



山东大学
SHANDONG UNIVERSITY

操作系统实验

实验名称 A simple file system

摘要

文件系统是操作系统用于明确存储设备或分区上的文件的方法和数据结构,即在存储设备上组织文件的方法。操作系统中负责管理和存储文件信息的软件机构称为文件管理系统,简称文件系统。文件系统由三部分组成:文件系统的接口,对对象操纵和管理的软件集合,对象及属性。从系统角度来看,文件系统是对文件存储设备的空间进行组织和分配,负责文件存储并对存入的文件进行保护和检索的系统。具体地说,它负责为用户建立文件,存入、读出、修改、转储文件,控制文件的存取,当用户不再使用时撤销文件等。

本次实验所要实现的简单文件系统,支持创建文件,写入数据,按名存取,读数据,删除文件,查看目录等功能。需要维护一个 FCB 数组,来记录文件位置,同时也要维护 FAT 表,来保存文件名。读写数据只需要知道文件名即可得到对应的在“硬盘”里的位置,完成读写操作。

目录

1 任务一	3
1.1 任务内容	3
1.2 实验过程	3
1.2.1 初始化操作	3
1.2.2 文件和目录项创建	4
1.2.3 ls 操作	6
1.2.4 cd 操作	6
1.2.5 open 和 delete 操作	7
1.2.6 write 操作	9
1.2.7 read 操作	11
1.3 结果展示	12
2 任务二	12
2.1 实验内容	12
2.2 实验过程	13
2.2.1 思路	13
2.2.2 switch 操作	13
2.2.3 进程读操作	14
2.2.4 进程写操作	15
2.2.5 文件关闭操作	16
2.3 结果展示	17
A 完整代码	18

1 任务一

1.1 任务内容

实现一个简单的类 FAT-16 文件系统，具有以下功能：

mkdir,ls,delete,open,read,write,close

系统是基于内存的：建立一个数组：

```
#define DISK_MAXLEN 2560
```

```
char Disk[DISK_MAXLEN];
```

把这个数组当成硬盘，实现文件系统。

假设只有一个进程使用。

1.2 实验过程

1.2.1 初始化操作

初始化文件系统时，首先初始化 FCB 控制块，初始化用户，并在磁盘中初始化 FAT 表，这里磁盘由于不设置分区，我们只设置 DBR 用来存储目录大小和 FAT 的长度。首先我们将根目录指针下的目录项设置为 0，然后我们计算 FAT 表的长度，并由 FAT 表长度推算数据存放的起始位置。

```
1 void Init()  
2 {  
3     /* 增加进程初始化 */  
4     UserInit[0] = true;  
5     cur = &User[0];  
6     User[0].id = 0;  
7     User[0].state = 0;  
8     strcpy_s(User[0].path, "SDU_user0:\\root\\");  
9     path = User[0].path;  
10    User[0].curtable = &rootdirtable;  
11    currentdirtable = &User[0].curtable;  
12  
13    rootdirtable.dirnumber = 0;
```

```

14     for (int i = 0; i < FAT_length; i++) {
15         FAT[i] = 0;
16     }
17     /*Current root table address*/
18     /*First set the DBR information
19     fat_length and dirtable_length
20     */
21     DBR[0] = FAT_length;
22     DBR[1] = dirtable_length;
23     //the start type number of memory
24
25 }

```

1.2.2 文件和目录项创建

我们在目录项中设置了用来标记是目录文件还是数据文件的标识位 type, 当 type=0 进行目录项增加操作, 当 type = 1 时我们作为数据文件进行创建, 在创建文件时, 我们首先要检查当前目录是否已满, 然后检查是否有同名文件, 最后更改目录维护文件中的相关字段。并且我们要给相应的文件分配一个新的 FCB 文件控制块项, 并在 FAT 中找到空闲块地址作为文件存储的起始位置。

```

1 void add_dir(char filename[], int type) {
2     int dirnumber = (*currentdirtable)->dirnumber;
3     /*check the dirunit is full or not*/
4     if (dirnumber == dirtable_length) {
5         cout << "Not have enough space to add file";
6         return;
7     }
8     /*check whether have the same file or not*/
9     if (Findunit(filename) != -1) {
10         cout << "The file has exist.";
11         return;
12     }
13     /*The dirtable isn't full and isn't have the same file*/
14     dirunit* newdirunit = &(*currentdirtable)->dirs[dirnumber];

```

```
15     (*currentdirttable)->dirnumber++;
16     strcpy_s(newdirunit->filename, filename);
17     newdirunit->type = type;
18     return;
19 }
20
21 void touch(char filename[], int flen, int type) {
22     if (strlen(filename) > 80) {
23         cout << "File name is illegal.";
24         return;
25     }
26     add_dir(filename, type);
27     int index = Findunit(filename);
28     fcb* curfcb = (fcb*) new fcb();
29     int i = Find_freeFAT();//find a free FAT
30     curfcb->blockindex = i;
31     FAT[i] = -1;
32     curfcb->datalength = 0;
33     FCB[FCBcnt] = curfcb;//将新创建的文件FCB保存
34     (*currentdirttable)->dirs[index].startfcb = FCBcnt;
35     (*currentdirttable)->dirs[index].startdir = -1;
36     ++FCBcnt;
37     //FAT creat
38
39
40     Fmux[i] = { 1,0,1 };
41
42     /*The type 0 is creat a new dir*/
43     if (type == 0) {
44         dirttable* newcurdirttable = (dirttable*)new dirttable();
45         /*set a new struct or class use ()*/
46         table[TAB_size] = newcurdirttable;
47         newcurdirttable->dirnumber = 0;
48         (*currentdirttable)->dirs[index].startdir = TAB_size;
49         TAB_size++;
```

```
50     }  
51 }
```

1.2.3 ls 操作

对于 ls 操作我们只需要将当前目录项指针所指目录中所有的文件进行遍历，将文件名进行打印输出。

```
1 void ls() {  
2     int uninum = (*currentdirtable)->dirnumber;  
3     for (int i = 0; i < uninum; i++) {  
4         dirunit curunit = (*currentdirtable)->dirs[i];  
5         cout << curunit.filename << " ";  
6     }  
7     cout << endl;  
8 }
```

1.2.4 cd 操作

cd 操作用来切换当前文件目录，当为 cd .. 时我们采用操作退回上一级文件目录，并将 nowtable 减一，实现物理指针上的目录回退。如果是 cd+filename 为进入下一级文件夹，这时候我们要判断所进入文件项是否是目录文件，如果不是目录文件，则返回错误信息，如果是目录文件则更新当前目录指针，将新目录名拼接在当前文件目录名之后。

```
1 void cd(char dirname[]) {  
2     if (strcmp(dirname, "..") == 0) {  
3         int length = strlen(path);  
4         for (int i = length - 2; i >= 0; i--) {  
5             if (path[i] == '\\') {  
6                 path[i + 1] = '\0';  
7                 break;  
8             }  
9         }  
10        if (now_table != 0){
```

```
11         now_table--;
12         (*currentdirtable) = table[now_table];
13         return;
14     }
15     else {
16         (*currentdirtable) = &rootdirtable;
17     }
18     return;
19 }
20 int index = Findunit(dirname);
21 if (index == -1) {
22     cout << "Not found the Dir." << endl;
23     return;
24 }
25 if ((*currentdirtable)->dirs[index].type == 1) {
26     cout << "Not a dir." << endl;
27     return;
28 }
29 int i = (*currentdirtable)->dirs[index].startdir;
30 (*currentdirtable) = table[i];
31 now_table = i;
32
33
34 strcat_s(cur->path, dirname);
35 strcat_s(cur->path, "\\");
36
37 }
```

1.2.5 open 和 delete 操作

open 操作我们需要检查当前目录下是否存在相应文件，如果存在则取出相应的 FCB 表项，然后根据 FCB 表项找到文件分配表项的起始位置。

delete 操作，我们首先也是要检查当前目录下是否存在相应的文件，如果存在，则

进行删除。并将相应的 FAT 表进行迭代删除处理。这里我们应该注意判断删除文件是一个文件目录还是一个数据文件，如果是文件目录，需要在文件目录表中将目录项进行删除。

```
1 void freeFAT(int unitindex) {
2     int i = (*currentdirtable)->dirs[unitindex].startfcb;
3     int k = FCB[i]->blockindex;
4     while (k == -1)
5     {
6         int temp = i;
7         i = FAT[i];
8         FAT[temp] = 0;
9     }
10    if (k == -1) {
11        FAT[i] = 0;
12        return;
13    }
14 }
15
16 void deleteunit(int unitindex) {
17     int dirnumber = (*currentdirtable)->dirnumber;
18     int FCBstart = (*currentdirtable)->dirs[unitindex].startfcb;
19     int DIRstart = (*currentdirtable)->dirs[unitindex].startdir;
20     int FATstart = FCB[FCBstart]->blockindex;
21     if (DIRstart != -1) {
22         for (int i = DIRstart; i < (*currentdirtable)->dirnumber -
23             2; i++) {
24             table[i] = table[i + 1];
25             TAB_size--;
26         }
27     }
28     for (int i = FCBstart; i < FCBcnt - 1; i++) {
29         FCB[i] = FCB[i + 1];
30         FCBcnt--;
```

```

31     freeFAT(unitindex);
32     for (int i = unitindex; i < dirnumber - 1; i++) {
33         (*currentdirtable)->dirs[i] = (*currentdirtable)->dirs[i +
34             1];
35     }
36     (*currentdirtable)->dirnumber--;
37 }
38
39 void deletefile(char filename[]) {
40     int unitindex = Findunit(filename);
41     if (unitindex == -1) {
42         cout << "Sorry, not found." << endl;
43         return;
44     }
45     deleteunit(unitindex);
46 }

```

1.2.6 write 操作

进行 write 操作时，首先检查当前目录下是否存在相应文件，并检查文件是否是数据文件。然后将文件打开，找到 FAT 表的开始位置，并计算数据长度，并给文件分配相应大小，然后根据 FAT 表逐步将硬盘的相应位置写入数据。这里磁盘我们以 32 个字节为一个文件簇，对于文件输入来说，首先我们要判断输入内容长度是否大于 32 个字节长度，如果是大于 32 个字节长度，则需要多个文件簇进行写入。

```

1 void fat_op(short number, int unitindex) {
2     int index = (*currentdirtable)->dirs[unitindex].startfcb;
3     int startfat = FCB[index]->blockindex;
4     if (number == 1) {
5         FAT[startfat] = -1;
6     }
7     else {
8         int index = startfat;

```

```
9         for (int i = 1; i < number; i++) {
10             int j = Find_freeFAT();
11             FAT[index] = j;
12             index = j;
13         }
14         FAT[index] = -1;
15     }
16 }
17
18
19 void write(char filename[], char content[]) {
20     int unitindex = Findunit(filename);
21     if (unitindex == -1) {
22         cout << "sorry,not found" << endl;
23         return;
24     }
25
26     int length = strlen(content);
27     int num = (length % 32 == 0) ? (length / 32) : (length / 32 + 1);
28     fat_op(num, unitindex);
29     FCB[(*currentdirtable)->dirs[unitindex].startfcb->datalength =
        num;
30     int index1 = (*currentdirtable)->dirs[unitindex].startfcb;
31     int startblock = FCB[index1]->blockindex;
32     int block = length / 32 + 1;
33     int index = startblock;
34     for (int j = 0; j < block-1; j++) {
35         for (int k = 0; k < 32; k++)
36         {
37             Disk[k + index * 32] = content[k+j*32];
38         }
39         index = FAT[index];
40     }
41     int j = block - 1;
42     for (int k = 0; k < length % 32; k++) {
```

```
43         Disk[k + index * 32] = content[k + j * 32];
44     }
45     cout << endl;
46 }
```

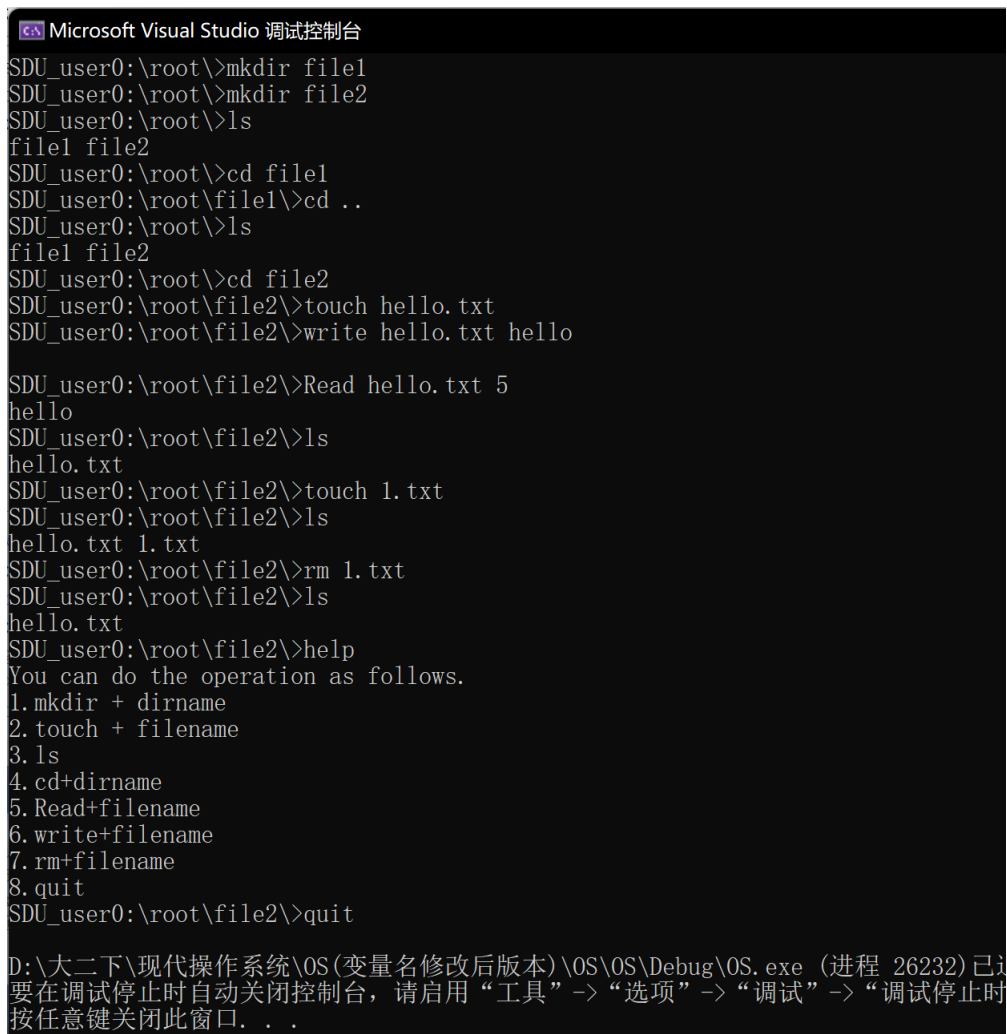
1.2.7 read 操作

读操作和写操作类似，首先在目录中检查文件是否存在，然后根据文件的 FCB 表项找到对应的 FAT 位置，再根据 FAT 表迭代对文件内容进行读取。这里 read 时要根据读取的数据长度对 FAT 表进行访问，FAT 表每一项代表一个文件簇，也就是 32 字节。

```
1 void read(char filename[], int length) {
2     int unitindex = Findunit(filename);
3     if (unitindex == -1) {
4         cout << "Sorry, not found." << endl;
5         return;
6     }
7     //read data in the file
8     int index1 = (*currentdirtable)->dirs[unitindex].startfcb;
9     int startblock = FCB[index1]->blockindex;
10    int num = (length % 32 == 0) ? (length / 32) : (length / 32 + 1);
11    int block = length / 32 + 1;
12    int index = startblock;
13    for (int j = 0; j < block - 1; j++) {
14        for (int k = 0; k < 32; k++)
15        {
16            cout<<Disk[k + index * 32];
17        }
18        index = FAT[index];
19    }
20    int j = block - 1;
21    for (int k = 0; k < length % 32; k++) {
22        cout<<Disk[k + index * 32];
23    }
```

```
24     cout << endl;  
25 }
```

1.3 结果展示



```
Microsoft Visual Studio 调试控制台  
SDU_user0:\root>mkdir file1  
SDU_user0:\root>mkdir file2  
SDU_user0:\root>ls  
file1 file2  
SDU_user0:\root>cd file1  
SDU_user0:\root\file1>cd ..  
SDU_user0:\root>ls  
file1 file2  
SDU_user0:\root>cd file2  
SDU_user0:\root\file2>touch hello.txt  
SDU_user0:\root\file2>write hello.txt hello  
  
SDU_user0:\root\file2>Read hello.txt 5  
hello  
SDU_user0:\root\file2>ls  
hello.txt  
SDU_user0:\root\file2>touch 1.txt  
SDU_user0:\root\file2>ls  
hello.txt 1.txt  
SDU_user0:\root\file2>rm 1.txt  
SDU_user0:\root\file2>ls  
hello.txt  
SDU_user0:\root\file2>help  
You can do the operation as follows.  
1.mkdir + dirname  
2.touch + filename  
3.ls  
4.cd+dirname  
5.Read+filename  
6.write+filename  
7.rm+filename  
8.quit  
SDU_user0:\root\file2>quit  
  
D:\大二下\现代操作系统\OS(变量名修改后版本)\OS\OS\Debug\OS.exe (进程 26232) 已  
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时  
按任意键关闭此窗口... ”
```

2 任务二

2.1 实验内容

实现多个进程同时访问文件、目录的功能。一个文件可以被多个进程读,但是只能被一个进程写。每个进程维护一个变量存放当前的目录位置。

2.2 实验过程

2.2.1 思路

首先,新增进程的结构,包括进程号 id,读写状态 state,进程当前目录位置 curtable,以及当前路径的字符地址 path。此外,还需要新增 fat 系统的信号量变量,由于要实现的问题类似于读者写者问题,故相关变量也类似:读写文件的互斥量 rm,记录读者数量的 count,关于 count 的互斥量 rc。

并且我们要求全局的 curtable 变量可以在不同进程的 curtable 自由切换,所以类型声明为 dirtable** 型,即指向 dirtable* 型指针的指针。

对于每个进程我们用 Pro 型数组 User 来表示,并用 swich 命令进行进程切换。

由于 VSC++ 环境下不支持进程级的阻塞和调度,所以我们的实现结构近似于常见的图形界面文件操作系统,使用 swich 新建或切换进程,相当于打开多个文件管理器的页面。为了模拟进程切换和文件读写需要增加命令, xread, ywirte, zclose, 为多进程下的读,写,关闭操作。有进程对一个文件读时,其他进程可读不可写,有进程写时,其他进程不可读写,并附加关闭操作。

2.2.2 swich 操作

对于 swich 操作,首先比较要切换的进程是否为现在进程。若为新进程,且未初始化过,那么对进程的各项值进行初始化,然后更改当前目录的指针,字符地址和进程。若已经初始化过,则直接切换。

```
1 void Swich(int i) {
2     if (cur->id == i) return;
3     if (UsrInit[i] == 0) {
4         UsrInit[i] = true;
5         User[i].id = i;
6         User[i].state = 0;
7         strcpy_s(User[i].path, 0rgin.c_str());
8         User[i].curtable = &rootdirtable;
9         User[i].path[8] = i + 48;
10    }
11 }
```

```

12     cur = &User[i];
13     path = User[i].path;
14     curdirrrtable = &User[i].curtable;
15 }

```

2.2.3 进程读操作

在进程读操作中，需传入待读的文件名和读入长度。首先根据 filename 找到文件的目录序号，然后转换为文件系统的 fat 地址序号，此时便可确定此文件对应的一组解决互斥同步的信号量。对于 P，V 操作我们的处理为判断资源是否可用，并返回判断结果。然后先进行对 count 的互斥访问，再进行 rm 的访问，若当前资源可读，则进行读入。

```

1 void OSRead(char filename[], int length) { //OS read
2
3     int i = findunit(filename);
4     if (i == -1) {
5         cout << "sorry,not found" << endl;
6         return;
7     }
8     i = (*curdirrrtable)->dirs[i].startfat;
9
10    bool f;
11    f=P(Fmux[i].rc);
12    if (!f) {
13        wrong();
14        return;
15    }
16    ++Fmux[i].count;
17    if (Fmux[i].count == 1)f=P(Fmux[i].rm);
18    if (!f) {
19        --Fmux[i].count;
20        V(Fmux[i].rc);
21        wrong();
22        return;

```

```
23     }
24
25     V(Fmux[i].rc);
26     cur->state = 1;
27     printf("Reading %s!\n", filename);
28     read(filename, length);
29
30 }
```

2.2.4 进程写操作

在进程写操作中，输入为待写的文件名和写入内容。也是先确定相关文件的信号量。然后通过信号量的结果对文件进行写。

```
1 void OSWrite(char filename[], char content[]) { // OS write
2
3     int i = findunit(filename);
4     if (i == -1) {
5         cout << "sorry, not found" << endl;
6         return;
7     }
8     i = (*curdirrrtable)->dirs[i].startfat;
9
10    bool f;
11    f = P(Fmux[i].rm);
12    if (!f) {
13        wrong();
14        return;
15    }
16    cur->state = 2;
17    printf("Writing %s!\n", filename);
18    write(filename, content);
19
20 }
```


2.2.5 文件关闭操作

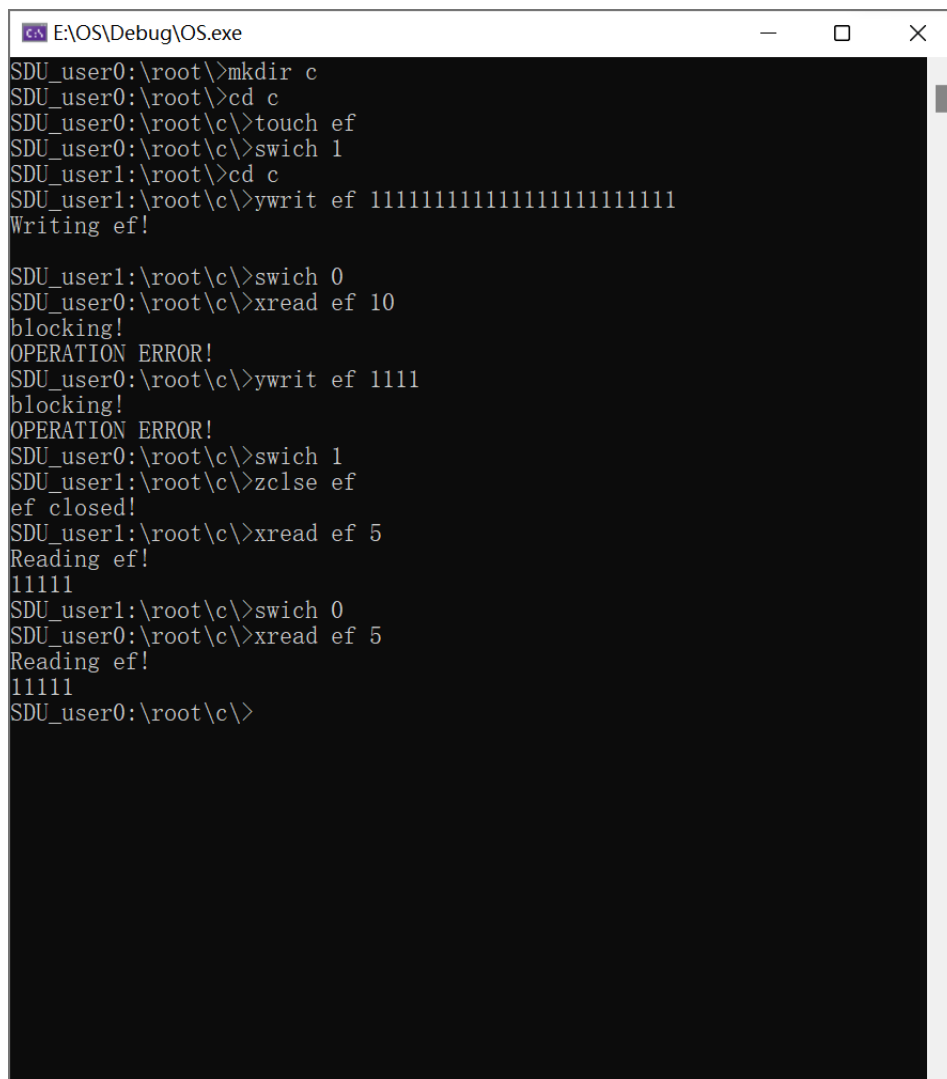
对于文件关闭操作，首先判断当前进程是否为读写状态，state 为 1 则为读，state 为 2 则为写，然后确定文件标识。由于读写时互斥量状态不同，故关闭时操作也不同。对于读文件进程的关闭需要修改文件的 count 变量，并在 count 为 0 时，即无进程读时，释放 rm，表示可以写或读。而对于写进程则直接释放 rm 即可。

并且对于一个进程，若当前在读写文件，下一步为打开新目录或创建新文件，那么会自动执行关闭当前文件的指令。

```
1 void Close(char filename[]) { //OS close
2     int& x = cur->state;
3     if (x==0) {
4         wrong();
5         return;
6     }
7
8     int i = findunit(filename);
9     if (i == -1) {
10        cout << "sorry,not found" << endl;
11        return;
12    }
13    i = (*curdirrrtable)->dirs[i].startfat;
14
15    bool f;
16    if (x == 1) {
17        f = P(Fmux[i].rc);
18        if (f)--Fmux[i].count;
19        else {
20            wrong();
21            return;
22        }
23        if (Fmux[i].count == 0)V(Fmux[i].rm);
24        V(Fmux[i].rc);
25    }
26    else {
```

```
27         V(Fmux[i].rm);
28     }
29     x = 0;
30     printf("%s closed!\n", filename);
31 }
```

2.3 结果展示



```
E:\OS\Debug\OS.exe
SDU_user0:\root\>mkdir c
SDU_user0:\root\>cd c
SDU_user0:\root\c\>touch ef
SDU_user0:\root\c\>swich 1
SDU_user1:\root\>cd c
SDU_user1:\root\c\>ywrit ef 11111111111111111111
Writing ef!

SDU_user1:\root\c\>swich 0
SDU_user0:\root\c\>xread ef 10
blocking!
OPERATION ERROR!
SDU_user0:\root\c\>ywrit ef 1111
blocking!
OPERATION ERROR!
SDU_user0:\root\c\>swich 1
SDU_user1:\root\c\>zclse ef
ef closed!
SDU_user1:\root\c\>xread ef 5
Reading ef!
11111
SDU_user1:\root\c\>swich 0
SDU_user0:\root\c\>xread ef 5
Reading ef!
11111
SDU_user0:\root\c\>
```

A 完整代码

OS_header.h

```
1 #pragma once
2 #include<iostream>
3 #include<stdio.h>
4 #include<stdlib.h>
5 #include<string.h>
6
7 using namespace std;
8
9 /*START*/
10 const int DISK_MAXLEN = 2560;
11 //size of the disk
12 const int dirtable_length = 80;
13 //FAT table length
14 const int FAT_length = 80;
15
16 /*Length of the disk set by byte*/
17 char Disk[DISK_MAXLEN];
18 int DBR[2];
19
20
21
22
23 /*
24 说明：一个磁盘对应一个FAT
25 并且保存着文件的FCB信息
26 这里文件目录实现按名存取的功能，存储文件名和对应FCB的位置
27 我们认为FAT和FCB已经提前加入到内存中
28
29 */
30 struct fcb {
31     short blockindex;
32     short datalength;
```

```
33 };
34
35 /*
36  the basic dirunit:
37  *fanme
38  *type
39  * the FAT start position
40  * the FCB start position
41  * the dir start position
42
43  */
44 struct dirunit {
45     char filename[80];
46     char type;
47     short startfcb;
48     short startdir;
49 };
50
51
52 struct dirtable {
53     int dirnumber;
54     dirunit dirs[dirtable_length];
55 };
56 /*dirtable consist the basic dirunit*/
57
58
59 struct Pro {
60     int id;
61     int state;
62     char path[100];
63     dirtable *curtable;
64 }User[11];
65
66
67 /* 最好放到fat里 */
```

```
68 struct FatMux {
69     int rm;
70     int count;
71     int rc;
72 }Fmux[FAT_length];
73
74 bool UserInit[11] = {0};
75
76 Pro* cur;
77
78 short FAT[FAT_length];
79 dirtable* table[55];
80 fcb* FCB[FAT_length];
81
82 dirtable rootdirtable;
83
84 /* 2阶指针 */
85 dirtable** currentdirtable;
86
87
88 int now_table = 0;
89
90 /* path更换 */
91 char *path;
92 string Origin("SDU_user0:\\root\\");
```

OS.cpp

```
1 #include "OS_header.h"
2
3
4 void Init()
5 {
6     /* 增加进程初始化 */
7     UserInit[0] = true;
8     cur = &User[0];
```

```
9      User[0].id = 0;
10     User[0].state = 0;
11     strcpy_s(User[0].path, "SDU_user0:\\root\\");
12     path = User[0].path;
13     User[0].curtable= &rootdirtable;
14     currentdirtable = &User[0].curtable;
15
16     rootdirtable.dirnumber = 0;
17     for (int i = 0; i < FAT_length; i++) {
18         FAT[i] = 0;
19     }
20     /*Current root table address*/
21     /*First set the DBR information
22     fat_length and dirtable_length
23     */
24     DBR[0] = FAT_length;
25     DBR[1] = dirtable_length;
26     //the start type number of memory
27
28 }
29
30 int FCBcnt = 0;
31 int TAB_size = 0;
32
33 int Findunit(char filename[]) {
34     int dirnumber = ((*currentdirtable)->dirnumber;
35     int unitIndex = -1;
36     //traversal
37     for (int i = 0; i < dirnumber; i++) {
38         if (strcmp(filename, (*currentdirtable)->dirs[i].filename)
39             == 0)
40             unitIndex = i;
41     }
42     return unitIndex;
43 }
```

```
43
44
45 void add_dir(char filename[], int type) {
46     int dirnumber = (*currentdirtable)->dirnumber;
47     /*check the dirunit is full or not*/
48     if (dirnumber == dirtable_length) {
49         cout << "Not have enough space to add file";
50         return;
51     }
52     /*check whether have the same file or not*/
53     if (Findunit(filename) != -1) {
54         cout << "The file has exist.";
55         return;
56     }
57     /*The dirtable isn't full and isn't have the same file*/
58     dirunit* newdirunit = &(*currentdirtable)->dirs[dirnumber];
59     (*currentdirtable)->dirnumber++;
60     strcpy_s(newdirunit->filename, filename);
61     newdirunit->type = type;
62     return;
63 }
64
65 /*Find free FAT the 0 is not allocate*/
66 int Find_freeFAT() {
67     for (short i = 0; i < FAT_length; i++) {
68         if (FAT[i] == 0)
69             return i;
70     }
71 }
72
73 void touch(char filename[], int flen, int type) {
74     if (strlen(filename) > 80) {
75         cout << "File name is illegal.";
76         return;
77     }
```

```
78     add_dir(filename, type);
79     int index = Findunit(filename);
80     fcb* curfcb = (fcb*) new fcb();
81     int i = Find_freeFAT();
82     curfcb->blockindex = i;
83     FAT[i] = -1;
84     curfcb->datalength = 0;
85     FCB[FCBcnt] = curfcb;///!!!
86     (*currentdirtable)->dirs[index].startfcb = FCBcnt;
87     (*currentdirtable)->dirs[index].startdir = -1;
88     ++FCBcnt;
89     //FAT creat
90
91
92     Fmux[i] = { 1,0,1 };
93
94     /*The type 0 is creat a new dir*/
95     if (type == 0) {
96         dirtable* newcurdirtable = (dirtable*)new dirtable();
97         /*set a new struct or class use ()*/
98         table[TAB_size] = newcurdirtable;
99         newcurdirtable->dirnumber = 0;
100        (*currentdirtable)->dirs[index].startdir = TAB_size;
101        TAB_size++;
102    }
103 }
104
105 void ls() {
106     int uninum = (*currentdirtable)->dirnumber;
107     for (int i = 0; i < uninum; i++) {
108         dirunit curunit = (*currentdirtable)->dirs[i];
109         cout << curunit.filename << " ";
110     }
111     cout << endl;
112 }
```



```

113
114 void freeFAT(int unitindex) {
115     int i = (*currentdirtable)->dirs[unitindex].startfcb;
116     int k = FCB[i]->blockindex;
117     while (k == -1)
118     {
119         int temp = i;
120         i = FAT[i];
121         FAT[temp] = 0;
122     }
123     if (k == -1) {
124         FAT[i] = 0;
125         return;
126     }
127 }
128
129 void deleteunit(int unitindex) {
130     int dirnumber = (*currentdirtable)->dirnumber;
131     int FCBstart = (*currentdirtable)->dirs[unitindex].startfcb;
132     int DIRstart = (*currentdirtable)->dirs[unitindex].startdir;
133     int FATstart = FCB[FCBstart]->blockindex;
134     if (DIRstart != -1) {
135         for (int i = DIRstart; i < (*currentdirtable)->dirnumber -
136             2; i++) {
137             table[i] = table[i + 1];
138             TAB_size--;
139         }
140     }
141     for (int i = FCBstart; i < FCBcnt - 1; i++) {
142         FCB[i] = FCB[i + 1];
143         FCBcnt--;
144     }
145     freeFAT(unitindex);
146     for (int i = unitindex; i < dirnumber - 1; i++) {
147         (*currentdirtable)->dirs[i] = (*currentdirtable)->dirs[i +

```

```
        1];
147     }
148     (*currentdirtable)->dirnumber--;
149
150 }
151
152 void deletefile(char filename[]) {
153     int unitindex = Findunit(filename);
154     if (unitindex == -1) {
155         cout << "Sorry, not found." << endl;
156         return;
157     }
158     deleteunit(unitindex);
159 }
160
161
162
163 void cd(char dirname[]) {
164     if (strcmp(dirname, "..") == 0) {
165         int length = strlen(path);
166         for (int i = length - 2; i >= 0; i--) {
167             if (path[i] == '\\') {
168                 path[i + 1] = '\0';
169                 break;
170             }
171         }
172         if (now_table != 0){
173             now_table--;
174             (*currentdirtable) = table[now_table];
175             return;
176         }
177         else {
178             (*currentdirtable) = &rootdirtable;
179         }
180         return;

```

```

181     }
182     int index = Findunit(dirname);
183     if (index == -1) {
184         cout << "Not found the Dir." << endl;
185         return;
186     }
187     if ((*currentdirtable)->dirs[index].type == 1) {
188         cout << "Not a dir." << endl;
189         return;
190     }
191     int i = (*currentdirtable)->dirs[index].startdir;
192     (*currentdirtable) = table[i];
193     now_table = i;
194
195
196     strcat_s(cur->path, dirname);
197     strcat_s(cur->path, "\\");
198
199 }
200
201 void read(char filename[], int length) {
202     int unitindex = Findunit(filename);
203     if (unitindex == -1) {
204         cout << "Sorry, not found." << endl;
205         return;
206     }
207     //read data in the file
208     int index1 = (*currentdirtable)->dirs[unitindex].startfcb;
209     int startblock = FCB[index1]->blockindex;
210     int num = (length % 32 == 0) ? (length / 32) : (length / 32 + 1);
211     int block = length / 32 + 1;
212     int index = startblock;
213     for (int j = 0; j < block - 1; j++) {
214         for (int k = 0; k < 32; k++)
215             {

```

```
216         cout<<Disk[k + index * 32];
217     }
218     index = FAT[index];
219 }
220 int j = block - 1;
221 for (int k = 0; k < length % 32; k++) {
222     cout<<Disk[k + index * 32];
223 }
224 cout << endl;
225 }
226
227
228
229 void fat_op(short number, int unitindex) {
230     int index = (*currentdirtable)->dirs[unitindex].startfcb;
231     int startfat = FCB[index]->blockindex;
232     if (number == 1) {
233         FAT[startfat] = -1;
234     }
235     else {
236         int index = startfat;
237         for (int i = 1; i < number; i++) {
238             int j = Find_freeFAT();
239             FAT[index] = j;
240             index = j;
241         }
242         FAT[index] = -1;
243     }
244 }
245
246
247 void write(char filename[], char content[]) {
248     int unitindex = Findunit(filename);
249     if (unitindex == -1) {
250         cout << "sorry,not found" << endl;
```

```

251         return;
252     }
253
254     int length = strlen(content);
255     int num = (length % 32 == 0) ? (length / 32) : (length / 32 + 1);
256     fat_op(num, unitindex);
257     FCB[(*currentdirtable)->dirs[unitindex].startfcb]->datalength =
        num;
258     int index1 = (*currentdirtable)->dirs[unitindex].startfcb;
259     int startblock = FCB[index1]->blockindex;
260     int block = length / 32 + 1;
261     int index = startblock;
262     for (int j = 0; j < block-1; j++) {
263         for (int k = 0; k < 32; k++)
264         {
265             Disk[k + index * 32] = content[k+j*32];
266         }
267         index = FAT[index];
268     }
269     int j = block - 1;
270     for (int k = 0; k < length % 32; k++) {
271         Disk[k + index * 32] = content[k + j * 32];
272     }
273     cout << endl;
274 }
275
276
277 /* 直接复制 */
278 void Swich(int i) {
279     if (cur->id == i) return;
280     if (UserInit[i] == 0) {
281         UserInit[i] = true;
282         User[i].id = i;
283         User[i].state = 0;
284         strcpy_s(User[i].path, Origin.c_str());

```

```
285         User[i].curtable = &rootdirtable;
286         User[i].path[8] = i + 48;
287     }
288
289     cur = &User[i];
290     path = User[i].path;
291     currentdirtable = &User[i].curtable;
292 }
293
294 bool P(int& mux) {
295     --mux;
296     //printf("%d\n", mux);
297     if (mux >= 0) {
298         return 1;
299     }
300     else {
301         printf("blocking!\n");
302         ++mux;
303         return 0;
304     }
305 }
306
307 void V(int& mux) {
308     ++mux;
309     if (mux >= 0) {
310         //return 1;
311     }
312     else {
313         printf("rising!\n");
314         //return 0;
315     }
316 }
317
318 void wrong() {
319     printf("OPERATION ERROR!\n");
```

```
320 }
321
322 void OSRead(char filename[], int length) { //OS read
323
324     int i = Findunit(filename);
325
326     if (i == -1) {
327         cout << "sorry,not found" << endl;
328         return;
329     }
330
331     i = (*currentdirttable)->dirs[i].startfcb;
332     i = FCB[i]->blockindex;
333
334     bool f;
335
336     f=P(Fmux[i].rc);
337     if (!f) {
338         wrong();
339         return;
340     }
341     ++Fmux[i].count;
342     if (Fmux[i].count == 1)f=P(Fmux[i].rm);
343     if (!f) {
344         --Fmux[i].count;
345         V(Fmux[i].rc);
346         wrong();
347         return;
348     }
349
350     V(Fmux[i].rc);
351     cur->state = 1;
352     printf("Reading %s!\n", filename);
353     read(filename, length);
354 }
```

```
355 }
356
357 void OSWrite(char filename[], char content[]) { //OS write
358
359     int i = Findunit(filename);
360
361     if (i == -1) {
362         cout << "sorry,not found" << endl;
363         return;
364     }
365
366
367     i = (*currentdirttable)->dirs[i].startfcb;
368     i = FCB[i]->blockindex;
369
370
371     bool f;
372
373     f=P(Fmux[i].rm);
374     if (!f) {
375         wrong();
376         return;
377     }
378     cur->state = 2;
379     printf("Writing %s!\n", filename);
380     write(filename, content);
381
382 }
383
384 void Close(char filename[]) { //OS close
385     int& x = cur->state;
386
387     if (x==0) {
388         wrong();
389         return;
```



```
390     }
391
392     int i = Findunit(filename);
393
394     if (i == -1) {
395         cout << "Sorry, not found." << endl;
396         return;
397     }
398
399
400     i = (*currentdirttable)->dirs[i].startfcb;
401     i = FCB[i]->blockindex;
402
403
404     bool f;
405
406     if (x == 1) {
407         f = P(Fmux[i].rc);
408         if (f)--Fmux[i].count;
409         else {
410             wrong();
411             return;
412         }
413         if (Fmux[i].count == 0)V(Fmux[i].rm);
414         V(Fmux[i].rc);
415     }
416     else {
417         V(Fmux[i].rm);
418     }
419
420     x = 0;
421     printf("%s closed!\n", filename);
422 }
423
424 int main() {
```

```
425     Init();
426     string s;
427     int d;
428     char operation[5],op;
429     char name[10] = { 0 };
430     char content[100] = { 0 };
431     int length;
432     while (1)
433     {
434
435         cout <<path<< '>';
436
437         op = getchar();
438         while(op!='\n'&& isalnum(op) == 0)op = getchar();
439         if(op=='\n')continue;
440         cin >> s;
441         memcpy(operation, s.c_str(), s.length());
442         if(cur->state!=0&&op!='s'&&op!='z')Close(name);
443         switch (op) {
444             case 'h':
445                 cout << "You can do the operation as follows." <<
                     endl;
446                 cout << "1.mkdir + dirname\n";
447                 cout << "2.touch + filename\n";
448                 cout << "3.ls\n";
449                 cout << "4.cd+dirname\n";
450                 cout << "5.Read+filename\n";
451                 cout << "6.write+filename\n";
452                 cout << "7.rm+filename\n";
453                 cout << "8.quit\n";
454
455                 //... ..
456
457
458                 getchar();
```

```
459         break;
460     case 'q':
461         return 0;
462     case 'm':
463         cin.get();
464         cin >> s;
465         memcpy(name, s.c_str(), 10);
466
467         //ls();
468
469         touch(name, 1, 0);
470         getchar();
471         break;
472     case 't':
473         cin >> s;
474         memcpy(name, s.c_str(), 10);
475         touch(name, 1, 1);
476         getchar();
477         break;
478     case 'l':
479         ls();
480         getchar();
481         break;
482     case 'c':
483         cin >> s;
484         memcpy(name, s.c_str(), 10);
485         //ls();
486         cd(name);
487         getchar();
488         break;
489     case 'w':
490         cin >> s;
491         memcpy(name, s.c_str(), 10);
492         cin >> s;
493         memcpy(content, s.c_str(), s.length());
```

```
494         write(name, content);
495         getchar();
496         break;
497     case 'R':
498         cin >> s;
499         memcpy(name, s.c_str(), 10);
500         scanf_s("%d", &length);
501         read(name, length);
502         getchar();
503         break;
504     case 'r':
505         cin >> s;
506         memcpy(name, s.c_str(), 10);
507         deletefile (name);
508         getchar();
509         break;
510     case 's':
511         cin >> d;
512         Swich(d);
513         getchar();
514         break;
515
516
517     /* 增加了命令 */
518     case 'x':
519         cin >> s;
520         memcpy(name, s.c_str(), 10);
521         scanf_s("%d", &length);
522         OSRead(name, length);
523         getchar();
524         break;
525     case 'y':
526         cin >> s;
527         memcpy(name, s.c_str(), 10);
528         cin >> s;
```

```
529         memcpy(content, s.c_str(), s.length());
530         OSWrite(name, content);
531         getchar();
532         break;
533     case 'z':
534         cin >> s;
535         memcpy(name, s.c_str(), 10);
536         Close(name);
537         getchar();
538         break;
539     default: cout << "ERROR INPUT\n";
540 }
541
542 }
543 }
```