

**本科实验报告**

课 程 名 称： 操作系统原理实验 课 程 编 号： 08060157 学 生 姓 名： 阮炜霖 学 号： 2020101603 学 院： 信息科学技术学院

系： 计算机科学系

专 业： 网络工程

指 导 教 师： 郝振明

教 师 单 位： 计算机科学系 开 课 时 间：2021 ~ 2022 学年度第 二 学期

## 暨南大学教务处

2022年 6 月 15 日

操作系统原理实验 **课程实验项目目录**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 实验项目 编号 | 实验项目名称 | 实验性质 | 成绩 | 指导教师 |
| 1 | 0806015701 | 实验环境的使用 | 验证 |  | 郝振明 |
| 2 | 0806015702 | 进程的控制 | 设计 |  | 郝振明 |
| 3 | 0806015703 | 进程间的通信 | 设计 |  | 郝振明 |
| 4 | 0806015704 | 生产者消费者问题 | 综合 |  | 郝振明 |
| 5 | 0806015705 | 简单文件系统实现 | 综合 |  | 郝振明 |
| 6 | 0806015706 | 非必须实验。若选择做此项目，则项目名称自定。 |  |  | 郝振明 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

\*此表由学生按顺序填写。

# 暨南大学本科实验报告专用纸

课程名称 操作系统原理实验 成绩评定

实验项目名称 简单文件系统实现 指导教师 郝振明

实验项目编号 0806015705 实验项目类型 设计型

学生姓名 阮炜霖 学号 2020101603

学院信息科学技术学院 系 计算机科学系 专业 网络工程

实验时间 2022 年 5 月 26 日 上 午 3、4节 实验地点 N116

**一、 实验目的**

实验目的：文件管理是操作系统的一个非常重要的组成部分。 学生应独立用高级语言编写和调试一个简单的文件系统，模拟文件管理的工作过程。从而对各种文件操作命令的实质内容和执行过程有比较深入的了解，掌握它们的实施方法，加深理解课堂上讲授过的知识。

1）掌握文件系统的工作原理。

2）理解文件系统的主要数据结构。

3）加深理解文件系统的内部功能和实现方法。

实验要求：任选 C 高级程序语言编写源程序，在 Linux 操作系统下调试通过，测试正确。

**二、 实验内容**

1.实验任务

设计并实现一个多用户的二级文件系统程序，要求提供以下操作。

①文件创建/删除命令： create/ delete  
②目录创建/删除命令： mkdir/rmdir  
③显示目录内容命令：ls

④文件打开/关闭open/ close

⑤文件读/写 read/write

⑥切换当前目录 cd  
拓展：①显示目录内容时要列出文件名和文件长度等信息。

②创建文件，查找空闲块时，不一定使用first\_fit算法，可以使用其他的磁盘调度算法。（如：SCAN、最短寻找时间优先算法等）

2.实验依据

可以在内存中开辟拍一个虚拟磁盘空间作为文件存储器，并将该虚拟文件系统以文件的xingshi保存到磁盘上，以便下次可以将其恢复到内存的虚拟磁盘空间中。对模拟文件系统的操作是对该文件的读写，创建磁盘即为创建该文件。

文件系统以块为基本分配单位，首先需要设置每个块( BLOCK\_SIZE)的大小（如512字  
节）。最开始的第0块为引导块；然后是FAT表，可以分配多个块存储FAT表；接下来就是根目录了。格式化时，根目录中需要增加两个目录项“.”和“..”,分别表示当前目录和上层目录。

需要使用的数据结构包括FAT表、文件控制块、文件打开表等，可参考以下内容进行设计。

①FAT表项中的内容表示文件的下一个盘块号，如果当前块为文件的最后一个盘块（即没有下一个盘块），则相应的表项内容可设置为0xFF( END\_OF\_FILE),表示文件结束。如果FAT表项的内容为0( NOT\_USED),则表示该表项对应的块未被使用。  
 ②每个文件都有一个文件控制块，需要包含文件名、文件属性、文件所在的第一个块的块号、文件大小等属性。  
 ③文件打开表用于记录文件访问的动态变化过程，应包含打开的文件指针、打开文件的权限、文件指针所在的块、文件指针所在块内偏移等信息。

系统中最主要的操作的实现方法可参考以下各项。

(1)文件系统格式化（format）

以读/写方式建立一个新的文件，返回一个文件指针。后续程序可以通过该文件指针对文进行相关的读操作。格式化过程中，文件系统读入内存，只需将位示图中的所有内容置为0,并在根目录中填入“.”和“..”项的内容即可。

(2)创建文件( create).

创建文件时，首先要判断该文件是否已存在，如不存在，再创建文件。创建文件时，首先査找该文件，如果找到表示文件存在，否则为该文件分配空闲的文件控制块，并填写相应的属性等信息。如果创建的是目录文件，则需要为其申请新的一块，并在块中填入“.”和“..”项的内容。由此可见，创建一个普通文件和创建一个目录文件是不同的，普通文件的初始大小为0,初始块号为 END\_OF\_FILE,而目录文件初始化时是有大小的。  
(3)删除文件(rm).  
 与创建文件 create操作类似。  
(4)获取文件控制块( get\_fcb).

这个操作主要根据路径找到它对应的文件的文件控制块。根据路径参数首先找到目录名，从当前查找目录找到该目录，如果找到，则把该目录设为当前查找目录，继续递归查找，直到找到路径的最后一项；如果没找到，则返回错误。

(5)打开文件(open)

首先检查文件是否已经打开。如果没有打开，则由文件对应的路径，利用 get\_fcb操作找到需打开的文件的文件控制块，并加到文件打开表中。文件打开表中的文件指针所在的块及块内偏移，表示当前对打开的文件的操作已经处于该文件所占磁盘空间的某个块的某个字节位置。文件刚打开时初值为0。

(6)关闭文件( close).  
与打开文件open操作类似

(7)文件读操作(read).  
 首先判断文件是否已经打开。若文件已打开，则由 get\_fcb操作找到相应的文件控制块，开始递归读取文件中的内容。读完之后，需要修改文件打开表中相应指针的值。如果读取的字节数超过文件最大长度，则只能读到文件结尾为止。

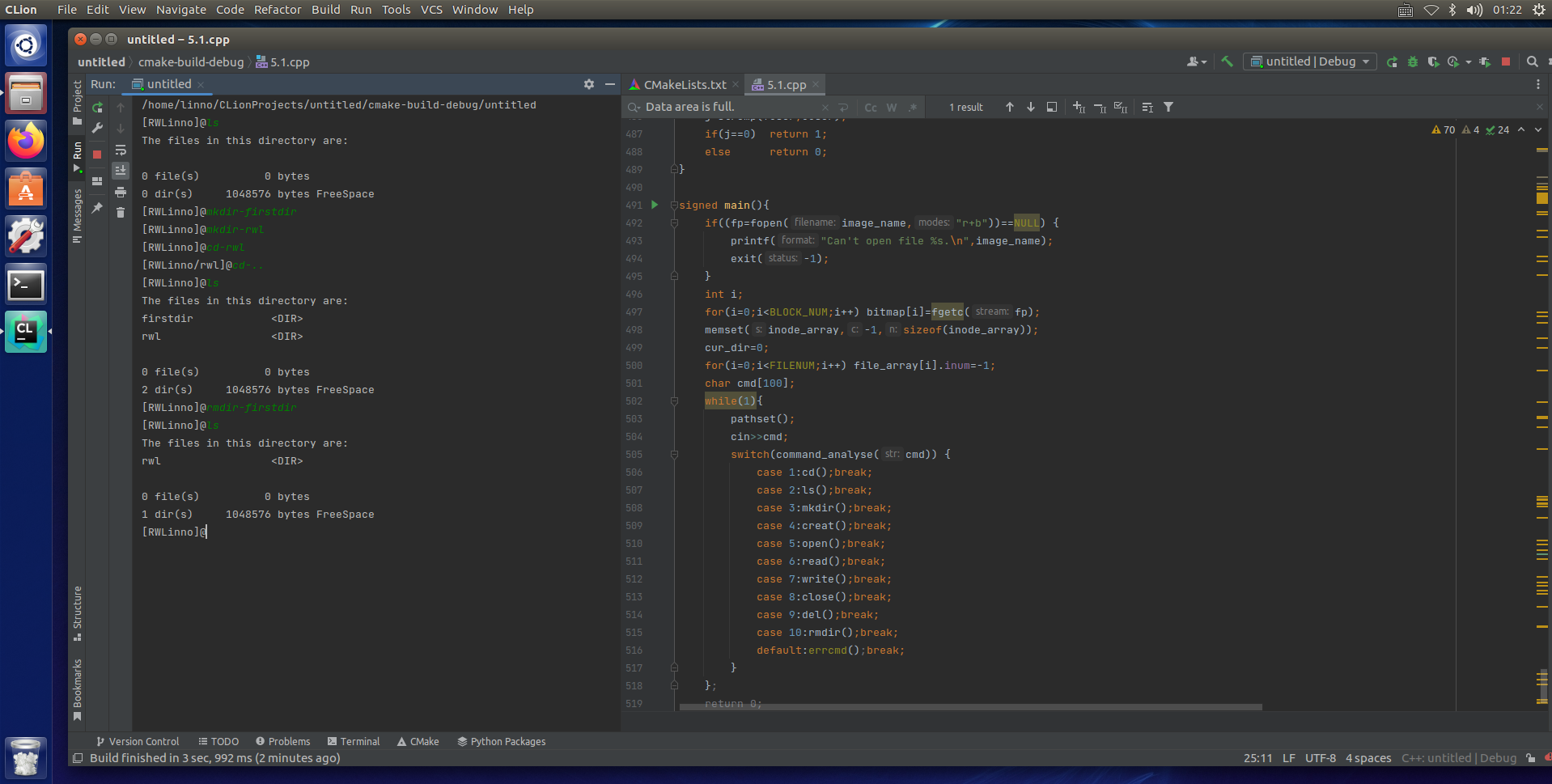
(8)文件写操作( write)

与文件读操作read类似。需注意：新创建的普通文件的大小为0,此时对该文件进行写操作时，必须先申请一个空闲块，再进行递归的写操作。

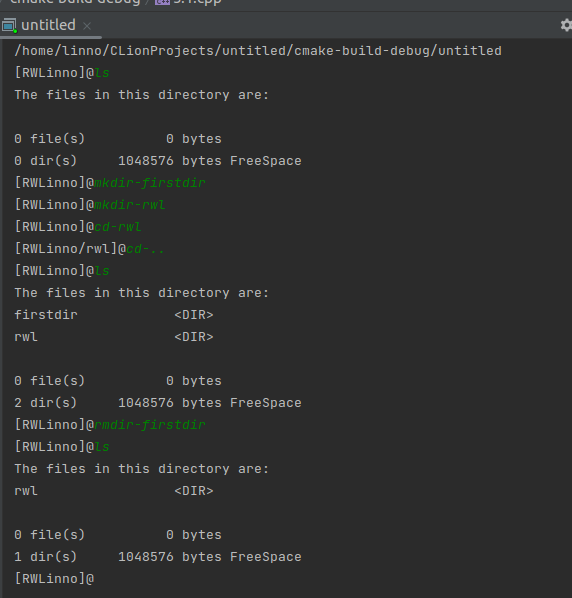
(9)显示当前目录下的文件和目录(1s)

遍历当前目录下所有的目录项，操作方法与 get\_fcb类似也可参考 Unix/Linux系统中的多级索引方式来设计文件的物理结构，请自行完成相关设计内容。

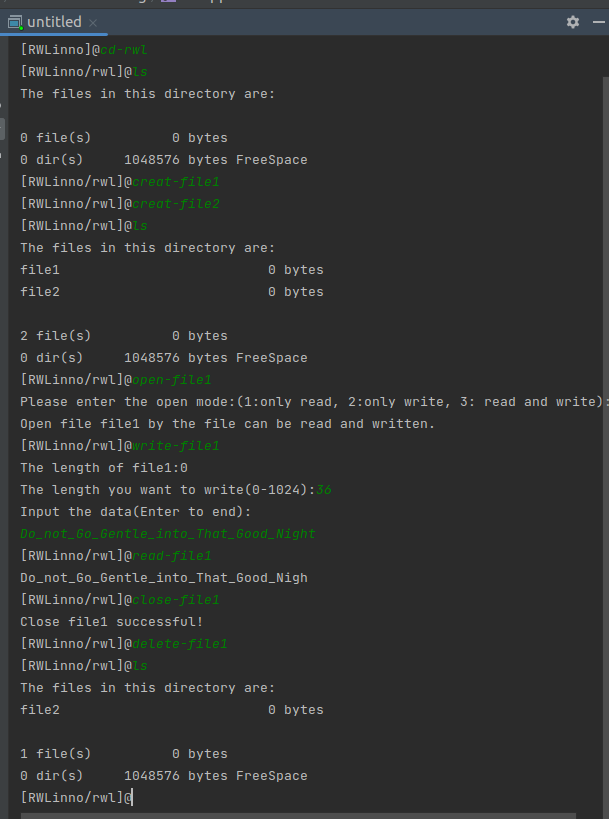
## 三、 实验步骤

该实验代码量较大，笔者用操作系统再结合c++文件操作的知识先Windows环境下编写了程序，然后再移植到linux系统环境下，因为一些库函数的方法发生了改变，再通过Clion进行修改并完成编译和运行。

Part1.关于文件系统的层次结构实现、虚拟空间的分配和创建/删除目录效果



Part2.关于文件管理和文件读写操作



Part3.对于一些违规操作的处理

1. 输入未知指令：退出程序
2. 指令参数数量不对或操作对象不存在：返回并重新输入指令
3. 找不到足够进行操作的剩余空间：异常退出程序
4. 跨目录操作文件或文件夹：输出“not yours”提示并返回。
5. 在只读文件中写或者在只写文件中读，输出“Can’t”提示并返回。
6. 不打开文件就进行读写操作，输出“Open Fitst”提示并返回。
7. 找不到文件系统，异常退出程序。
8. 再次打开某个未关闭的文件文件，输出“Can’t”提示并返回

## 五、附录

程序 5-1.cpp

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. #define BLOCK\_SIZE    2048        // 数据块的大小
4. #define BLOCK\_NUM    2048        // 数据块的块数
5. #define INODE\_NUM   128        // i节点的数目
6. #define FILENUM    8        // 打开文件表的数目
7. #define errcmd() printf("This command is not allowed\n");
9. **typedef** **struct**{
10. **char** user\_name[10]="RWLinno"; // 用户名
11. }User;
13. // i节点(32B)
14. **typedef** **struct** {
15. **short** inum;          // 文件i节点号
16. **char**  file\_name[10]; // 文件名
17. **char**  type;          // 文件类型
18. **char**  user\_name[10]; // 文件所有者
19. **short** iparent;       // 父目录的i节点号
20. **short** length;        // 文件长度
21. **short** address[2];    // 存放文件的地址
22. } Inode;
24. // 打开文件表(16B)
25. **typedef** **struct**{
26. **short** inum;             // i节点号
27. **char**  file\_name[10]; // 文件名
28. **short** mode;             // 读写模式(1:read,2:write,3:read and write)
29. **short** offset;        // 偏移量
30. } File\_table;
32. **int** argc;                           // 命令的参数的个数
33. **char** \*argv[5];                      // 命令的参数
34. **int** cur\_dir;                        // 当前目录
35. **char** the\_buffer[2\*BLOCK\_SIZE];      // 缓冲区
36. User user;                          // 当前用户
37. **char** bitmap[BLOCK\_NUM];             // 位示图数组
38. Inode inode\_array[INODE\_NUM];       // i节点数组
39. File\_table file\_array[FILENUM];     // 打开文件表数组
40. **char** image\_name[10]="file.dat";     // 文件系统名称
41. **FILE** \*fp;                           // 打开文件指针
43. //函数声明
44. **int**  analyse(**char** \*);  //命令分析
45. **void** save\_inode(**int**);  //保存i结点
46. **int** get\_BLOCK\_NUM(**void**); //获取块数
47. **void** pathset();        //设置路径
48. **void** delet(**int** innum); //删除文件
49. **int**  check(**int** i); //是否为根目录
50. **void** cd(**void**);     //跳转到某目录
51. **void** ls(**void**);     //显示当前目录下的文件
52. **void** mkdir(**void**);  //新建目录
53. **void** rmdir(**void**);  //删除目录
54. **void** creat(**void**);  //新建文件
55. **void** del(**void**);    //删除文件
56. **void** open(**void**);   //打开文件
57. **void** read(**void**);   //读文件
58. **void** write(**void**);  //写文件
59. **void** close(**void**);  //关闭文件
60. **void** read\_blk(**int**);
61. **void** write\_blk(**int**);
62. **void** release\_blk(**int**);
64. //命令解析：
65. **int** command\_analyse(**char** \*str){
66. **int** i;
67. **char** the\_buffer[20];
68. **char** \*ptr\_char;
69. **char** \*syscmd[]={"","cd","ls","mkdir","creat","open","read","write","close","delete","rmdir"};   //, "logout", "clear","format","quit","rmdir"};
70. argc = 0;
71. **for**(i = 0, ptr\_char = str; \*ptr\_char != '\0'; ptr\_char++) {
72. **if**(\*ptr\_char!= '-'){
73. **while**(\*ptr\_char != '-' && (\*ptr\_char != '\0'))
74. the\_buffer[i++] = \*ptr\_char++;
75. argv[argc] = (**char** \*)malloc(i+1);
76. strncpy(argv[argc], the\_buffer, i);
77. argv[argc][i] = '\0';
78. argc++;
79. i = 0;
80. **if**(\*ptr\_char=='\0') **break**;
81. }
82. }
83. **if**(argc != 0){
84. **for**(i = 0;(i<12)&&strcmp(argv[0],syscmd[i]);i++);
85. **return** i;
86. } **else** **return** 12;
87. }
89. //文件创建/删除命令
90. **void** creat(**void**){
91. **int** i;
92. **if**(argc!=2){
93. printf("command creat must have one args. \n");
94. **return**;
95. }
96. **for**(i=1;i<INODE\_NUM;i++){
97. **if**((inode\_array[i].inum > 0) && (inode\_array[i].type == '-') && !strcmp(inode\_array[i].file\_name, argv[1])) {
98. printf("The file has been exsited.\n");
99. **return**;
100. }
101. }
102. **for**(i=1;i<INODE\_NUM;i++)
103. **if**(inode\_array[i].inum<0) **break**;
104. **if**(i==INODE\_NUM){
105. printf("There is no else INode.\n");
106. **return**;
107. }
108. inode\_array[i].inum=i;
109. strcpy(inode\_array[i].file\_name,argv[1]);
110. inode\_array[i].type='-';
111. strcpy(inode\_array[i].user\_name,user.user\_name);
112. inode\_array[i].iparent=cur\_dir;
113. inode\_array[i].length=0;
114. save\_inode(i);
115. }
116. **void** delet(**int** innum) {
117. inode\_array[innum].inum = -1;
118. **if**(inode\_array[innum].length >= 0) {
119. release\_blk(inode\_array[innum].address[0]);
120. **if**(inode\_array[innum].length >= 512)
121. release\_blk(inode\_array[innum].address[1]);
122. }
123. save\_inode(innum);
124. }
125. **void** del(**void**) {
126. **int** i,chk;
127. **if**(argc != 2) {
128. printf("Command delete must have one args. \n");
129. **return** ;
130. }
131. **for**(i = 0; i < INODE\_NUM; i++)
132. **if**((inode\_array[i].inum > 0) && (inode\_array[i].type == '-') && !strcmp(inode\_array[i].file\_name, argv[1]))
133. **break**;
134. **if**(i == INODE\_NUM) {
135. printf("This file doesn't exist.\n");
136. **return** ;
137. }
138. chk=check(i);
139. **if**(chk!=1) {
140. printf("This file is not yours !\n");
141. **return** ;
142. }
143. delet(i);
144. }
146. //目录创建/删除命令
147. **void** mkdir(){
148. **int** i;
149. **if**(argc!=2){
150. printf("command mkdir must have two args. \n");
151. **return** ;
152. }
154. **for**(i=1;i<INODE\_NUM;i++) **if**(inode\_array[i].inum<0) **break**;
155. **if**(i==INODE\_NUM){
156. printf("Inode is full.\n");
157. exit(-1);
158. }
159. inode\_array[i].inum = i;
160. strcpy(inode\_array[i].file\_name, argv[1]);
161. inode\_array[i].type = 'd';
162. strcpy(inode\_array[i].user\_name,user.user\_name);
163. inode\_array[i].iparent = cur\_dir;
164. inode\_array[i].length = 0;
165. save\_inode(i);
166. }
168. **void** rmdir(){
169. **int** i,j,t,flag=0;
170. **if**(argc != 2) {
171. printf("Command delete must have one args. \n");
172. **return** ;
173. }
174. **for**(i = 0; i < INODE\_NUM; i++)
175. **if**((inode\_array[i].inum > 0) && (inode\_array[i].iparent == cur\_dir) && (inode\_array[i].type == 'd')&&(!strcmp(inode\_array[i].file\_name,argv[1]))) {
176. **int** chk=check(i);
177. **if**(chk!=1) {
178. printf("This directory is not your !\n");
179. **return** ;
180. } **else** j=inode\_array[i].inum;
181. **for**(t=0; t<INODE\_NUM; t++) {
182. **if**((inode\_array[t].inum>0)&&(inode\_array[t].iparent==j)&&(inode\_array[i].type == '-'))
183. delet(t);
184. **else** **if**((inode\_array[t].inum>0)&&(inode\_array[t].iparent==j)&&(inode\_array[i].type == 'd'))
185. delet(t);
186. }
187. **if**(t == INODE\_NUM)
188. delet(j);
189. }
190. **if**(i == INODE\_NUM)
191. delet(i);
192. **return**;
193. }
195. //显示目录内容命令
196. **void** ls(){
197. printf("The files in this directory are:\n");
198. **int** i;
199. **int** dcount=0,fcount=0;
200. **short** bcount=0;
201. **if**(argc != 1) {
202. printf("Command ls must have one args. \n");
203. **return** ;
204. }
205. **for**(i = 0; i < INODE\_NUM; i++)
206. **if**((inode\_array[i].inum> 0) && (inode\_array[i].iparent == cur\_dir)) {
207. **if**(inode\_array[i].type == 'd' && check(i)) {
208. dcount++;
209. printf("%-20s<DIR>\n", inode\_array[i].file\_name);
210. }
211. **if**(inode\_array[i].type == '-' && check(i)) {
212. fcount++;
213. bcount+=inode\_array[i].length;
214. printf("%-20s%12d bytes\n", inode\_array[i].file\_name,inode\_array[i].length);
215. }
216. }
217. printf("\n%d file(s)%11d bytes\n",fcount,bcount);
218. printf("%d dir(s) %11d bytes FreeSpace\n",dcount,1024\*1024-bcount);
219. }
221. //文件打开/关闭open/ close
222. **void** open() {
223. **int** i, inum, mode, filenum,chk;
224. **if**(argc != 2) {
225. printf("command open must have one args. \n");
226. **return** ;
227. }
228. **for**(i = 1; i < INODE\_NUM; i++)
229. **if**((inode\_array[i].inum > 0) && (inode\_array[i].type == '-') && !strcmp(inode\_array[i].file\_name,argv[1]))
230. **break**;
231. **if**(i == INODE\_NUM) {
232. printf("The file you want to open doesn't exsited.\n");
233. **return** ;
234. }
235. inum = i;
236. chk=check(i);
237. **if**(chk!=1) {
238. printf("This file is not your !\n");
239. **return** ;
240. }
241. printf("Please enter the open mode:(1:only read, 2:only write, 3: read and write):");
242. scanf("%d", &mode);
243. cin>>the\_buffer;
244. **if**((mode < 1) || (mode > 3)) {
245. printf("the number you entered is wrong.\n");
246. **return**;
247. }
248. **for**(i = 1; i < FILENUM; i++)
249. **if**(file\_array[i].inum < 0) **break**;
250. **if**(i == FILENUM) {
251. printf("The file table is full, please close some file.\n");
252. **return** ;
253. }
254. filenum = i;
255. file\_array[filenum].inum = inum;
256. strcpy(file\_array[filenum].file\_name, inode\_array[inum].file\_name);
257. file\_array[filenum].mode = mode;
258. file\_array[filenum].offset = 0;
259. printf("Open file %s by ", file\_array[filenum].file\_name);
260. **if**(mode == 1) printf("the file can be only read.\n");
261. **else** **if**(mode == 2) printf("the file can be only written.\n");
262. **else** printf("the file can be read and written.\n");
263. }
265. **void** close(**void**) {
266. **int** i;
267. **if**(argc != 2) {
268. printf("Command close must have one args. \n");
269. **return** ;
270. }
271. **for**(i = 0; i < FILENUM; i++)
272. **if**((file\_array[i].inum > 0) && !strcmp(file\_array[i].file\_name, argv[1]))
273. **break**;
274. **if**(i == FILENUM) {
275. printf("This file doesn't be opened.\n");
276. **return** ;
277. } **else** {
278. file\_array[i].inum = -1;
279. printf("Close %s successful!\n", argv[1]);
280. }
281. }
283. //文件读/写 read/write
284. **void** read() {
285. **int** i, start=0, num, inum;
286. **if**(argc != 2) {
287. printf("command read must have one args. \n");
288. **return**;
289. }
290. **for**(i = 0; i < FILENUM; i++)
291. **if**((file\_array[i].inum > 0) && !strcmp(file\_array[i].file\_name,argv[1]))
292. **break**;
293. **if**(i == FILENUM) {
294. printf("Open %s first.\n", argv[1]);
295. **return** ;
296. } **else** **if**(file\_array[i].mode == 2) {
297. printf("Can't read %s.\n", argv[1]);
298. **return** ;
299. }
300. inum = file\_array[i].inum;
301. **if**(inode\_array[inum].length > 0) {
302. cin>>the\_buffer;
303. read\_blk(inum);
304. **for**(i = 0; the\_buffer[i] != '\0'; i++)
305. printf("%c", the\_buffer[i]);
306. printf("\n");
307. }
308. }
309. **void** write() {
310. **int** i, inum, length=0;
311. **if**(argc != 2) {
312. printf("Command write must have one args. \n");
313. **return** ;
314. }
315. **for**(i = 0; i < FILENUM; i++)
316. **if**((file\_array[i].inum>0)&&!strcmp(file\_array[i].file\_name,argv[1]))
317. **break**;
318. **if**(i == FILENUM) {
319. printf("Open %s first.\n", argv[1]);
320. **return** ;
321. } **else** **if**(file\_array[i].mode == 1) {
322. printf("Can't write %s.\n", argv[1]);
323. **return** ;
324. }
325. inum = file\_array[i].inum;
326. printf("The length of %s:%d\n", inode\_array[inum].file\_name, inode\_array[inum].length);
327. **if**(inode\_array[inum].length == 0) {
328. printf("The length you want to write(0-1024):");
329. scanf("%d", &length);
330. **if**((length < 0) && (length >1024)) {
331. printf("Input wrong.\n");
332. **return** ;
333. }
334. inode\_array[inum].length = length;
335. //inode\_array[inum].address[0] = get\_BLOCK\_NUM();
336. //if(length > 512)
337. inode\_array[inum].address[1]=get\_BLOCK\_NUM();
338. save\_inode(inum);
339. printf("Input the data(Enter to end):\n");
340. cin>>the\_buffer;
341. write\_blk(inum);
342. } **else**
343. printf("This file can't be written.\n");
344. }
346. //切换当前目录
347. **void** cd(**void**) {
348. **int** i;
349. **if**(argc != 2) {
350. printf("Command cd must have two args. \n");
351. **return** ;
352. }
353. **if**(!strcmp(argv[1], ".."))
354. cur\_dir = inode\_array[cur\_dir].iparent;
355. **else** {
356. // 遍历i节点数组
357. **for**(i = 0; i < INODE\_NUM; i++)
358. **if**((inode\_array[i].inum>0)&&(inode\_array[i].type=='d')&&(inode\_array[i].iparent==cur\_dir)&&!strcmp(inode\_array[i].file\_name,argv[1])&&check(i))
359. **break**;
360. **if**(i == INODE\_NUM)
361. printf("This directory isn't exsited.\n");
362. **else**
363. cur\_dir = i;
364. }
365. }
367. **void** save\_inode(**int** num){
368. **if**((fp=fopen(image\_name,"r+b"))==NULL){
369. printf("Can't open file %s\n", image\_name);
370. exit(-1);
371. }
372. fseek(fp, 512+num\***sizeof**(Inode), SEEK\_SET);
373. fwrite(&inode\_array[num], **sizeof**(Inode), 1, fp);
374. fclose(fp);
375. }
377. **int** get\_BLOCK\_NUM(**void**){
378. **int** i;
379. **for**(i=1;i<BLOCK\_NUM;i++) **if**(bitmap[i]=='0') **break**;
380. **if**(i==BLOCK\_NUM){
381. printf("Data area is full.\n");
382. exit(-1);
383. }
384. bitmap[i]='1';
385. **if**((fp=fopen(image\_name, "r+b"))==NULL) {
386. printf("Can't open file %s\n",image\_name);
387. exit(-1);
388. }
389. fseek(fp,i,SEEK\_SET);
390. fputc('1',fp);
391. fclose(fp);
392. **return** i;
393. }
395. **void** read\_blk(**int** num) {
396. **int** i,len;
397. **char** ch;
398. **int** add0,add1;
399. len=inode\_array[num].length;
400. add0=inode\_array[num].address[0];
401. **if**(len>512) add1 = inode\_array[num].address[1];
402. **if**((fp=fopen(image\_name, "r+b"))==NULL){
403. printf("Can't open file %s.\n",image\_name);
404. exit(-1);
405. }
406. fseek(fp, 1536+add0\*BLOCK\_SIZE, SEEK\_SET);
407. ch = fgetc(fp);
408. **for**(i=0;(i<len)&&(ch!='\0')&&(i<512);i++) {
409. the\_buffer[i] = ch;
410. ch = fgetc(fp);
411. }
412. **if**(i >= 512) {
413. fseek(fp, 1536+add1\*BLOCK\_SIZE, SEEK\_SET);
414. ch = fgetc(fp);
415. **for**(; (i < len) && (ch != '\0'); i++) {
416. the\_buffer[i] = ch;
417. ch = fgetc(fp);
418. }
419. }
420. the\_buffer[i] = '\0';
421. fclose(fp);
422. }
424. **void** write\_blk(**int** num) {
425. **int**  i, len;
426. **int**  add0, add1;
427. add0 = inode\_array[num].address[0];
428. len  = inode\_array[num].length;
429. **if**((fp = fopen(image\_name, "r+b")) == NULL) {
430. printf("Can't open file %s.\n", image\_name);
431. exit(-1);
432. }
433. fseek(fp, 1536+add0\*BLOCK\_SIZE, SEEK\_SET);
434. **for**(i=0; (i<len)&&(the\_buffer[i]!='\0')&&(i < 512); i++)
435. fputc(the\_buffer[i], fp);
436. **if**(i == 512) {
437. add1 = inode\_array[num].address[1];
438. fseek(fp, 1536+add1\*BLOCK\_SIZE, SEEK\_SET);
439. **for**(; (i < len) && (the\_buffer[i] != '\0'); i++)
440. fputc(the\_buffer[i], fp);
441. }
442. fputc('\0', fp);
443. fclose(fp);
444. }
446. **void** release\_blk(**int** num){
447. **FILE** \*fp;
448. **if**((fp=fopen(image\_name, "r+b")) == NULL) {
449. printf("Can't open file %s\n", image\_name);
450. exit(-1);
451. }
452. bitmap[num] = '0';
453. fseek(fp, num, SEEK\_SET);
454. fputc('0', fp);
455. fclose(fp);
456. }
458. **void** pathset() {
459. **char** path[50];
460. **int** m,n;
461. **if**(inode\_array[cur\_dir].inum == 0)
462. strcpy(path,user.user\_name);
463. **else** {
464. strcpy(path,user.user\_name);
465. m=0;
466. n=cur\_dir;
467. **while**(m != cur\_dir) {
468. **while**(inode\_array[n].iparent != m) {
469. n = inode\_array[n].iparent;
470. }
471. strcat(path,"/");
472. strcat(path,inode\_array[n].file\_name);
473. m = n;
474. n = cur\_dir;
475. }
476. }
477. printf("[%s]@",path);
478. }
480. **int** check(**int** i){
481. **int** j;
482. **char** \*uuser,\*fuser;
483. uuser=user.user\_name;
484. fuser=inode\_array[i].user\_name;
485. j=strcmp(fuser,uuser);
486. **if**(j==0)  **return** 1;
487. **else**      **return** 0;
488. }
490. **signed** main(){
491. **if**((fp=fopen(image\_name,"r+b"))==NULL) {
492. printf("Can't open file %s.\n",image\_name);
493. exit(-1);
494. }
495. **int** i;
496. **for**(i=0;i<BLOCK\_NUM;i++) bitmap[i]='0';
497. memset(inode\_array,-1,**sizeof**(inode\_array));
498. cur\_dir=0;
499. **for**(i=0;i<FILENUM;i++) file\_array[i].inum=-1;
500. **char** cmd[100];
501. **while**(1){
502. pathset();
503. cin>>cmd;
504. **switch**(command\_analyse(cmd)) {
505. **case** 1:cd();**break**;
506. **case** 2:ls();**break**;
507. **case** 3:mkdir();**break**;
508. **case** 4:creat();**break**;
509. **case** 5:open();**break**;
510. **case** 6:read();**break**;
511. **case** 7:write();**break**;
512. **case** 8:close();**break**;
513. **case** 9:del();**break**;
514. **case** 10:rmdir();**break**;
515. **default**:errcmd();**break**;
516. }
517. };
518. **return** 0;
519. }