|  |
| --- |
| 操作系统实验报告 |
| 实验四：文件系统 |

|  |
| --- |
| 负责人：刘润泽 14061120  2016-5-18 |

**小组成员及分工：**

|  |  |  |
| --- | --- | --- |
| 姓名 | 学号 | 分工 |
| 陈鸿超 | 14061216 | 提高要求3 |
| 刘润泽 | 14061120 | 提高要求1、2、5 |
| 叶旭诚 | 14061215 | 提高要求4 |
| 杨佳琦 | 14061124 | 测试、debug |

目录

[一． 实验目的 3](#_Toc451518180)

[二． 实验要求 3](#_Toc451518181)

[1. 基本要求 3](#_Toc451518182)

[2. 提高要求 3](#_Toc451518183)

[3. 完成情况 3](#_Toc451518184)

[三． 设计说明 4](#_Toc451518185)

[1. 程序流程图 4](#_Toc451518186)

[2. 更改与新增内容 5](#_Toc451518187)

[修改ScanEntry函数 5](#_Toc451518188)

[修改fd\_cf函数 6](#_Toc451518189)

[修改fd\_df函数 20](#_Toc451518190)

[修改fd\_ls函数 26](#_Toc451518191)

[修改main函数 30](#_Toc451518192)

# 实验目的

1. 了解文件管理系统的作用和工作方式。
2. 了解FAT文件系统的结构。
3. 学习文件管理系统的一般开发方法。

# 实验要求

## 基本要求

准备一个FAT16格式的U盘，在Linux下编写一个文件系统管理程序，对U盘上的文件进行管理。具体要求如下：

1. 设计并实现一个目录列表函数（无须支持选项，如ls -a、 ls -l等），用来显示当前目录下包含的文件信息。
2. 设计并实现一个改变目录函数（无须处理路径名，如../../directoryName等），用来把当前目录切换到上一层目录或当前目录的子目录。
3. 设计并实现一个删除文件函数， 使用当前目录中的要删除的文件名作为参数，用来删除指定的文件，要注意文件的隐藏、只读和其他系统属性。
4. 设计并实现一个创建文件函数，使用要创建的文件名和文件大小作为参数，用来创建一个新的文件。

## 提高要求

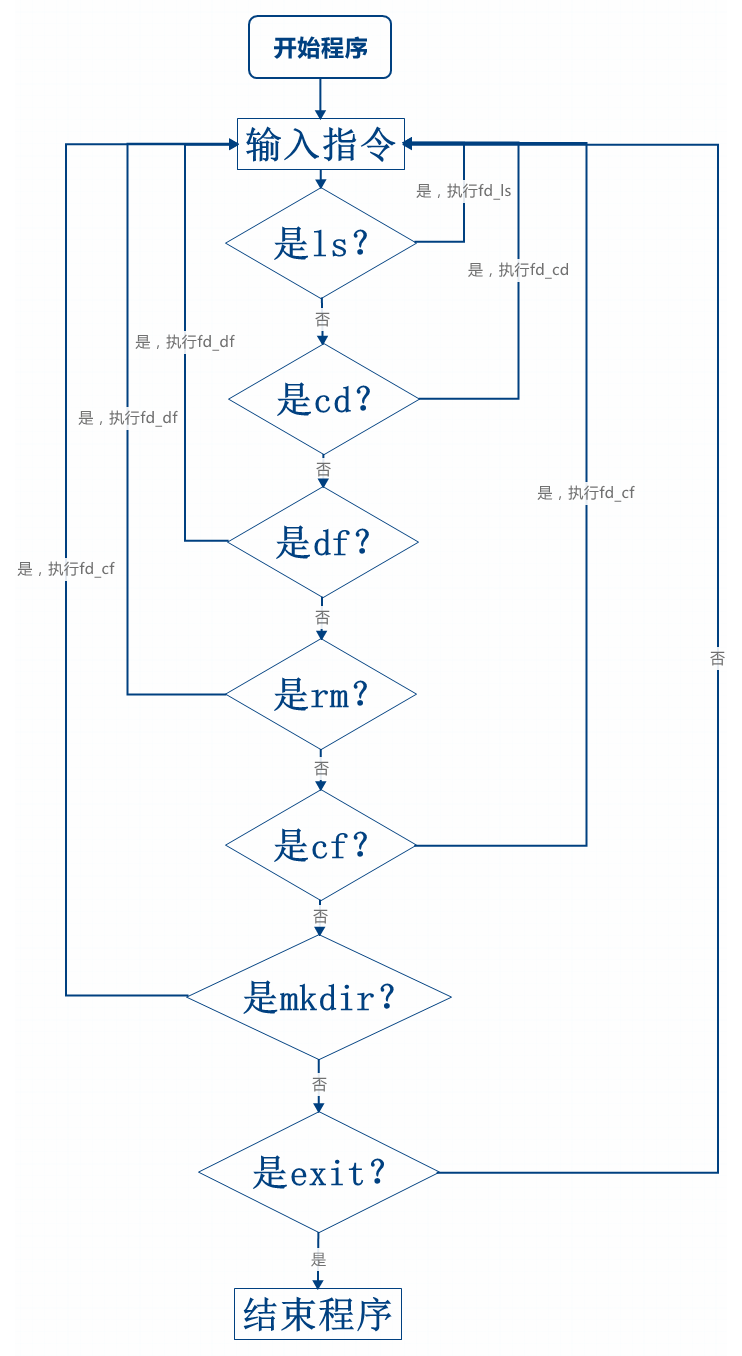
1. 增加创建目录的功能。
2. 增加删除目录的功能：通常需要先判断目录是否为空目录，若目录不为空，则需给出提示，并删除其包含的所有子目录和文件；若是空目录则可以直接删除。
3. 增加绝对路径和多级目录的支持：这里需要对输入的目录路径字符串进行解析，然后逐级查找目录。
4. 对ud\_cf()函数进行改进，使其可以向文件中写入实际内容，并根据写入的内容计算文件的实际大小。
5. 对ud\_ls()函数进行改进，增加对全部非根目录信息的读取（基本要求中仅读取一个扇区的非根目录细信息）。

## 完成情况

完成所有实验要求。

# 设计说明

## 程序流程图



## 更改与新增内容

### 修改ScanEntry函数

* 原本ScanEntry函数在扫描子目录的时候只读一簇，现在改为读所有簇。

实现方法：读第一簇->从从fat表中找下一簇并读->…->fat表项内容是ffff，结束。

|  |
| --- |
| int ScanEntry(char \*entryname, struct Entry \*pentry, int mode)  {  int ret, offset, i;  int cluster\_addr;  char uppername[80];  for (i = 0; i < strlen(entryname); i++)  uppername[i] = toupper(entryname[i]);  memset(pentry, 0, sizeof(pentry));  uppername[i] = '\0';  */\*扫描根目录\*/*  if (curdir == NULL)  {  if ((ret = lseek(fd, ROOTDIR\_OFFSET, SEEK\_SET)) < 0)  perror("lseek ROOTDIR\_OFFSET failed");  offset = ROOTDIR\_OFFSET;    *//从根目录区起始位置开始一次读取32字节数据保存在pentry中*  while (offset < DATA\_OFFSET)  {  ret = GetEntry(pentry);  offset += abs(ret);  if (pentry->subdir == mode &&!strcmp((char\*)pentry->short\_name, uppername))  return offset;    }  return -1;  }    */\*扫描子目录\*/*  *//////////////////////////////原本只读一簇，改为读所有簇//////////////////////////////////////////////*  else  {  short cur\_cluster = curdir->FirstCluster;  while (1)  {  cluster\_addr = DATA\_OFFSET + (cur\_cluster - 2) \* CLUSTER\_SIZE;  if ((ret = lseek(fd, cluster\_addr, SEEK\_SET)) < 0)  perror("lseek cluster\_addr failed");    offset = cluster\_addr;  *//printf("%d\n", offset);*  while (offset < cluster\_addr + CLUSTER\_SIZE)  {  lseek(fd, offset, SEEK\_SET);  ret = GetEntry(pentry);  offset += abs(ret);    *//printf("%d\n", offset);*  if (pentry->subdir == mode &&!strcmp((char\*)pentry->short\_name, uppername))  return offset;    }  if (GetFatCluster(cur\_cluster) != 0xffff)  {  cur\_cluster = GetFatCluster(cur\_cluster);  }  else  {  break;  }  }    }  return -1;  *///////////////////////////////////////////////////////////////////////////////////////*  } |

### 修改fd\_cf函数

* 添加创建目录功能（实现提高要求一）

实现方法：与创建文件不同的是目录项中代表子目录的那位填1

* 添加向文件中写入实际内容（实现提高要求四）

实现方法：向对应的簇里写。

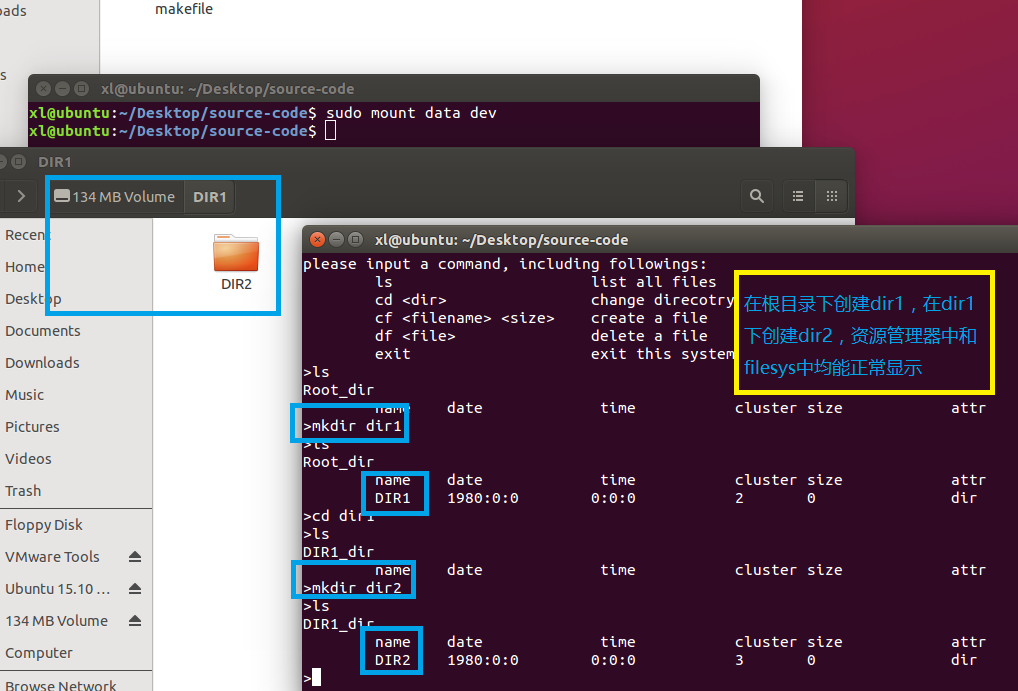
* 修改：原本在非根目录下创建文件或目录的时候，只会在一个簇中找空的目录项，现改为：在所有簇中找空目录项，若没有空目录项，则先分配新的空簇，再在空簇中找空的目录项

实现方法：在所有簇中找空目录项，若没找到，找空簇，找到后更改fat表，把空簇对应的fat表项填ffff，原来的最后一簇对应的fat表项填新簇的簇号。

|  |
| --- |
| int fd\_cf(char \*filename, int size, int is\_dir)  {  int write\_flag = size < 0 ? 1 : 0;  unsigned char \*stringaddr, inputstring[CLUSTER\_SIZE \* 10] = { "\0" };  unsigned char cin;  if (!is\_dir&&size < 0)  {  int j = 0;  cin = getchar();  if (size < 0) {  for (; (cin = getchar()) != '\n';) {  inputstring[j++] = cin;  }  size = j;  }  }    struct Entry \*pentry;  int ret, i = 0, cluster\_addr, offset;  unsigned short cluster, clusterno[100];  unsigned char c[DIR\_ENTRY\_SIZE];  int index, clustersize;  unsigned char buf[DIR\_ENTRY\_SIZE];  pentry = (struct Entry\*)malloc(sizeof(struct Entry));      clustersize = (size / (CLUSTER\_SIZE));    if (size % (CLUSTER\_SIZE) != 0)  clustersize++;    if (clustersize == 0)  clustersize = 1;    *//////////////////////////区别目录和文件/////////////////////////////////////*  *//扫描根目录，是否已存在该文件名*  if (is\_dir)  ret = ScanEntry(filename, pentry, 1);  else  ret = ScanEntry(filename, pentry, 0);  *////////////////////////////////////////////////////////////////////////////*    if (ret < 0)  {  */\*查询fat表，找到空白簇，保存在clusterno[]中\*/*  for (cluster = 2; cluster < 1000; cluster++)  {  index = cluster \* 2;  if (fatbuf[index] == 0x00 && fatbuf[index + 1] == 0x00)  {  clusterno[i] = cluster;    i++;  if (i == clustersize)  break;    }    }    */\*在fat表中写入下一簇信息\*/*  for (i = 0; i < clustersize - 1; i++)  {  index = clusterno[i] \* 2;    fatbuf[index] = (clusterno[i + 1] & 0x00ff);  fatbuf[index + 1] = ((clusterno[i + 1] & 0xff00) >> 8);      }  */\*最后一簇写入0xffff\*/*  index = clusterno[i] \* 2;  fatbuf[index] = 0xff;  fatbuf[index + 1] = 0xff;    if (curdir == NULL)  */\*往根目录下写文件\*/*  {    if ((ret = lseek(fd, ROOTDIR\_OFFSET, SEEK\_SET)) < 0)  perror("lseek ROOTDIR\_OFFSET failed");  offset = ROOTDIR\_OFFSET;  while (offset < DATA\_OFFSET)  {  if ((ret = read(fd, buf, DIR\_ENTRY\_SIZE)) < 0)  perror("read entry failed");    offset += abs(ret);    if (buf[0] != 0xe5 && buf[0] != 0x00)  {  while (buf[11] == 0x0f)  {  if ((ret = read(fd, buf, DIR\_ENTRY\_SIZE)) < 0)  perror("read root dir failed");  offset += abs(ret);  }  }      */\*找出空目录项或已删除的目录项\*/*  else  {  offset = offset - abs(ret);  for (i = 0; i <= strlen(filename); i++)  {  c[i] = toupper(filename[i]);  }  for (; i <= 10; i++)  c[i] = ' ';    *//////////////////////////区别目录和文件/////////////////////////////////////*  if (is\_dir) *//子目录那位填1*  c[11] = 0x11;  else  c[11] = 0x01;  *////////////////////////////////////////////////////////////////////////////*    */\*写第一簇的值\*/*  c[26] = (clusterno[0] & 0x00ff);  c[27] = ((clusterno[0] & 0xff00) >> 8);    */\*写文件的大小\*/*  c[28] = (size & 0x000000ff);  c[29] = ((size & 0x0000ff00) >> 8);  c[30] = ((size & 0x00ff0000) >> 16);  c[31] = ((size & 0xff000000) >> 24);    if (lseek(fd, offset, SEEK\_SET) < 0)  perror("lseek fd\_cf failed");  if (write(fd, &c, DIR\_ENTRY\_SIZE) < 0)  perror("write failed");    */////////////////////////////写入实际内容///////////////////////////////////////*  if (!is\_dir&&write\_flag)  {  short temp\_cur\_cluster = RevByte(c[26], c[27]);  stringaddr = inputstring;  for (i = 0; i < clustersize; ++i) {  offset = DATA\_OFFSET + (temp\_cur\_cluster - 2) \* CLUSTER\_SIZE;  if (lseek(fd, offset, SEEK\_SET) < 0)  perror("lseek fd\_cf failed");  if (write(fd, stringaddr, CLUSTER\_SIZE) < 0)  perror("write failed");  stringaddr = stringaddr + CLUSTER\_SIZE;  temp\_cur\_cluster = GetFatCluster(temp\_cur\_cluster);  }  }  *//////////////////////////////////////////////////////////////////////////////*  free(pentry);  if (WriteFat() < 0)  exit(1);    return 1;  }    }  }  else *//非根目录*  {  *///////////////////////////////////////////////////////////////////////////////*  *//改为在所有簇里找空位置，如果所有簇里都没有空位置，给目录添加新簇*  short cur\_cluster = curdir->FirstCluster;  while (1)  {    cluster\_addr = (cur\_cluster - 2)\*CLUSTER\_SIZE + DATA\_OFFSET;  if ((ret = lseek(fd, cluster\_addr, SEEK\_SET)) < 0)  perror("lseek cluster\_addr failed");  offset = cluster\_addr;    while (offset < cluster\_addr + CLUSTER\_SIZE)  {  if ((ret = read(fd, buf, DIR\_ENTRY\_SIZE)) < 0)  perror("read entry failed");    offset += abs(ret); *//read返回成功读取的字节数*    if (buf[0] != 0xe5 && buf[0] != 0x00)  {  while (buf[11] == 0x0f)  {  if ((ret = read(fd, buf, DIR\_ENTRY\_SIZE)) < 0)  perror("read root dir failed");  offset += abs(ret);  }  }  else  {  offset = offset - abs(ret);  for (i = 0; i <= strlen(filename); i++)  {  c[i] = toupper(filename[i]);  }  for (; i <= 10; i++)  c[i] = ' ';    *//////////////////////////区别目录和文件/////////////////////////////////////*  if (is\_dir) *//子目录那位填1*  c[11] = 0x11;  else  c[11] = 0x01;  *////////////////////////////////////////////////////////////////////////////*    c[26] = (clusterno[0] & 0x00ff);  c[27] = ((clusterno[0] & 0xff00) >> 8);    c[28] = (size & 0x000000ff);  c[29] = ((size & 0x0000ff00) >> 8);  c[30] = ((size & 0x00ff0000) >> 16);  c[31] = ((size & 0xff000000) >> 24);    if (lseek(fd, offset, SEEK\_SET) < 0)  perror("lseek fd\_cf failed");  if (write(fd, &c, DIR\_ENTRY\_SIZE) < 0)  perror("write failed");    */////////////////////////////写入实际内容///////////////////////////////////////*  if (!is\_dir&&write\_flag)  {  short temp\_cur\_cluster = RevByte(c[26], c[27]);  stringaddr = inputstring;  for (i = 0; i < clustersize; ++i) {  offset = DATA\_OFFSET + (temp\_cur\_cluster - 2) \* CLUSTER\_SIZE;  if (lseek(fd, offset, SEEK\_SET) < 0)  perror("lseek fd\_cf failed");  if (write(fd, stringaddr, CLUSTER\_SIZE) < 0)  perror("write failed");  stringaddr = stringaddr + CLUSTER\_SIZE;  temp\_cur\_cluster = GetFatCluster(temp\_cur\_cluster);  }  }  *///////////////////////////////////////////////////////////////////////////////*    free(pentry);  if (WriteFat() < 0)  exit(1);    return 1;  }    }    if (GetFatCluster(cur\_cluster) != 0xffff)  {  cur\_cluster = GetFatCluster(cur\_cluster);  }  else  {  *//////////////////////////////////////////////////////////////*  *//最后一个cluster了还没找到能分配的，先撸新的cluster给目录*  *//查询fat表，找到空白簇*  for (cluster = 2; cluster < 1000; cluster++)  {  index = cluster \* 2;  *//みつけた！*  if (fatbuf[index] == 0x00 && fatbuf[index + 1] == 0x00)  {  *//新的cluster对应的fat表填ffff*  fatbuf[index] = 0xff;  fatbuf[index + 1] = 0xff;    *//cur\_cluster下一个变成新撸的*  fatbuf[cur\_cluster \* 2] = (cluster & 0x00ff);  fatbuf[cur\_cluster \* 2 + 1] = ((cluster & 0xff00) >> 8);  break;  }    }  *//////////////////////////////////////////////////////////////*  printf("%d\n%d\n", GetFatCluster(cur\_cluster), GetFatCluster(GetFatCluster(cur\_cluster)));    *//分配新的簇之后，把entry弄进去*  cur\_cluster = GetFatCluster(cur\_cluster);    cluster\_addr = (cur\_cluster - 2)\*CLUSTER\_SIZE + DATA\_OFFSET;  if ((ret = lseek(fd, cluster\_addr, SEEK\_SET)) < 0)  perror("lseek cluster\_addr failed");  offset = cluster\_addr;    while (offset < cluster\_addr + CLUSTER\_SIZE)  {  if ((ret = read(fd, buf, DIR\_ENTRY\_SIZE)) < 0)  perror("read entry failed");    offset += abs(ret); *//read返回成功读取的字节数*    if (buf[0] != 0xe5 && buf[0] != 0x00)  {  while (buf[11] == 0x0f)  {  if ((ret = read(fd, buf, DIR\_ENTRY\_SIZE)) < 0)  perror("read root dir failed");  offset += abs(ret);  }  }  else  {  offset = offset - abs(ret);  for (i = 0; i <= strlen(filename); i++)  {  c[i] = toupper(filename[i]);  }  for (; i <= 10; i++)  c[i] = ' ';    *//////////////////////////区别目录和文件/////////////////////////////////////*  if (is\_dir) *//子目录那位填1*  c[11] = 0x11;  else  c[11] = 0x01;  *////////////////////////////////////////////////////////////////////////////*    c[26] = (clusterno[0] & 0x00ff);  c[27] = ((clusterno[0] & 0xff00) >> 8);    c[28] = (size & 0x000000ff);  c[29] = ((size & 0x0000ff00) >> 8);  c[30] = ((size & 0x00ff0000) >> 16);  c[31] = ((size & 0xff000000) >> 24);    if (lseek(fd, offset, SEEK\_SET) < 0)  perror("lseek fd\_cf failed");  if (write(fd, &c, DIR\_ENTRY\_SIZE) < 0)  perror("write failed");    */////////////////////////////写入实际内容///////////////////////////////////////*  if (!is\_dir&&write\_flag)  {  short temp\_cur\_cluster = RevByte(c[26], c[27]);  stringaddr = inputstring;  for (i = 0; i < clustersize; ++i) {  offset = DATA\_OFFSET + (temp\_cur\_cluster - 2) \* CLUSTER\_SIZE;  if (lseek(fd, offset, SEEK\_SET) < 0)  perror("lseek fd\_cf failed");  if (write(fd, stringaddr, CLUSTER\_SIZE) < 0)  perror("write failed");  stringaddr = stringaddr + CLUSTER\_SIZE;  temp\_cur\_cluster = GetFatCluster(temp\_cur\_cluster);  }  }  *///////////////////////////////////////////////////////////////////////////////*  free(pentry);  if (WriteFat() < 0)  exit(1);    return 1;  }    }  }  }    *///////////////////////////////////////////////////////////////////////////////*  }  }  else  {  printf("This filename is exist\n");  free(pentry);  return -1;  }  return 1;    } |

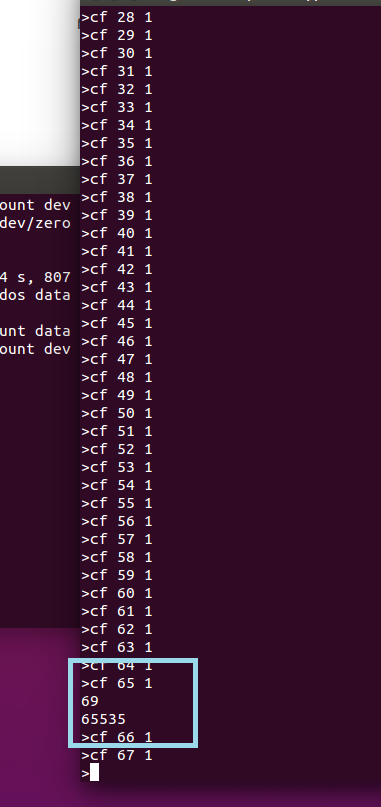
#### 效果图：

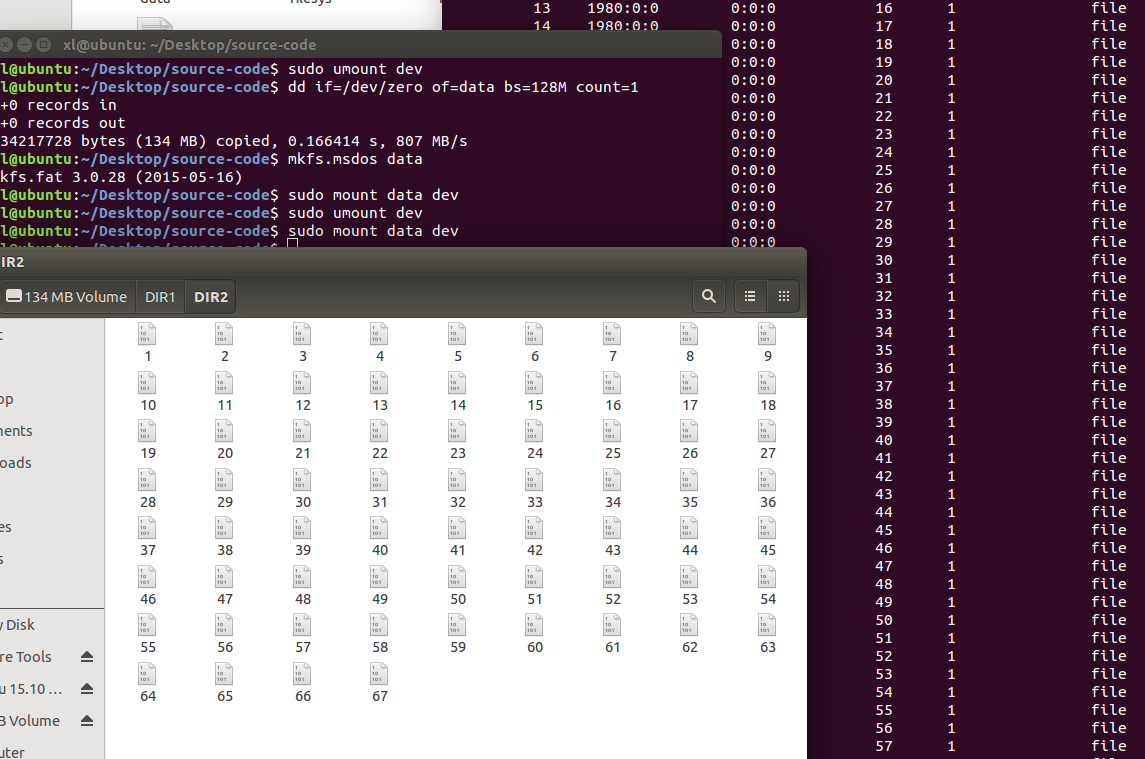
* 创建文件夹效果：



* 文件夹簇的扩充效果：

一个簇2K，一个目录项32字节，所以目录在只有一簇的情况下最多有64个目录项。如图，创建第65个文件的时候输出了给目录添加新簇的提示信息，表明文件夹添加簇成功。而且所有创建的文件均显示正常。



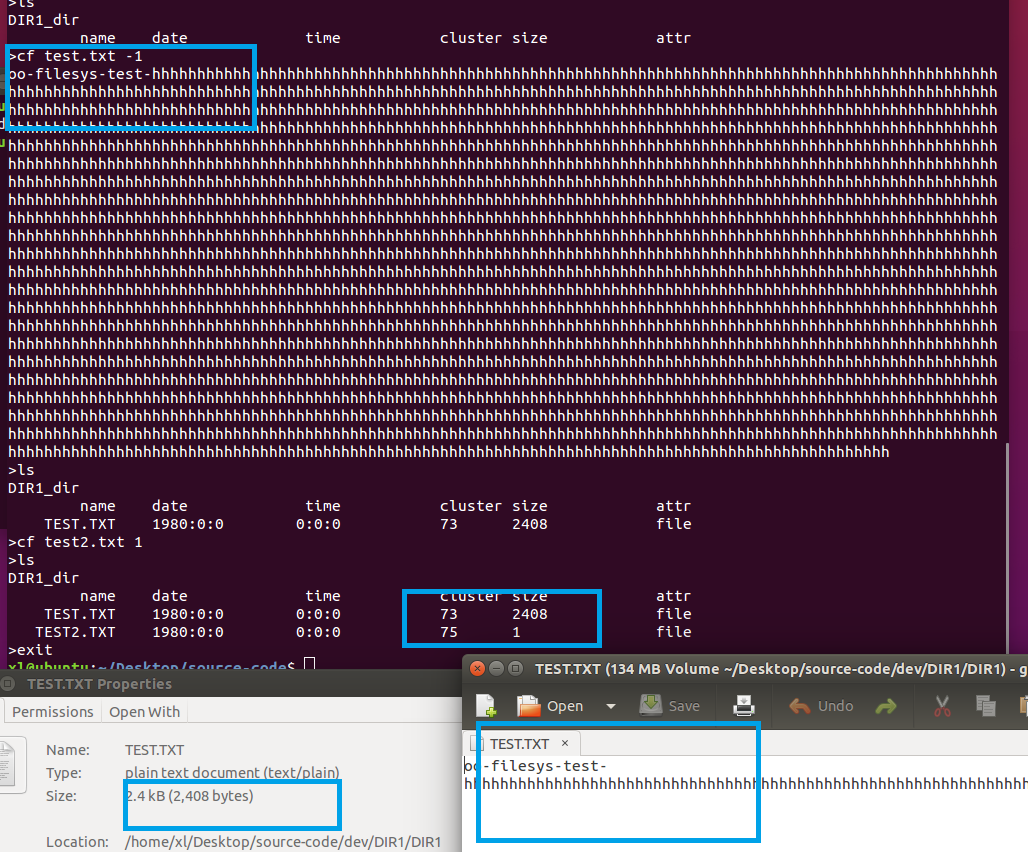


添加簇对应代码：

|  |
| --- |
| *//////////////////////////////////////////////////////////////*  *//最后一个cluster了还没找到能分配的，先撸新的cluster给目录*  *//查询fat表，找到空白簇*  for (cluster = 2; cluster < 1000; cluster++)  {  index = cluster \* 2;  *//みつけた！*  if (fatbuf[index] == 0x00 && fatbuf[index + 1] == 0x00)  {  *//新的cluster对应的fat表填ffff*  fatbuf[index] = 0xff;  fatbuf[index + 1] = 0xff;    *//cur\_cluster下一个变成新撸的*  fatbuf[cur\_cluster \* 2] = (cluster & 0x00ff);  fatbuf[cur\_cluster \* 2 + 1] = ((cluster & 0xff00) >> 8);  break;  }    }  *//////////////////////////////////////////////////////////////*  printf("%d\n%d\n", GetFatCluster(cur\_cluster), GetFatCluster(GetFatCluster(cur\_cluster))); |

* 文件写入实际内容的效果：

创建文件输入大小为负数则会继续输入文件内容，如下图可见，输入内容可以正常在gpedit中显示，而且输入内容超过2k（一簇大小）也能正常分为多簇写入，可见test.txt占了73和74两个簇。



写入实际内容对应代码：

|  |
| --- |
| */////////////////////////////写入实际内容///////////////////////////////////////*  if (!is\_dir&&write\_flag)  {  short temp\_cur\_cluster = RevByte(c[26], c[27]);  stringaddr = inputstring;  for (i = 0; i < clustersize; ++i) {  offset = DATA\_OFFSET + (temp\_cur\_cluster - 2) \* CLUSTER\_SIZE;  if (lseek(fd, offset, SEEK\_SET) < 0)  perror("lseek fd\_cf failed");  if (write(fd, stringaddr, CLUSTER\_SIZE) < 0)  perror("write failed");  stringaddr = stringaddr + CLUSTER\_SIZE;  temp\_cur\_cluster = GetFatCluster(temp\_cur\_cluster);  }  }  *///////////////////////////////////////////////////////////////////////////////* |

### 修改fd\_df函数

* 添加删除目录功能（实现提高要求二）

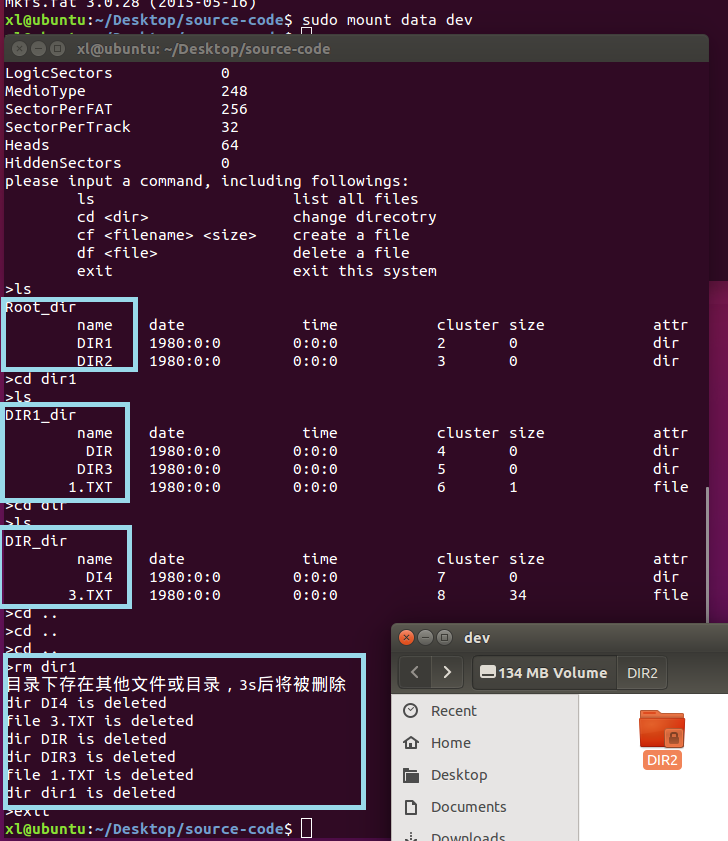
实现方法：扫描要被删除的目录的所有目录项，若为文件直接删掉，若为目录递归调用自己删掉，随后删除目标的目录

|  |
| --- |
| int fd\_df(char \*filename, int is\_dir)  {  int output\_flag = 1;  struct Entry \*pentry;  int ret;  unsigned char c;  unsigned short seed, next;  int cluster\_addr;  int offset;  struct Entry \*entry;    pentry = (struct Entry\*)malloc(sizeof(struct Entry));  entry = (struct Entry\*)malloc(sizeof(struct Entry));      if (is\_dir == 1 || is\_dir == 2)  {  ret = ScanEntry(filename, pentry, 1);  *//printf("----------------------\cluster:\t%d\n", curdir->FirstCluster);*  if (ret < 0)  {  printf("-----no such file-----\n");  free(pentry);  return -1;  }    *//备份当前目录*  struct Entry \*cur\_backup;  cur\_backup = (struct Entry\*)malloc(sizeof(struct Entry));  if (curdir == NULL)  cur\_backup = NULL;  else  memcpy(cur\_backup, curdir, sizeof(struct Entry));    *//pentry是找到的目录*  ret = ScanEntry(filename, pentry, 1);    if (ret < 0)  {  printf("no such dir------\n");  free(pentry);  return -1;  }    short cur\_cluster = pentry->FirstCluster;  *//printf("%s\t%d\n", pentry->short\_name, pentry->FirstCluster);*  while (1)  {  cluster\_addr = DATA\_OFFSET + (cur\_cluster - 2) \* CLUSTER\_SIZE;  if ((ret = lseek(fd, cluster\_addr, SEEK\_SET)) < 0)  perror("lseek cluster\_addr failed");    offset = cluster\_addr;    */\*读一簇的内容\*/*  while (offset < cluster\_addr + CLUSTER\_SIZE)  {  lseek(fd, offset, SEEK\_SET);  ret = GetEntry(entry);  offset += abs(ret);  if (ret > 0)  {  if (is\_dir == 1 && output\_flag == 1)  {  printf("目录下存在其他文件或目录，3s后将被删除\n");  sleep(3);  output\_flag = 0;  }  *//printf("-----------------------------\n");*  *//printf("%d\n", offset);*  *//printf("%s\t%d\t%d\n", entry->short\_name, entry->FirstCluster, GetFatCluster(entry->FirstCluster));*  if (entry->subdir)  {  *//printf("dir--------------\n");*  free(curdir);  curdir = (struct Entry\*)malloc(sizeof(struct Entry));  memcpy(curdir, pentry, sizeof(struct Entry));  fd\_df(entry->short\_name, 2);  *//sleep(2);*  if (cur\_backup == NULL)  curdir = NULL;  else  memcpy(curdir, cur\_backup, sizeof(struct Entry));  }  else  {  printf("file %s is deleted\n", entry->short\_name);  */\*清除fat表项\*/*  seed = entry->FirstCluster;  while ((next = GetFatCluster(seed)) != 0xffff)  {  ClearFatCluster(seed);  seed = next;  }    ClearFatCluster(seed);    if (WriteFat() < 0)  exit(1);    */\*清除目录表项\*/*  c = 0xe5;    if (lseek(fd, offset - 0x20, SEEK\_SET) < 0)  perror("lseek fd\_df failed");  if (write(fd, &c, 1) < 0)  perror("write failed");      }  }  }  *//  printf("%d\n", GetFatCluster(cur\_cluster));*  *//  sleep(5);*  if (GetFatCluster(cur\_cluster) != 0xffff)  {  *//printf("%d\t%d\n", pentry->FirstCluster, GetFatCluster(pentry->FirstCluster));*  *//printf("%d\t%d\n", cur\_cluster, GetFatCluster(cur\_cluster));*  *//printf("1113\n");*  cur\_cluster = GetFatCluster(cur\_cluster);  *//printf("2223\n");*  }  else  {  *//printf("12333\n");*  if (cur\_backup == NULL)  curdir = NULL;  else  memcpy(curdir, cur\_backup, sizeof(struct Entry));  free(cur\_backup);    ret = ScanEntry(filename, pentry, 1);  if (ret < 0)  {  printf("no such file\n");  free(pentry);  return -1;  }    */\*清除fat表项\*/*  *//printf("%s\t%d\n", pentry->short\_name, pentry->FirstCluster);*  seed = pentry->FirstCluster;  while ((next = GetFatCluster(seed)) != 0xffff)  {  ClearFatCluster(seed);  seed = next;    }  ClearFatCluster(seed);    */\*清除目录表项\*/*  c = 0xe5;    *//printf("%d\t%d\t%d\n", ret,pentry->FirstCluster, DATA\_OFFSET + (pentry->FirstCluster - 2) \* CLUSTER\_SIZE);*  if (lseek(fd, ret - 0x20, SEEK\_SET) < 0)  perror("lseek fd\_df failed");  if (write(fd, &c, 1) < 0)  perror("write failed");    free(pentry);  if (WriteFat() < 0)  exit(1);      break;  }  }  printf("dir %s is deleted\n", filename);  }  else  {  */\*扫描当前目录查找文件\*/*  ret = ScanEntry(filename, pentry, 0);    if (ret < 0)  {  printf("no such file\n");  free(pentry);  return -1;  }    */\*清除fat表项\*/*  seed = pentry->FirstCluster;  while ((next = GetFatCluster(seed)) != 0xffff)  {  ClearFatCluster(seed);  seed = next;    }    ClearFatCluster(seed);    */\*清除目录表项\*/*  c = 0xe5;    if (lseek(fd, ret - 0x20, SEEK\_SET) < 0)  perror("lseek fd\_df failed");  if (write(fd, &c, 1) < 0)  perror("write failed");    free(pentry);  if (WriteFat() < 0)  exit(1);  return 1;  }  } |

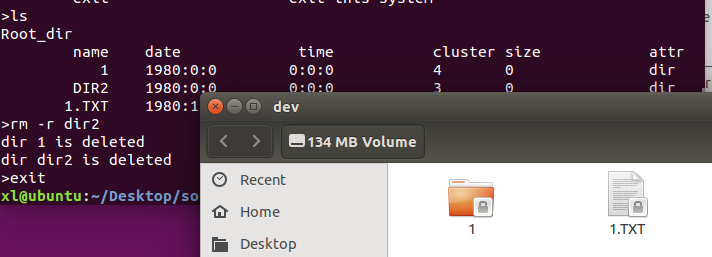
#### 效果图：

* rm:

如下图可见，原本根目录下有目录dir1和dir2，dir1中有目录dir和dir3，有文件1.txt，目录dir中有目录dir4和文件3.txt，现删除根目录下的dir1，输出提示信息后将dir1和其中所有内容删除。



* rm –r:与rm相同，但不输出提示信息，只输出删掉的内容



### 修改fd\_ls函数

* 非根目录由读一簇变为读所有簇（实现提高要求五）

实现方法：扫描要被删除的目录的所有目录项，若为文件直接删掉，若为目录递归调用自己删掉，随后删除目标的目录

|  |
| --- |
| int fd\_ls()  {    int ret, offset, cluster\_addr;  struct Entry entry;  unsigned char buf[DIR\_ENTRY\_SIZE];  if ((ret = read(fd, buf, DIR\_ENTRY\_SIZE)) < 0)  perror("read entry failed");    if (curdir == NULL)  printf("Root\_dir\n");  else  printf("%s\_dir\n", curdir->short\_name);  printf("\tname\tdate\t\t time\t\tcluster\tsize\t\tattr\n");    if (curdir == NULL)  */\*显示根目录区\*/*  {  */\*将fd定位到根目录区的起始地址\*/*  if ((ret = lseek(fd, ROOTDIR\_OFFSET, SEEK\_SET)) < 0)  perror("lseek ROOTDIR\_OFFSET failed");    offset = ROOTDIR\_OFFSET;    */\*从根目录区开始遍历，直到数据区起始地址\*/*  while (offset < (DATA\_OFFSET))  {  ret = GetEntry(&entry);    offset += abs(ret);  if (ret > 0)  {  printf("%12s\t"  "%d:%d:%d\t"  "%d:%d:%d   \t"  "%d\t"  "%d\t\t"  "%s\n",  entry.short\_name,  entry.year, entry.month, entry.day,  entry.hour, entry.min, entry.sec,  entry.FirstCluster,  entry.size,  (entry.subdir) ? "dir" : "file");  }  }  }    else */\*显示子目录\*/*  {  *//读取目录的所有cluster,而不是只读第一个cluster*  short cur\_cluster = curdir->FirstCluster;  while (1)  {    cluster\_addr = DATA\_OFFSET + (cur\_cluster - 2) \* CLUSTER\_SIZE;  if ((ret = lseek(fd, cluster\_addr, SEEK\_SET)) < 0)  perror("lseek cluster\_addr failed");    offset = cluster\_addr;  if (GetFatCluster(cur\_cluster) == 0)  {  fatbuf[cur\_cluster \* 2] = 0xff;  fatbuf[cur\_cluster \* 2 + 1] = 0xff;  WriteFat();  }  */\*读一簇的内容\*/*  while (offset < cluster\_addr + CLUSTER\_SIZE)  {  ret = GetEntry(&entry);  offset += abs(ret);  if (ret > 0)  {  printf("%12s\t"  "%d:%d:%d\t"  "%d:%d:%d   \t"  "%d\t"  "%d\t\t"  "%s\n",  entry.short\_name,  entry.year, entry.month, entry.day,  entry.hour, entry.min, entry.sec,  entry.FirstCluster,  entry.size,  (entry.subdir) ? "dir" : "file");  }  }  *//  printf("%d\n", GetFatCluster(cur\_cluster));*  *//  sleep(5);*  if (GetFatCluster(cur\_cluster) != 0xffff)  {  cur\_cluster = GetFatCluster(cur\_cluster);  }  else  {  break;  }  }  }  return 0;  } |

#### 效果图：

原本ls只读一簇，最多2k/32=64个目录项，现在改为读所有簇，能将所有目录项显示，如图。

#### 

### 修改main函数

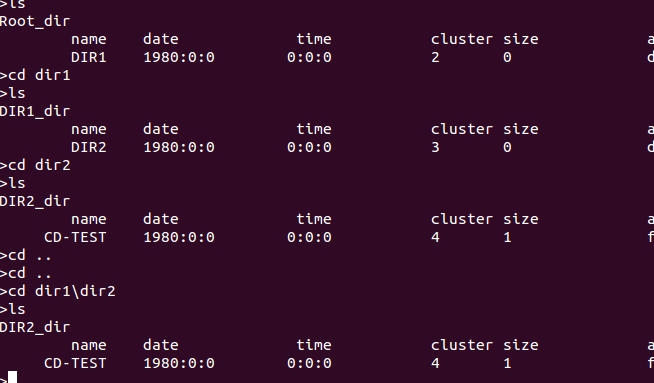
* 增加mkdir，rm，rm -r

增加绝对路径和多级路径的支持（实现提高要求三）

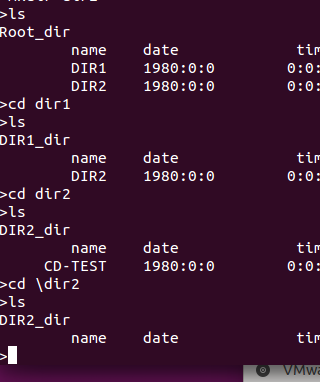
|  |
| --- |
| int main()  {  char input[10];  int size = 0;  char name[20];    */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*绝对路径和多层目录需要用到的\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  char tempname[20];  *//strtok函数会把name给改了,所以要来个副本*  int n = 0, i = 0, j = 0;  char \*path[10] = { NULL }; *//将绝对路径分解*  int pathNumber = 0;   *//绝对路径分成几部分,最后一部分是最终目录*  char \*p = "\\";  char \*nowp = NULL;  struct Entry \*tempcurdir = NULL;  int ret = 0;  struct Entry \*pentry;  pentry = (struct Entry\*)malloc(sizeof(struct Entry));  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*        if ((fd = open(DEVNAME, O\_RDWR)) < 0)  perror("open failed");  ScanBootSector(); *//打印启动项记录*  if (ReadFat() < 0) *//读fat表的信息，存入fatbuf[]中*  exit(1);  do\_usage(); *//打印提示信息*      while (1)  {  printf(">");  scanf("%s", input);    if (strcmp(input, "exit") == 0)  break;  else if (strcmp(input, "ls") == 0)  fd\_ls();  else if (strcmp(input, "cd") == 0)  {  scanf("%s", name);  for (i = 0; i < 20; i++) {  tempname[i] = name[i];  }  *//name中含有*  if (strstr(name, p)) {  i = 0;  path[i] = strtok(tempname, p);  while (path[i] != NULL) {  i++;  path[i] = strtok(NULL, p);  }  pathNumber = i;    *//如果第一个目录是根目录下的,就是绝对路径*  tempcurdir = curdir;  curdir = NULL;  ret = ScanEntry(path[0], pentry, 1);  *//相对路径*  if (ret < 0) {  curdir = tempcurdir;  for (i = 0; i < pathNumber; i++) {  fd\_cd(path[i]);  }  }  else {  dirno = 0;  for (i = 0; i < pathNumber; i++) {  fd\_cd(path[i]);  }  }  }  else {  fd\_cd(name);  }  }  else if (strcmp(input, "df") == 0)  {  scanf("%s", name);  fd\_df(name, 0);  }  else if (strcmp(input, "rm") == 0)  {  scanf("%s", name);  if (strcmp(name, "-r") == 0)  {  scanf("%s", name);  fd\_df(name, 2);  }  else  {  fd\_df(name, 1);  }  }  else if (strcmp(input, "cf") == 0)  {  scanf("%s", name);  scanf("%s", input);  size = atoi(input);  fd\_cf(name, size, 0);  }  else if (strcmp(input, "mkdir") == 0)  {  scanf("%s", name);    fd\_cf(name, 0, 1);  }  else  do\_usage();  }    return 0;  } |

#### 效果图：

* 多级路径：



* 绝对路径：



设计思路：

对输入的路径进行判断，如果含有“\”，说明输入的是绝对路径，否则是一般的目录或文件名。对后者不进行特殊处理。

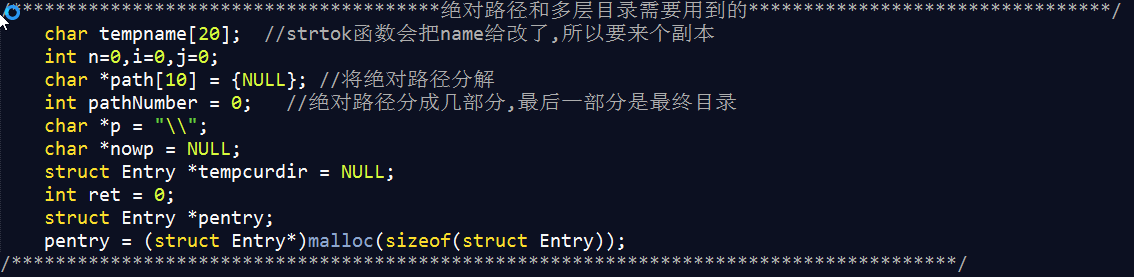
对于前者，将输入的路径中的目录都提取出来，如果路径中的第一个目录是根目录下的一个目录的话，说明这是个绝对路径，否则是个相对路径。

对于绝对路径，只要将curdir=NULL，dirno=0，然后从路径中的第一个目录开始调用fd\_cd()函数即可。

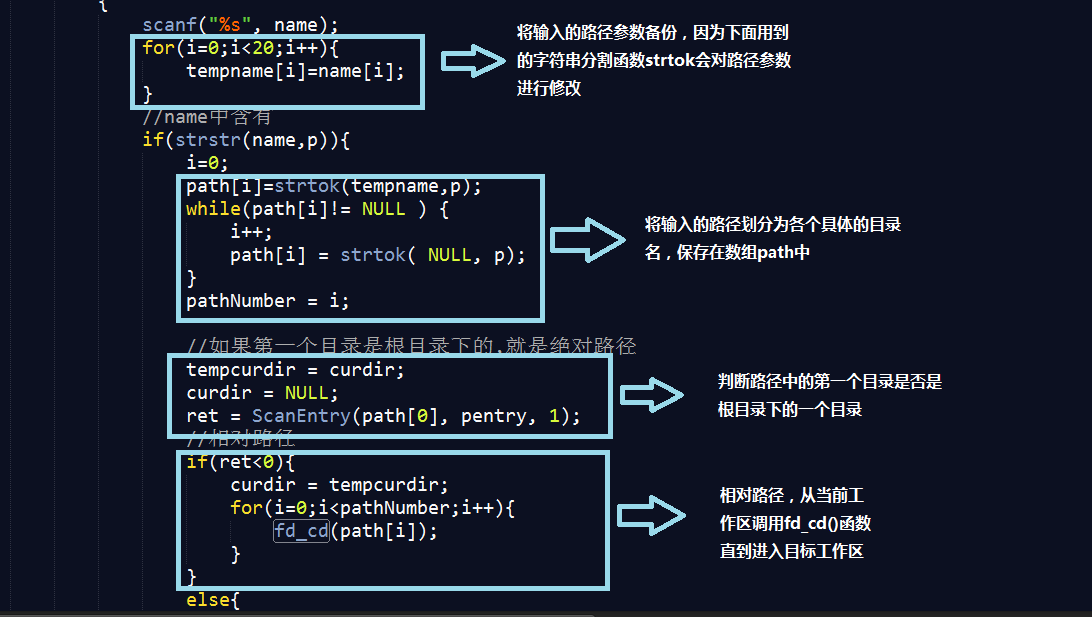
对于相对路径，更简单，之间从路径中的第一个目录开始调用fd\_cd()函数。

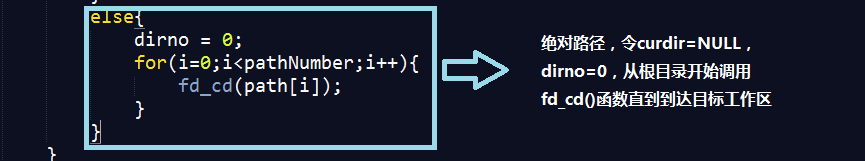
代码实现：

需要用到的参数:



操作代码:





## 最新修改

增加功能：创建文件时，写入当前系统时间

增加了scan函数，在程序运行开始递归所有目录，将空目录的firstcluster的fat表项从0改回ffff