

# 操作系统课程设计

## Project3: File System

周旭东 521021910829

2023.05.23

### 摘要

在 Project3 中，首先设计和实现了一个基本的类似磁盘的辅助存储服务器；接着实现了一个基本的 FAT 结构文件系统；在此基础上用磁盘当服务器，用 socket 套接字使得文件系统能够和磁盘服务器交互写作，形成一个非易失文件系统，并将其作为第二级服务器，用客户端连接访问。并在 linux 系统上进行了测试。

## 目录

<b>1 Step1: Design a basic disk-storage system</b>	<b>3</b>
1.1 实现要点 . . . . .	3
1.1.1 存储映射 . . . . .	3
1.1.2 指令处理 . . . . .	4
1.1.3 日志维护 . . . . .	6
1.2 功能测试 . . . . .	6
<b>2 Step2: Design a basic file system</b>	<b>7</b>
2.1 实现要点 . . . . .	7
2.1.1 FAT 系统基础构建 . . . . .	7
2.1.2 指令处理 . . . . .	9
2.2 命令解析 . . . . .	17
2.3 功能测试 . . . . .	19
<b>3 Step3: Work together</b>	<b>21</b>
3.1 实现要点 . . . . .	21
3.1.1 socket 结构构建 . . . . .	21
3.1.2 虚拟磁盘与物理磁盘交互 . . . . .	22

3.1.3	FileEntryTable 的存储 . . . . .	24
3.1.4	FileSystem 的初始化 . . . . .	25
3.2	功能测试 . . . . .	27
3.2.1	基础功能测试 . . . . .	27
3.2.2	恢复功能测试 . . . . .	29
	总结与反思	30

# 1 Step1: Design a basic disk-storage system

实现物理磁盘的模拟。模拟磁盘按柱面和扇区进行组织，并考虑轨道到轨道的时间。(使用 usleep (3C), nanosleep (3R) 等)。以微秒为单位的值(可能不是整数)，作为命令行参数传递给磁盘存储系统。此外，让柱面数和每个柱面的扇区数作为命令行参数。扇区大小固定为 256 字节。将此文件的文件名作为另一个命令行选项。

## 1.1 实现要点

### 1.1.1 存储映射

首先，使用 open 函数打开指定的文件 filename，使用 O\_RDWR 标志表示以可读可写的方式打开文件，O\_CREAT 标志表示如果文件不存在则创建它，S\_IRUSR | S\_IWUSR 表示设置文件权限为用户可读可写。

接下来，使用 lseek 函数将文件指针移动到文件末尾前一个字节的位置。这样做是为了“扩展”文件大小，确保映射的内存空间足够容纳整个文件。

然后，代码使用 write 函数向文件写入一个空字节，以实际改变文件大小。

```
1 // Open disk file and map it into memory
2 fd = open(filename, O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
3 if (fd == -1) {
4     printf("Error: Could not open file '%s'.\n", filename);
5     exit(-1);
6 }
7 if (lseek(fd, FILESIZE - 1, SEEK_SET) == -1) {
8     printf("Error calling lseek() to 'stretch' the file\n");
9     exit(-1);
10 }
11 if (write(fd, "", 1) == -1) {
12     printf("Error writing last byte of the file\n");
13     exit(-1);
14 }
```

使用 mmap 函数将文件映射到内存中。参数 NULL 表示由系统选择合适的映射地址，FILESIZE 指定映射的长度为文件大小，PROT\_READ | PROT\_WRITE 表示内存区域可读可写，MAP\_SHARED 表示对映射区的修改对其他映射同一文件的进程可见，fd 是打开的文件描述符，0 表示映射起始位置相对于文件开头的偏移量。

```
1 disk_data = mmap(NULL, FILESIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
2 if (disk_data == MAP_FAILED) {
3     close(fd);
```

```

4     printf("Error: Could not map file.\n");
5     exit(-1);
6 }
```

在处理完所有指令后，解除文件的内存映射并关闭相关的文件描述符。

```

1 // Unmap and close
2 if (disk_data != NULL && munmap(disk_data, FILESIZE) == -1) {
3     perror("munmap");
4     return -1;
5 }
6 if (fd != -1 && close(fd) == -1) {
7     perror("close");
8     return -1;
9 }
10 close(fd_log);
```

### 1.1.2 指令处理

依照要求，需要支持的指令有：

- I：信息请求。磁盘返回两个表示磁盘几何的整数：柱面数和每个柱面的扇区数。
- R c s：对柱面 c 扇区 s 的内容的读取请求。磁盘返回 Yes，后跟写空间和这 256 字节的信息，如果不存在此类块，则返回 No。
- W c s data：编写圆柱体 c 扇区 s 的请求。磁盘返回 Yes 并将数据写入柱面 c 扇区 s（如果是有效的写入请求）或返回 No。
- E：退出磁盘存储系统。

在一个循环中持续支持指令，直到读取到“E”，将 break 出循环体。用 scanf 对输入的字符串进行解析，并做扇区越界判断。符合要求时进行磁盘操作，所操作位置为

`disk_data + (c * SECTORS + s) * BLOCKSIZE`

并使用 usleep 函数根据两次读写位置的差值进行磁盘读写的延迟模拟。

`usleep(track_to_track_time * abs(c - last_c));`

```

1 while (1) {
2     scanf("%c", &cmd);
3     if (cmd == 'E') {
4         // Exit
5         printf("Goodbye!\n");
```

```

6         break;
7     }
8     else if (cmd == 'I') {
9         // Information request
10        printf("%d %d\n", CYLINDERS, SECTORS);
11    }
12    else if (cmd == 'R') {
13        // Read request
14        scanf("%d %d", &c, &s);
15        if (c < 0 || c >= CYLINDERS || s < 0 || s >= SECTORS) {
16            printf("No\n");
17        }
18        else {
19            printf("Yes ");
20            memcpy(data, disk_data + (c * SECTORS + s) * BLOCKSIZE, BLOCKSIZE);
21            fwrite(data, sizeof(char), strlen(data), stdout);
22            printf("\n");
23            usleep(track_to_track_time * abs(c - last_c));
24        }
25    }
26    else if (cmd == 'W') {
27        // Write request
28        scanf("%d %d", &c, &s);
29        if (c < 0 || c >= CYLINDERS || s < 0 || s >= SECTORS) {
30            printf("No\n");
31        }
32        else {
33            fgets(data, BLOCKSIZE, stdin);
34            memcpy(disk_data + (c * SECTORS + s) * BLOCKSIZE, data, strlen(data)-1);
35            printf("Yes\n");
36            usleep(track_to_track_time * abs(c - last_c));
37        }
38    }
39    else if(cmd == '\n')continue;
40    else {
41        printf("Instructions Error!\n");
42    }
43    last_c = c;
44}

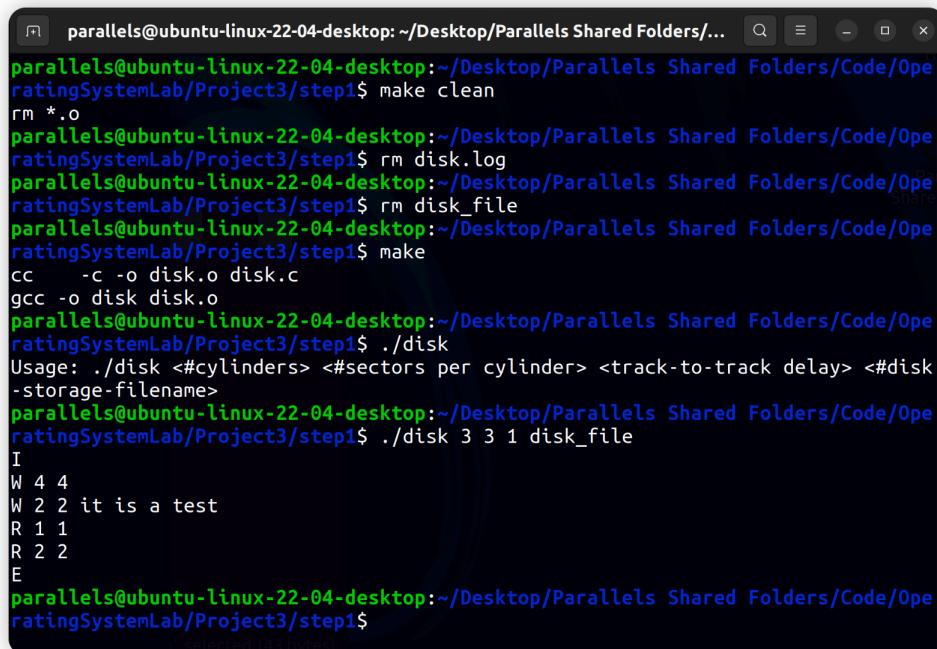
```

### 1.1.3 日志维护

由于前期调试的需要，将所有信息使用 printf 直接输出到标准输出流，为转而进行日志记录，使用 dup2 重定向输出流到日志中。在此之前需要设置日志文件：使用 open 函数打开名为“disk.log”的文件。使用参数 O\_WRONLY 表示以只写方式打开文件，O\_CREAT 表示如果文件不存在则创建它，S\_IRUSR | S\_IWUSR 表示设置文件权限为用户可读可写。

```
1 int fd_log = open("disk.log", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
2 if (fd_log == -1) {
3     perror("Error: Could not open file 'disk.log'.\n");
4     exit(-1);
5 }
6 // redirect using dup2
7 if (dup2(fd_log, STDOUT_FILENO) == -1) {
8     perror("Error: Could not dup.");
9     exit(-1);
10 }
```

## 1.2 功能测试

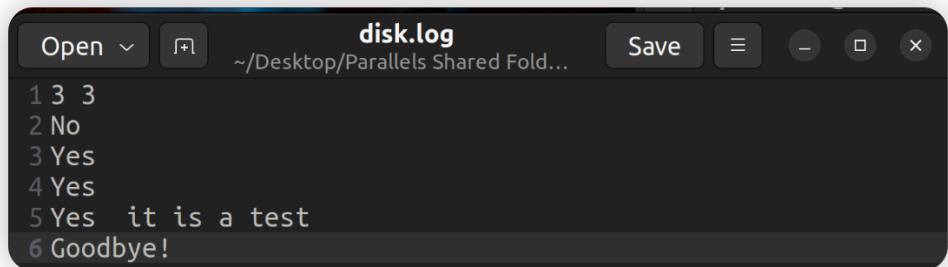


```
parallels@ubuntu-linux-22-04-desktop:~/Desktop/Parallels Shared Folders/Code/OperatingSystemLab/Project3/step1$ make clean
rm *.o
parallels@ubuntu-linux-22-04-desktop:~/Desktop/Parallels Shared Folders/Code/OperatingSystemLab/Project3/step1$ rm disk.log
parallels@ubuntu-linux-22-04-desktop:~/Desktop/Parallels Shared Folders/Code/OperatingSystemLab/Project3/step1$ rm disk_file
parallels@ubuntu-linux-22-04-desktop:~/Desktop/Parallels Shared Folders/Code/OperatingSystemLab/Project3/step1$ make
cc -c -o disk.o disk.c
gcc -o disk disk.o
parallels@ubuntu-linux-22-04-desktop:~/Desktop/Parallels Shared Folders/Code/OperatingSystemLab/Project3/step1$ ./disk
Usage: ./disk <#cylinders> <#sectors per cylinder> <track-to-track delay> <#disk -storage-filename>
parallels@ubuntu-linux-22-04-desktop:~/Desktop/Parallels Shared Folders/Code/OperatingSystemLab/Project3/step1$ ./disk 3 3 1 disk_file
I
W 4 4
W 2 2 it is a test
R 1 1
R 2 2
E
parallels@ubuntu-linux-22-04-desktop:~/Desktop/Parallels Shared Folders/Code/OperatingSystemLab/Project3/step1$
```

图 1: Step1 测试

如图 1 进行测试，顺序为：用 I 读取磁盘信息，写越界位置，写正常位置内容，读空白位置，读刚写入的位置，退出。

日志内容如图 2 所示，结果符合要求。



The screenshot shows a terminal window titled "disk.log". The window has standard OS X-style controls at the top: "Open", "Save", and window control buttons. The file path is listed as "~/Desktop/Parallels Shared Fold...". The terminal output contains the following text:

```
1 3 3
2 No
3 Yes
4 Yes
5 Yes it is a test
6 Goodbye!
```

图 2: Step1 测试

## 2 Step2: Design a basic file system

### 2.1 实现要点

#### 2.1.1 FAT 系统基础构建

文件系统构建思路为将目录和文件都看作 file 存储，以 type 进行辨别，好处是在后期与物理磁盘交互时打包数据块更为方便简洁。

目录项包含了文件的相关信息。其结构如图 3 所示。

DirectoryEntry	
1	filename
2	type
3	start_block
4	file_size
5	last_modified

图 3: 目录项结构

filename: 文件名。数组的大小为文件名的最大长度。type: 表示文件类型。取值为“F”，“D”，表示普通文件或目录。start\_block: 一个整数，表示文件起始块的位置或索引。它指示文件在存储介质（例如磁盘）上的起始位置。file\_size: 文件的大小。last\_modified: 文件的最后修改时间。

`root_directory[MAX_FILE_ENTRIES]` 是目录项的实例化。

`char disk[NUM_BLOCKS][BLOCK_SIZE];` 为构建的虚拟磁盘，存储每个文件的相应内容，大小为 `[BLOCK_SIZE]`

```
1 // DirectoryEntry struct
2 typedef struct {
3     char filename[MAX_FILENAME_LENGTH];
4     char type;
5     int start_block;
6     int file_size;
7     char* last_modified;
8 } DirectoryEntry;
9
10 DirectoryEntry root_directory[MAX_FILE_ENTRIES];
11 char disk[NUM_BLOCKS][BLOCK_SIZE];
```

由于每个文件大小不做要求，假定都为 256bytes 以内，将 StartBlock 作为文件分配的唯一数据块，给出此系统的结构示意图 4

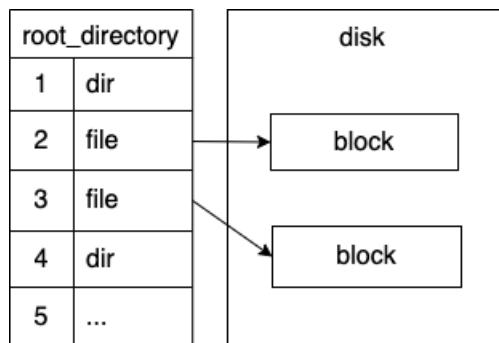


图 4: 目录和文件块分配

另外，针对目录项和虚拟磁盘，给出两个函数用于在给定的数据结构中查找空闲的目录项和空闲的数据块。以方便文件的创建和删除等操作。

```
1 int find_free_entry() {
2     for (int i = 0; i < MAX_FILE_ENTRIES; i++) {
3         if (root_directory[i].filename[0] == '\0')
4             return i;
5     }
6     return -1;
7 }
```

```

9 int find_free_block() {
10    for (int i = 0; i < NUM_BLOCKS; i++) {
11        if (disk[i][0] == '\0')
12            return i;
13    }
14    return -1;
15}

```

### 2.1.2 指令处理

根据指令要求给出实现函数。

**f:** 格式化。这将通过初始化文件系统所依赖的任何/所有表来格式化磁盘上的文件系统。

将所有的数据块和目录项 table 用空字符串替换，即格式化整个文件系统。

```

1 void clear_all() {
2     for(int i = 0; i < MAX_FILE_ENTRIES; i++){
3         strcpy(root_directory[i].filename, "");
4         if(root_directory[i].type == 'F')
5             strcpy(disk[root_directory[i].start_block], "\0");
6         root_directory[i].file_size = 0;
7     }
8     printf("Done\n");
9 }

```

**mk f :** 创建文件。这将在文件系统中创建一个名为 f 的文件。

**mkdir d:** 创建目录。这将在当前目录中创建一个名为 d 的子目录。

create\_file 函数负责创建文件，并将文件名、类型、起始块位置、文件大小和最后修改时间等信息存储到文件系统的目录中。寻找到空白目录项之后，对所创建的文件类型分情况，若为目录，则不分配数据块，若为文件则分配一个 block 大小的数据块并用 tab 占位防止后续查找空余块时发生重合。

在创建文件时用 time 函数设置时间并用 ctime 转为字符串后保存到 last\_modified 中。需要注意的是，这里的时间末尾带有一个换行符，使用字符串操作手动去除。

此外需定义全局变量 char current\_directory[MAX\_PATH\_LENGTH] = "/";，每次启动文件系统初始化当前目录为 /。使用该变量作为查找文件和设置文件名称。文件命名时会带上当前目录，方便查找和后续目录切换功能。

例如，位于 /test 目录下的文件 /main.c 的名称为 /test/main.c

```

1 void create_file(const char* name, char type) {
2     char* filename= concat_strings(current_directory, name);
3     if(find_file(name) != -1){

```

```

4         free(filename);
5
6     }
7
8     int entry_index = find_free_entry();
9
10    if (entry_index == -1) {
11        printf("No: full\n");
12        free(filename);
13        return;
14    }
15
16    strcpy(root_directory[entry_index].filename, filename);
17    root_directory[entry_index].type = type;
18
19    if(type == 'D'){
20        root_directory[entry_index].start_block = 0;
21        root_directory[entry_index].file_size = 0;
22    }
23
24    else{
25        root_directory[entry_index].start_block = find_free_block();
26        strcpy(disk[root_directory[entry_index].start_block], "0");
27        root_directory[entry_index].file_size = 1;
28    }
29
30
31    printf("Yes\n");
32
33    // set last modified time
34    time_t current_time;
35
36    time(&current_time);
37
38    root_directory[entry_index].last_modified = ctime(&current_time);
39    root_directory[entry_index].last_modified[strlen(root_directory[entry_index].
40        last_modified) - 1] = '\0';
41
42
43    free(filename);
44}

```

**cd path:** 更改目录。这会将当前工作目录更改为路径，初始工作路径为 /。需要处理 cd . 和 cd .. 。

切换目录在该文件系统下为字符串操作，cd.. 进入上一目录时用 strrchr 读取上一个 / 以内的内容作为新目录入口。当输入指定路径时在末尾加上 / 以符合目录规范。进入目录成功后输出 Yes 并告知当前所在目录。

```

1 void change_dir(char* path) {
2     if (strcmp(path, "..") == 0) {

```

```

3      // Switch to parent directory
4      strcpy(current_directory, strrchr(current_directory, '/') );
5  }
6  else if (strcmp(path, "/") == 0) {
7      strcpy(current_directory, "/");
8  }
9  else if (strcmp(path, ".") == 0) {
10     strcpy(current_directory, current_directory);
11 }
12 else {
13     if (root_directory[find_file(path)].type == 'D') {
14         // Switch to the specified subdirectory
15         strcat(current_directory, path);
16         strcat(current_directory, "/");
17     }
18     else {
19         printf("Not a Directory\n");
20         return;
21     }
22 }
23 printf("Yes: %s\n", current_directory);
24 }
```

**rm f** : 删除文件。这将从当前目录中删除名为 f 的文件。

**rmdir d**: 删除目录。这将删除当前目录中名为 d 的目录。

由于设计的 `find_free_entry` 用文件名决定是否占用某个目录项空间，只需将文件名置为空字符串即可。F 类文件可能在 disk 中分配有 block 空间，也需将占位字符串收回并置其大小为 0。

```

1 void delete_file(const char* name) {
2     char* filename= concat_strings(current_directory, name);
3     for (int i = 0; i < MAX_FILE_ENTRIES; i++) {
4         if(strcmp(root_directory[i].filename, filename) == 0) {
5             root_directory[i].filename[0] = '\0';
6             root_directory[i].file_size = 0;
7             printf("Yes\n");
8             free(filename);
9             strcpy(disk[root_directory[i].start_block], "\0");
10            return;
11        }
12    }
```

```
13     printf("No\n");
14     free(filename);
15 }
```

ls：目录列表。这将返回当前目录中的文件和目录的列表。还需要输出一些其他信息，例如文件大小、上次更新时间等。

要求：首先，按词典顺序输出所有文件，用空格分隔。然后按词典顺序输出所有目录，用空格分隔。在文件和目录之间输出一个“”。

根据要求进行字符串操作，先将所有在当前目录下的所有文件找出存入相应数组中；其后，使用选择排序算法对文件名进行字典顺序的排序；最后，用另外的数组进行字符串拼接操作并按照要求进行输出。

```
1 void list_file() {
2     int flag = 1;
3     char tmp[256] = "";
4     char file_tmp[1024] = "";
5     char dir_tmp[256] = "";
6
7     // Sort file and directory names in alphabetical order
8     char sorted_files[MAX_FILE_ENTRIES][MAX_FILENAME_LENGTH];
9     char sorted_dirs[MAX_FILE_ENTRIES][MAX_FILENAME_LENGTH];
10    int num_files = 0;
11    int num_dirs = 0;
12
13    // Iterate over root_directory to populate sorted_files and sorted_dirs arrays
14    for (int i = 0; i < MAX_FILE_ENTRIES; i++) {
15        if (strcmp(root_directory[i].filename, current_directory, strlen(
16            current_directory)) == 0) {
17            flag = 0;
18            if (root_directory[i].type == 'F') {
19                strcpy(sorted_files[num_files], root_directory[i].filename);
20                num_files++;
21            } else {
22                strcpy(sorted_dirs[num_dirs], root_directory[i].filename);
23                num_dirs++;
24            }
25        }
26    }
27
28    // Sort file names in alphabetical order
29    for (int i = 0; i < num_files - 1; i++) {
```

```

29     for (int j = i + 1; j < num_files; j++) {
30         if (strcmp(sorted_files[i], sorted_files[j]) > 0) {
31             char temp[MAX_FILENAME_LENGTH];
32             strcpy(temp, sorted_files[i]);
33             strcpy(sorted_files[i], sorted_files[j]);
34             strcpy(sorted_files[j], temp);
35         }
36     }
37 }
38
39 // Sort directory names in alphabetical order
40 for (int i = 0; i < num_dirs - 1; i++) {
41     for (int j = i + 1; j < num_dirs; j++) {
42         if (strcmp(sorted_dirs[i], sorted_dirs[j]) > 0) {
43             char temp[MAX_FILENAME_LENGTH];
44             strcpy(temp, sorted_dirs[i]);
45             strcpy(sorted_dirs[i], sorted_dirs[j]);
46             strcpy(sorted_dirs[j], temp);
47         }
48     }
49 }
50
51 // Create file_tmp string with sorted file names
52 for (int i = 0; i < num_files; i++) {
53     sprintf(tmp, "%s, Size:%d, LastModifiedTime:%s", sorted_files[i], root_directory
54 [i].file_size, root_directory[i].last_modified);
55     strcat(file_tmp, tmp);
56     if (i < num_files - 1) {
57         strcat(file_tmp, " ");
58     }
59 }
60
61 // Create dir_tmp string with sorted directory names
62 for (int i = 0; i < num_dirs; i++) {
63     sprintf(tmp, "%s", sorted_dirs[i]);
64     strcat(dir_tmp, tmp);
65     if (i < num_dirs - 1) {
66         strcat(dir_tmp, " ");
67     }
68 }
69
70 // Print the final output

```

```

70     if (flag) {
71         printf("\n");
72     } else {
73         if (dir_tmp[0] != '\0') {
74             strcat(file_tmp, " & ");
75         }
76         strcat(file_tmp, dir_tmp);
77         printf("%s\n", file_tmp);
78     }
79 }
```

**cat f** : 捕获文件。这将读取名为 f 的文件，并返回来自该文件的数据。

读取文件需要进行查找，并判断其类型是否符合要求；此后根据 start\_block 信息进行 disk 内容的读取并输出文件大小的数据。

```

1 void read_file(const char* name) {
2     char* filename= concat_strings(current_directory, name);
3     for (int i = 0; i < MAX_FILE_ENTRIES; i++) {
4         if (strcmp(root_directory[i].filename, filename) == 0 && root_directory[i].type
5 == 'F') {
6             int start_block = root_directory[i].start_block;
7             int file_size = root_directory[i].file_size;
8
9             printf("Yes: ");
10            for (int block = start_block; block < start_block + (file_size / BLOCK_SIZE)
11 + 1; block++) {
12                printf("%.8s", file_size - (block - start_block) * BLOCK_SIZE, disk[
13 block]);
14            }
15            printf("\n");
16
17            free(filename);
18            return;
19        }
20    }
21    free(filename);
22    printf("No\n");
23 }
```

**w f1 data**: 写入文件。这将用 1 个字节的数据覆盖名为 f 的文件的内容。如果新数据比文件中以前的数据长，则文件将变得更长。如果新数据短于文件中以前的数据，则文件将被截

断为新长度。

实现为读取文件的逆操作，同时还需进行文件大小与时间信息的更新。

```
1 void write_file(const char* name, int len, const char* data) {
2     char* filename= concat_strings(current_directory, name);
3     for (int i = 0; i < MAX_FILE_ENTRIES; i++) {
4         if (strcmp(root_directory[i].filename, filename) == 0 && root_directory[i].type
5             == 'F') {
6             int start_block = root_directory[i].start_block;
7
8             strncpy(disk[start_block], data, len);
9             root_directory[i].file_size = len;
10            printf("Yes\n");
11
12            time_t current_time;
13            time(&current_time);
14            root_directory[i].last_modified = ctime(&current_time);
15            root_directory[i].last_modified[strlen(root_directory[i].last_modified) - 1]
16            = '\0';
17
18            free(filename);
19            return;
20        }
21    }
22}
```

**i f pos l data:** 插入到文件。这将插入到文件中的 posth 字符 (0 索引) 之前的位置，其中包含 1 个字节的数据。如果 pos 大于文件大小。只需将其插入文件末尾即可。

在进行插入操作时用到 memmove 函数，其功能是在内存中移动数据块。在这里，它被用于将 start\_block 位置开始的文件数据向后移动，为新的数据腾出空间。之后用 strncpy 将需要插入内容拷贝到对应位置 disk 中。同时，也需要进行文件大小，时间等信息更新。

```
1 void insert_file(const char* name, int position, int length, const char* data) {
2     char* filename= concat_strings(current_directory, name);
3     for (int i = 0; i < MAX_FILE_ENTRIES; i++) {
4         if (strcmp(root_directory[i].filename, filename) == 0 && root_directory[i].type
5             == 'F') {
6             int start_block = root_directory[i].start_block;
7             if (position > root_directory[i].file_size)
8                 position = root_directory[i].file_size;
```

```

8
9     if (root_directory[i].file_size + length > MAX_DATA_LENGTH) {
10        printf("Error: Inserted data exceeds maximum file size.\n");
11        return;
12    }
13
14    memmove(disk[start_block] + position + length, disk[start_block] + position,
15    root_directory[i].file_size - position);
16    strncpy(disk[start_block] + position, data, length);
17    root_directory[i].file_size += length;
18    printf("Yes\n");
19
20    time_t current_time;
21    time(&current_time);
22    root_directory[i].last_modified = ctime(&current_time);
23    root_directory[i].last_modified[strlen(root_directory[i].last_modified) - 1]
24    = '\0';
25
26    free(filename);
27    return;
28}
29
30}

```

**d f pos l:** 在文件中删除。这将从 pos 字符（0 索引）中删除内容，删除 l 字节或直到文件末尾。

使用 memmove 函数将 start\_block 位置开始的文件数据块中的一段数据向前移动，覆盖被删除的数据块。并更新文件大小，时间等信息。

```

1 void delete_content(const char* name, int position, int length) {
2     char* filename= concat_strings(current_directory, name);
3     for (int i = 0; i < MAX_FILE_ENTRIES; i++) {
4         if (strcmp(root_directory[i].filename, filename) == 0 && root_directory[i].type
5 == 'F') {
6             int start_block = root_directory[i].start_block;
7             if (position >= root_directory[i].file_size) {
8                 printf("Error: Invalid position.\n");
9                 return;
10            }
11        }
12    }
13
14    free(filename);
15    return;
16}
17
18}
19
20}

```

```

11     if (position + length > root_directory[i].file_size)
12         length = root_directory[i].file_size - position;
13
14     memmove(disk[start_block] + position, disk[start_block] + position + length,
15             root_directory[i].file_size - position - length);
16     root_directory[i].file_size -= length;
17     printf("Yes\n");
18
19     time_t current_time;
20     time(&current_time);
21     root_directory[i].last_modified = ctime(&current_time);
22     root_directory[i].last_modified[strlen(root_directory[i].last_modified) - 1]
23 = '\0';
24
25     free(filename);
26     return;
27 }
28 free(filename);
29 printf("No\n");
}

```

## 2.2 命令解析

使用 sscanf 进行各个参数的读入并对各个指令调用相应函数进行处理。

```

1 while (running) {
2     printf("My File System > ");
3     fgets(input, MAX_INPUT_SIZE, stdin);
4
5     if (sscanf(input, "%s %s %s %s %s", command, arg1, arg2, arg3, arg4) < 1) {
6         continue;
7     }
8     if (strcmp(command, "e") == 0) {
9         printf("Goodbye!\n");
10        running = 0;
11        return 0;
12    }
13    if (strcmp(command, "f") == 0) {
14        clear_all();
15    }

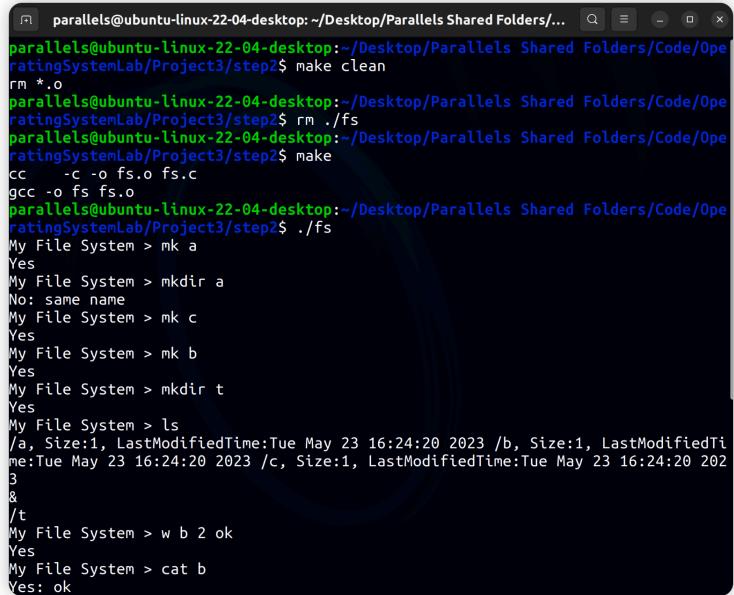
```

```

16     else if (strcmp(command, "mk") == 0) {
17         create_file(arg1, 'F');
18     }
19     else if (strcmp(command, "mkdir") == 0) {
20         create_file(arg1, 'D');
21     }
22     else if (strcmp(command, "rm") == 0) {
23         delete_file(arg1);
24     }
25     else if (strcmp(command, "rmdir") == 0) {
26         delete_file(arg1);
27     }
28     else if (strcmp(command, "cd") == 0) {
29         change_dir(arg1);
30     }
31     else if (strcmp(command, "ls") == 0) {
32         list_file();
33     }
34     else if (strcmp(command, "cat") == 0) {
35         read_file(arg1);
36     }
37     else if (strcmp(command, "w") == 0) {
38         int len = atoi(arg2);
39         sscanf(input, "%*s %*s %*s %[^\\n]", arg3);
40         write_file(arg1, len, arg3);
41     }
42     else if (strcmp(command, "i") == 0) {
43         int pos = atoi(arg2);
44         int len = atoi(arg3);
45         sscanf(input, "%*s %*s %*s %*s %[^\\n]", arg4);
46         insert_file(arg1, pos, len, arg4);
47     }
48     else if (strcmp(command, "d") == 0) {
49         int pos = atoi(arg2);
50         int len = atoi(arg3);
51         delete_content(arg1, pos, len);
52     }
53     else {
54         printf("Invalid command.\n");
55     }
56 }
```

## 2.3 功能测试

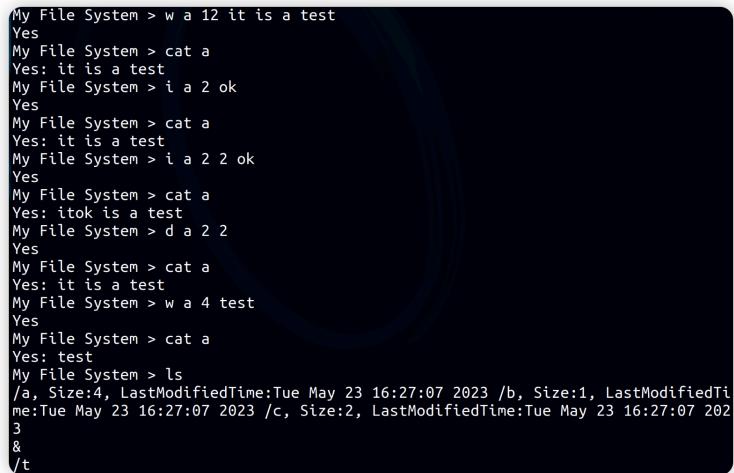
对指令 mk, mkdir, ls, w f l data, cat f 进行了相应测试，输出符合要求。



```
parallels@ubuntu-linux-22-04-desktop:~/Desktop/Parallels Shared Folders/...$ make clean
rm *.o
parallels@ubuntu-linux-22-04-desktop:~/Desktop/Parallels Shared Folders/Code/OpenRatingSystemLab/Project3/step2$ rm ./fs
parallels@ubuntu-linux-22-04-desktop:~/Desktop/Parallels Shared Folders/Code/OpenRatingSystemLab/Project3/step2$ make
cc -c -o fs.o fs.c
gcc -o fs fs.o
parallels@ubuntu-linux-22-04-desktop:~/Desktop/Parallels Shared Folders/Code/OpenRatingSystemLab/Project3/step2$ ./fs
My File System > mk a
Yes
My File System > mkdir a
No: same name
My File System > mk c
Yes
My File System > mk b
Yes
My File System > mkdir t
Yes
My File System > ls
/a, Size:1, LastModifiedTime:Tue May 23 16:24:20 2023 /b, Size:1, LastModifiedTime:Tue May 23 16:24:20 2023 /c, Size:1, LastModifiedTime:Tue May 23 16:24:20 2023
&
/t
My File System > w b 2 ok
Yes
My File System > cat b
Yes: ok
```

图 5: Step2 测试 1

对指令 i f pos l data, d f pos l 进行了相应测试，输出符合要求。



```
My File System > w a 12 it is a test
Yes
My File System > cat a
Yes: it is a test
My File System > i a 2 ok
Yes
My File System > cat a
Yes: it is a test
My File System > i a 2 2 ok
Yes
My File System > cat a
Yes: itok is a test
My File System > d a 2 2
Yes
My File System > cat a
Yes: itok is a test
My File System > w a 4 test
Yes
My File System > cat a
Yes: test
My File System > ls
/a, Size:4, LastModifiedTime:Tue May 23 16:27:07 2023 /b, Size:1, LastModifiedTime:Tue May 23 16:27:07 2023 /c, Size:2, LastModifiedTime:Tue May 23 16:27:07 2023
&
/t
```

图 6: Step2 测试 2

对指令 f ,e 进行了相应测试，输出符合要求。

```
My File System > ls
/a, Size:4, LastModifiedTime:Tue May 23 16:27:20 2023 /b, Size:1, LastModifiedTi
me:Tue May 23 16:27:20 2023 /c, Size:2, LastModifiedTime:Tue May 23 16:27:20 202
3 /t/a, Size:1, LastModifiedTime:Tue May 23 16:27:20 2023
&
/t
My File System > f
Done
My File System > ls
My File System > e
Goodbye!
```

图 7: Step2 测试 3

对指令 rm ,rmdir 进行了相应测试，输出符合要求。

```
parallels@ubuntu-linux-22-04-desktop:~/Desktop/Parallels Shared Folders/Code/Ope
ratingSystemLab/Project3/step2$ ./fs
My File System > mk c
Yes
My File System > mk a
Yes
My File System > mkdir t
Yes
My File System > ls
/a, Size:1, LastModifiedTime:Tue May 23 16:31:46 2023 /c, Size:1, LastModifiedTi
me:Tue May 23 16:31:46 2023
&
/t
My File System > rmdir t
Yes
My File System > ls
/a, Size:1, LastModifiedTime:Tue May 23 16:31:46 2023 /c, Size:1, LastModifiedTi
me:Tue May 23 16:31:46 2023
My File System > rm a
Yes
My File System > ls
/c, Size:1, LastModifiedTime:Tue May 23 16:31:46 2023
My File System > e
Goodbye!
```

图 8: Step2 测试 4

对指令 cd path ,cd., cd.. 进行了相应测试，输出符合要求。

```
parallels@ubuntu-linux-22-04-desktop:~/Desktop/Parallels Shared Folders/Code/Ope
ratingSystemLab/Project3/step2$ ./fs
My File System > mkdir f
Yes
My File System > mkdir t
Yes
My File System > cd f
Yes: /f/
My File System > cd .
Yes: /f/
My File System > cd ..
Yes: /
My File System > ls
&
/f /t
My File System > e
Goodbye!
```

图 9: Step2 测试 5

### 3 Step3: Work together

#### 3.1 实现要点

##### 3.1.1 socket 结构构建

与 Project1 类似，进行 Client 构建，修改其中读写顺序为“写-读-写-读...”使之与 fs 相匹配。

disk 作为服务器端，构建亦与 Project1 类似，这里不再赘述。修改其中读写顺序为“读-写-读-写...”。

在 fs 文件中需要进行客户端和服务器的构建，使用两套 socket 构建。对客户端读写顺序为“读-写-读-写...”，对 disk 端进行读写顺序为“写-读-写-读...”。

```
1 // client part
2 disk_portno = atoi(argv[2]);
3 disk_sockfd = socket(AF_INET, SOCK_STREAM, 0);
4 if (disk_sockfd < 0)
5     error("ERROR opening socket");
6 disk_server = gethostbyname(argv[1]);
7 if (disk_server == NULL) {
8     fprintf(stderr,"ERROR, no such host\n");
9     exit(0);
10 }
11 bzero((char *) &disk_serv_addr, sizeof(disk_serv_addr));
12 disk_serv_addr.sin_family = AF_INET;
13 bcopy((char *)disk_server->h_addr,
14     (char *)&disk_serv_addr.sin_addr.s_addr,
15     disk_server->h_length);
16 disk_serv_addr.sin_port = htons(disk_portno);
17 if (connect(disk_sockfd,(struct sockaddr *) &disk_serv_addr,sizeof(disk_serv_addr))
18 < 0)
19     error("ERROR connecting");
20
21 // server part
22 sockfd = socket(AF_INET, SOCK_STREAM, 0);
23 if (sockfd < 0)
24     error("Error opening socket");
25 printf("Accepting connections ... \n");
26 bzero((char *) &serv_addr, sizeof(serv_addr));
27 portno = atoi(argv[3]);
28 serv_addr.sin_family = AF_INET;
29 serv_addr.sin_addr.s_addr = INADDR_ANY;
```

```

29     serv_addr.sin_port = htons(portno);
30
31     if (bind(sockfd, (struct sockaddr *) &serv_addr,sizeof(serv_addr)) < 0)
32         error("Error on binding");
33     listen(sockfd,5);

```

用 dup2 将输出内容重定向到相应的客户端，并用 fprintf 维护日志的内容。

```

1 // redirect to client
2 dup2(newsockfd, STDOUT_FILENO);
3 ...
4 // print to log
5 fprintf(fd_log, "Receive from PORT(%d): %s", cli_addr.sin_port, buffer);

```

### 3.1.2 虚拟磁盘与物理磁盘交互

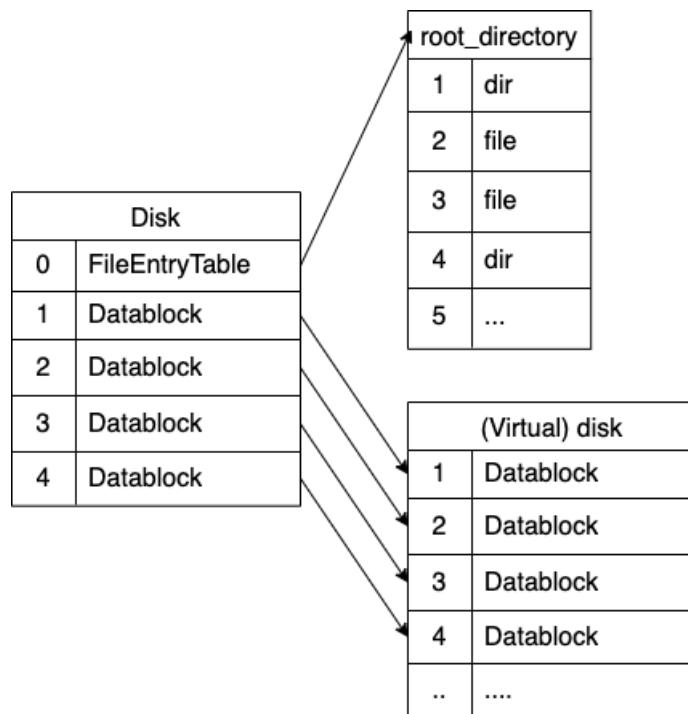


图 10: 物理磁盘对应存储内容

图 10 展示了虚拟 disk 与构建的服务端 disk 之间映射对应关系，其中第一个 block 存储 FileTable。受限于 block 大小，该文件系统一共支持至多 6 个文件（含目录）。

在创建文件，读文件，插入文件，删除内容，删除文件这些部分需要对虚拟 disk 进行读写，增加服务端 disk 后，用 R c s data 指令对相应数据块同步进行更新。

```
1 ...
2         sprintf(disk_buffer, "W %d %d %s\n", root_directory[i].start_block/CYLINDERS
3 , root_directory[i].start_block%CYLINDERS, disk[start_block]);
4         disk_n = write(disk_sockfd, disk_buffer, strlen(disk_buffer));
5         if (disk_n < 0) error("Error reading from socket");
6         bzero(disk_buffer, 1024);
7         disk_n = read(disk_sockfd, disk_buffer, 1023);
8         if (disk_n < 0) error("Error reading from socket");
9         printf("%s", disk_buffer);
10        bzero(disk_buffer, 1024);
11 ...
12
```

修改 disk.c 文件，增加对磁盘某一块的删除 D c s 操作和整体清空操作 C 以方便格式化文件系统操作和删除文件操作。并将写磁盘操作更改为在写之前重置磁盘内容。

```
1 ...
2         else if (cmd == 'W') {
3             // Write request
4             sscanf(buf, "%d %d %[^\n]", &c, &s, instr);
5             fprintf(fd_log, "command: W %d %d %s\n", c, s, instr);
6             if (c < 0 || c >= CYLINDERS || s < 0 || s >= SECTORS) {
7                 printf("No\n");
8             }
9             else {
10                 strcpy(data, instr);
11                 memcpy(disk_data + (c * SECTORS + s) * BLOCKSIZE, data, strlen(data));
12                 printf("Yes\n");
13                 fprintf(fd_log, "Yes\n");
14                 usleep(track_to_track_time * abs(c - last_c));
15             }
16         }
17         else if (cmd == 'D') {
18             // Delete block
19             sscanf(buf, "%d %d", &c, &s);
20             fprintf(fd_log, "command: D %d %d\n", c, s);
21             if (c < 0 || c >= CYLINDERS || s < 0 || s >= SECTORS) {
22                 printf("No\n");
23             }
24             else {
25                 strcpy(data, "\0");
26             }
27         }
28     }
29 }
```

```

25         memcpy(disk_data + (c * SECTORS + s) * BLOCKSIZE, data, 1);
26         printf("Yes\n");
27         fprintf(fd_log, "Yes\n");
28         usleep(track_to_track_time * abs(c - last_c));
29     }
30 }
31 else if (cmd == 'C') {
32     fprintf(fd_log, "command: C\n");
33     // Clear all
34     for(int i = 0; i < CYLINDERS; i ++){
35         for(int j = 0; j < SECTORS; j ++){
36             strcpy(data, "");
37             memcpy(disk_data + (i * SECTORS + j) * BLOCKSIZE, data, 1);
38         }
39     }
40     printf("Yes\n");
41     fprintf(fd_log, "Yes\n");
42 }
```

### 3.1.3 FileEntryTable 的存储

DirectoryEntry 为一个结构体，在存入服务端 disk 的 block 块中需要先打包为字符串

```

1 // pack DirectoryEntry in to a string
2 char* pack_dir(DirectoryEntry dir_entry) {
3     char* packed_string = (char*)malloc(BLOCK_SIZE * sizeof(char));
4
5     sprintf(packed_string, " %s %c %d %d %s|",
6             dir_entry.filename, dir_entry.type, dir_entry.start_block, dir_entry.
7             file_size, dir_entry.last_modified);
8     return packed_string;
}
```

在此基础上，将整个 root\_directory 转化为字符串，并在开头添加 file\_count 以确定文件数量。

```

1 void store_table() {
2     char entry_table[BLOCK_SIZE] = "";
3     if (file_count != 0){
4         sprintf(entry_table, "%d", file_count);
5         for(int i = 0; i < MAX_FILE_ENTRIES; i++){
6             if(strcmp(root_directory[i].filename, "\0") == 0) continue;
```

```

7         strncat(entry_table, pack_dir(root_directory[i]), strlen(pack_dir(
root_directory[i])));
8     }
9     sprintf(disk_buffer, "W %d %d %s\n", 0, 0, entry_table);
10    disk_n = write(disk_sockfd, disk_buffer, strlen(disk_buffer));
11    if (disk_n < 0) error("Error reading from socket");
12 }
13 else{
14     sprintf(disk_buffer, "C\n");
15     disk_n = write(disk_sockfd, disk_buffer, strlen(disk_buffer));
16     if (disk_n < 0) error("Error reading from socket");
17 }
18 }
```

此函数在退出时调用，将最新的 FileEntryTable 存入物理磁盘中。

### 3.1.4 FileSystem 的初始化

在文件系统开始时需要获取磁盘信息以得知存储大小信息；利用上一次存入的 FileEntryTable 初始化目录表，并将数据块读入虚拟磁盘从而形成非易失的文件系统。

```

1 // initial CYLINDERS and SECTORS
2 strcpy(disk_buffer, "I\n");
3 disk_n = write(disk_sockfd, disk_buffer, strlen(disk_buffer));
4 if (disk_n < 0) error("Error writing to socket");
5 disk_n = read(disk_sockfd, disk_buffer, 1024);
6 if (disk_n < 0) error("Error reading from socket");
7 sscanf(disk_buffer, "%d %d", &CYLINDERS, &SECTORS);
8 bzero(disk_buffer, 1024);

9
10 // initial file allocation table
11 sprintf(disk_buffer, "R %d %d\n", 0, 0);
12 disk_n = write(disk_sockfd, disk_buffer, strlen(disk_buffer));
13 if (disk_n < 0) error("Error writing to socket");
14 disk_n = read(disk_sockfd, disk_buffer, 1024);
15 if (disk_n < 0) error("Error reading from socket");
16 unpack_dir(getString(disk_buffer), root_directory);
17 bzero(disk_buffer, 1024);

18
19 // initial virtual disk
20 for(int i = 1; i < CYLINDERS * SECTORS; i++){
21     sprintf(disk_buffer, "R %d %d\n", i/CYLINDERS, i%CYLINDERS);
```

```

22     disk_n = write(disk_sockfd, disk_buffer, strlen(disk_buffer));
23     if (disk_n < 0) error("Error reading from socket");
24     bzero(disk_buffer, 1024);
25     disk_n = read(disk_sockfd, disk_buffer, 1023);
26     if (disk_n < 0) error("Error reading from socket");
27     if(disk_buffer[0] == '\n') disk_buffer[0] = '\0';
28     strncpy(disk[i], disk_buffer, BLOCK_SIZE);
29     bzero(disk_buffer, 1024);
30 }
```

在读取目录表时用到解析函数 `unpack_dir` 函数，将字符串信息恢复成结构体数组。由于每一项在打包时用 | 进行分隔，在解析时可以使用 `strchr` 定位下一项内容。

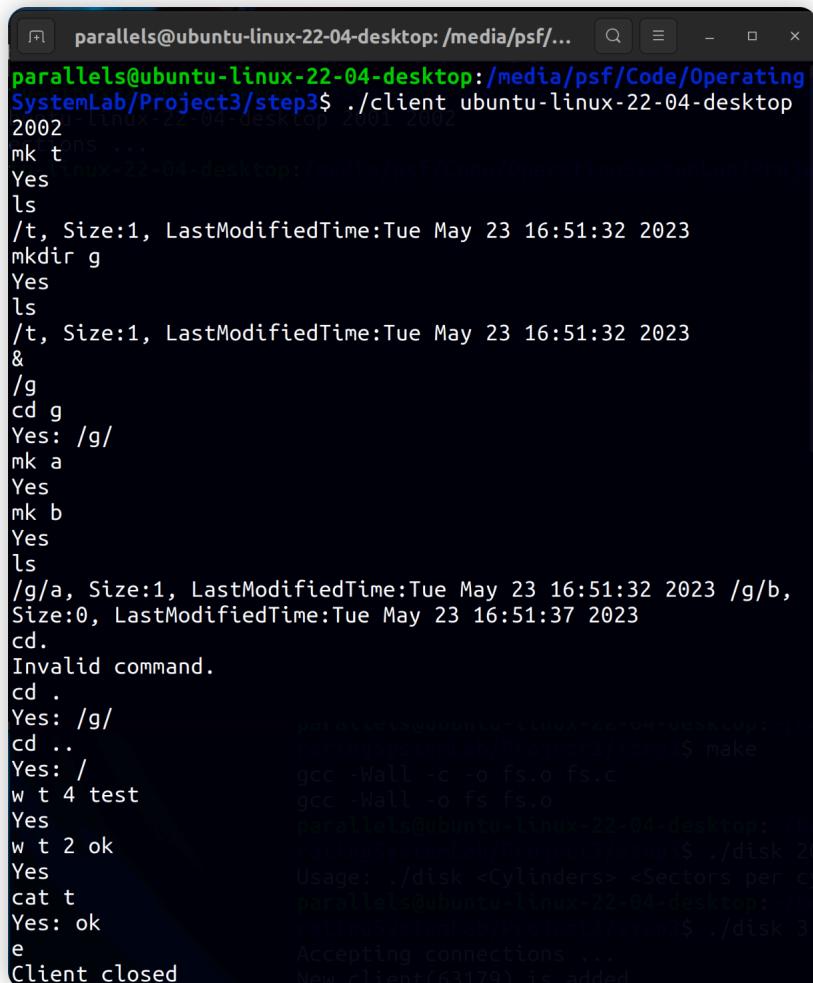
```

1 // Parse the struct from the string
2 void unpack_dir(const char* packed_string, DirectoryEntry* dir_entry) {
3     if(strcmp(packed_string, "\0") == 0){
4         file_count = 0;
5         return;
6     }
7     sscanf(packed_string, "%d", &file_count);
8     const char* current_position = packed_string + 1;
9
10    for (int i = 0; i < file_count; i++) {
11        sscanf(current_position, " %s %c %d %d %[^\n]",
12               &dir_entry[i].filename, &dir_entry[i].type, &dir_entry[i].start_block, &
13               dir_entry[i].file_size, &dir_entry[i].last_modified);
14
15        // Navigate to the location of the next directory entry
16        current_position = strchr(current_position, '|');
17        if (current_position == NULL) {
18            printf("Invalid packed_string format\n");
19            return;
20        }
21        current_position++; // Jump '/'
22    }
23 }
```

## 3.2 功能测试

### 3.2.1 基础功能测试

由于 Step2 进行了完善的测试，这里进行简单的测试，相应内容符合要求。

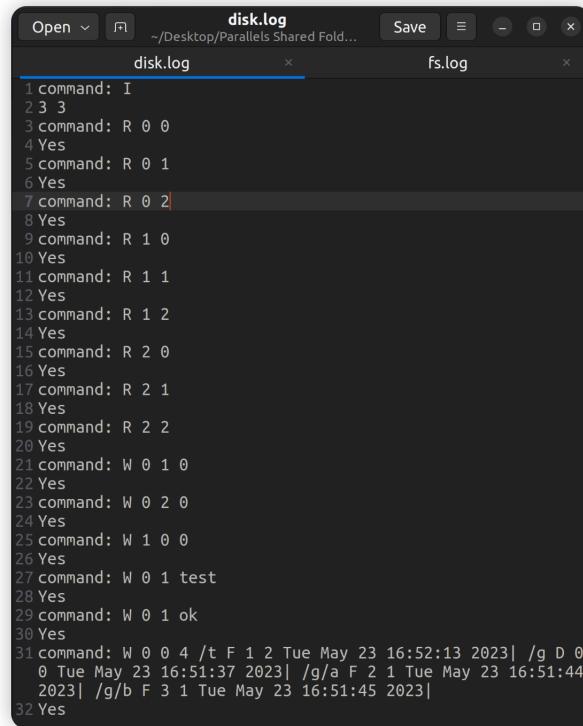


```
parallels@ubuntu-linux-22-04-desktop:/media/psf/...$ ./client ubuntu-linux-22-04-desktop
2002
mk t
Yes
ls
/t, Size:1, LastModifiedTime:Tue May 23 16:51:32 2023
mkdir g
Yes
ls
/t, Size:1, LastModifiedTime:Tue May 23 16:51:32 2023
&
/g
cd g
Yes: /g/
mk a
Yes
mk b
Yes
ls
/g/a, Size:1, LastModifiedTime:Tue May 23 16:51:32 2023 /g/b,
Size:0, LastModifiedTime:Tue May 23 16:51:37 2023
cd.
Invalid command.
cd .
Yes: /g/
cd ..
Yes: /
w t 4 test
Yes
w t 2 ok
Yes
cat t
Yes: ok
e
Client closed
```

```
parallels@ubuntu-linux-22-04-desktop:~/OperatingSystemLab/Project3/step3$ make
gcc -Wall -c -o fs.o fs.c
gcc -Wall -o fs fs.o
parallels@ubuntu-linux-22-04-desktop:~/OperatingSystemLab/Project3/step3$ ./disk 20
Usage: ./disk <Cylinders> <Sectors per cylinder>
parallels@ubuntu-linux-22-04-desktop:~/OperatingSystemLab/Project3/step3$ ./disk 3
Accepting connections ...
New client(63179) is added
```

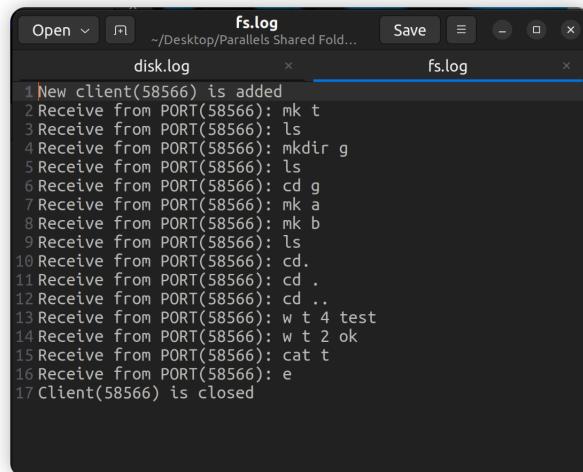
图 11: Step3 测试 1

所维护的两个日志文件内容如图 1213所示。



```
disk.log          fs.log
1 command: I
2 3 3
3 command: R 0 0
4 Yes
5 command: R 0 1
6 Yes
7 command: R 0 2
8 Yes
9 command: R 1 0
10 Yes
11 command: R 1 1
12 Yes
13 command: R 1 2
14 Yes
15 command: R 2 0
16 Yes
17 command: R 2 1
18 Yes
19 command: R 2 2
20 Yes
21 command: W 0 1 0
22 Yes
23 command: W 0 2 0
24 Yes
25 command: W 1 0 0
26 Yes
27 command: W 0 1 test
28 Yes
29 command: W 0 1 ok
30 Yes
31 command: W 0 0 4 /t F 1 2 Tue May 23 16:52:13 2023| /g D 0
   0 Tue May 23 16:51:37 2023| /g/a F 2 1 Tue May 23 16:51:44
   2023| /g/b F 3 1 Tue May 23 16:51:45 2023|
32 Yes
```

图 12: disk.log 内容



```
fs.log
1 New client(58566) is added
2 Receive from PORT(58566): mk t
3 Receive from PORT(58566): ls
4 Receive from PORT(58566): mkdir g
5 Receive from PORT(58566): ls
6 Receive from PORT(58566): cd g
7 Receive from PORT(58566): mk a
8 Receive from PORT(58566): mk b
9 Receive from PORT(58566): ls
10 Receive from PORT(58566): cd .
11 Receive from PORT(58566): cd ..
12 Receive from PORT(58566): cd ..
13 Receive from PORT(58566): w t 4 test
14 Receive from PORT(58566): w t 2 ok
15 Receive from PORT(58566): cat t
16 Receive from PORT(58566): e
17 Client(58566) is closed
```

图 13: fs.log 内容

### 3.2.2 恢复功能测试

在进行联合后，文件系统拥有非易失特性，可从上一次的系统情况继续操作，测试结果如图 14。

```
parallels@ubuntu-linux-22-04-desktop:~/Desktop/Parallels Share$ parallels@ubuntu-linux-22-04-desktop:/media/psf/Code/OperatingSystemLab/Project3$ ratingSystemLab/Project3/step3$ make
gcc -Wall -c -o disk.o disk.c
gcc -Wall -o disk disk.o
gcc -Wall -c -o fs.o fs.c
gcc -Wall -o fs fs.o
gcc -Wall -c -o client.o client.c
gcc -Wall -o client client.o
parallels@ubuntu-linux-22-04-desktop:~/Desktop/Parallels Share$ ratingSystemLab/Project3/step3$ ./disk 3 3 1 2001
Accepting connections ...
New client(43696) is added
parallels@ubuntu-linux-22-04-desktop:~/Desktop/Parallels Share$ ratingSystemLab/Project3/step3$ ./disk 3 3 1 2003
Accepting connections ...
New client(33469) is added
parallels@ubuntu-linux-22-04-desktop:~/Desktop/Parallels Share$ ratingSystemLab/Project3/step3$ |
parallels@ubuntu-linux-22-04-desktop:/media/psf/Code/OperatingSystemLab/Project3$ /step3$ ./client ubuntu-linux-22-04-desktop 2002
mk t
Yes
ls
/t, Size:1, LastModifiedTime:Tue May 23 17:47:26 2023
w t 2 ok
Yes
cat t
Yes: ok
e
Client closed
parallels@ubuntu-linux-22-04-desktop:/media/psf/Code/OperatingSystemLab/Project3$ /step3$ ./client ubuntu-linux-22-04-desktop 2004
ls
/t, Size:2, LastModifiedTime:Tue May 23 17:47:36 2023
i t 2 2 ok
Yes
cat t
Yes: okok
d t 2 2
parallels@ubuntu-linux-22-04-desktop:/media/psf/Code/OperatingSystemLab/Project3$ yes
cat t
Yes: ok
mkdir g
Yes
ls
/t, Size:2, LastModifiedTime:Tue May 23 17:48:35 2023
&
/g
cd g
Yes: /g
cd ..
Yes: /
f
Done
ls

e
Client closed
```

图 14: 恢复功能测试

## 总结与反思

本次实验首先设计和实现了一个基本的类似磁盘的辅助存储服务器；接着实现了一个基本的 FAT 结构文件系统；在此基础上用磁盘当服务器，用 socket 套接字使得文件系统能够和磁盘服务器交互写作，形成一个非易失文件系统，并将其作为第二级服务器，用客户端连接访问。并在 linux 系统上进行了测试，证明了该文件系统基本实现了相关要求。

由于多服务端端特性，调试过程困难繁琐。在此总结实验过程中的易错问题及其解决：

- buffer 以换行符作为一次读取的标准，需要在输出时谨慎，例如 ctime 转换的字符串末尾会带上换行符。
- 由于服务器和客户端采用阻塞读写，双方需协调好读写顺序，若出现多读，多写与少读，少写的情况会导致整个系统卡死。
- 日志文件采用 int fd\_log = open("disk.log", O\_WRONLY | O\_CREAT, S\_IRUSR | S\_IWUSR); 的方式打开操作繁琐且会出现乱码的情况，读写情况不透明。后期改为 File\* fd\_log = fopen("fs.log", "w"); 的方式创建文件。
- disk 并不真正“删除”某些数据，通过在开头添加\0 作为格式化方案。

本次实验使我对文件系统特性的了解大大加深，也大大提高我 c 语言处理内存，字符串和调试的能力。

该文件系统仍然存在不少不足的地方：

- 采用的 FAT 结构相较于 I-node 在效率上和拓展性上存在不足。
- 支持的文件数量较少且文件大小有限制，拓展性能不足。
- 未对输出进行进一步的美化工作，较为简陋。
- 文件的写入不支持回车等特殊情况。

总之，这是一个微型的文件系统，能够在一定程度上实现功能，却仍有很大进步空间。