OS Lab Project 5: Defragmentation

10152130113 王玉凤
10152130121 吕波尔
10152130122 钱庭涵

defrag.h

存放 superblock 和 inode 的数据结构

defrag.c

打开输入磁盘映像文件，获取文件名，添加后缀得到输出文件的名称，创建并打开输出磁盘映像文件，打开文件时作错误检查

```
21      if ((fin = fopen(argv[1], "r")) == NULL) {
22          fprintf(stderr, "error: open file <%s> fail\n", argv[1]);
23          exit(1);
24      }
25      strcpy(outfilename, argv[1]);
26      strcpy(outfilename + strlen(argv[1]) - 4, "-defrag.txt");
27      if ((fout = fopen(outfilename, "w+")) == NULL) {
28          fprintf(stderr, "error: open file <%s> fail\n", outfilename);
29          exit(1);
30      }
```

读取输入文件的引导块和 superblock 并复制到输出文件中

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

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

为 inode region 预留内存空间

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

遍历所有 inode，进入循环
因为在后面调用的递归函数中 fin 指针发生过变动，所以在循环刚开始时要先找到当前 inode 存放的位置

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

先判断当前 inode 是否为有效，若有效，开始遍历它的索引块
利用当前 inode 指向的文件大小中还未被读取到的大小，来遍历所有直接块、一级块、二级块、三级块，并调用递归函数 new_block

```
61          if (inode->nlink != 0) {
62              size_not_read = inode->size;
63              for (j = 0; j < N_DBLOCKS &&  size_not_read > 0; j ++)
64                  new_block(0, &inode->dblocks[j]);
65              for (j = 0; j < N_IBLOCKS &&  size_not_read > 0; j ++)
66                  new_block(1, &inode->iblocks[j]);
67              if (size_not_read > 0)
68                  new_block(2, &inode->i2block);
69              if (size_not_read > 0)
70                  new_block(3, &inode->i3block);
71          }
```

index 是目录级数，为 0 表示直接块，为 1 表示一级块，为 2 表示二级块，为 3 表示三级块；*inode_iblock 是 inode 中指向对应数据块的指针

```
118   void new_block(int index, int *inode_iblock)
```

调用 fseek 找到当前指针指向的数据块在内存中的位置，并读取

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

判断 index 是否为 0，即是否为直接块，data_block_count 变量作为数据块的计数
若为直接块，则计数加 1，当前 inode 指向的文件中未被读取到的大小减去一个块大小，并更新指针值为新的数据块存放位置

```
124      if (index == 0) {
125          *inode_iblock = data_block_count;
126          data_block_count ++;
127          size_not_read -= block_size;
128      }
```

若不为直接块，则对于该间接块能够指向的最多块个数（比如示例文件的一个块大小为 512 字节，记录一个整数索引需要 4 字节，那么间接块最多能指向 512 / 4 = 128 个块），调用递归函数，调用时 index 减 1，buffer 为当前读入的块，buffer + i 遍历同一个间接块指向的所有块
因为间接块也需要存放地址，所以遍历该间接块指向的所有块后，计数加 1，并更新指针值

```
129      else {
130          for (int i = 0; i < block_size / sizeof(int) && size_not_read > 0; i ++) {
131              new_block(index - 1, (buffer + i));
132          }
133          *inode_iblock = data_block_count;
134          data_block_count ++;
135      }
```

执行以上 if else 判断及判断内的代码语句之后，将 buffer 读入的块写入输出文件

结束对当前 inode 中所有直接块、一级块、二级块、三级块的遍历之后，此时 inode 指向的数据块已经写入到输出文件的新的位置，且 inode 的索引指针值也更新完毕，则将 inode 写入输出文件，对于无效的 inode 则不经过遍历直接写入输出文件
因为在递归函数内是按照内存顺序写入的，所以在写入 inode 时需要先记录当前 fout 指针偏移量，再将 fout 指向需要写入 inode 的位置，写入完毕之后再将 fout 指回原来的位置

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

结束循环后，此时所有 inode 及存储数据的数据块已经写入完毕，接下来写入剩余的空闲块，并建立空闲块链表

```
80      for (i = data_block_count; i < data_block_number; i ++) {
81          buffer = (int *)malloc(block_size);
82          if (i != data_block_number -1)
83              *buffer = i + 1;
84          else
85              *buffer = -1;
86          fwrite(buffer, block_size, 1, fout);
87          free(buffer);
88      }
```

读取输入文件的交换区并复制到输出文件中，考虑到示例文件出现的问题（实际文件有 10241 个块，但 swap_offset 为 10243，算上引导块和 superblock，即示例文件少了 4 个块），先对于计算出来的 swap_size 进行判断，若大于 0，再进行复制

```
93      if (swap_size > 0) {
94          char *swap_region = malloc(swap_size);
95          fseek(fin, 1024 + superblock->swap_offset * block_size, SEEK_SET);
96          fread(swap_region, swap_size, 1, fin);
97          fwrite(swap_region, swap_size, 1, fout);
98          free(swap_region);
99      }
```

更新 superblock 的空闲块列表头

```
102     fseek(fout, 512, SEEK_SET);
103     fread(superblock, 512, 1, fout);
104     superblock->free_iblock = data_block_count;
105     fseek(fout, 512, SEEK_SET);
106     fwrite(superblock, 512, 1, fout);
```

对于磁盘映像文件的碎片整理完毕！

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 = 2KB

check.c

对于磁盘映像文件，打印 superblock 的相关参数和所有 inode 的部分参数

Makefile

进行编译

进行测试：

```
qth@qth-virtual-machine:~/oslab/project5$ make
gcc -Wall -Werror -o defrag defrag.c defrag.h
gcc -Wall -Werror -o check check.c defrag.h
qth@qth-virtual-machine:~/oslab/project5$ ./defrag datafile-frag.txt
qth@qth-virtual-machine:~/oslab/project5$ ./check datafile-frag.txt > check-inpu
t.txt
qth@qth-virtual-machine:~/oslab/project5$ ./check datafile-frag-defrag.txt > che
ck-output.txt
```

打开 check-input.txt，观察到文件中数据块位置较乱，存在碎片

```
1  superblock:
2    size = 512
3    inode_offset = 0
4    data_offset = 4
5    swap_offset = 10243
6    free_inode = 14
7    free_iblock = 10133
8
9  inode:
10   next_inode = 0
11   nlink = 1
12   dblocks[N_DBLOCKS] = 1120 8393 1579 9539 7108 7883 4762 1980 1030
     8610
13   iblocks[N_IBLOCKS] = 4500 3711 0 0
14   i2block = 0
15   i3block = 0
16
17 inode:
18   next_inode = 0
19   nlink = 1
20   dblocks[N_DBLOCKS] = 3830 2687 4963 2929 4463 9865 5972 4996 179 9419
21   iblocks[N_IBLOCKS] = 1082 9196 4895 0
22   i2block = 0
23   i3block = 0
24
25 inode:
26   next_inode = 0
27   nlink = 1
28   dblocks[N_DBLOCKS] = 4008 1963 5253 8446 6304 9449 796 7281 3164 9751
29   iblocks[N_IBLOCKS] = 9672 0 0 0
30   i2block = 0
31   i3block = 0
```

打开 check-output.txt，观察到文件中数据块位置排列整齐，碎片整理完毕

```
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
```

检验是否有内存泄漏：

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

观察到 defrag.c 无内存泄漏

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

同样，check.c 也无内存泄漏

```
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
```