



**Unix与linux操作系统课程设计1**

|  |  |  |
| --- | --- | --- |
| 项目名称 | **：** | **文件系统** |
| 组员 | **：** |  |
| 班级 | **：** |  |
| 指导教师 | **：** |  |
| 完成时间 | **：** |  |

目录

[1.项目成员分工表 4](#_Toc57752868)

[2.开发设计文档 4](#_Toc57752869)

**[2.1定义的常量](#_Toc57752870)** [4](#_Toc57752870)

**[2.2数据结构](#_Toc57752871)** [4](#_Toc57752871)

**[2.3全局变量定义](#_Toc57752872)** [6](#_Toc57752872)

**[2.5系统主函数main()](#_Toc57752873)** [7](#_Toc57752873)

**[2.6进入文件系统函数my\_startsys()](#_Toc57752874)** [13](#_Toc57752874)

**[2.7磁盘格式化函数my\_format()](#_Toc57752875)** [15](#_Toc57752875)

**[2.8创建子目录函数my\_mkdir()](#_Toc57752876)** [17](#_Toc57752876)

**[2.9更改当前目录函数my\_cd()](#_Toc57752877)** [22](#_Toc57752877)

**[2.10删除子目录函数my\_rmdir()](#_Toc57752878)** [25](#_Toc57752878)

**[2.11显示目录函数my\_ls()](#_Toc57752879)** [28](#_Toc57752879)

**[2.12创建文件函数my\_create()](#_Toc57752880)** [30](#_Toc57752880)

**[2.13申请新的空闲磁盘块函数get\_free\_block()](#_Toc57752881)** [33](#_Toc57752881)

**[2.14申请新的打开文件列表函数get\_free\_openfilelist()](#_Toc57752882)** [34](#_Toc57752882)

**[2.15删除文件函数my\_rm()](#_Toc57752883)** [35](#_Toc57752883)

**[2.16打开文件函数my\_open()](#_Toc57752884)** [37](#_Toc57752884)

[2.17](#_Toc57752885)**[关闭文件函数my\_close()](#_Toc57752885)** [38](#_Toc57752885)

**[2.19实际写文件函数do\_write()](#_Toc57752886)** [42](#_Toc57752886)

**[2.20读文件函数my\_read()](#_Toc57752887)** [45](#_Toc57752887)

**[2.22退出文件系统函数my\_exitsys()](#_Toc57752888)** [47](#_Toc57752888)

**1.项目成员分工表**

|  |  |
| --- | --- |
| **成员** | **分工** |
| **张艺腾** | 代码设计与调试、开发文档排版 |
| **罗琪源** | 代码设计与调试、开发文档排版 |
| **万嘉正** | 代码测试、开发文档编写 |
| **刘爱军** | 代码测试、开发文档编写 |

**2.开发设计文档(测试截图已经包含)**

**2.1定义的常量**

#define BLOCKSIZE 1024 // 一个块的大小

#define SIZE 1024000

#define END 65535

#define FREE 0

#define ROOTBLOCKNUM 2

#define MAXOPENFILE 10 // 同时打开最大文件数（表项数目）

#define MAX\_TEXT\_SIZE 10000

对系统在磁盘中的一些使用数据先进行定义，方便后续使用，根据系统设计思路添加文件最大数加以控制。

**2.2数据结构**

开辟虚拟磁盘空间进行文件存储分区，我们仿造FAT16文件系统进行设计。

typedef struct FCB {

char filename[8]; //文件名

char exname[3]; //文件扩展名

unsigned char attribute; // 0: dir file, 1: data file 文件属性字段（目录或文件）

unsigned short time; //创建时间

unsigned short date; //创建日期

unsigned short first; //起始盘块号

unsigned short length; //文件长度

char free; // 0: 空 fcb //表示目录项是否为空

} fcb;

对文件和目录的信息进行了上述设计。我们同样设计了用户打开文件表来对有关文件操作的动态信息进行记录。

typedef struct USEROPEN {

//内存FCB表的所有属性

char filename[8]; //文件名

char exname[3]; //文件扩展名

unsigned char attribute; //文件属性：值为0时表示目录文件，值为1时表示数据文件

unsigned short time; //文件创建时间

unsigned short date; //文件创建日期

unsigned short first; //文件起始盘块号

unsigned short length; //文件长度（对数据文件是字节数，对目录文件可以是目录项个数）

char free; //表示目录项是否为空，若值为0，表示空，值为1，表示已分配

int dirno; // 父目录文件起始盘块号（父目录指针位置）

int diroff; // 该文件对应的 fcb 在父目录中的逻辑序号（子目录在父目录中的指针位置）

char dir[MAXOPENFILE][80]; // 全路径信息

int count;

char fcbstate; // 是否修改 1是 0否

char topenfile; // 0: 空 openfile

} useropen;

文件分配表记录了FAT中的信息，包括是否空闲和块号，每个FAT占据两个磁盘块。

typedef struct FAT {

unsigned short id; //fat表的信息

} fat;

在结构的引导块存储磁盘的相关信息。

typedef struct BLOCK {

char magic\_number[8]; //文件魔数

char information[200]; //磁盘块大小

unsigned short root; //根目录文件开始位置

unsigned char\* startblock; ////数据区开始位置

} block0;

**2.3全局变量定义**

extern unsigned char\* myvhard; //指向虚拟磁盘的起始地点

extern useropen openfilelist[MAXOPENFILE]; //打开文件表数组

extern int currfd; //方便记录指针当前位置

extern unsigned char\* startp; //记录虚拟磁盘上数据区开始位置

extern unsigned char buffer[SIZE]; //当前目录名

**2.4虚拟磁盘空间布局**

系统申请内存作为虚拟磁盘，一共划分1000个磁盘块，每个块1024个字节，引导块占一个盘块，两张FAT各占两个盘块，剩余为数据区，并将数据区第一块分配给根目录。如图所示：

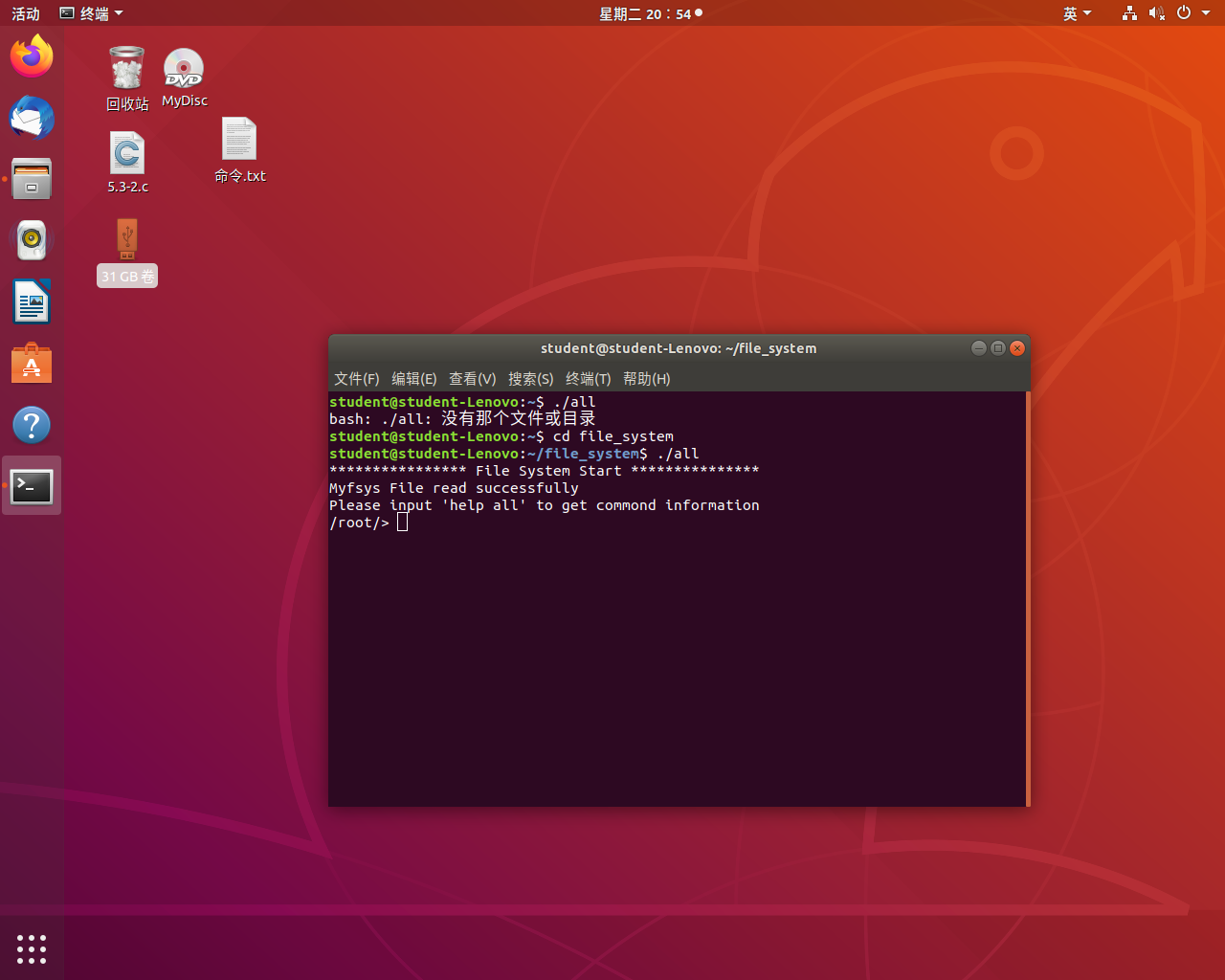
块数 1块 2块 2块 1块 994块

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 引导块 | FAT1 | FAT2 | 根目录 | 其他数据区 |

| 数据区 |

**2.5系统主函数main()**

系统主函数main()文件中首先定义函数void test()，这个函数是用来测试文件系统是否正确启动。Main()中定义数组dict用于存放定义的命令字符，然后调用my\_startsys()来进入文件系统，创建识别函数同时对命令正确错误的情况进行定义，若正确则根据输入的命令查找函数中的指令调用相关的文件进行处理，如输入read后，主函数将调用my\_read()函数来处理命令。若指令错误，则根据指令格式予以错误提醒。



1. #include "head.h"
3. //debug func
4. **void** test()
5. {
6. **int** openfile\_num = 0;
7. **int** i;
8. printf("debug area ############\n");
9. // print open file number
10. **for** (i = 0; i < MAXOPENFILE; i++) {
11. **if** (openfilelist[i].topenfile == 1) {
12. openfile\_num++;
13. }
14. }
15. printf("  open file number: %d\n", openfile\_num);
16. printf("  curr file name: %s\n", openfilelist[currfd].filename);
17. printf("  curr file length: %d\n", openfilelist[currfd].length);
19. printf("debug end  ############\n");
20. }
22. **int** main(**void**)
23. {
25. **char** dict[13][20] = {
26. "mkdir", "rmdir", "ls", "cd", "create",
27. "rm", "open", "close", "write", "read",
28. "exit", "help", "test"
29. };
30. **char** command[30], \*sp;
31. **int** cmd\_idx, i;
32. printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* File System Start \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");
33. my\_startsys();
35. **while** (1) {
36. printf("%s> ", openfilelist[currfd].dir);
37. gets(command);
38. cmd\_idx = -1;
39. **if** (strcmp(command, "") == 0) {
40. printf("\n");
41. **continue**;
42. }
43. sp = strtok(command, " ");
44. **for** (i = 0; i < 13; i++) {
45. **if** (strcmp(sp, dict[i]) == 0) {
46. cmd\_idx = i;
47. **break**;
48. }
49. }
50. **switch** (cmd\_idx) {
51. **case** 0: // mkdir
52. sp = strtok(NULL, " ");
53. **if** (sp != NULL)
54. my\_mkdir(sp);
55. **else**
56. printf("mkdir error\n");
57. sp = strtok(NULL, " ");
58. **if** (sp != NULL)
59. printf("\nwarning:you only need to INPUT mkdir  dir\_name\n");
60. **break**;
61. **case** 1: // rmdir
62. sp = strtok(NULL, " ");
63. **if** (sp != NULL)
64. my\_rmdir(sp);
65. **else**
66. printf("rmdir error\n");
67. sp = strtok(NULL, " ");
68. **if** (sp != NULL)
69. printf("\nwarning:you only need to INPUT rmdir dir\_name\n");
70. **break**;
71. **case** 2: // ls
72. my\_ls();
73. sp = strtok(NULL, " ");
74. **if** (sp != NULL)
75. printf("\nwarning:you only need to INPUT ls\n");
76. **break**;
77. **case** 3: // cd
78. sp = strtok(NULL, " ");
79. **if** (sp != NULL)
80. my\_cd(sp);
81. **else**
82. printf("cd error\n");
83. sp = strtok(NULL, " ");
84. **if** (sp != NULL)
85. printf("\nwarning:you only need to INPUT cd dir\_name\n");
86. **break**;
87. **case** 4: // create
88. sp = strtok(NULL, " ");
89. **if** (sp != NULL)
90. my\_create(sp);
91. **else**
92. printf("create error\n");
93. sp = strtok(NULL, " ");
94. **if** (sp != NULL)
95. printf("\nwarning:you only need to INPUT create file\_name\n");
96. **break**;
97. **case** 5: // rm
98. sp = strtok(NULL, " ");
99. **if** (sp != NULL)
100. my\_rm(sp);
101. **else**
102. printf("rm error\n");
103. sp = strtok(NULL, " ");
104. **if** (sp != NULL)
105. printf("\nwarning:you only need to INPUT rm file\_name\n");
106. **break**;
107. **case** 6: // open
108. sp = strtok(NULL, " ");
109. **if** (sp != NULL)
110. my\_open(sp);
111. **else**
112. printf("open error\n");
113. sp = strtok(NULL, " ");
114. **if** (sp != NULL)
115. printf("\nwarning:you only need to INPUT open file\_name\n");
116. **break**;
117. **case** 7: // close
118. **if** (openfilelist[currfd].attribute == 1)
119. my\_close(currfd);
120. **else**
121. printf("there is not openning file\n");
122. sp = strtok(NULL, " ");
123. **if** (sp != NULL)
124. printf("\nwarning:you only need to INPUT close fd\n");
125. **break**;
126. **case** 8: // write
127. **if** (openfilelist[currfd].attribute == 1)
128. my\_write(currfd);
129. **else**
130. printf("please open file first, then write\n");
131. sp = strtok(NULL, " ");
132. **if** (sp != NULL)
133. printf("\nwarning:you only need to INPUT write fd\n");
134. **break**;
135. **case** 9: // read
136. **if** (openfilelist[currfd].attribute == 1)
137. my\_read(currfd);
138. **else**
139. printf("please open file first, then read\n");
140. sp = strtok(NULL, " ");
141. **if** (sp != NULL)
142. printf("\nwarning:you only need to INPUT read fd\n");
143. **break**;
144. **case** 10: // exit
145. my\_exitsys();
146. **return** 0;
147. **break**;
148. **case** 11: // help
149. sp = strtok(NULL, " ");
150. **if** (sp != NULL)
151. help(sp);
152. **else**
153. printf("rm error\n");
154. sp = strtok(NULL, " ");
155. **if** (sp != NULL)
156. printf("\nwarning:you only need to INPUT help function\_name\n");
157. **break**;
158. **case** 12: // test
159. test();
160. **break**;
161. **default**:
162. printf("wrong command: %s\n", command);
163. **break**;
164. }
165. }
166. **return** 0;
167. }

**2.6进入文件系统函数my\_startsys()**

这个函数用于进入文件系统，如果文件系统存在则载入相关的root目录，如果不存在就调用my\_format格式化磁盘来创建文件系统。主要判断手段是通过FILENAEM是否存在且开头是否为魔数，判断成功后定义fcb指针root来载入根目录初始信息，如文件名，拓展名，创建日期等等。

1. #include "head.h"
3. **void** my\_startsys()
4. {
5. /\*\*
6. \* 如果存在文件系统（存在 FILENAME 这个文件 且 开头为魔数）则
7. \* 将 root 目录载入打开文件表。
8. \* 否则，调用 my\_format 创建文件系统，再载入。
9. \*/
10. myvhard = (unsigned **char**\*)malloc(SIZE);
11. **FILE**\* file;
12. **if** ((file = fopen(FILENAME, "r")) != NULL) {
13. fread(buffer, SIZE, 1, file);
14. fclose(file);
15. **if** (memcmp(buffer, "10101010", 8) == 0) {
16. memcpy(myvhard, buffer, SIZE);
17. printf("Myfsys File read successfully\n");
18. printf("Please input 'help all' to get commond information\n");
19. } **else** {
20. printf("invalid myfsys magic num, create file system\n");
21. my\_format();
22. memcpy(buffer, myvhard, SIZE);
23. }
24. } **else** {
25. printf("Myfsys not find, create file system\n");
26. my\_format();
27. memcpy(buffer, myvhard, SIZE);
28. }
30. fcb\* root;
31. root = (fcb\*)(myvhard + 5 \* BLOCKSIZE);
32. strcpy(openfilelist[0].filename, root->filename);
33. strcpy(openfilelist[0].exname, root->exname);
34. openfilelist[0].attribute = root->attribute;
35. openfilelist[0].time = root->time;
36. openfilelist[0].date = root->date;
37. openfilelist[0].first = root->first;
38. openfilelist[0].length = root->length;
39. openfilelist[0].free = root->free;
41. openfilelist[0].dirno = 5;
42. openfilelist[0].diroff = 0;
43. strcpy(openfilelist[0].dir, "/root/");
44. openfilelist[0].count = 0;
45. openfilelist[0].fcbstate = 0;
46. openfilelist[0].topenfile = 1;
48. startp = ((block0\*)myvhard)->startblock;
49. currfd = 0;
50. **return**;
51. }

**2.7磁盘格式化函数my\_format()**

创建引导块boot指针，创建文件系统FILENAME,赋予魔数10101010，然后写入到磁盘内存空间，初始化前五个磁盘块建立两个FAT表，将前五定为已用，后面定为空闲，然后将第六个磁盘块指定为根目录，，创建根目录.和..。然后读取localtime改变格式定义为文件系统和根目录.及..的创立时间。

1. #include "head.h"
3. **void** my\_format()
4. {
5. /\*\*
6. \* 初始化前五个磁盘块
7. \* 设定第六个磁盘块为根目录磁盘块
8. \* 初始化 root 目录： 创建 . 和 .. 目录
9. \* 写入 FILENAME 文件 （写入磁盘空间）
10. \*/
11. block0\* boot = (block0\*)myvhard;
12. strcpy(boot->magic\_number, "10101010");
13. strcpy(boot->information, "fat file system");
14. boot->root = 5;
15. boot->startblock = myvhard + BLOCKSIZE \* 5;
17. fat\* fat1 = (fat\*)(myvhard + BLOCKSIZE);
18. fat\* fat2 = (fat\*)(myvhard + BLOCKSIZE \* 3);
19. **int** i;
20. **for** (i = 0; i < 6; i++) {
21. fat1[i].id = END;
22. fat2[i].id = END;
23. }
24. **for** (i = 6; i < 1000; i++) {
25. fat1[i].id = FREE;
26. fat2[i].id = FREE;
27. }
29. // 5th block is root
30. fcb\* root = (fcb\*)(myvhard + BLOCKSIZE \* 5);
31. strcpy(root->filename, ".");
32. strcpy(root->exname, "di");
33. root->attribute = 0; // dir file
35. **time\_t** rawTime = time(NULL);
36. **struct** **tm**\* time = localtime(&rawTime);
37. // 5 6 5 bits
38. root->time = time->tm\_hour \* 2048 + time->tm\_min \* 32 + time->tm\_sec / 2;
39. // 7 4 5 bits; year from 2000
40. root->date = (time->tm\_year - 100) \* 512 + (time->tm\_mon + 1) \* 32 + (time->tm\_mday);
41. root->first = 5;
42. root->free = 1;
43. root->length = 2 \* **sizeof**(fcb);
45. fcb\* root2 = root + 1;
46. memcpy(root2, root, **sizeof**(fcb));
47. strcpy(root2->filename, "..");
48. **for** (i = 2; i < (**int**)(BLOCKSIZE / **sizeof**(fcb)); i++) {
49. root2++;
50. strcpy(root2->filename, "");
51. root2->free = 0;
52. }
54. **FILE**\* fp = fopen(FILENAME, "w");
55. fwrite(myvhard, SIZE, 1, fp);
56. fclose(fp);
57. }

**2.8创建子目录函数my\_mkdir()**

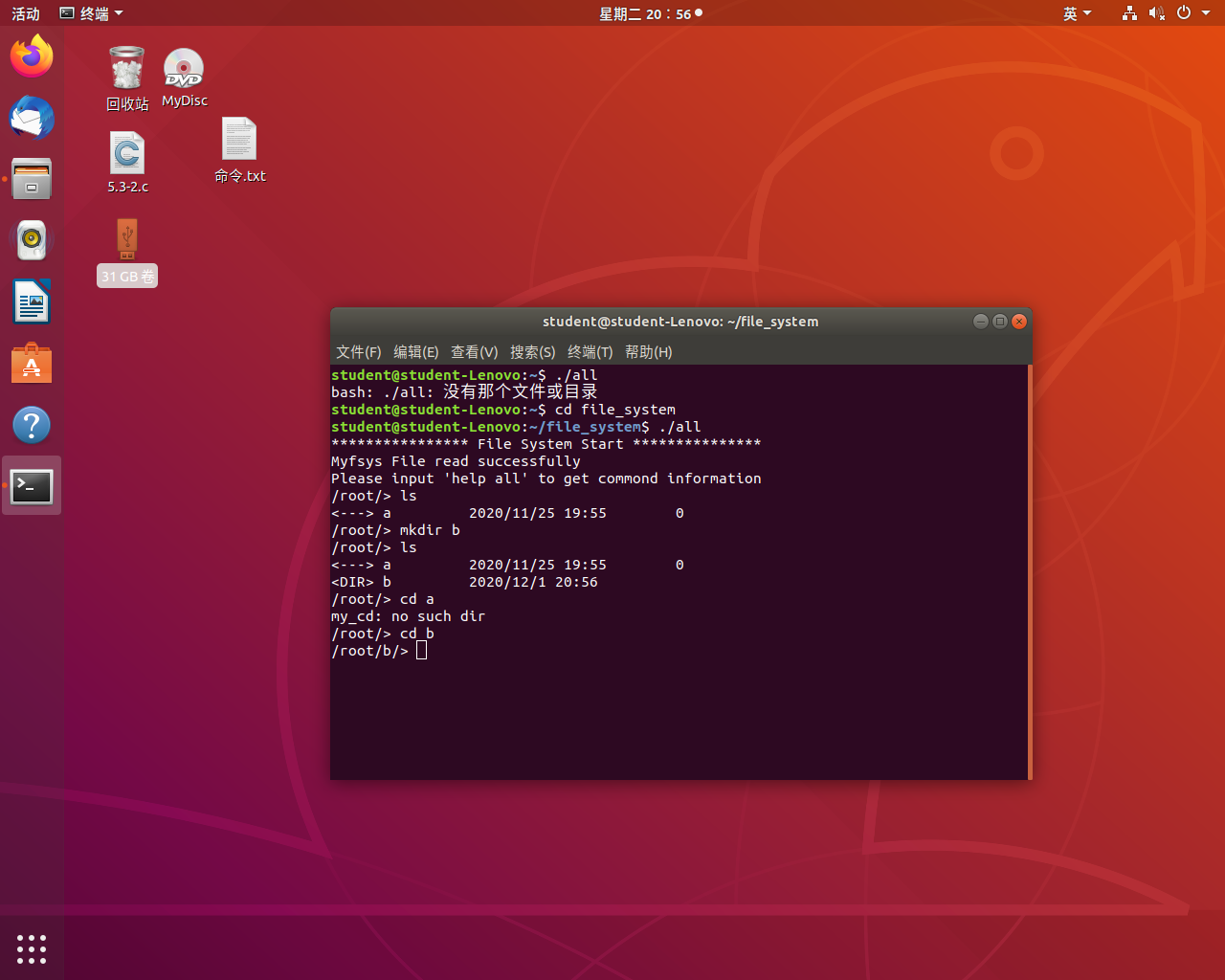
my\_mkdir函数用于创建子目录，通过调用do\_read函数读取当前的父目录信息，输入信息中的新建目录名用来在父目录中进行重名检查，若文件名已经在父目录中存在，则错误信息并返回。若不存在则在父目录下申请新的目录表项，申请磁盘块，更新FAT表，将父目录中空的fcb分配给该目录，并初始化新目录的信息，读取时间作为创建时间，还会将目录路径上的所有目录信息进行更新修改，包括.和..目录等。



1. #include "head.h"
3. **void** my\_mkdir(**char**\* dirname)
4. {
5. /\*\*
6. \* 当前目录：当前打开目录项表示的目录
7. \* 该目录：以下指创建的目录
8. \* 父目录：指该目录的父目录
9. \* 如:
10. \* 我现在在 root 目录下， 输入命令 mkdir a/b/bb
11. \* 表示 在 root 目录下的 a 目录下的 b 目录中创建 bb 目录
12. \* 这时，父目录指 b，该目录指 bb，当前目录指 root
13. \* 以下都用这个表达，简单情况下，当前目录和父目录是一个目录
14. \* 来不及了，先讨论简单情况，即 mkdir bb
15. \*/
16. **int** i = 0;
17. **char** text[MAX\_TEXT\_SIZE];
19. **char**\* fname = strtok(dirname, ".");
20. **char**\* exname = strtok(NULL, ".");
21. **if** (exname != NULL) {
22. printf("you can not use extension\n");
23. **return**;
24. }
25. // 读取父目录信息
26. openfilelist[currfd].count = 0;
27. **int** filelen = do\_read(currfd, openfilelist[currfd].length, text);
28. fcb\* fcbptr = (fcb\*)text;
30. // 查找是否重名
31. **for** (i = 0; i < (**int**)(filelen / **sizeof**(fcb)); i++) {
32. **if** (strcmp(dirname, fcbptr[i].filename) == 0 && fcbptr->attribute == 0) {
33. printf("dir has existed\n");
34. **return**;
35. }
36. }
38. // 申请一个打开目录表项
39. **int** fd = get\_free\_openfilelist();
40. **if** (fd == -1) {
41. printf("openfilelist is full\n");
42. **return**;
43. }
44. // 申请一个磁盘块
45. unsigned **short** **int** block\_num = get\_free\_block();
46. **if** (block\_num == END) {
47. printf("blocks are full\n");
48. openfilelist[fd].topenfile = 0;
49. **return**;
50. }
51. // 更新 fat 表
52. fat\* fat1 = (fat\*)(myvhard + BLOCKSIZE);
53. fat\* fat2 = (fat\*)(myvhard + BLOCKSIZE \* 3);
54. fat1[block\_num].id = END;
55. fat2[block\_num].id = END;
57. // 在父目录中找一个空的 fcb，分配给该目录  ??未考虑父目录满的情况??
58. **for** (i = 0; i < (**int**)(filelen / **sizeof**(fcb)); i++) {
59. **if** (fcbptr[i].free == 0) {
60. **break**;
61. }
62. }
63. openfilelist[currfd].count = i \* **sizeof**(fcb);
64. openfilelist[currfd].fcbstate = 1;
65. // 初始化该 fcb
66. fcb\* fcbtmp = (fcb\*)malloc(**sizeof**(fcb));
67. fcbtmp->attribute = 0;
68. **time\_t** rawtime = time(NULL);
69. **struct** **tm**\* time = localtime(&rawtime);
70. fcbtmp->date = (time->tm\_year - 100) \* 512 + (time->tm\_mon + 1) \* 32 + (time->tm\_mday);
71. fcbtmp->time = (time->tm\_hour) \* 2048 + (time->tm\_min) \* 32 + (time->tm\_sec) / 2;
72. strcpy(fcbtmp->filename, dirname);
73. strcpy(fcbtmp->exname, "di");
74. fcbtmp->first = block\_num;
75. fcbtmp->length = 2 \* **sizeof**(fcb);
76. fcbtmp->free = 1;
77. do\_write(currfd, (**char**\*)fcbtmp, **sizeof**(fcb), 2);
79. // 设置打开文件表项
80. openfilelist[fd].attribute = 0;
81. openfilelist[fd].count = 0;
82. openfilelist[fd].date = fcbtmp->date;
83. openfilelist[fd].time = fcbtmp->time;
84. openfilelist[fd].dirno = openfilelist[currfd].first;
85. openfilelist[fd].diroff = i;
86. strcpy(openfilelist[fd].exname, "di");
87. strcpy(openfilelist[fd].filename, dirname);
88. openfilelist[fd].fcbstate = 0;
89. openfilelist[fd].first = fcbtmp->first;
90. openfilelist[fd].free = fcbtmp->free;
91. openfilelist[fd].length = fcbtmp->length;
92. openfilelist[fd].topenfile = 1;
93. strcat(strcat(strcpy(openfilelist[fd].dir, (**char**\*)(openfilelist[currfd].dir)), dirname), "/");
95. // 设置 . 和 .. 目录
96. fcbtmp->attribute = 0;
97. fcbtmp->date = fcbtmp->date;
98. fcbtmp->time = fcbtmp->time;
99. strcpy(fcbtmp->filename, ".");
100. strcpy(fcbtmp->exname, "di");
101. fcbtmp->first = block\_num;
102. fcbtmp->length = 2 \* **sizeof**(fcb);
103. do\_write(fd, (**char**\*)fcbtmp, **sizeof**(fcb), 2);
105. fcb\* fcbtmp2 = (fcb\*)malloc(**sizeof**(fcb));
106. memcpy(fcbtmp2, fcbtmp, **sizeof**(fcb));
107. strcpy(fcbtmp2->filename, "..");
108. fcbtmp2->first = openfilelist[currfd].first;
109. fcbtmp2->length = openfilelist[currfd].length;
110. fcbtmp2->date = openfilelist[currfd].date;
111. fcbtmp2->time = openfilelist[currfd].time;
112. do\_write(fd, (**char**\*)fcbtmp2, **sizeof**(fcb), 2);
114. // 关闭该目录的打开文件表项，close 会修改父目录中对应该目录的 fcb 信息
115. /\*\*
116. \* 这里注意，一个目录存在 2 个 fcb 信息，一个为该目录下的 . 目录文件，一个为父目录下的 fcb。
117. \* 因此，这俩个fcb均需要修改，前一个 fcb 由各个函数自己完成，后一个 fcb 修改由 close 完成。
118. \* 所以这里，需要打开文件表，再关闭文件表，实际上更新了后一个 fcb 信息。
119. \*/
120. my\_close(fd);
122. free(fcbtmp);
123. free(fcbtmp2);
125. // 修改父目录 fcb
126. fcbptr = (fcb\*)text;
127. fcbptr->length = openfilelist[currfd].length;
128. openfilelist[currfd].count = 0;
129. do\_write(currfd, (**char**\*)fcbptr, **sizeof**(fcb), 2);
130. openfilelist[currfd].fcbstate = 1;
131. }

**2.9更改当前目录函数my\_cd()**

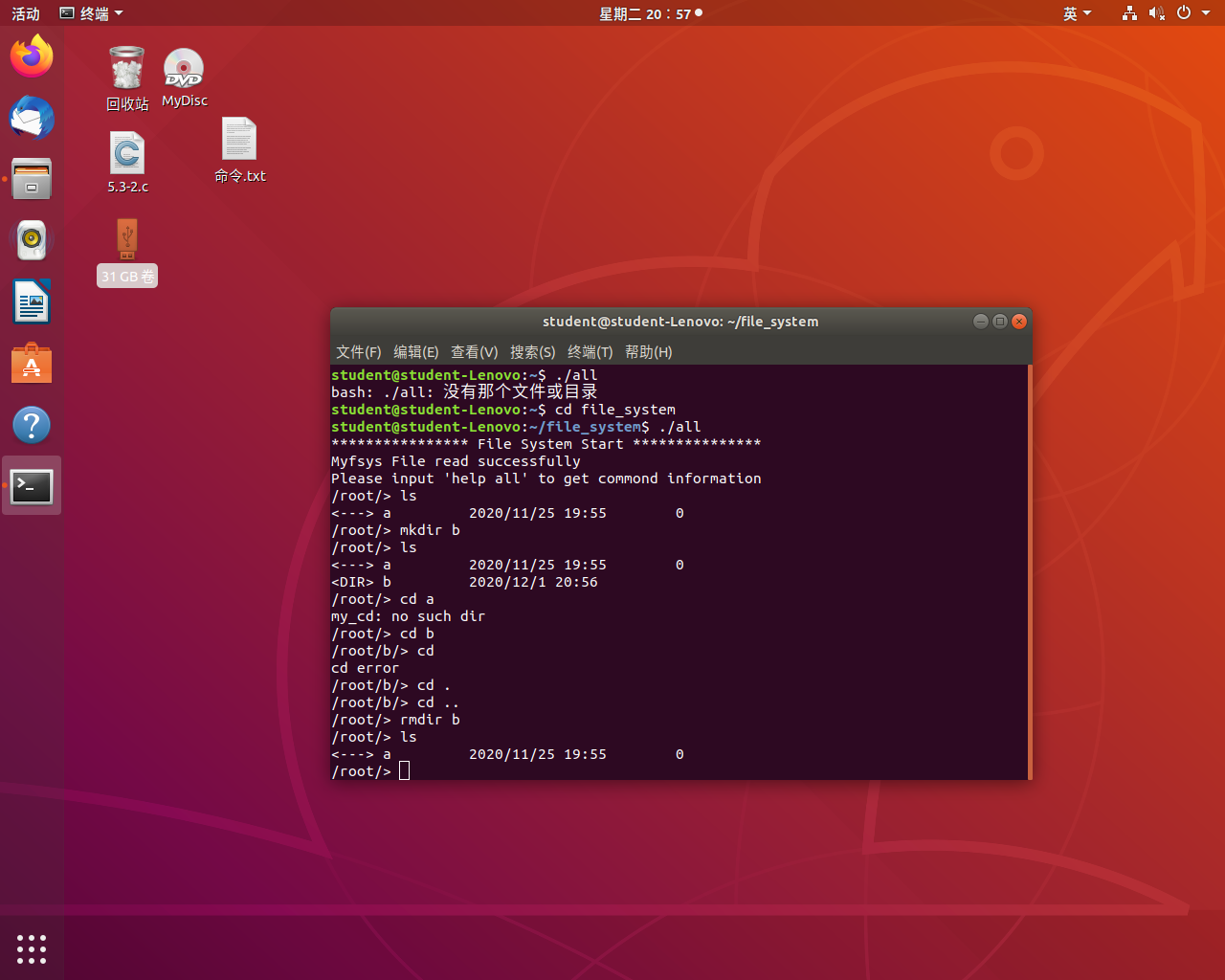
my\_cd函数用于对当前目录进行更改，从而进入新的目录下。首先会对当前位置的dir进行判断，若不为dir类型则返回错误信息，否则调用do\_read函数读取命令cd后的目录路径。然后根据路径读取相关的fcb信息直到进入目的目录，若未查找到相关的fcb信息，则输出路径错误并返回。



1. #include "head.h"
3. **void** my\_cd(**char**\* dirname)
4. {
5. **int** i = 0;
6. **int** tag = -1;
7. **int** fd;
8. **if** (openfilelist[currfd].attribute == 1) {
9. // if not a dir
10. printf("you are in a data file, you could use 'my\_close' to exit this file\n");
11. **return**;
12. }
13. **char**\* buf = (**char**\*)malloc(10000);
14. openfilelist[currfd].count = 0;
15. do\_read(currfd, openfilelist[currfd].length, buf);
17. fcb\* fcbptr = (fcb\*)buf;
18. // 查找目标 fcb
19. **for** (i = 0; i < (**int**)(openfilelist[currfd].length / **sizeof**(fcb)); i++, fcbptr++) {
20. **if** (strcmp(fcbptr->filename, dirname) == 0 && fcbptr->attribute == 0) {
21. tag = 1;
22. **break**;
23. }
24. }
25. **if** (tag != 1) {
26. printf("my\_cd: no such dir\n");
27. **return**;
28. } **else** {
29. // . 和 .. 检查
30. **if** (strcmp(fcbptr->filename, ".") == 0) {
31. **return**;
32. } **else** **if** (strcmp(fcbptr->filename, "..") == 0) {
33. **if** (currfd == 0) {
34. // root
35. **return**;
36. } **else** {
37. currfd = my\_close(currfd);
38. **return**;
39. }
40. } **else** {
41. // 其他目录
42. fd = get\_free\_openfilelist();
43. **if** (fd == -1) {
44. **return**;
45. }
46. openfilelist[fd].attribute = fcbptr->attribute;
47. openfilelist[fd].count = 0;
48. openfilelist[fd].date = fcbptr->date;
49. openfilelist[fd].time = fcbptr->time;
50. strcpy(openfilelist[fd].filename, fcbptr->filename);
51. strcpy(openfilelist[fd].exname, fcbptr->exname);
52. openfilelist[fd].first = fcbptr->first;
53. openfilelist[fd].free = fcbptr->free;
55. openfilelist[fd].fcbstate = 0;
56. openfilelist[fd].length = fcbptr->length;
57. strcat(strcat(strcpy(openfilelist[fd].dir, (**char**\*)(openfilelist[currfd].dir)), dirname), "/");
58. openfilelist[fd].topenfile = 1;
59. openfilelist[fd].dirno = openfilelist[currfd].first;
60. openfilelist[fd].diroff = i;
61. currfd = fd;
62. }
63. }
64. }

**2.10删除子目录函数my\_rmdir()**

my\_rmdir函数用于删除某个空的子目录。调用do\_read函数读取命令rmdir后的子目录名称，排除.和..这两个不能删除的目录的可能性，然后在fcb查找子目录的名称，如果没有则报错返回，如果查找到了根据目录的长度信息中是否为空判断能否删除，不为空则返回错误信息，为空则更新删除fcb中的信息，包括父目录的长度等信息。



1. #include "head.h"
3. **void** my\_rmdir(**char**\* dirname)
4. {
5. **int** i, tag = 0;
6. **char** buf[MAX\_TEXT\_SIZE];
8. // 排除 . 和 .. 目录
9. **if** (strcmp(dirname, ".") == 0 || strcmp(dirname, "..") == 0) {
10. printf("can not remove . and .. special dir\n");
11. **return**;
12. }
13. openfilelist[currfd].count = 0;
14. do\_read(currfd, openfilelist[currfd].length, buf);
16. // 查找要删除的目录
17. fcb\* fcbptr = (fcb\*)buf;
18. **for** (i = 0; i < (**int**)(openfilelist[currfd].length / **sizeof**(fcb)); i++, fcbptr++) {
19. **if** (fcbptr->free == 0)
20. **continue**;
21. **if** (strcmp(fcbptr->filename, dirname) == 0 && fcbptr->attribute == 0) {
22. tag = 1;
23. **break**;
24. }
25. }
26. **if** (tag != 1) {
27. printf("no such dir\n");
28. **return**;
29. }
30. // 无法删除非空目录
31. **if** (fcbptr->length > 2 \* **sizeof**(fcb)) {
32. printf("can not remove a non empty dir\n");
33. **return**;
34. }
36. // 更新 fat 表
37. **int** block\_num = fcbptr->first;
38. fat\* fat1 = (fat\*)(myvhard + BLOCKSIZE);
39. **int** nxt\_num = 0;
40. **while** (1) {
41. nxt\_num = fat1[block\_num].id;
42. fat1[block\_num].id = FREE;
43. **if** (nxt\_num != END) {
44. block\_num = nxt\_num;
45. } **else** {
46. **break**;
47. }
48. }
49. fat1 = (fat\*)(myvhard + BLOCKSIZE);
50. fat\* fat2 = (fat\*)(myvhard + BLOCKSIZE \* 3);
51. memcpy(fat2, fat1, BLOCKSIZE \* 2);
53. // 更新 fcb
54. fcbptr->date = 0;
55. fcbptr->time = 0;
56. fcbptr->exname[0] = '\0';
57. fcbptr->filename[0] = '\0';
58. fcbptr->first = 0;
59. fcbptr->free = 0;
60. fcbptr->length = 0;
62. openfilelist[currfd].count = i \* **sizeof**(fcb);
63. do\_write(currfd, (**char**\*)fcbptr, **sizeof**(fcb), 2);
65. // 删除目录需要相应考虑可能删除 fcb，也就是修改父目录 length
66. // 这里需要注意：因为删除中间的 fcb，目录有效长度不变，即 length 不变
67. // 因此需要考虑特殊情况，即删除最后一个 fcb 时，极有可能之前的 fcb 都是空的，这是需要
68. // 循环删除 fcb (以下代码完成)，可能需要回收 block 修改 fat 表等过程(do\_write 完成)
69. **int** lognum = i;
70. **if** ((lognum + 1) \* **sizeof**(fcb) == openfilelist[currfd].length) {
71. openfilelist[currfd].length -= **sizeof**(fcb);
72. lognum--;
73. fcbptr = (fcb \*)buf + lognum;
74. **while** (fcbptr->free == 0) {
75. fcbptr--;
76. openfilelist[currfd].length -= **sizeof**(fcb);
77. }
78. }
80. // 更新父目录 fcb
81. fcbptr = (fcb\*)buf;
82. fcbptr->length = openfilelist[currfd].length;
83. openfilelist[currfd].count = 0;
84. do\_write(currfd, (**char**\*)fcbptr, **sizeof**(fcb), 2);
86. openfilelist[currfd].fcbstate = 1;
87. }

**2.11显示目录函数my\_ls()**

my\_ls函数用于显示当前目录下的所有子目录。首先根据标识判断是否为目录，然后调用do\_read函数将当前目录的fcb所有信息读取到内存，然后从当前目录开始遍历目录下的所有子目录，并将子目录展示。

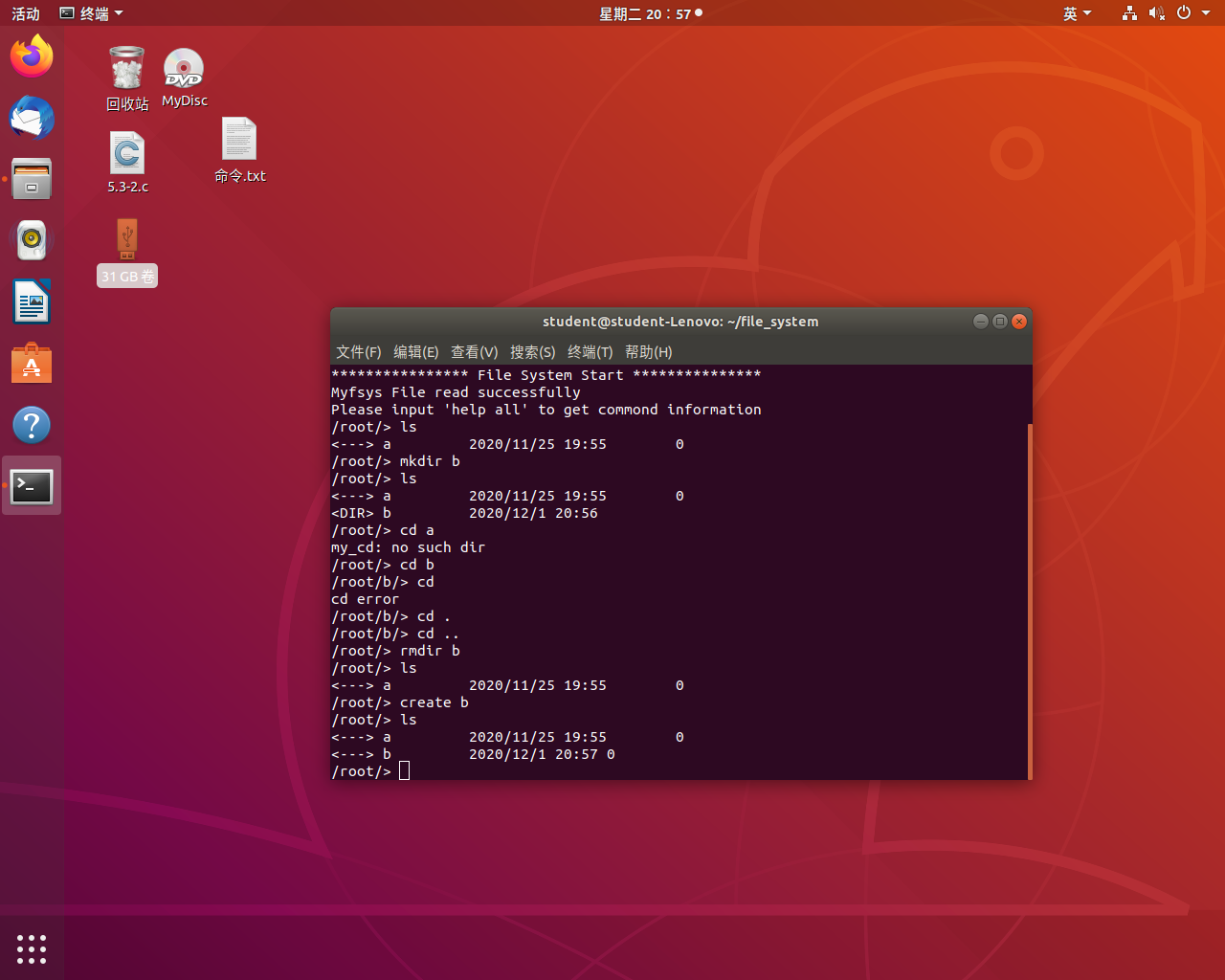


1. #include "head.h"
3. **void** my\_ls()
4. {
5. // 判断是否是目录
6. **if** (openfilelist[currfd].attribute == 1) {
7. printf("data file\n");
8. **return**;
9. }
10. **char** buf[MAX\_TEXT\_SIZE];
11. **int** i;
13. // 读取当前目录文件信息(一个个fcb), 载入内存
14. openfilelist[currfd].count = 0;
15. do\_read(currfd, openfilelist[currfd].length, buf);
17. // 遍历当前目录 fcb
18. fcb\* fcbptr = (fcb\*)buf;
19. **for** (i = 0; i < (**int**)(openfilelist[currfd].length / **sizeof**(fcb)); i++, fcbptr++) {
20. **if** (fcbptr->free == 1) {
21. **if** (fcbptr->attribute == 0){
22. **if**(strcmp(fcbptr->filename,".") && strcmp(fcbptr->filename,"..")){
23. printf("<DIR> %-8s\t%d/%d/%d %d:%d\n",
24. fcbptr->filename,
25. (fcbptr->date >> 9) + 2000,
26. (fcbptr->date >> 5) & 0x000f,
27. (fcbptr->date) & 0x001f,
28. (fcbptr->time >> 11),
29. (fcbptr->time >> 5) & 0x003f);
30. }

33. } **else** {
34. printf("<---> %-8s\t%d/%d/%d %d:%d\t%d\n",
35. fcbptr->filename,
36. (fcbptr->date >> 9) + 2000,
37. (fcbptr->date >> 5) & 0x000f,
38. (fcbptr->date) & 0x001f,
39. (fcbptr->time >> 11),
40. (fcbptr->time >> 5) & 0x003f,
41. fcbptr->length);
42. }
43. }
44. }
45. }

**2.12创建文件函数my\_create()**

my\_create函数用于创建新的文件。根据输入的文件名先进行文件名命名合法性的判断，然后判断当前所在的属性，若为0目录才能继续创建。调用do\_read读取当前目录信息，然后判断文件名是否在目录中重名，若重名则返回错误信息，否则申请新的fcb和磁盘块，同时更新fat表中的信息，修改文件的fcb信息，同时更新相关目录的fcb信息。



1. #include "head.h"

4. **int** my\_create(**char**\* filename)
5. {
6. // 非法判断
7. **if** (strcmp(filename, "") == 0) {
8. printf("please input filename\n");
9. **return** -1;
10. }
11. **if** (openfilelist[currfd].attribute == 1) {
12. printf("you are in data file now\n");
13. **return** -1;
14. }
16. openfilelist[currfd].count = 0;
17. **char** buf[MAX\_TEXT\_SIZE];
18. do\_read(currfd, openfilelist[currfd].length, buf);
20. **int** i;
21. fcb\* fcbptr = (fcb\*)buf;
22. // 检查重名
23. **for** (i = 0; i < (**int**)(openfilelist[currfd].length / **sizeof**(fcb)); i++, fcbptr++) {
24. **if** (fcbptr->free == 0) {
25. **continue**;
26. }
27. **if** (strcmp(fcbptr->filename, filename) == 0 && fcbptr->attribute == 1) {
28. printf("the same filename error\n");
29. **return** -1;
30. }
31. }
33. // 申请空 fcb;
34. fcbptr = (fcb\*)buf;
35. **for** (i = 0; i < (**int**)(openfilelist[currfd].length / **sizeof**(fcb)); i++, fcbptr++) {
36. **if** (fcbptr->free == 0)
37. **break**;
38. }
39. // 申请磁盘块并更新 fat 表
40. **int** block\_num = get\_free\_block();
41. **if** (block\_num == -1) {
42. **return** -1;
43. }
44. fat\* fat1 = (fat\*)(myvhard + BLOCKSIZE);
45. fat\* fat2 = (fat\*)(myvhard + BLOCKSIZE \* 3);
46. fat1[block\_num].id = END;
47. memcpy(fat2, fat1, BLOCKSIZE \* 2);
49. // 修改 fcb 信息
50. strcpy(fcbptr->filename, filename);
51. **time\_t** rawtime = time(NULL);
52. **struct** **tm**\* time = localtime(&rawtime);
53. fcbptr->date = (time->tm\_year - 100) \* 512 + (time->tm\_mon + 1) \* 32 + (time->tm\_mday);
54. fcbptr->time = (time->tm\_hour) \* 2048 + (time->tm\_min) \* 32 + (time->tm\_sec) / 2;
55. fcbptr->first = block\_num;
56. fcbptr->free = 1;
57. fcbptr->attribute = 1;
58. fcbptr->length = 0;
60. openfilelist[currfd].count = i \* **sizeof**(fcb);
61. do\_write(currfd, (**char**\*)fcbptr, **sizeof**(fcb), 2);
63. // 修改父目录 fcb
64. fcbptr = (fcb\*)buf;
65. fcbptr->length = openfilelist[currfd].length;
66. openfilelist[currfd].count = 0;
67. do\_write(currfd, (**char**\*)fcbptr, **sizeof**(fcb), 2);
68. openfilelist[currfd].fcbstate = 1;
69. }

**2.13申请新的空闲磁盘块函数get\_free\_block()**

定义这个函数用于在空间中申请空闲的新磁盘块用于使用。定义fat指针通过循环返回磁盘块的位置数，从而修改该磁盘块的使用信息为占有。

1. #include "head.h"
3. unsigned **short** **int** get\_free\_block()
4. {
5. **int** i;
6. fat\* fat1 = (fat\*)(myvhard + BLOCKSIZE);
7. **for** (i = 0; i < (**int**)(SIZE / BLOCKSIZE); i++) {
8. **if** (fat1[i].id == FREE) {
9. **return** i;
10. }
11. }
12. **return** END;
13. }

**2.14申请新的打开文件列表函数get\_free\_openfilelist()**

函数用于申请打开文件或目录。在未达到最大打开文件数之前，函数可以将需要打开的文件打开信息更改为1，表示以及打开。

1. #include "head.h"
3. **int** get\_free\_openfilelist()
4. {
5. **int** i;
6. **for** (i = 0; i < MAXOPENFILE; i++) {
7. **if** (openfilelist[i].topenfile == 0) {
8. openfilelist[i].topenfile = 1;
9. **return** i;
10. }
11. }
12. **return** -1;
13. }

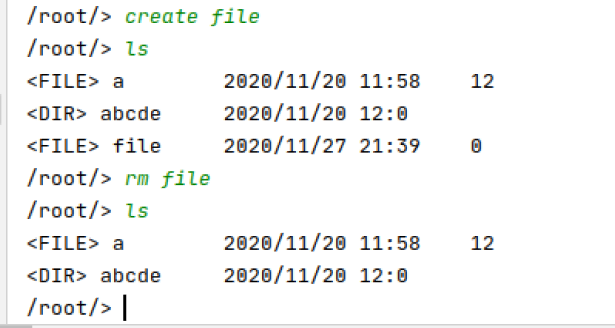
**2.15删除文件函数my\_rm()**

my\_rm函数的主要功能是删除文件，首先通过for循环判断父目录文件是否已经打开，若欲删除文件的父目录文件还没有打开，则调用my\_open()打开；若打开失败，则返回，并显示错误信息；调用do\_read()读出该父目录文件内容到内存，检查该目录下欲删除文件是否存在，若不存在则返回，并输出no such file； 用一个while语句检查该文件是否已经打开，若已打开则关闭掉； 回收该文件所占据的磁盘块，修改fat,更新fat表；从文件的父目录文件中清空该文件的目录项，清空fcb，然后以覆盖写方式调用do\_write函数来修改父目录文件的fcb，最后修改该父目录文件的用户打开文件表项中的长度信息，并将该表项中的fcbstate置为1。

**程序代码：**

1. #include "head.h"
3. **void** my\_rm(**char**\* filename)
4. {
5. **char** buf[MAX\_TEXT\_SIZE];
6. openfilelist[currfd].count = 0;
7. do\_read(currfd, openfilelist[currfd].length, buf);
9. **int** i, flag = 0;
10. fcb\* fcbptr = (fcb\*)buf;
11. // 查询
12. **for** (i = 0; i < (**int**)(openfilelist[currfd].length / **sizeof**(fcb)); i++, fcbptr++) {
13. **if** (strcmp(fcbptr->filename, filename) == 0 && fcbptr->attribute == 1) {
14. flag = 1;
15. **break**;
16. }
17. }
18. **if** (flag != 1) {
19. printf("no such file\n");
20. **return**;
21. }
23. // 更新 fat 表
24. **int** block\_num = fcbptr->first;
25. fat\* fat1 = (fat\*)(myvhard + BLOCKSIZE);
26. **int** nxt\_num = 0;
27. **while** (1) {
28. nxt\_num = fat1[block\_num].id;
29. fat1[block\_num].id = FREE;
30. **if** (nxt\_num != END)
31. block\_num = nxt\_num;
32. **else**
33. **break**;
34. }
35. fat1 = (fat\*)(myvhard + BLOCKSIZE);
36. fat\* fat2 = (fat\*)(myvhard + BLOCKSIZE \* 3);
37. memcpy(fat2, fat1, BLOCKSIZE \* 2);
39. // 清空 fcb
40. fcbptr->date = 0;
41. fcbptr->time = 0;
42. fcbptr->exname[0] = '\0';
43. fcbptr->filename[0] = '\0';
44. fcbptr->first = 0;
45. fcbptr->free = 0;
46. fcbptr->length = 0;
47. openfilelist[currfd].count = i \* **sizeof**(fcb);
48. do\_write(currfd, (**char**\*)fcbptr, **sizeof**(fcb), 2);
49. //
50. **int** lognum = i;
51. **if** ((lognum + 1) \* **sizeof**(fcb) == openfilelist[currfd].length) {
52. openfilelist[currfd].length -= **sizeof**(fcb);
53. lognum--;
54. fcbptr = (fcb \*)buf + lognum;
55. **while** (fcbptr->free == 0) {
56. fcbptr--;
57. openfilelist[currfd].length -= **sizeof**(fcb);
58. }
59. }
61. // 修改父目录 . 目录文件的 fcb
62. fcbptr = (fcb\*)buf;
63. fcbptr->length = openfilelist[currfd].length;
64. openfilelist[currfd].count = 0;
65. do\_write(currfd, (**char**\*)fcbptr, **sizeof**(fcb), 2);
67. openfilelist[currfd].fcbstate = 1;
68. }

**运行截图：**



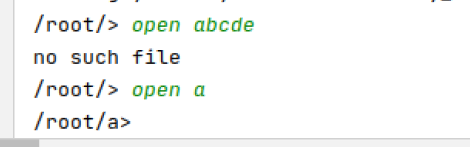
**2.16打开文件函数my\_open()**

my\_open函数的主要功能是打开文件，检查该文件是否已经打开，若已打开则返回-1，并显示错误信息； 调用do\_read()读出父目录文件的内容到内存，用if语句判断该目录下欲打开文件是否存在，若不存在则返回-1，并输出no such file；申请新的打开目录项并初始化该目录项，检查用户打开文件表中是否有空表项，若有则为欲打开文件分配一个空表项，若没有则返回-1，并输出"my\_open: full openfilelist"；为该文件填写空白用户打开文件表表项内容，读写指针置为0；将该文件所分配到的空白用户打开文件表表项序号（数组下标）作为文件描述符fd，返回1。

**程序代码：**

1. #include "head.h"
3. **int** my\_open(**char**\* filename)
4. {
5. **char** buf[MAX\_TEXT\_SIZE];
6. openfilelist[currfd].count = 0;
7. do\_read(currfd, openfilelist[currfd].length, buf);
9. **int** i, flag = 0;
10. fcb\* fcbptr = (fcb\*)buf;
11. // 重名检查
12. **for** (i = 0; i < (**int**)(openfilelist[currfd].length / **sizeof**(fcb)); i++, fcbptr++) {
13. **if** (strcmp(fcbptr->filename, filename) == 0 && fcbptr->attribute == 1) {
14. flag = 1;
15. **break**;
16. }
17. }
18. **if** (flag != 1) {
19. printf("no such file\n");
20. **return** -1;
21. }
23. // 申请新的打开目录项并初始化该目录项
24. **int** fd = get\_free\_openfilelist();
25. **if** (fd == -1) {
26. printf("my\_open: full openfilelist\n");
27. **return** -1;
28. }
30. openfilelist[fd].attribute = 1;
31. openfilelist[fd].count = 0;
32. openfilelist[fd].date = fcbptr->date;
33. openfilelist[fd].time = fcbptr->time;
34. openfilelist[fd].length = fcbptr->length;
35. openfilelist[fd].first = fcbptr->first;
36. openfilelist[fd].free = 1;
37. strcpy(openfilelist[fd].filename, fcbptr->filename);
38. strcat(strcpy(openfilelist[fd].dir, (**char**\*)(openfilelist[currfd].dir)), filename);
39. openfilelist[fd].dirno = openfilelist[currfd].first;
40. openfilelist[fd].diroff = i;
41. openfilelist[fd].topenfile = 1;
43. openfilelist[fd].fcbstate = 0;
45. currfd = fd;
46. **return** 1;
47. }

**运行截图：**



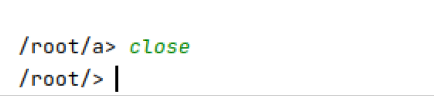
2.17**关闭文件函数my\_close()**

my\_close函数的主要功能是关闭前面由my\_open()打开的文件。首先使用if语句检查fd的有效性，如果fd超出用户打开文件表所在数组的最大下标或者小于0，则无效，返回-1，并输出my\_close: fd error；然后检查用户打开文件表表项中的fcbstate字段的值，如果为1则打开该文件的父目录文件，以覆盖写方式调用do\_write()将欲关闭文件的FCB写入父目录文件的相应盘块中；以此来将该文件的FCB的内容保存到虚拟磁盘上该文件的目录项中，最后释放打开文件表，回收该文件占据的用户打开文件表表项（进行清空操作），并将topenfile字段置为0，返回father\_fd。

**程序代码：**

1. #include "head.h"
3. **int** my\_close(**int** fd)
4. {
5. **if** (fd > MAXOPENFILE || fd < 0) {
6. printf("my\_close: fd error\n");
7. **return** -1;
8. }
10. **int** i;
11. **char** buf[MAX\_TEXT\_SIZE];
12. **int** father\_fd = -1;
13. fcb\* fcbptr;
14. **for** (i = 0; i < MAXOPENFILE; i++) {
15. **if** (openfilelist[i].first == openfilelist[fd].dirno) {
16. father\_fd = i;
17. **break**;
18. }
19. }
20. **if** (father\_fd == -1) {
21. printf("my\_close: no father dir\n");
22. **return** -1;
23. }
24. **if** (openfilelist[fd].fcbstate == 1) {
25. do\_read(father\_fd, openfilelist[father\_fd].length, buf);
26. // update fcb
27. fcbptr = (fcb\*)(buf + **sizeof**(fcb) \* openfilelist[fd].diroff);
28. strcpy(fcbptr->exname, openfilelist[fd].exname);
29. strcpy(fcbptr->filename, openfilelist[fd].filename);
30. fcbptr->first = openfilelist[fd].first;
31. fcbptr->free = openfilelist[fd].free;
32. fcbptr->length = openfilelist[fd].length;
33. fcbptr->time = openfilelist[fd].time;
34. fcbptr->date = openfilelist[fd].date;
35. fcbptr->attribute = openfilelist[fd].attribute;
36. openfilelist[father\_fd].count = openfilelist[fd].diroff \* **sizeof**(fcb);
37. do\_write(father\_fd, (**char**\*)fcbptr, **sizeof**(fcb), 2);
38. }
39. // 释放打开文件表
40. memset(&openfilelist[fd], 0, **sizeof**(useropen));
41. currfd = father\_fd;
42. **return** father\_fd;
43. }

**运行截图：**

****

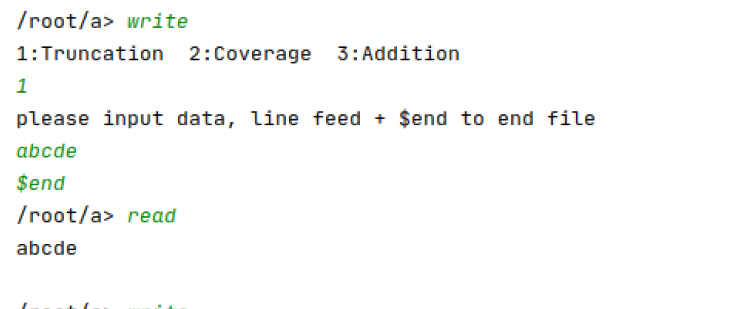
**2.18写文件函数my\_write()**

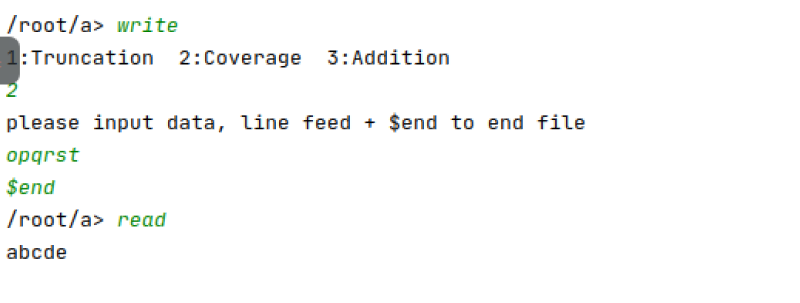
my\_write函数的功能是将用户通过键盘输入的内容写到fd所指定的文件中。磁盘文件的读写操作都必须以完整的数据块为单位进行，在写操作时，先将数据写在缓冲区中，缓冲区的大小与磁盘块的大小相同，然后再将缓冲区中的数据一次性写到磁盘块中；读出时先将一个磁盘块中的内容读到缓冲区中，然后再传送到用户区。本实例为了简便起见，没有设置缓冲区管理，只是在读写文件时由用户使用malloc()申请一块空间作为缓冲区，读写操作结束后使用free()释放掉。写操作有三种方式：截断写、覆盖写和追加写。截断写是放弃原来文件的内容，重新写文件；覆盖写是修改文件在当前读写指针所指的位置开始的部分内容；追加写是在原文件的最后添加新的内容。在本实例中，输入写文件命令后，系统会出现提示让用户选择其中的一种写方式，并将随后键盘输入的内容按照所选的方式写到文件中。首先检查fd的有效性，如果fd超出用户打开文件表所在数组的最大下标或小于0，则返回-1，并输出my\_write: no such file；有效则提示并等待用户输入写方式：（1：截断写；2：覆盖写；3：追加写）如果用户要求的写方式是截断写，则释放文件除第一块外的其他磁盘空间内容（查找并修改FAT表），将内存用户打开文件表项中文件长度修改为0，将读写指针置为0；如果用户要求的写方式是追加写，则修改文件的当前读写指针位置到文件的末尾；如果写方式是覆盖写，则从文件指针处开始写；用户可分多次输入写入内容，每次用回车结束；每次用户的输入内容保存到一临时变量text[]中，要每次输入以回车结束；my\_write调用do\_write()函数将通过键盘键入的内容写到文件中。如果do\_write()函数的返回值为非负值，则将实际写入字节数增加do\_write()函数返回值，否则显示出错信息；如果text[]中最后一个字符不是结束字符，则继续进行写操作；如果当前读写指针位置大于用户打开文件表项中的文件长度，则修改打开文件表项中的文件长度信息，并将fcbstate的值定为1；

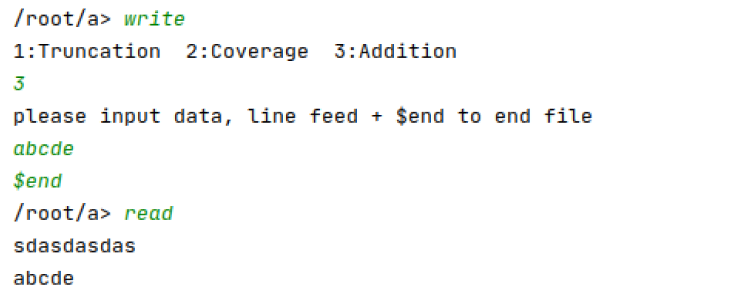
**程序代码：**

1. #include "head.h"
3. **int** my\_write(**int** fd)
4. {
5. **if** (fd < 0 || fd >= MAXOPENFILE) {
6. printf("my\_write: no such file\n");
7. **return** -1;
8. }
9. **int** wstyle;
10. **while** (1) {
11. // 1: 截断写，清空全部内容，从头开始写
12. // 2. 覆盖写，从文件指针处开始写
13. // 3. 追加写，字面意思
14. printf("1:Truncation  2:Coverage  3:Addition\n");
15. scanf("%d", &wstyle);
16. **if** (wstyle > 3 || wstyle < 1) {
17. printf("input error\n");
18. } **else** {
19. **break**;
20. }
21. }
22. **char** text[MAX\_TEXT\_SIZE] = "\0";
23. **char** texttmp[MAX\_TEXT\_SIZE] = "\0";
24. printf("please input data, line feed + $end to end file\n");
25. getchar();
26. **while** (gets(texttmp)) {
27. **if** (strcmp(texttmp, "$end") == 0) {
28. **break**;
29. }
30. texttmp[strlen(texttmp)] = '\n';
31. strcat(text, texttmp);
32. }
34. text[strlen(text)] = '\0';
35. do\_write(fd, text, strlen(text) + 1, wstyle);
36. openfilelist[fd].fcbstate = 1;
37. **return** 1;
38. }

**运行截图：**

****

****



**2.19实际写文件函数do\_write()**

do\_write是实际写文件函数，它会被写文件函数my\_write()调用，它的功能是用来将键盘输入的内容写到相应的文件中。首先用malloc()申请1024B的内存空间作为读写磁盘的缓冲区buf，申请失败则返回-1，并显示出错信息；将读写指针转化为逻辑块块号和块内偏移off，并利用打开文件表表项中的首块号及FAT表的相关内容将逻辑块块号转换成对应的磁盘块块号blkno；如果找不到对应的磁盘块，则需要检索FAT为该逻辑块分配一新的磁盘块，并将对应的磁盘块块号blkno登记到FAT中，若分配失败，则返回-1，并显示出错信息；如果是覆盖写，或者如果当前读写指针所对应的块内偏移off不等于0，则将块号为blkno的虚拟磁盘块全部1024B的内容读到缓冲区buf中，否则便用清空buf；将text中未写入的内容暂存到缓冲区buff的第off字节开始的位置，直到缓冲区满，或者接收到结束字符为止，将本次写入字节数记录到tmplen中，buf中1024B的内容写入到块号为blkno的虚拟磁盘块中，然后将当前读写指针修改为原来的值加上tmplen；并将本次实际写入的字节数增加tmplen，如果tmplen小于len，则继续写入，否则返回本次实际写入的字节数。

**程序代码：**

1. #include "head.h"
2. **int** do\_write(**int** fd, **char**\* text, **int** len, **char** wstyle)
3. {
4. **int** block\_num = openfilelist[fd].first;
5. **int** i, tmp\_num;
6. **int** lentmp = 0;
7. **char**\* textptr = text;
8. **char** buf[BLOCKSIZE];
9. fat\* fatptr = (fat\*)(myvhard + BLOCKSIZE) + block\_num;
10. unsigned **char**\* blockptr;
12. **if** (wstyle == 1) {
13. openfilelist[fd].count = 0;
14. openfilelist[fd].length = 0;
15. } **else** **if** (wstyle == 3) {
16. // 追加写，如果是一般文件，则需要先删除末尾 \0，即将指针移到末位减一个字节处
17. openfilelist[fd].count = openfilelist[fd].length;
18. **if** (openfilelist[fd].attribute == 1) {
19. **if** (openfilelist[fd].length != 0) {
20. // 非空文件
21. openfilelist[fd].count = openfilelist[fd].length - 1;
22. }
23. }
24. }
26. **int** off = openfilelist[fd].count;
28. // 定位磁盘块和块内偏移量
29. **while** (off >= BLOCKSIZE) {
30. block\_num = fatptr->id;
31. **if** (block\_num == END) {
32. printf("do\_write: off error\n");
33. **return** -1;
34. }
35. fatptr = (fat\*)(myvhard + BLOCKSIZE) + block\_num;
36. off -= BLOCKSIZE;
37. }
39. blockptr = (unsigned **char**\*)(myvhard + BLOCKSIZE \* block\_num);
40. // 写入磁盘
41. **while** (len > lentmp) {
42. memcpy(buf, blockptr, BLOCKSIZE);
43. **for** (; off < BLOCKSIZE; off++) {
44. \*(buf + off) = \*textptr;
45. textptr++;
46. lentmp++;
47. **if** (len == lentmp)
48. **break**;
49. }
50. memcpy(blockptr, buf, BLOCKSIZE);
52. // 写入的内容太多，需要写到下一个磁盘块，如果没有磁盘块，就申请一个
53. **if** (off == BLOCKSIZE && len != lentmp) {
54. off = 0;
55. block\_num = fatptr->id;
56. **if** (block\_num == END) {
57. block\_num = get\_free\_block();
58. **if** (block\_num == END) {
59. printf("do\_write: block full\n");
60. **return** -1;
61. }
62. blockptr = (unsigned **char**\*)(myvhard + BLOCKSIZE \* block\_num);
63. fatptr->id = block\_num;
64. fatptr = (fat\*)(myvhard + BLOCKSIZE) + block\_num;
65. fatptr->id = END;
66. } **else** {
67. blockptr = (unsigned **char**\*)(myvhard + BLOCKSIZE \* block\_num);
68. fatptr = (fat\*)(myvhard + BLOCKSIZE) + block\_num;
69. }
70. }
71. }
73. openfilelist[fd].count += len;
74. **if** (openfilelist[fd].count > openfilelist[fd].length)
75. openfilelist[fd].length = openfilelist[fd].count;
77. // 删除多余的磁盘块
78. **if** (wstyle == 1 || (wstyle == 2 && openfilelist[fd].attribute == 0)) {
79. off = openfilelist[fd].length;
80. fatptr = (fat \*)(myvhard + BLOCKSIZE) + openfilelist[fd].first;
81. **while** (off >= BLOCKSIZE) {
82. block\_num = fatptr->id;
83. off -= BLOCKSIZE;
84. fatptr = (fat \*)(myvhard + BLOCKSIZE) + block\_num;
85. }
86. **while** (1) {
87. **if** (fatptr->id != END) {
88. i = fatptr->id;
89. fatptr->id = FREE;
90. fatptr = (fat \*)(myvhard + BLOCKSIZE) + i;
91. } **else** {
92. fatptr->id = FREE;
93. **break**;
94. }
95. }
96. fatptr = (fat \*)(myvhard + BLOCKSIZE) + block\_num;
97. fatptr->id = END;
98. }
100. memcpy((fat\*)(myvhard + BLOCKSIZE \* 3), (fat\*)(myvhard + BLOCKSIZE), BLOCKSIZE \* 2);
101. **return** len;
102. }

**2.20读文件函数my\_read()**

my\_read() 函数的功能是读出指定文件中从读写指针开始的内容到用户空间中。通过定义一个字符型数组text[MAX\_TEXT\_SIZE]，来接收用户从文件中读出的文件内容，如果返回值超出用户打开文件表所在数组的最大下标或者小于零，则返回为-1,并输出no such file，然后调用do\_read()将指定文件中的内容读出到text中并输出**。**

**程序代码：**

1. #include "head.h"
3. **int** my\_read(**int** fd)
4. {
5. **if** (fd < 0 || fd >= MAXOPENFILE) {
6. printf("no such file\n");
7. **return** -1;
8. }
10. openfilelist[fd].count = 0;
11. **char** text[MAX\_TEXT\_SIZE] = "\0";
12. do\_read(fd, openfilelist[fd].length, text);
13. printf("%s\n", text);
14. **return** 1;
15. }

**2.21实际读文件函数do\_read()**

do\_read函数是实际读文件函数，它会被my\_read()调用，读出指定文件中从读写指针开始的长度为len的内容到用户空间的text中。首先使用malloc()函数申请一个1024B的空间作为缓冲区buf，申请失败则返回-1，并输出do\_read reg mem error；然后将读写指针转化为逻辑块块号及块内偏移量off，利用打开文件表表项中的首块号查找FAT表，找到该逻辑块所在的磁盘块块号；将该磁盘块块号转化为虚拟磁盘上的内存位置；将该内存位置开始的一个磁盘块的内容读入buf中；比较buf中从偏移量off开始的剩余字节数是否大于等于应读写的字节数len，如果是，则将从off开始的buf中的len长度的内容读入到text中；否则，将从off开始的buf中的剩余内容读入到text中；使用free()释放开始申请的缓冲区。返回实际读出的字节数。

**程序代码：**

1. #include "head.h"
3. **int** do\_read(**int** fd, **int** len, **char**\* text)
4. {
5. **int** len\_tmp = len;
6. **char**\* textptr = text;
7. unsigned **char**\* buf = (unsigned **char**\*)malloc(1024);
8. **if** (buf == NULL) {
9. printf("do\_read reg mem error\n");
10. **return** -1;
11. }
12. **int** off = openfilelist[fd].count;
13. **int** block\_num = openfilelist[fd].first;
14. fat\* fatptr = (fat\*)(myvhard + BLOCKSIZE) + block\_num;
16. // 定位读取目标磁盘块和块内地址
17. **while** (off >= BLOCKSIZE) {
18. off -= BLOCKSIZE;
19. block\_num = fatptr->id;
20. **if** (block\_num == END) {
21. printf("do\_read: block not exist\n");
22. **return** -1;
23. }
24. fatptr = (fat\*)(myvhard + BLOCKSIZE) + block\_num;
25. }
27. unsigned **char**\* blockptr = myvhard + BLOCKSIZE \* block\_num;
28. memcpy(buf, blockptr, BLOCKSIZE);
30. // 读取内容
31. **while** (len > 0) {
32. **if** (BLOCKSIZE - off > len) {
33. memcpy(textptr, buf + off, len);
34. textptr += len;
35. off += len;
36. openfilelist[fd].count += len;
37. len = 0;
38. } **else** {
39. memcpy(textptr, buf + off, BLOCKSIZE - off);
40. textptr += BLOCKSIZE - off;
41. len -= BLOCKSIZE - off;
43. block\_num = fatptr->id;
44. **if** (block\_num == END) {
45. printf("do\_read: len is lager then file\n");
46. **break**;
47. }
48. fatptr = (fat\*)(myvhard + BLOCKSIZE) + block\_num;
49. blockptr = myvhard + BLOCKSIZE \* block\_num;
50. memcpy(buf, blockptr, BLOCKSIZE);
51. }
52. }
53. free(buf);
54. **return** len\_tmp - len;
55. }

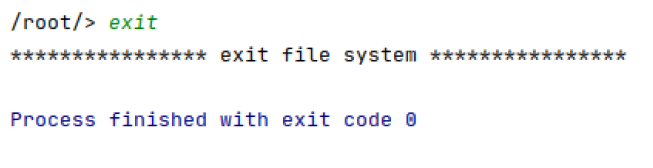
**2.22退出文件系统函数my\_exitsys()**

my\_exitsys()函数的主要功能是退出文件系统，通过调用库函数fopen()来打开myfsys文件，并将内容写入到FILENAME文件中，并且可以通过调用库函数fclose()关闭myfsys，从而释放虚拟磁盘空间，最后输出exit file system。

**程序代码：**

1. # include "head.h"
2. **void** my\_exitsys()
3. {
4. /\*\*
5. \* 依次关闭 打开文件。 写入 FILENAME 文件
6. \*/
7. **while** (currfd) {
8. my\_close(currfd);
9. }
10. **FILE**\* fp = fopen(FILENAME, "w");
11. fwrite(myvhard, SIZE, 1, fp);
12. fclose(fp);
13. printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* exit file system \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");
14. }

**运行截图：**

****