Assignment9 Report

姓名：Yitong WANG(王奕童) 11910104@mail.sustech.edu.cn

学号：11910104

实验课时段：周五5-6节

实验课教师：Yun SHEN(沈昀) sheny@mail.sustech.edu.cn

实验课SA:

- Yining TANG(汤怡宁) 11811237@mail.sustech.edu.cn
- Yushan WANG(王宇杉) 11813002@mail.sustech.edu.cn

1. Disk scheduling

(1) READ/WRITE data time = Seek Time + Rotational Latency + Transfer Time

(2)

a. Track Access Sequence:

- FIFO: 70 => 30 => 90 => 120 => 60 => 20
- SSTF: 90 => 70 => 60 => 30 => 20 => 120
- SCAN: 120 => 90 => 70 => 60 => 30 => 20
- CSCAN: 120 => 20 => 30 => 60 => 70 => 90

b. Time Calculation

For the seek time, first calculate their tracks:

- FIFO: (100-70) + (70-30) + (90-30) + (120-90) + (120-60) + (60-20) = 260
- SSTF: (100-90) + (90-70) + (70-60) + (60-30) + (30-20) + (120-20) = 180
- SCAN: (120-100) + (199-120) + (199-90) + (90-70) + (70-60) + (60-30) + (30-20) = 278
- CSCAN: (120-100) + (199-120) + (199-0) + (20-0) + (30-20) + (60-30) + (70-60) + (90-70) = 388

Thus time is:

- FIFO: 260ms

- SSTF: 180ms
- SCAN: 278ms
- CSCAN: 388ms

For FIFO\SSTF\SCAN\CSCAN algorithm, their rotational latency is the same.

12000r/min => 200r/s => 0.2r/ms => 5ms/r

Since it is randomly distributed access, we treat it as half round: 2.5ms

Since 6 accesses, total time = 2.5ms*6 = 15ms

The question does not tell us the transfer time, thus we omitted.

Total Time:

- FIFO: 260+15=275ms
- SSTF: 180+15=195ms
- SCAN: 278+15=293ms
- CSCAN: 388+15=403ms

2. Simple File System

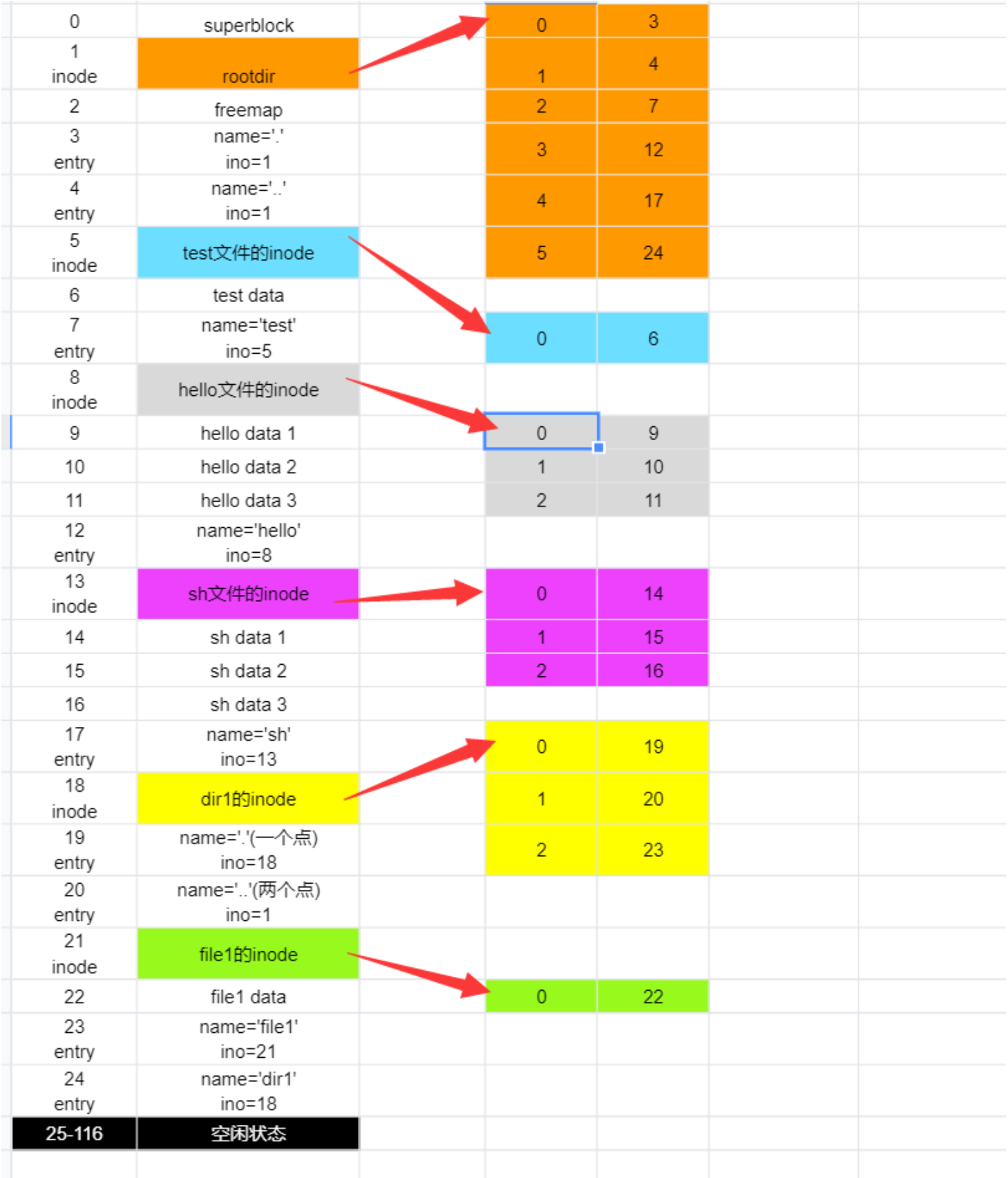
Consider adding one statement in `tools/mksfs.c` :

```
428 static void
429 add_entry(struct sfs_fs *sfs, struct cache_inode *current, struct cache_inode *file, const char *name) {
430     static struct sfs_entry __entry, *entry = &__entry;
431     assert(current->inode.type == SFS_TYPE_DIR && strlen(name) <= SFS_MAX_FNAME_LEN);
432     entry->ino = file->ino, strcpy(entry->name, name);
433     uint32_t entry_ino = sfs_alloc_ino(sfs);
434     write_block(sfs, entry, sizeof(struct sfs_entry), entry_ino);
435     append_block(sfs, current, sizeof(struct sfs_entry), entry_ino, name);
436     printf("entry is %d, name is %s, file->ino is %d\n", entry_ino, name, file->ino);
437     file->inode.nlinks ++;
438 
439 }
440 
```

And the partial output of `make qemu` is:

```
480000 bytes (480 kB, 469 KiB) copied, 0.00130072 s, 369 MB/s
entry is 3, name is ., file->ino is 1
entry is 4, name is .., file->ino is 1
entry is 7, name is test, file->ino is 5
entry is 12, name is hello, file->ino is 8
entry is 17, name is sh, file->ino is 13
entry is 19, name is ., file->ino is 18
entry is 20, name is .., file->ino is 1
entry is 23, name is file1, file->ino is 21
entry is 24, name is dir1, file->ino is 18
create bin/sfs.img (disk0) successfully.
+ ld bin/kernel
riscv64-unknown-elf-objcopy bin/kernel --strip-all -O binary bin/ucore.img
```

Thus we draw the diagram according to the running result:



Since 25-116 are free, the 25-116 bits are 1.

Here is a brief explanation:

第一个Table (root dir inode)

.的entry

..的entry

dir1的entry

test的entry

sh的entry

hello的entry

第二个Table (dir inode)

.的entry

..的entry

file1的entry

第三个Table (file1的inode)

file1的data x1

第四个Table' (test的inode)

test的data x1

第五个Table (sh的inode)

sh的data x3

第六个Table (hello的~~dat~~ inode)

hello的data x3

By 11910104