
{
  char *path;
  struct inode *ip;

  begin_op();
  if(argstr(0, &path) < 0 ||
     (ip = create(path, T_DIR, 0, 0)) == 0){
    end_op();
    return -1;
  }
  iunlockput(ip);
  end_op();
  return 0;
}
```

- 功能：创建一个文件目录
- argstr(n,addr) 功能是把第n个系统调用的参数读进addr地址里面
- argstr<0说明当前系统调用的参数范围小于n，也就是说没传够n个参数，所以也就相当于报错
- 此处调用Create函数表示在path这个路径里创建一个目录，并使ip指向创建的目录的inode，如果创建失败会返回0，此时退出，否则释放缓存并退出

Q1：请逐行介绍sys_mkdir, sys_chdir和sys_mknod函数的功能。

```
sys_chdir(void)
{
  char *path;
  struct inode *ip;
  struct proc *curproc = myproc();

  begin_op();
  if(argstr(0, &path) < 0 || (ip = namei(path)) == 0){
    end_op();
    return -1;
  }
  ilock(ip);
  if(ip->type != T_DIR){
    iunlockput(ip);
    end_op();
    return -1;
  }
  iunlock(ip);
  iput(curproc->cwd);
  end_op();
  curproc->cwd = ip;
  return 0;
}
```

- 功能：将当前的工作目录改变成以参数路径所指的目录
- 先检查有没有把新目录赋值给path
- 然后用namei检查有没有这个目录并找到该path目录
- 检查是不是目录
- 然后把curproc的cwd(进程当前工作的目录)改成刚刚namei找到的inode

Q1：请逐行介绍sys_mkdir, sys_chdir和sys_mknod函数的功能。

```
sys_mknod(void)
{
  struct inode *ip;
  char *path;
  int major, minor;

  begin_op();
  if((argstr(0, &path)) < 0 ||
     argint(1, &major) < 0 ||
     argint(2, &minor) < 0 ||
     (ip = create(path, T_DEV, major, minor)) == 0){
    end_op();
    return -1;
  }
  iunlockput(ip);
  end_op();
  return 0;
}
```

- 功能：创建一个设备文件
- 三个参数分别表示路径，主设备号和辅设备号，用来创建设备文件
- 三个argstr函数检查是否传入足够的参数同时把参数拷贝进path、major、minor三个临时变量
- 然后调用create创建设备文件

Q2：create函数调用了dirlookup函数，它实现了什么功能，是怎么实现的？

- 功能：在目录dp内检查名为name的文件是否存在；
  - 如果存在，则返回该文件的inode;
  - 否则返回0
  - poff如果不为0，则将找到的目录项在dp数据块内的偏移量存入*poff内

```
// Look for a directory entry in a directory.
// If found, set *poff to byte offset of entry.
struct inode*
dirlookup(struct inode *dp, char *name, uint *poff)
{
  uint off, inum;
  struct dirent de;

  if(dp->type != T_DIR)
    panic("dirlookup not DIR");

  for(off = 0; off < dp->size; off += sizeof(de)){
    if(readi(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
      panic("dirlookup read");
    if(de.inum == 0)
      continue;
    if(namecmp(name, de.name) == 0){
      // entry matches path element
      if(poff)
        *poff = off;
      inum = de.inum;
      return iget(dp->dev, inum);
    }
  }

  return 0;
}
```

Q2：create函数调用了dirlookup函数，它实现了什么功能，是怎么实现的？

```
for(off = 0; off < dp->size; off += sizeof(de)){
  if(readi(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
    panic("dirlookup read");
  if(de.inum == 0)
    continue;
  if(namecmp(name, de.name) == 0){
    // entry matches path element
    if(poff)
      *poff = off;
    inum = de.inum;
    return iget(dp->dev, inum);
  }
}
```

- 以一个目录项长度为单位进行for循环
- Readi函数：将dp中的off位置读出一个目录项长度的数据写入de的位置中。
- 如果此时de的inode number为0时，该目录项无效，因此直接跳过该次循环
- 如果要比较的name与目录项de的name一样，则返回该目录项的inode。
- 另外，如果poff不为0，还要将该目录项在数据块中的偏移量off存入*poff中。

Q3：create函数还调用了ialloc函数，它的功能是什么？

```
ialloc(uint dev, short type)
{
  int inum;
  struct buf *bp;
  struct dinode *dip;

  for(inum = 1; inum < sb.ninodes; inum++){
    bp = bread(dev, IBLOCK(inum, sb));
    dip = (struct dinode*)bp->data + inum%IPB;
    if(dip->type == 0){  // a free inode
      memset(dip, 0, sizeof(*dip));
      dip->type = type;
      log_write(bp);   // mark it allocated on the disk
      brelse(bp);
      return iget(dev, inum);
    }
    brelse(bp);
  }
  panic("ialloc: no inodes");
}
```

- ialloc函数：在磁盘上分配一个给定dev和type的inode，若分配成功则返回该ip，更新缓存。

- 用for循环和bread函数将每一块中的对应dev的inode调入缓存，并检查是否是free的。
- 若找到，则对其进行初始化，并用iget函数返回对应ip。
- 最后再用brelse释放每一块被调入占用的缓存。若找不到，则产生panic故障。

Q3：create函数还调用了ialloc函数，它的功能是什么？

```
if((ip = ialloc(dp->dev, type)) == 0)
  panic("create: ialloc");
```

- Create调用iallock函数作用：

  当create函数找不到传入的文件名时，则需要在磁盘上分配一个新的inode并为其命名。

Q4：namei函数和create函数最终都调用fs.c文件中的namex函数，请逐行介绍它的功能。

```
static struct inode*
namex(char *path, int nameiparent, char *name)
{
  struct inode *ip, *next;
```

```
struct inode*
namei(char *path)
{
  char name[DIRSIZ];
  return namex(path, 0, name);
}
```

```
struct inode*
nameiparent(char *path, char *name)
{
  return namex(path, 1, name);
}
```

- namex函数的参数分别表示
  - *path: 要解析的文件路径
  - nameiparent: 找上级目录的inode还是当前目录的inode。
    - 被namei函数调用时该参数为0，而被nameiparent函数调用时设为1.
  - *name用于存放查询过程中的最后一项

Q4：namei函数和create函数最终都调用fs.c文件中的namex函数，请逐行介绍它的功能。

```
if(*path == '/')
  ip = iget(ROOTDEV, ROOTINO);
else
  ip = idup(myproc()->cwd);
```

对路径首个字符判断，
如果是“/”，说明是绝对路径，此时将ip设为根目录的inode;
否则将ip设为当前进程所在的目录的inode

```
while((path = skipelem(path, name)) != 0){
```

通过while对路径进行解析，
通过调用skipelem函数将路径最前的名称放入name中，并将剩余部分放入path中

```
// Examples:
//   skipelem("a/bb/c", name) = "bb/c", setting name = "a"
//   skipelem("///a//bb", name) = "bb", setting name = "a"
//   skipelem("a", name) = "", setting name = "a"
//   skipelem("", name) = skipelem("////", name) = 0
//
static char*
skipelem(char *path, char *name)
```

Q4：namei函数和create函数最终都调用fs.c文件中的namex函数，请逐行介绍它的功能。

```
while((path = skipelem(path, name)) != 0){
  if(nameiparent && *path == '\0'){
    // Stop one level early.
    iunlock(ip);
    return ip;
  }
  if((next = dirlookup(ip, name, 0)) == 0){
    iunlockput(ip);
    return 0;
  }
  ip = next;
}
if(nameiparent){
  iput(ip);
  return 0;
}
return ip;
```

- 如果nameiparent为1，说明要找的是上一级目录的inode，此时要少执行一次skipelem函数。
  - 以a/b/c为例，退出时，ip应该指向上次循环中dirlookup函数返回的inode，也就是b
- 在Q2中说明了dirlookup函数的作用是在目录ip内检查名为name的文件是否存在
  - 如果存在，则让next指向函数返回的inode，否则就说明该目录下不存在要找的文件，于是释放缓存并退出
- 然后让ip指向next所指的内容