Project 5

Defragmentation

10152130113 王玉凤
10152130121 吕波尔
10152130122 钱庭涵

汇报：钱庭涵

Index

Part 1 Project目的

Part 2 数据结构

Part 3 算法实现

Part 4 观察结果

Part 5 检验内存泄漏

PART 1

Project目的

Project目的

A 为类 Unix 文件系统编写一个磁盘碎片整理程序

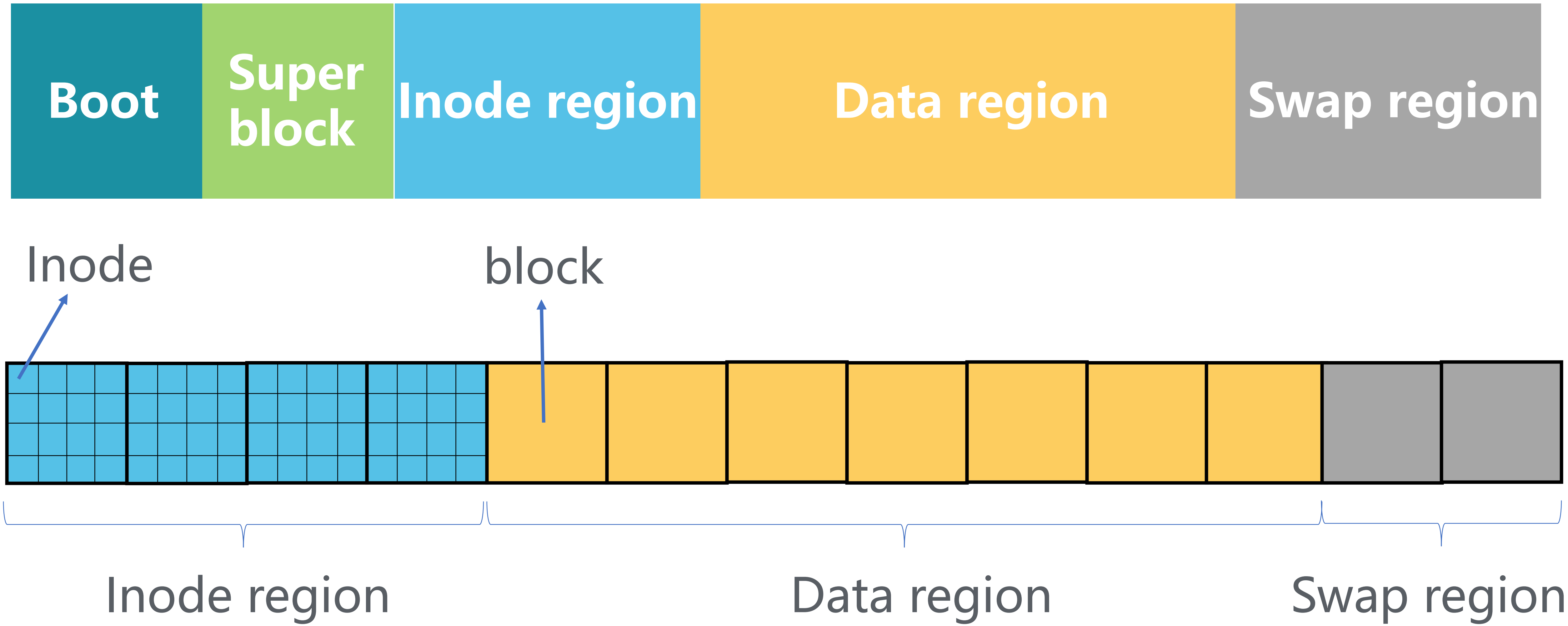
B 通过在磁盘上按顺序排列文件的所有块来提高性能

C 没有内存泄漏

PART 2

数据结构

数据结构



Boot

Super block

Inode region

Data region

Swap region

Inode

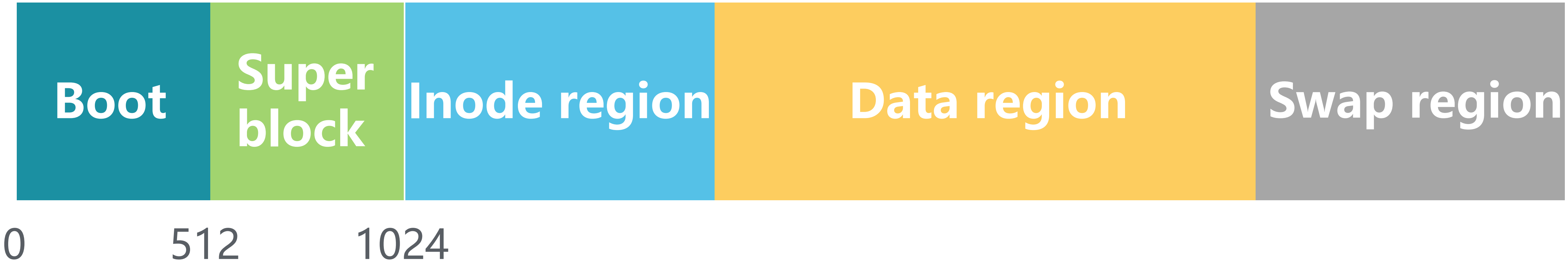
block

Inode region

Data region

Swap region

数据结构



```
typedef struct SUPERBLOCK {
    int size;
    int inode_offset;
    int data_offset;
    int swap_offset;
    int free_inode;
    int free_iblock;
} SuperBlock;
```

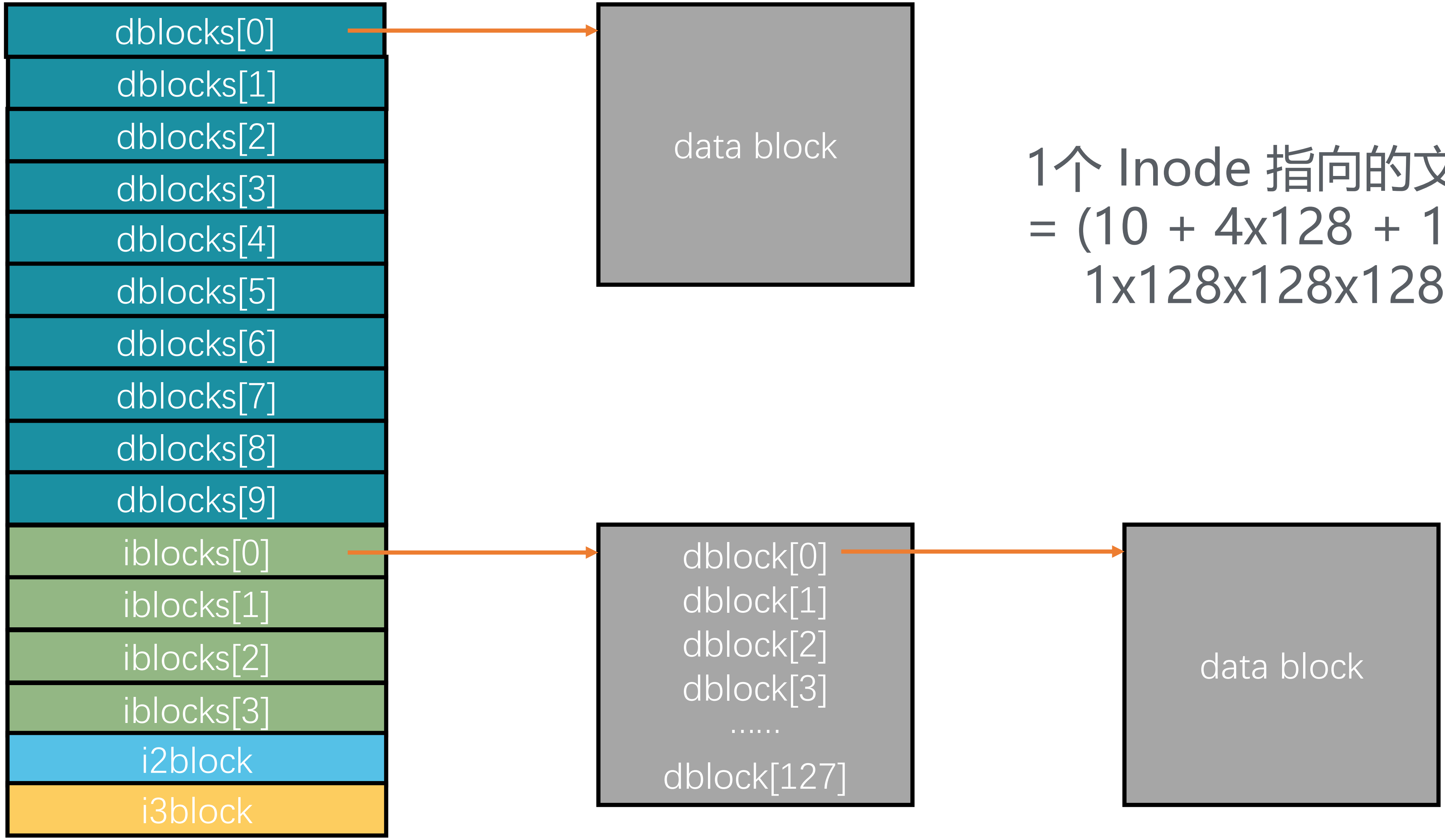
Inode/Data/Swap region 在磁盘内存中的起始区域
= 1024 + offset * block_size (bytes)

数据结构

```
#define N_DBLOCKS 10
#define N_IBLOCKS 4
```

```
typedef struct INODE {
    int next_inode;
    int protect;
    int nlink;     // 0 = not use
    int size;
    int uid;
    int gid;
    int ctime;
    int mtime;
    int atime;
    int dblocks[N_DBLOCKS];
    int iblocks[N_IBLOCKS];
    int i2block;
    int i3block;
} Inode;
```

数据结构



1个 Inode 指向的文件最大大小
= (10 + 4x128 + 1x128x128 +
1x128x128x128) * 512 (bytes)

PART 3

算法实现

算法实现

- 打开输入文件，创建并打开输出文件，错误检查

```
if ((fin = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "error: open file <%s> fail\n", argv[1]);
        exit(1);
}
strcpy(outfilename, argv[1]);
strcpy(outfilename + strlen(argv[1]) - 4, "-defrag.txt");
if ((fout = fopen(outfilename, "w+")) == NULL) {
        fprintf(stderr, "error: open file <%s> fail\n", outfilename);
        exit(1);
}
```

算法实现

- 复制引导块和 superblock

```
char *bootblock = malloc(512);
fread(bootblock, 512, 1, fin);
fwrite(bootblock, 512, 1, fout);
free(bootblock);

SuperBlock *superblock = (SuperBlock*)malloc(512);
fread(superblock, 512, 1, fin);
fwrite(superblock, 512, 1, fout);
```

算法实现

- 为 inode region 预留空间

```
char *inode_region = malloc(inode_block_number * block_size);
fwrite(inode_region, inode_block_number * block_size, 1, fout);
free(inode_region);
```

- 遍历所有 inode ， 进入循环，找到当前 inode 位置

```
for (i = 0; i < inode_number; i ++) {
        fseek(fin, 1024 + i * sizeof(Inode), SEEK_SET);
        fread(inode, sizeof(Inode), 1, fin);
```

算法实现

- 判断当前 inode 是否有效，利用 size_not_read 遍历索引块

```
if (inode->nlink != 0) {
        size_not_read = inode->size;
        for (j = 0; j < N_DBLOCKS &&  size_not_read > 0; j ++)
                new_block(0, &inode->dblocks[j]);
        for (j = 0; j < N_IBLOCKS &&  size_not_read > 0; j ++)
                new_block(1, &inode->iblocks[j]);
        if (size_not_read > 0)
                new_block(2, &inode->i2block);
        if (size_not_read > 0)
                new_block(3, &inode->i3block);
}
```

算法实现

- **void new_block(int index, int *inode_iblock) 内部**

- 读取当前指针指向的数据块

```
fseek(fin, data_region_start + (*inode_iblock) * block_size, SEEK_SET);
fread(buffer, block_size, 1, fin);
```

- 判断是否为直接块，若是，更新变量

```
if (index == 0) {
        *inode_iblock = data_block_count;
        data_block_count ++;
        size_not_read -= block_size;
}
```

算法实现

- **void new_block(int index, int *inode_iblock) 内部**

- 若为间接块，对其指向的最多块数调用递归函数

```
else {
        for (int i = 0; i < block_size / sizeof(int) && size_not_read > 0; i ++) {
                new_block(index - 1, (buffer + i));
        }
        *inode_iblock = data_block_count;
        data_block_count ++;
}
```

- 将 buffer 读入的块写入输出文件

算法实现

- 将无效的和更新后的 inode 写入输出文件，注意 fout 指针偏移量

```
int offset = ftell(fout);
fseek(fout, 1024 + i * sizeof(Inode), SEEK_SET);
fwrite(inode, sizeof(Inode), 1, fout);
fseek(fout, offset, SEEK_SET);
```

算法实现

- 建立新的空闲块列表并写入输出文件

```
for (i = data_block_count; i < data_block_number; i ++) {
        buffer = (int *)malloc(block_size);
        if (i != data_block_number -1)
                *buffer = i + 1;
        else
                *buffer = -1;
        fwrite(buffer, block_size, 1, fout);
        free(buffer);
}
```

算法实现

- 复制交换区

```
fseek(fin, 0, SEEK_END);
int swap_size = ftell(fin) - (1024 + superblock->swap_offset * block_size);
if (swap_size > 0) {
        char *swap_region = malloc(swap_size);
        fseek(fin, 1024 + superblock->swap_offset * block_size, SEEK_SET);
        fread(swap_region, swap_size, 1, fin);
        fwrite(swap_region, swap_size, 1, fout);
        free(swap_region);
}
```

算法实现

- 更新 superblock 的空闲块列表头

```
fseek(fout, 512, SEEK_SET);
fread(superblock, 512, 1, fout);
superblock->free_iblock = data_block_count;
fseek(fout, 512, SEEK_SET);
fwrite(superblock, 512, 1, fout);
```

- 碎片整理完毕!

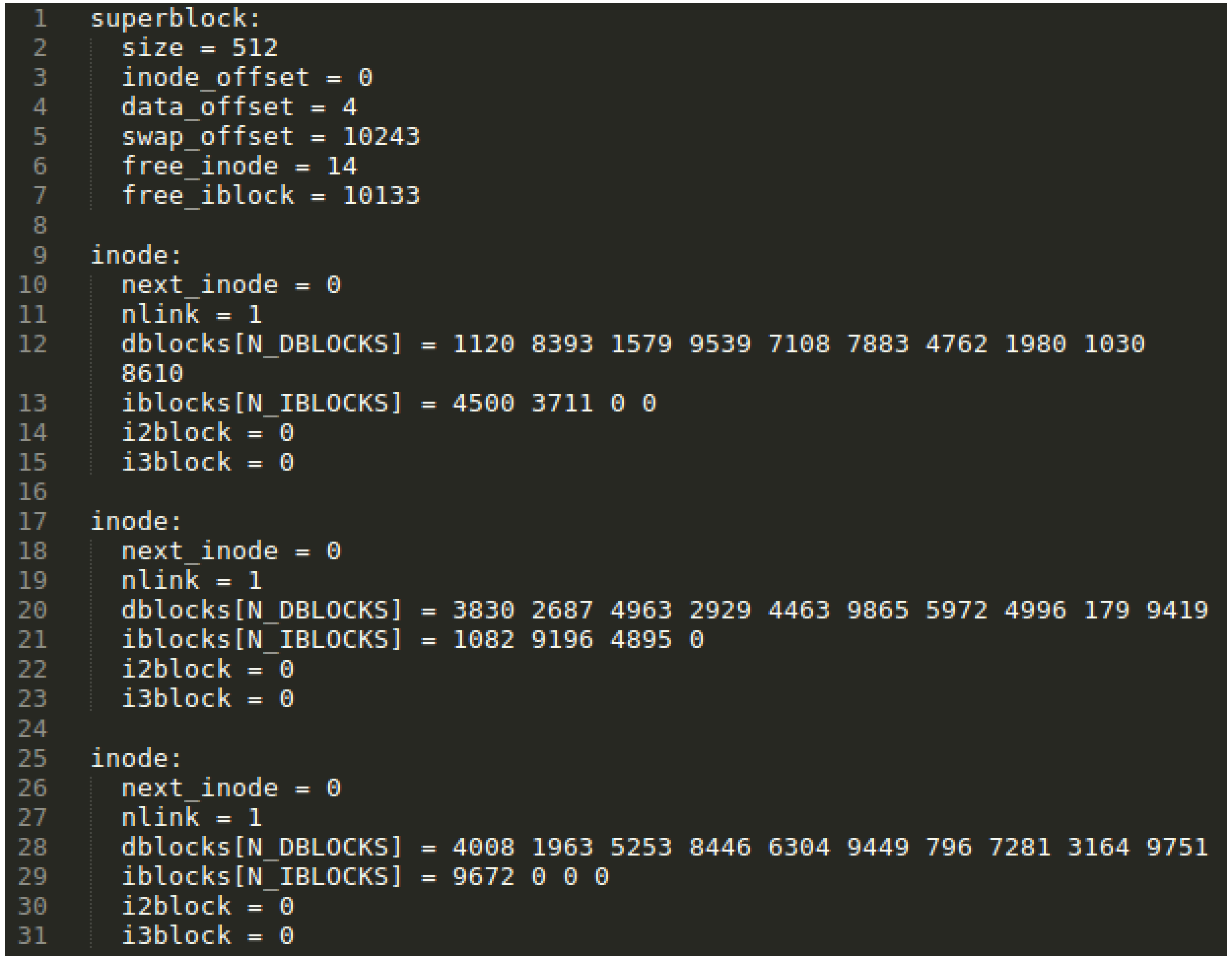
因为示例文件少了 4 个块，所以输出文件比示例文件大 4 * 512B = 2K

PART 4

观察结果

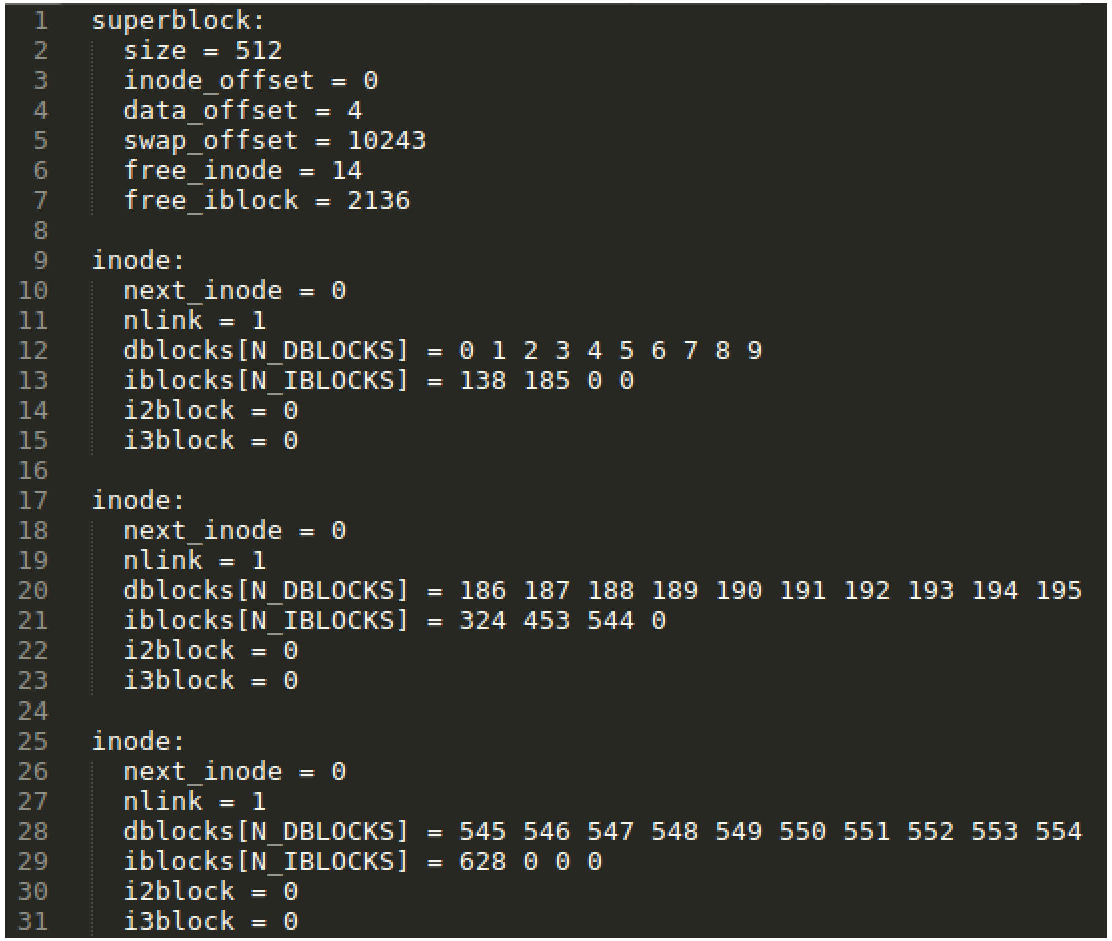
观察结果

check 原文件：



```
superblock:
  size = 512
  inode_offset = 0
  data_offset = 4
  swap_offset = 10243
  free_inode = 14
  free_iblock = 10133

inode:
  next_inode = 0
  nlink = 1
  dblocks[N_DBLOCKS] = 1120 8393 1579 9539 7108 7883 4762 1980 1030 8610
  iblocks[N_IBLOCKS] = 4500 3711 0 0
  i2block = 0
  i3block = 0

inode:
  next_inode = 0
  nlink = 1
  dblocks[N_DBLOCKS] = 3830 2687 4963 2929 4463 9865 5972 4996 179 9419
  iblocks[N_IBLOCKS] = 1082 9196 4895 0
  i2block = 0
  i3block = 0

inode:
  next_inode = 0
  nlink = 1
  dblocks[N_DBLOCKS] = 4008 1963 5253 8446 6304 9449 796 7281 3164 9751
  iblocks[N_IBLOCKS] = 9672 0 0 0
  i2block = 0
  i3block = 0
```

check 输出文件：



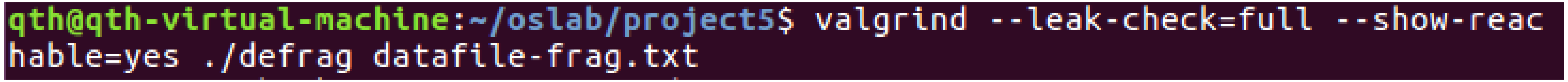
```
superblock:
  size = 512
  inode_offset = 0
  data_offset = 4
  swap_offset = 10243
  free_inode = 14
  free_iblock = 2136

inode:
  next_inode = 0
  nlink = 1
  dblocks[N_DBLOCKS] = 0 1 2 3 4 5 6 7 8 9
  iblocks[N_IBLOCKS] = 138 185 0 0
  i2block = 0
  i3block = 0

inode:
  next_inode = 0
  nlink = 1
  dblocks[N_DBLOCKS] = 186 187 188 189 190 191 192 193 194 195
  iblocks[N_IBLOCKS] = 324 453 544 0
  i2block = 0
  i3block = 0

inode:
  next_inode = 0
  nlink = 1
  dblocks[N_DBLOCKS] = 545 546 547 548 549 550 551 552 553 554
  iblocks[N_IBLOCKS] = 628 0 0 0
  i2block = 0
  i3block = 0
```

PART 5

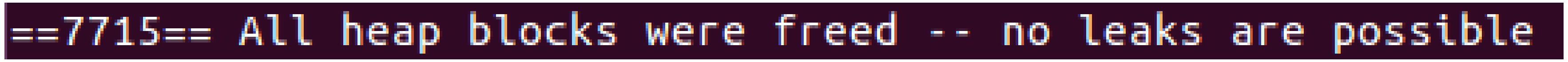
检验内存泄漏

检验内存泄漏

- 用 valgrind 运行 defrag.c

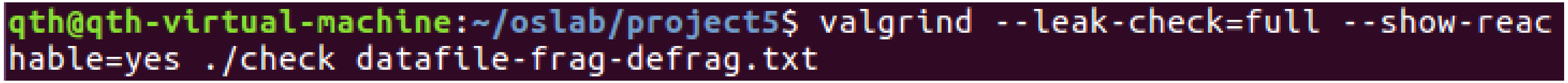


```
qth@qth-virtual-machine:~/oslab/project5$ valgrind --leak-check=full --show-reac
hable=yes ./defrag datafile-frag.txt
```

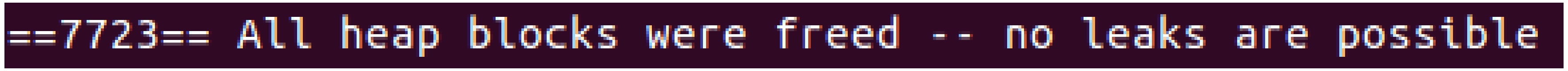


```
==7715== All heap blocks were freed -- no leaks are possible
```

- 用 valgrind 运行 check.c



```
qth@qth-virtual-machine:~/oslab/project5$ valgrind --leak-check=full --show-reac
hable=yes ./check datafile-frag-defrag.txt
```



```
==7723== All heap blocks were freed -- no leaks are possible
```

Thank you