PRJ 6

一. Debug记录

TASK 1

（1）未pad镜像

```
------------------- COMMAND -------------------
blocks read error!fs
[FS] Start initialize filesystem!
[FS] Setting superblock...
  magic: 0x20221205
  num sector: 1049346, start sector: 1048576
  sector map offset: 1(256)
  inode map offset: 257(1)
blocks write error!1048576)
blocks write error!map...
blocks write error!-map...
blocks read error!...
blocks write error!
blocks read error!
blocks write error!
blocks write error!
blocks read error!
blocks write error!
```

开局不利……意识到此时应该暂时给镜像pad一下，按照prj4修改Makefile：

```
image: $(ELF_CREATEIMAGE) $(ELF_BOOT) $(ELF_MAIN) $(ELF_USER)
#   cd $(DIR_BUILD) && ./$(<F) --extended $(filter-out $(<F), $(^F))
    cd $(DIR_BUILD) && ./$(<F) --extended $(filter-out $(<F), $(^F))
&& dd if=/dev/zero of=image oflag=append conv=notrunc bs=512MB
count=2
```

```
------------------- COMMAND -------------------
> root@UCAS_OS: mkfs
[FS] Start initialize filesystem!
[FS] Setting superblock...
  magic: 0x20221205
  num sector: 1049346, start sector: 1048576
  block map offset: 1(256)
  inode map offset: 257(1)
  inode offset: 258(512)
  data offset: 770(1048576)
[FS] Setting inode-map...
[FS] Setting sector-map...
blocks write error!..
[FS] Initialize filesystem finished!
> root@UCAS_OS:
```

（2）分配inode时犯的低级错误……

```
------------------- COMMAND -------------------
> root@UCAS_OS:~/$ mkfs
[FS] Start initialize filesystem!
[FS] Setting superblock...
  magic: 0x20221205
  num sector: 1049346, start sector: 1048576
  block map offset: 1(256)
  inode map offset: 257(1)
  inode offset: 258(512)
  data offset: 770(1048576)
[FS] Setting inode-map...
[FS] Setting sector-map...
[FS] Setting inode...
[FS] Initialize filesystem finished!
> root@UCAS_OS:~/$ mkdir hey
> root@UCAS_OS:~/$ ls
 hey
> root@UCAS_OS:~/$ mkdir hi
> root@UCAS_OS:~/$ ls
 hi
> root@UCAS_OS:~/$
```

mkdir时似乎覆盖了之前创建的目录，gdb跟踪：

```
Breakpoint 1, do_mkdir (path=0x1269c "hey")
    at ./kernel/fs/fs.c:238
238         if(get_inode_from_name(current_inode, path, NULL))
(gdb) n
242         int ino = alloc_inode();
(gdb)
244         bzero(buffer, 512);
(gdb)
245         dentry_t * de = (dentry_t*) buffer;
(gdb)
246         strcpy(de[0].name, ".");
(gdb)
247         strcpy(de[1].name, "..");
(gdb)
248         de[0].ino = ino;
(gdb)
249         de[1].ino = current_inode.ino;
(gdb)
250         uint32_t data_blk_addr = alloc_block();
(gdb)
251         bios_sdwrite(kva2pa(buffer), 1,  data_blk_addr);
(gdb) p data_blk_addr
$1 = 131072
(gdb) n
253         inode_t *node = ino2inode(ino);
(gdb) p ino
$2 = 513
(gdb)
    Breakpoint 1, do_mkdir (path=0x1269c "hi")
    at ./kernel/fs/fs.c:238
238         if(get_inode_from_name(current_inode, path, NULL))
(gdb) n
```

```
242         int ino = alloc_inode();
(gdb)
244         bzero(buffer, 512);
(gdb) p ino
$3 = 513
(gdb)
```

意识到分配inode的函数出了问题：

```
for(i=0; i<INODE_MAP_NUM*SECTOR_SIZE; i++){
        for(j=0, mask=1; j<sizeof(char); j++, mask<<1){
            if(imap[i] & mask == 0)
                break;
        }
    }
    // 将未使用的inode结点置为已用
    imap[i] |= mask;
    bios_sdwrite(kva2pa(imap), INODE_MAP_NUM, FS_START_SEC +
INODE_MAP_OFFSET);
```

意识到break只跳出了一重循环，同时注意到按位与的优先级低于==。

（3）忽视目录项被回收后目录块中的空泡

修改后可以正常创建目录，但删除文件夹会出错：

```
------------------- COMMAND -------------------
> root@UCAS_OS:~/$ mkfs
[FS] Start initialize filesystem!
[FS] Setting superblock...
  magic: 0x20221205
  num sector: 1049346, start sector: 1048576
  block map offset: 1(256)
  inode map offset: 257(1)
  inode offset: 258(512)
  data offset: 770(1048576)
[FS] Setting inode-map...
[FS] Setting sector-map...
[FS] Setting inode...
[FS] Initialize filesystem finished!
> root@UCAS_OS:~/$ mkdir hi
> root@UCAS_OS:~/$ mkdir hey
> root@UCAS_OS:~/$ ls
 hi hey
> root@UCAS_OS:~/$ rmdir hi
> root@UCAS_OS:~/$ ls

> root@UCAS_OS:~/$
```

原因是起初遇到空结点就会跳出循环：

```
    // 跳过.和..
    for(int i=2; i<SECTOR_SIZE/sizeof(dentry_t); i++){
        if(de[i].name[0]==0)
            break;
        else if(option){ // 需打印详细信息
```

而对于位于其后的结点就可能无法被遍历，修改为continue后就可正常进行。

（4）上板后的一些迷死

发现在ls-l时总是会卡住，查看实现：

```
for(int i=2; i<DPSEC; i++){
        if(de[i].name[0]==0)
            continue;
        else if(option){ // 需打印详细信息
            inode_t tmp = *ino2inode(de[i].ino);
            printk("%c%c%c nlink:%d ctime:%d atime:%d mtime:%d
size:%d %s\n",
                    tmp.type == T_DIR ? 'd' : '-',
                    tmp.mode & O_RDONLY ? 'r' : '-',
                    tmp.mode & O_WRONLY ? 'w' : '-',
                    tmp.nlink, tmp.ctime, tmp.atime, tmp.mtime,
tmp.size,
                    de[i].name
                    );
        }
        else
            printk("\t%s", de[i].name);
    }
```

原本是在读到名字为空时就不会根据其ino域去获取inode，但由于sd卡中可能原本就有些脏数据，导致此时可能会获取到很诡异的ino。故需要在每次分配block时先将数据清空。

TASK 2

（1）读写文件的实际长度?

运行测试用例卡住，gdb跟踪：

```
(gdb) n
608             if(write_ptr % BLOCK_SIZE)
<p write_ptr
$3 = 0
(gdb) n
610             memcpy(buffer + (write_ptr%BLOCK_SIZE),
buff,partial_len);
(gdb)
^C
Program received signal SIGINT, Interrupt.
atomic_swap (
```

```
    val=1,
    mem_addr=18446743800176394880) at
./arch/riscv/include/atomic.h:9
9       {
(gdb)
```

意识到此时len的设置不准确：

```
// 以block为单位读取
    for(int read_ptr = fdesc_array[fd].read_ptr;
read_ptr<fdesc_array[fd].read_ptr + len;){
        int partial_len = read_ptr % BLOCK_SIZE ? (BLOCK_SIZE -
(read_ptr % BLOCK_SIZE)) : BLOCK_SIZE;
        uint32_t read_addr =  get_data_block_addr(node, read_ptr);
        bios_sdread(kva2pa(buffer), BLOCK_SIZE/SECTOR_SIZE,
read_addr);
        memcpy(buff, buffer+(read_ptr%BLOCK_SIZE), partial_len);
        read_ptr += partial_len;
        buff += partial_len;
    }
```

还需要判断 `partial_len` 和要求读入的 `len` 的关系：

```
int partial_len = read_ptr % BLOCK_SIZE ? (BLOCK_SIZE - (read_ptr %
BLOCK_SIZE)) : BLOCK_SIZE;
        int tmp = fdesc_array[fd].read_ptr + len -
fdesc_array[fd].read_ptr;
        partial_len = partial_len > tmp ? tmp : partial_len;
```

修改后可正常输出（请忽略间歇性抽风的终端输出）：

```
hello world!   c             v         : 00000004
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
------------------[12[[1[11;25H----------------
> root@UCAS_OS:~$ e ec rwfile &
Info: excute rwfile successfully, pid = 2
> root@UCAS_OS:~$
```

重启后调用cat也能正常输出：

```
------------------[11H1;23H ------------------
> root@UCAS_OS:~$ cat 1.txt
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!     QEMU: Terminated
```

（2）多级索引考虑不周

同时为了测试大文件的情况，在测试用例中增加lseek：

```
//------------------------------******REVISE START******------------
------------------------
    assert(fd>=0);
    assert(sys_lseek(fd, RW_START, SEEK_SET)==RW_START);
//------------------------------*******REVISE END*******------------
------------------------

```

```
 e "  * t2hellocworld!        v          : 00000004
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!

------------------- COMMAND -------------------
> root@UCAS_OS:~$ mkfs
[FS] Start initialize filesystem!
[FS] Setting superblock...
  magic: 0x20221205
  num sector: 1049346, start sector: 1048576
  block map offset: 1(256)
  inode map offset: 257(1)
  inode offset: 258(512)
  data offset: 770(1048576)
[FS] Setting inode-map...
[FS] Setting sector-map...
[FS] Setting inode...
[FS] Initialize filesystem finished!
> root@UCAS_OS:~$ touch 1.txt
> root@UCAS_OS:~$ exec rwfile &
Info: excute rwfile successfully, pid = 2
> root@UCAS_OS:~$
```

发现输出不对。gdb跟踪：

```
(gdb) p tmp
```

```
$2 = 13
(gdb) n
636             uint32_t write_addr =  get_data_block_addr(node,
write_ptr);
(gdb) n
637             if(write_ptr % BLOCK_SIZE ||
partial_len<BLOCK_SIZE)
(gdb) p write_addr
$3 = 1057458
(gdb) p write_ptr
$4 = 8388608
(gdb) p/x write_ptr
$5 = 0x800000
(gdb) n
638                 bios_sdread(kva2pa(buffer),
BLOCK_SIZE/SECTOR_SIZE, write_addr);
......

Breakpoint 2, do_fread (fd=0,
    buff=0x117f0 "", length=13)
    at ./kernel/fs/fs.c:602
602         for(int read_ptr = fdesc_array[fd].read_ptr;
read_ptr<fdesc_array[fd].read_ptr + len;){
(gdb) n
603             int partial_len = read_ptr % BLOCK_SIZE ?
(BLOCK_SIZE - (read_ptr % BLOCK_SIZE)) : BLOCK_SIZE;
(gdb)
604             int tmp = fdesc_array[fd].read_ptr + len -
read_ptr;
(gdb)
605             partial_len = partial_len > tmp ? tmp :
partial_len;
(gdb)
606             uint32_t read_addr =  get_data_block_addr(node,
read_ptr);
(gdb)
607             bios_sdread(kva2pa(buffer), BLOCK_SIZE/SECTOR_SIZE,
read_addr);
(gdb) p read_addr
$15 = 1049362
(gdb) p read_ptr
$16 = 8388608
(gdb) p/x read_ptr
$17 = 0x800000
(gdb)
```

发现前后分配的地址不对，查看原先多级索引的实现：

```
    // 使用一级索引
```

```
    else if(size < DIRECT_SIZE + INDIRECT_1ST_SIZE){
        size -= DIRECT_SIZE;
        int index1 = size/(BLOCK_SIZE*IA_PER_BLOCK);
        int index2 = (size - index1*BLOCK_SIZE*IA_PER_BLOCK)/BLOCK_SIZE;
        if(node.indirect_addrs_1st[index1]==0){
            uint32_t data_blk_addr = alloc_block();
            setup_level_index(data_blk_addr, 0);
            inode_t* node_ptr = ino2inode(node.ino);
            node_ptr->indirect_addrs_1st[index1] = data_blk_addr;
            int offset = node.ino / IPSEC;
            bios_sdwrite(kva2pa(buffer), 1, FS_START_SEC + INODE_OFFSET + offset);
            bios_sdread(kva2pa(buffer), BLOCK_SIZE/SECTOR_SIZE, node_ptr->indirect_addrs_1st[index1]);
            uint32_t* addr_array = buffer;
            return addr_array[index2];
        }
    }
    // 使用二级索引
    else if(size < DIRECT_SIZE + INDIRECT_1ST_SIZE + INDIRECT_2ND_SIZE){
        size -= (DIRECT_SIZE + INDIRECT_1ST_SIZE);
        int index1 = size/(BLOCK_SIZE*IA_PER_BLOCK*IA_PER_BLOCK);
        int index2 = (size - index1*BLOCK_SIZE*IA_PER_BLOCK*IA_PER_BLOCK)/(BLOCK_SIZE*IA_PER_BLOCK);
        int index3 = (size - index1*BLOCK_SIZE*IA_PER_BLOCK*IA_PER_BLOCK - index2*IA_PER_BLOCK)/BLOCK_SIZE;
        if(node.indirect_addrs_2nd[index1]==0){
            uint32_t data_blk_addr = alloc_block();
            setup_level_index(data_blk_addr, 1);
            inode_t* node_ptr = ino2inode(node.ino);
            node_ptr->indirect_addrs_2nd[index1] = data_blk_addr;
            int offset = node.ino / IPSEC;
            bios_sdwrite(kva2pa(buffer), 1, FS_START_SEC + INODE_OFFSET + offset);
            // 获取二级索引项
            bios_sdread(kva2pa(buffer), BLOCK_SIZE/SECTOR_SIZE, data_blk_addr);
            uint32_t* addr_array = buffer;
            data_blk_addr = addr_array[index2];
            // 获取一级索引项
            bios_sdread(kva2pa(buffer), BLOCK_SIZE/SECTOR_SIZE, data_blk_addr);
            return addr_array[index3];
        }
    }
```

```
    // 使用三级索引
    else if(size < DIRECT_SIZE + INDIRECT_1ST_SIZE + INDIRECT_2ND_SIZE + INDIRECT_3RD_SIZE){
        size -= (DIRECT_SIZE + INDIRECT_1ST_SIZE + INDIRECT_2ND_SIZE);
        int index1 = size/(BLOCK_SIZE*IA_PER_BLOCK*IA_PER_BLOCK);
        int index2 = (size - index1*BLOCK_SIZE*IA_PER_BLOCK*IA_PER_BLOCK)/(BLOCK_SIZE*IA_PER_BLOCK);
        if(node.indirect_addrs_3rd==0){
            uint32_t data_blk_addr = alloc_block();
            setup_level_index(data_blk_addr, 1);
            inode_t* node_ptr = ino2inode(node.ino);
            node_ptr->indirect_addrs_3rd = data_blk_addr;
            int offset = node.ino / IPSEC;
            bios_sdwrite(kva2pa(buffer), 1, FS_START_SEC + INODE_OFFSET + offset);
            // 获取二级索引项
            bios_sdread(kva2pa(buffer), BLOCK_SIZE/SECTOR_SIZE, data_blk_addr);
            uint32_t* addr_array = buffer;
            data_blk_addr = addr_array[index1];
            // 获取一级索引项
            bios_sdread(kva2pa(buffer), BLOCK_SIZE/SECTOR_SIZE, data_blk_addr);
            return addr_array[index2];
        }
    }
```

意识到上述都忘记考虑索引项已经分配的情况，应该修改为如下（只附上一级索引的代码，二级、三级同理）：

```
    // 使用一级索引
    else if(size < DIRECT_SIZE + INDIRECT_1ST_SIZE){
        size -= DIRECT_SIZE;
        int index1 = size/(BLOCK_SIZE*IA_PER_BLOCK);
        int index2 = (size - index1*BLOCK_SIZE*IA_PER_BLOCK)/BLOCK_SIZE;
        uint32_t data_blk_addr;
        if(node.indirect_addrs_1st[index1]==0){
            data_blk_addr = alloc_block();
            setup_level_index(data_blk_addr, 0);
            inode_t* node_ptr = ino2inode(node.ino);
            node_ptr->indirect_addrs_1st[index1] = data_blk_addr;
            int offset = node.ino / IPSEC;
            bios_sdwrite(kva2pa(buffer), 1, FS_START_SEC + INODE_OFFSET + offset);
        }
```

```
        else
            data_blk_addr = node.indirect_addrs_1st[index1];
        bios_sdread(kva2pa(buffer), BLOCK_SIZE/SECTOR_SIZE,
data_blk_addr);
        uint32_t* addr_array = buffer;
        return addr_array[index2];
    }
```

修改后如下：

```
hello world!   c             v          : 0006
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
------------------- COMMAND -------------------
> root@UCAS_OS:~$ mkfs
[FS] Start initialize filesystem!
[FS] Setting superblock...
  magic: 0x20221205
  num sector: 1049346, start sector: 1048576
  block map offset: 1(256)
  inode map offset: 257(1)
  inode offset: 258(512)
  data offset: 770(1048576)
[FS] Setting inode-map...
[FS] Setting sector-map...
[FS] Setting inode...
[FS] Initialize filesystem finished!
> root@UCAS_OS:~$ touch 1.txt
> root@UCAS_OS:~$ exec rwfile &
Info: excute rwfile successfully, pid = 2
> root@UCAS_OS:~$
```

虽然在输出上一模一样，但是我们可以通过查看block的使用情况来判断索引的分配情况：

```
------------------- COMMAND -------------------
> root@UCAS_OS:~$ statfs
[FS] state:
  magic: 0x20221205
  total sector: 1049346, start sector: 1048576(00100000)
  block map offset: 1, occupied sector: 256
  inode map offset: 257, occupied sector: 1
  inode offset: 258, usage 2/512
  data offset: 770, usage 1027/1048576
  inode size: 100B, dir entry size: 20B
> root@UCAS_OS:~$
```

为建立多级索引，其会使用多个数据块。此外，还可以通过ls -l查看详细信息：

```
> root@UCAS_OS:~$ ls -l
-rw nlink:01 ctime:14 atime:14 mtime:14 size:8388738 1.txt
                 ▉
> root@UCAS_OS:~$
```

可见其size符合预期值。但注意到读写时还未更新时间戳信息，修改后效果如下：

```
------------------- COMMAND -------------------
> root@UCAS_OS:~$ ls -l
-rw nlink:1 ctime:28 atime:41 mtime:39 size:8388738 1.txt
```

简名	全名	中文	作用
atime	Access Time	访问时间	最后一次访问文件（读取或执行）的时间
ctime	Change Time	变化时间	最后一次改变文件（属性或权限）或者目录（属性或权限）的时间
mtime	Modify Time	修改时间	最后一次修改文件（内容）或者目录（内容）的时间

回收不完全：

```
------------------- COMMAND -------------------
> root@UCAS_OS:~$ rm 1.txt
> root@UCAS_OS:~$ statfs
[FS] state:
  magic: 0x20221205
  total sector: 1049346, start sector: 1048576(00100000)
  block map offset: 1, occupied sector: 256
  inode map offset: 257, occupied sector: 1
  inode offset: 258, usage 1/512
  data offset: 770, usage 899/1048576
  inode size: 100B, dir entry size: 20B
> root@UCAS_OS:~$
```

gdb查看回收后的bmap情况：

```
(gdb) x/1000x bmap
0xffffffc050250520 <bmap>:      0xfefefeff      0xfefefefe      0xfefefefe      0xfefefefe
0xffffffc050250530 <bmap+16>:   0xfefefefe      0xfefefefe      0xfefefefe      0xfefefefe
0xffffffc050250540 <bmap+32>:   0xfefefefe      0xfefefefe      0xfefefefe      0xfefefefe
0xffffffc050250550 <bmap+48>:   0xfefefefe      0xfefefefe      0xfefefefe      0xfefefefe
0xffffffc050250560 <bmap+64>:   0xfefefefe      0xfefefefe      0xfefefefe      0xfefefefe
0xffffffc050250570 <bmap+80>:   0xfefefefe      0xfefefefe      0xfefefefe      0xfefefefe
0xffffffc050250580 <bmap+96>:   0xfefefefe      0xfefefefe      0xfefefefe      0xfefefefe
0xffffffc050250590 <bmap+112>:  0xfefefefe      0xfefefefe      0xfefefefe      0xfefefefe
0xffffffc0502505a0 <bmap+128>:  0x00000006      0x00000000      0x00000000      0x00000000
0xffffffc0502505b0 <bmap+144>:  0x00000000      0x00000000      0x00000000      0x00000000
0xffffffc0502505c0 <bmap+160>:  0x00000000      0x00000000      0x00000000      0x00000000
0xffffffc0502505d0 <bmap+176>:  0x00000000      0x00000000      0x00000000      0x00000000
```

查看回收部分代码：

```
    int bno = data_blk_addr - FS_START_SEC - DATA_OFFSET;
    bmap[bno/8] &= ~(1 << (bno%8));
```

意识到回收时忽略了分配是以块为单位进行的，故bno应该修改为：

```
int bno = (data_blk_addr - FS_START_SEC - DATA_OFFSET) *SECTOR_SIZE / BLOCK_SIZE;
bmap[bno/8] &= ~(1 << (bno%8));
```

修改后可以正常删除（请再次忽略抽风的终端）：

```
-----------------[11;22HH[11;26H---------------
> root@UCAS_OS:~$ [1212;22H
> root@UCAS_OS:~$ statfs
[FS] state:
  magic: 0x20221205
  total sector: 1049346, start sector: 1048576(00100000)
  block map offset: 1, occupied sector: 256
  inode map offset: 257, occupied sector: 1
  inode offset: 258, usage 1/512
  data offset: 770, usage 1/1048576
  inode size: 100B, dir entry size: 20B
> root@UCAS_OS:~$
```

TASK 3 [Undone]

小程序发送包时显示如下：

```
stu@stu:/$ cd ~
stu@stu:~$ cd p6-pktRxTx_elf/
stu@stu:~/p6-pktRxTx_elf$ sudo ./pktRxTx -m 1 -t 5
[sudo] password for stu:
Info: pktRxTx was built at Dec 17 2022, 07:21:43
1. tap0 (No Description Avaliable)
2. br0 (No Description Avaliable)
3. enp0s3 (No Description Avaliable)
4. lo (No Description Avaliable)
5. Pseudo-device that captures on all interfaces
6. Bluetooth Linux Monitor
7. Linux netfilter log (NFLOG) interface
8. Linux netfilter queue (NFQUEUE) interface
Enter the interface number (1-8): 1
Info: Here, MAC Address is 128:250:91:51:86:239, listening on device tap0 ...
Info: MAC Address of the opposite is 00:10:53:00:30:83 ...
Info: Input command. For example:
        --- 'send 60': send 60 packets to the opposite
        --- 'test 60': keep sending packets in 60 seconds
        --- 'quit': quit this program
> send
> Please input file name: plus.elf
> Opening 'plus.elf'.
> Size is 34B
  [INFO] sending head packet, size = 8B.
  [INFO] sending 0 packet, size = 34B.
>
> quit
Info: Sender finishes its task and exits!
```

我个亲娘欸这包都没发两个咋触发网卡中断啊？难不成多发几个废包？

3. 未完待续

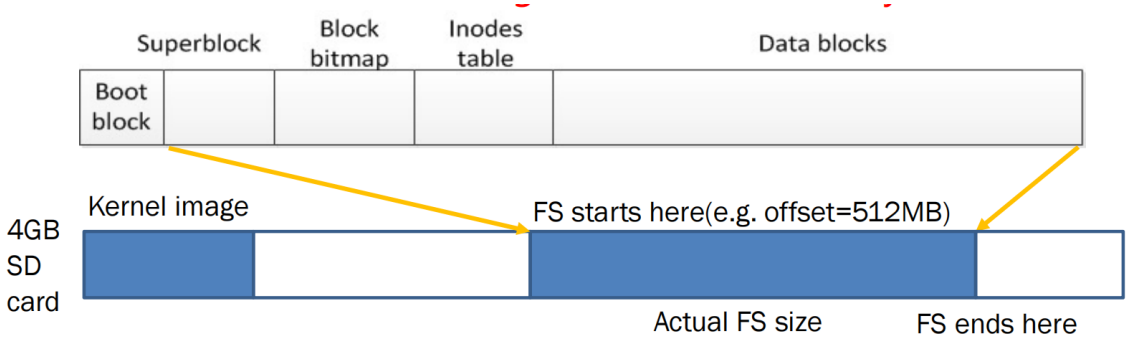
目前还需考虑如下问题：

- 考虑发包未满的情况?
- 考虑test文件在kernel内的处理方式：
  - 在task info数组里加上相关信息?
  - elf头是否要仿照createimage中处理?

二. Design Review

1. Q &A

- What is the disk layout in your design?



- Show the structures of your FS metadata, including superblock, inode, dentry, and file descriptor

```c
/* data structures of file system */
typedef struct superblock_t{
    // TODO [P6-task1]: Implement the data structure of superblock
    uint32_t magic_number;
    uint32_t fs_start_sec;
    uint32_t fs_size;       // Size of file system image (blocks)
    uint32_t block_map_offset;
    uint32_t block_map_num;
    uint32_t inode_map_offset;
    uint32_t inode_map_num;
    uint32_t inode_offset;
    uint32_t inode_num;
    uint32_t data_offset;
    uint32_t data_num;
} superblock_t;

```

```c
typedef struct dentry_t{
    // TODO [P6-task1]: Implement the data structure of directory entry
    char name[16];
    int ino;
} dentry_t;

typedef struct inode_t{
    // TODO [P6-task1]: Implement the data structure of inode
    char type;
    char mode;
    short nlink;       // Number of links to inode in file system
    uint32_t ino;
    uint32_t ctime;
    uint32_t atime;
    uint32_t mtime;
    uint32_t size;
    uint32_t direct_addrs[NDIRECT];
    uint32_t indirect_addrs_1st[3];
    uint32_t indirect_addrs_2nd[2];
    uint32_t indirect_addrs_3rd;
} inode_t;

typedef struct fdesc_t{
    // TODO [P6-task2]: Implement the data structure of file descriptor
    uint8_t valid;
    uint8_t mode;
    short ref; // reference count
    int ino;
    uint32_t write_ptr;
    uint32_t read_ptr;
} fdesc_t;


```

> 着重讲一下nlink和ref用处区别

- How many files and directories do you file system support?

```
#define IA_PER_BLOCK (BLOCK_SIZE/sizeof(uint32_t))
#define DIRECT_SIZE (NDIRECT*BLOCK_SIZE)
#define INDIRECT_1ST_SIZE (3*BLOCK_SIZE*IA_PER_BLOCK)
#define INDIRECT_2ND_SIZE
(2*BLOCK_SIZE*IA_PER_BLOCK*IA_PER_BLOCK)
#define INDIRECT_3RD_SIZE
(1*BLOCK_SIZE*IA_PER_BLOCK*IA_PER_BLOCK*IA_PER_BLOCK)
#define MAX_FILE_SIZE (DIRECT_SIZE + INDIRECT_1ST_SIZE +
INDIRECT_2ND_SIZE + INDIRECT_3RD_SIZE)
```

一个inode最多：

- 索引 `MAX_FILE_SIZE` 大小的文件；
- 含 `MAX_FILE_SIZE/sizeof(dentry_t)` 个目录项；

文件系统使用 `256KB` 作inode的存储空间，故最多 `256K/sizeof(inode_t)` 个inode。

- What do you do when initializing a file system?
  - superblock的初始化
    > 从磁盘特定位置载入后先check是否已建立fs
  - inode和block map的 初始化
    > 数据块以4KB为粒度
  - 创建根目录
    - 分配inode
    - 分配目录索引页
  - 初始化文件描述符
- Given an operation, for example *ls /home/student*, How do you handle path lookup?
  - 路径解析：以 `/` 为分割
  - 相对路径特性：递归查找对应inode并返回

---

2. Design Review后的改进

1）元数据的个数&一致性

在design review中意识到此前自己对元数据个数的理解尚不清晰，出现了inode数和inode map的体量不匹配的情况（原先是划定给inode的内存大小，反推inode数和inode map，后成功把自己算晕）。

后将inode数目划定，再利用相对关系定出inode map和inode内存大小，如下：

```
#define INODE_NUM           512         // INODE个数
#define DATA_BLOCK_NUM      (1<<20)     // 数据块个数（4KB为单位）,共4GB
#define INODE_MAP_SEC_NUM   CEIL_DIV(INODE_NUM, SECTOR_SIZE*8)  // inode map所占sector个数
#define BLOCK_MAP_SEC_NUM   CEIL_DIV(DATA_BLOCK_NUM, SECTOR_SIZE*8)   // data block map所占sector个数
#define INODE_SEC_NUM       CEIL_DIV(sizeof(inode_t)*INODE_NUM, SECTOR_SIZE)       // inode占据的sector个数
#define DATA_BLOCK_SEC      (DATA_BLOCK_NUM/SECTOR_SIZE*BLOCK_SIZE)            // data block占据的sector个数
```

2）删除时出现的问题

- 硬链接文件的删除情况：原本设置为支持多层删除，后续发现删除了前一层文件后，本层目录下的硬链接文件反而会被删除，原因是根据name索引到了文件后我返回的是ino号，然后再在本层目录下以ino号做匹配删除。只需改为严格限制为单级删除即可。
- nlink数忘记更改：原本只是将之在内存中的缓存的nlink数减一，忘记写回。

3）一个南死人的bug：内核栈溢出

在qemu时跑大文件都没有出问题，但是上板一直会卡住，后来擦边改了很多无关紧要的代码……最后发现是函数开了过大的局部变量做buffer，在递归建立多级索引的时候把内核栈搞溢出了。后修改为每层递归使用不同的buffer即可：

```
static char level_buffer[3][BLOCK_SIZE];
void setup_level_index(uint32_t data_blk_addr, int level){
    uint32_t* addr_array = level_buffer[level];
    // printk("I am in setup_level_index\n");
    alloc_block(addr_array, IA_PER_BLOCK);
    if(level)
        for(int i=0; i<IA_PER_BLOCK; i++)
            setup_level_index(addr_array[i], level-1);
    bios_sdwrite(kva2pa(level_buffer[level]), BLOCK_SIZE/SECTOR_SIZE, data_blk_addr);
}
```

> 很怪的是之前也开了一些4KB的局部buffer（我们的内核栈大小就4KB……），但上板也没跑出问题。大概板子想出错时才出错，怪错只能自己抗😭。