CS356 **Operating System Projects Report** Spring 2014

Project 2: Design and Implementation of File System

5120309491 ChenShangJie

1.Objectives:

¬ Design and implement a basic disk-like secondary storage server.

¬ Design and implement a basic file system to act as a client using the disk services provided by the server designed above.

¬ Evaluate the performance of the disk server you implemented for various disk IO scheduling algorithms.

¬ Study and learn to use the socket API. Sockets will be used to provide the communication mechanism between (i) the client processes and the file system, and (ii) the file system and the disk storage server.

2. Basic disk-storage system:

Implement, as an Internet-domain socket server, a simulation of a physical disk. The simulated disk is organized by cylinder and sector. You should include in your simulation something to account for track-to-track time (using usleep(3C), nanosleep(3R) etc.). Let this be a value in microseconds, passed as a command-line parameter to the disk server program. Also, let the number of cylinders and number of sectors per cylinder be command line parameters. Assume the sector size is fixed at 256 bytes. Your simulation should store the actual data in a real disk file, so you'll want a filename for this file as another command line option.

¬ The Disk Protocol

The server must understand the following commands, and give the following responses:

1) I: information request. The disk returns two integers representing the disk geometry: the number of cylinders, and the number of sectors per cylinder.

2) R c s: read request for the contents of cylinder c sector s. The disk returns 'Yes' followed by a whitespace and those 256 bytes of information, or 'No' if no such block exists. (This will return whatever data happens to be on the disk in a given sector, even if nothing has ever been explicitly written there before.)

3) W c s l data: write request for cylinder c sector s. l is the number of bytes being provided, with a maximum of 256. The data is those 1 bytes of data. The disk returns 'Yes' to the client if it is a valid write request (legal values of c, s and l), or returns a 'No' otherwise. In cases where l < 256, the contents of those bytes of the sector between byte l and byte 256 is undefined, use zero-fill.

¬ Program Logic

BDS(Basic disk-storage system) is the initial part of the whole project. My program’s logic follows several steps:

1. Init all global variables you are going to use, like filename, cylinder number, sector number and so on.

2. Open a file to map the memory area(if not exists create one), map it to a char array named ”disk-file”.

3.Open a socket and listen, until the program BDC connecting to it.

4. Run a cycle: receive the request from BDC, exe-cute it and send the reply message.

5. Enter the loop(Ensure multiple users).

¬ **BDS Core code:**

```
while(1){  // enter loop
        bzero(buf,300);
        char *command_array[4] = {0};
        int nread = read(client_sockfd, buf, 300);
        if (nread < 0) error("ERROR reading from socket");
        count = parseLine(buf, command_array);//divide buf
        ctemp = c; // record current head
        switch (buf[0]){
                case 'I':
                        sprintf(buf,"%d %d", CYLINDERS,SECTORS);
                        n = write(client_sockfd,buf,256);
                        if (n < 0) error("ERROR writing to socket");
                        break;
                case 'R':
                        if(count != 3 ) {
                                char s[256] = "Wrong R command: ";
                                strcat(s, buf);
                                write(client_sockfd,buf,256);
                                break;}
                        c = atoi(command_array[1]);
                        s = atoi(command_array[2]);
                        if(c >= CYLINDERS || s >= SECTORS) write(client_sockfd,"No",256);
                        else {
                                char d[300] = "Yes ";
                                memcpy (buf, &diskfile[BLOCKSIZE * (c * SECTORS + s)],
BLOCKSIZE);
                                strcat(d, buf);
                                usleep(delay * abs(ctemp - c)); // simulate disk delay
                                n = write(client_sockfd,d,300);
                                if (n < 0) error("ERROR writing to socket");
                        }
                        break;
                case 'W':
                        if(count != 4 ) {write(client_sockfd,"Wrong W command",256);break;}
                        c = atoi(command_array[1]);
                        s = atoi(command_array[2]);
                        l = atoi(command_array[3]);
                        if(c >= CYLINDERS || s >= SECTORS || l > 256)
                                write(client_sockfd,"No",256);
                        else {
                                usleep(delay * abs(ctemp - c));
                                char buff[300] = {0};///
                                memcpy (buff, command_array[4],l);

                                memcpy (&diskfile[BLOCKSIZE * (c * SECTORS + s)], buff,
BLOCKSIZE);
                                n = write(client_sockfd,"Yes",300);
                                if (n < 0) error("ERROR writing to socket");
                        }
                        break;
```

```
                    case 'Q': printf("Q"); goto flag; break;
                    default: {
                            char s[300] = "Wrong command: ";
                            write(client_sockfd,s,300); break;}
            }//switch
        }//while
```

¬ **BDC_command core code:**

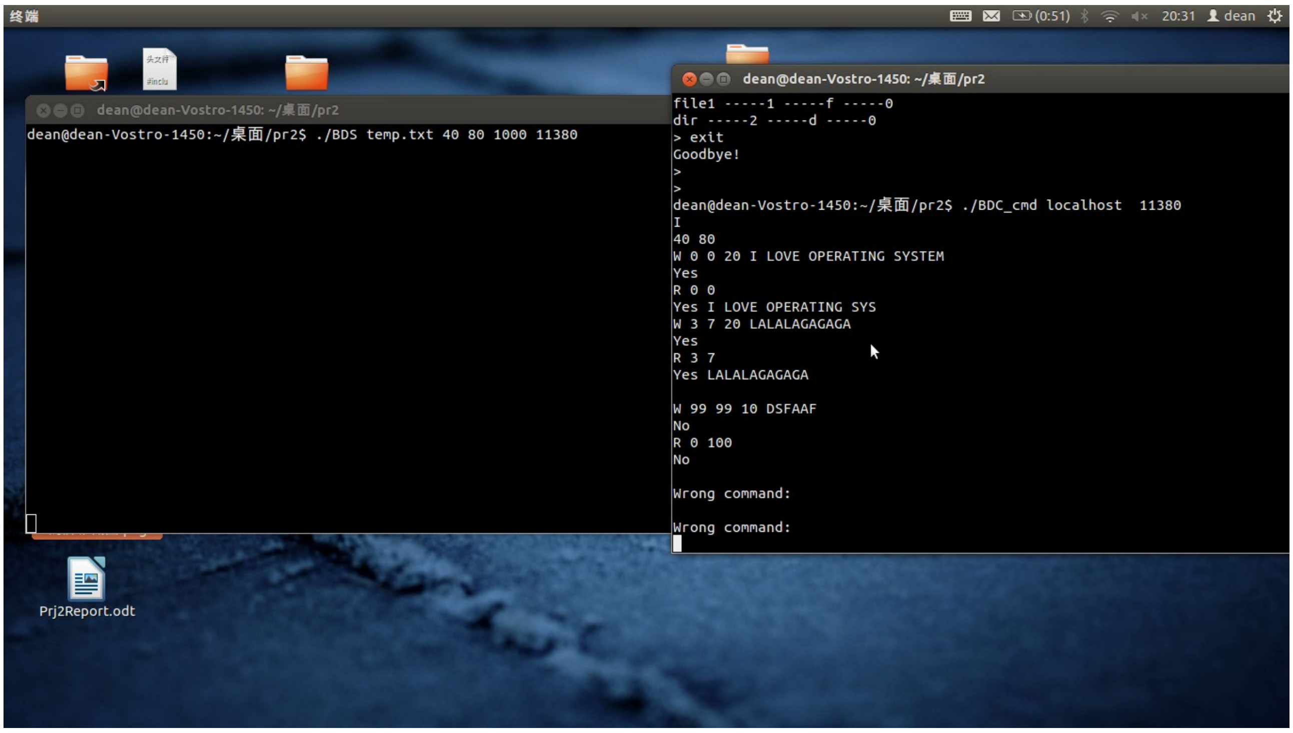
```
    while(1){
            for(i = 0; i < cache_size; i++){ //write command
                    bzero(buffer_write,300);
                    fgets(buffer_write,300,stdin);
                    n = write(sockfd,buffer_write,300);
                    if (n < 0) error("Error wrie");
                    if(buffer_write[0] == 'Q' || buffer_write[0] == 'q') {exit = 1; break;}
                    if(buffer_write[0] == 'I' || buffer_write[0] == 'i'){
                            bzero(buffer_read,300);
                            n = read(sockfd,buffer_read,300);
                            if (n < 0)  error("Error read");
                            printf("%s\n",buffer_read);
                            bzero(buffer_write,300);
                            fgets(buffer_write,300,stdin);
                            n = write(sockfd,buffer_write,300);
                            if (n < 0) error("Error write");
                    }
            }
            for(j = 0; j < i; j++){  // when send i command ,client must read n times
                    bzero(buffer_read,300);
                    n = read(sockfd,buffer_read,300);
                    if (n < 0)  error("Error read");
                    printf("%s\n",buffer_read);
                    }
            if (exit) break;
    }
```

¬ **BDC_rand core code:**

```
while(N > 0){
            for(i = 0; i < cache_size; i++){ //write command
                    bzero(buffer_write,300);
                    int c = random_num(Cylinders);
                    int s = random_num(Sectors);
                    int t = random_num(2);
                    switch (t){
                    case 0:
                            sprintf(buffer_write,"R %d %d",c,s);
                            printf(">R %d %d\n",c,s);//debug
                            break;
                    case 1:
                            {
                            char data[257] = {0}; int k;
                            for(k = 0; k< 256; k++) data[k] = rand()%95+32;
                            data[256] = '\0';
                            sprintf(buffer_write,"W %d %d 256 %s",c,s,data);
                            printf(">W %d %d\n",c,s);//debug
                            break;
                            }
                    }
                    n = write(sockfd,buffer_write,300);
                    if (n < 0) error("Error wrie");
                    if (N-- == 1){
                            sprintf(buffer_write,"Q");
                            n = write(sockfd,buffer_write,300);
                            break;
                    }
            }
            for(i = 0; i < cache_size; i++){
                    bzero(buffer_read,300);
                    n = read(sockfd,buffer_read,300);
                    if (n < 0)  error("Error read");
            }

    }
```

¬ **Sample Run:**



```
dean@dean-Vostro-1450:~/桌面/pr2$ ./BDS temp.txt 40 80 1000 11380
```

```
file1 -----1 -----f -----0
dir -----2 -----d -----0
> exit
Goodbye!
>
>
dean@dean-Vostro-1450:~/桌面/pr2$ ./BDC_cmd localhost  11380
I
40 80
W 0 0 20 I LOVE OPERATING SYSTEM
Yes
R 0 0
Yes I LOVE OPERATING SYS
W 3 7 20 LALALAGAGAGA
Yes
R 3 7
Yes LALALAGAGAGA

W 99 99 10 DSFAAF
No
R 0 100
No

Wrong command:

Wrong command:
```

BDS linking with BDC_cmd

3.Intelligent disk-storage system:

In this part the disk server will be made slightly more intelligent. It should obey exactly the same protocol as in the previous part. However, the disk server will gather (cache) a series of up to n requests and schedule/re-order them in such a way that the blocks can be read/written in an optimal way so that time spent seeking between cylinders is reduced. Let n be specified as yet another command line parameter. Note that this means that with n set to 1, this part will act exactly as the previous simple disk system. For the scheduler, use three different schedulers: first come first serve (FCFS), shortest seek time first (SSTF), and C-LOOK.

¬ The Disk Protocol

The IDS is very simular to BDS, the only difference is that you have to add the cache size in the command-line , and when you enter into the loop , the porgramm will ask you to select scheduling algorithms, such as FCFS / SSTF or C-LOOK.

¬ Program Logic

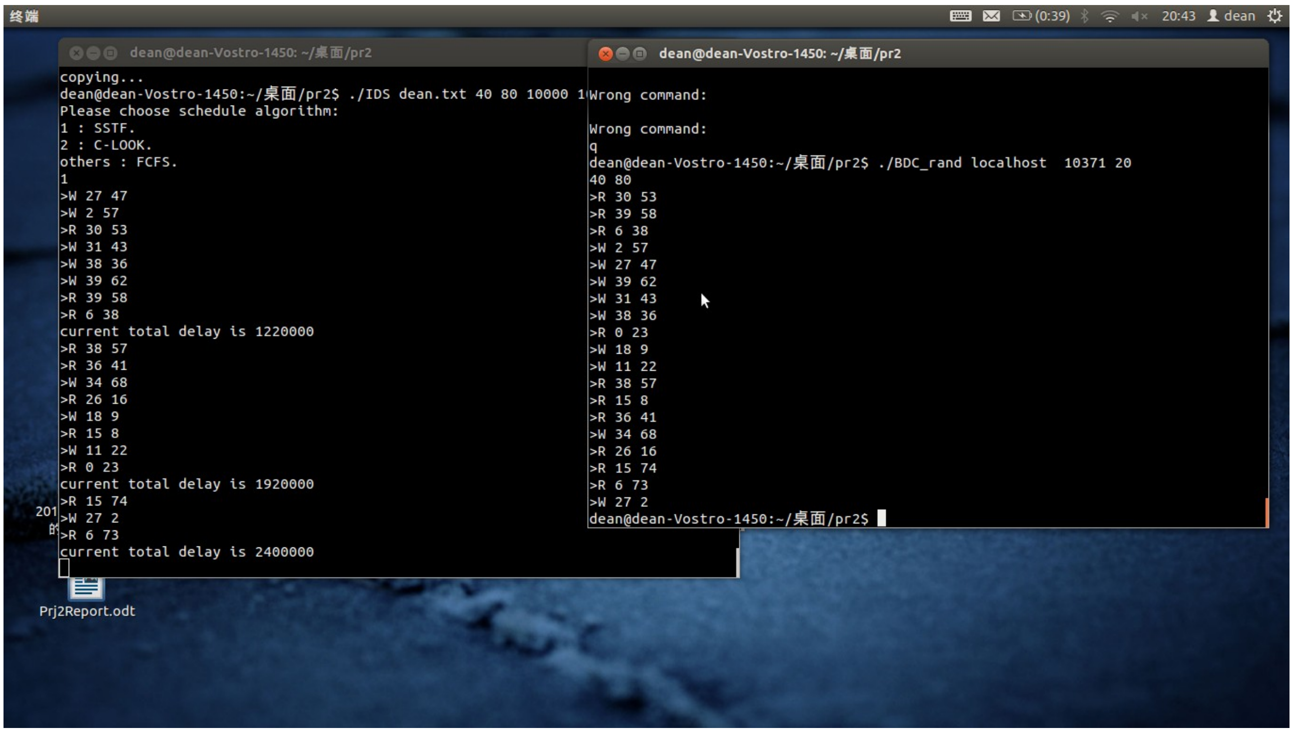
1. **FCFS**: Simply execute commands in sequence.
2. **SSTF**: The location of disk arm will always be recorded after an I/O operation. Before deciding which command to run, a traverse will be run to find the nearest location. And only after that will the next command be chosen.Then execute it.
3. **C-LOOK**: Firstly all commands will be sorted using Quick sort or Bubble sort(We use this one for simplicity). And then we start traversing from the current location and the previous direction. When int nread = 0; the arm hit the bottom(last location of this direction), it turns around, goes directly to the first location and continue traversing in the original direction.

¬ Core code:

```
void SSTF(){
// caculate the shortest move time and take it as the next command
        char temp[CACHES][300];
        int flag[CACHES];
        int queue[CACHES];
        int i; int j; int k;
        for(i = 0; i < CACHES; i++) memcpy(temp[i], cache[i], 300), flag[i] = 0 , queue[i] =
extract(cache[i]);
        for(i = 0; i < CACHES; i++){
                int seektime[CACHES];
                for(j = 0; j < CACHES; j++) seektime[j] = abs(queue[j] - head);// caculate seektime
                j = 0;
                while(flag[j] != 0 && j < CACHES) j++;
                int min = seektime[j]; k = j;
                for(j = 0; j < CACHES; j++){
                        if (seektime[j] <= min && flag[j] == 0 ) min = seektime[j], k = j;}
                flag[k] = 1;
                bzero(cache[i],300);
                memcpy(cache[i], temp[k], 300);
                head = queue[k];
        }
}
```

```
void C_LOOK(){
//sort the cache and move the head to the first place
        char temp[CACHES][300];
        int order[CACHES]; int queue[CACHES];
        int i;int j; int temp1; int temp2; int k;
        for(i = 0; i < CACHES; i++) memcpy(temp[i], cache[i], 300), queue[i] = extract(temp[i]),
order[i] = i;
        for(j = 0; j < CACHES - 1; j++){  // buble sort
                for(i = 0; i < CACHES - 1 - j; i++){
                   if(queue[i] > queue[i+1]){
                      temp1=queue[i]; temp2 = order[i];
                      queue[i]=queue[i+1]; order[i] = order[i+1];
                      queue[i+1]=temp1; order[i+1] = temp2;
                   }
                }
        }
        k = CACHES;
        for(i = 0; i < CACHES; i++)
                if(queue[i] >= head){
                        k = i;
                        break;}
        for(i = 0; i < CACHES; i++){
                memcpy(cache[i], temp[order[k%CACHES]], 300);
                head = extract(cache[i]); // find the head
                k++;
        }
}
```

¬ Sample Run



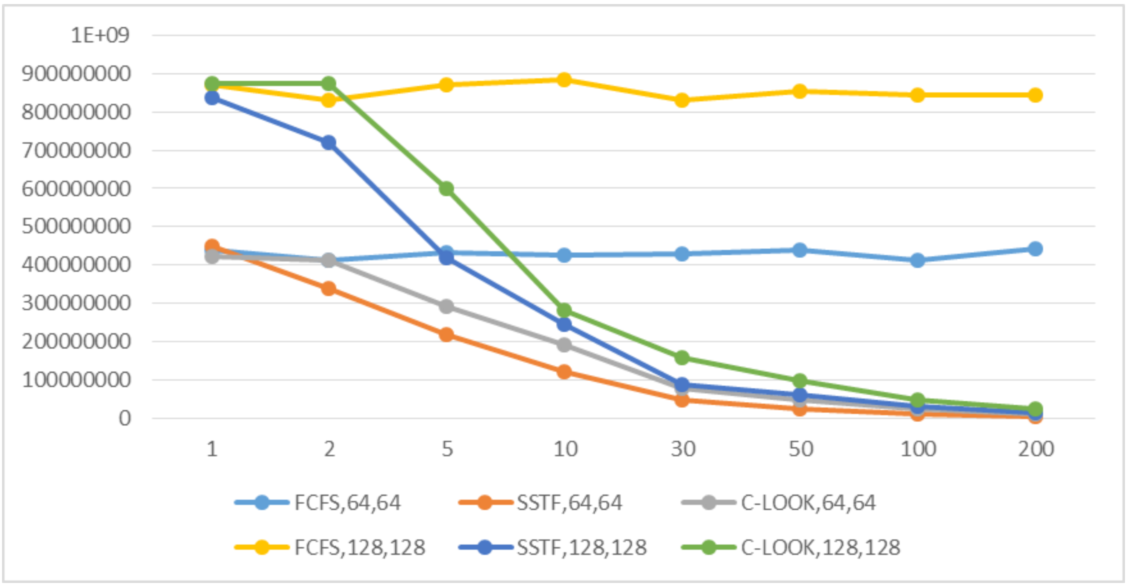
```
copying...
dean@dean-Vostro-1450:~/桌面/pr2$ ./IDS dean.txt 40 80 10000 1
Please choose schedule algorithm:
1 : SSTF.
2 : C-LOOK.
others : FCFS.
1
>W 27 47
>W 2 57
>R 30 53
>W 31 43
>W 38 36
>W 39 62
>R 39 58
>R 6 38
current total delay is 1220000
>R 38 57
>R 36 41
>W 34 68
>R 26 16
>W 18 9
>R 15 8
>W 11 22
>R 0 23
current total delay is 1920000
>R 15 74
>W 27 2
>R 6 73
current total delay is 2400000
```

```
Wrong command:

Wrong command:
q
dean@dean-Vostro-1450:~/桌面/pr2$ ./BDC_rand localhost  10371 20
40 80
>R 30 53
>R 39 58
>R 6 38
>W 2 57
>W 27 47
>W 39 62
>W 31 43
>W 38 36
>R 0 23
>W 18 9
>W 11 22
>R 38 57
>R 15 8
>R 36 41
>W 34 68
>R 26 16
>R 15 74
>R 6 73
>W 27 2
dean@dean-Vostro-1450:~/桌面/pr2$
```

IDS linking with BDS_rand with cachesize = 5 ,total commad = 20

¬ **Simulation Result**



From the graph we can see ,SSTF is the most effective algorithms , and then C-LOOK, FCFS is the worst. The total delay will decrease when cache size is bigger. The total delay will increase when total Cylinder increased.

4.File system:

Implement a inode file system. The file system should provide operations such as: initialize the file-system, create a file, read the data from a file, write a file with given data, append data to a file, remove a file, create directories, etc.

¬ The File-system Protocol

The server must understand and response to the following commands:

1) f: format. This will format the file-system on the disk, by initializing any/all of tables that the file-system relies on.

2) mk f: create file. This will create a file named f in the file-system. Possible return codes: 0 = successfully created the file; 1 = a file of this name already existed; 2 = some other failure (such as no space left, etc.).

3) mkdir d: create directory. This will create a subdirectory named d in the current directory. Possible return codes: 0 = successfully created the directory; 1 = a directory of this name already existed; 2 = some other failure.

4) rm f: delete file. This will delete the file named f from the file-system. Possible return codes: 0 = successfully deleted the file; 1 = a file of this name did not exist; 2 = some other failure.

5) cd path: change directory. This will change the current working directory to the path. The path can be either relative path or absolute path. When the file system starts, the initial working path is “/”. You need to handle “.” and “..”.

6) rmdir d: delete directory. This will delete the directory named d in the current directory. Possible return codes: 0 = successfully deleted the directory; 1 = a directory of this name did not exist; 2 = some other failure.

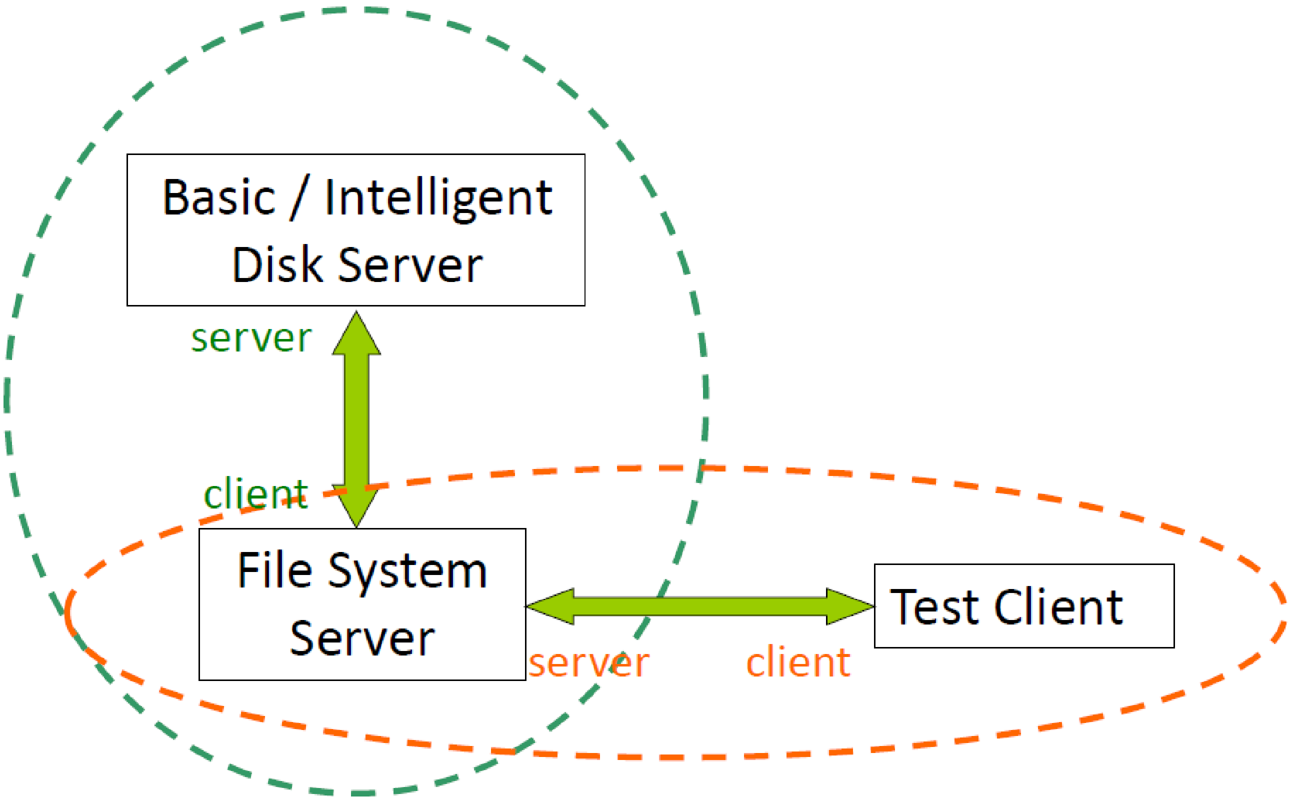
7) ls b: directory listing. This will return a listing of the files and directories in the file-systems. b is a Boolean flag: if '0', it lists just the names of all the files and directories, one per line; if '1', it includes other information about each file, such as file length, plus anything else your file-system might store about it.

8) cat f: catch file. This will read the entire contents of the file named f, and return the data that came from it. The message sent back to the client is, in order: a return code, a white-space, the number of bytes in the file (in ASCII), a white-space, and finally the data from the file. Possible return codes: 0 = successfully read file; 1 = no such filename exists; 2 = some other failure.

9) w f l data: write file. This will overwrite the contents of the file named f with the l bytes of data. If the new data is longer than the data previous in the file, the file will be made longer. If the new data is shorter than the data previously in the file, the file will be truncated to the new length. A return code is sent back to the client. Possible return codes: 0 = successfully written file; 1 = no such filename exists; 2 = some other failure (such as no space left, etc.).

10) a f l data: append to file. This will append the l bytes of data to the end of the current contents of the file named f. A return code is sent back to the client. Possible return codes: 0 = successfully appended to file; 1 = no such filename exists; 2 = some other failure (such as no space left, etc.).

11) exit: exit the file system.



The relations between each part

¬ I-node structure:

int	father_dir : the high directory of file
int	inode_num
int	Size : size for file and Capacity for directory
char	file_or_dir 'f' stands for file / 'd' stands for directory
int	index_num : current avaliable index
int[27]	index
	(20 single direct)
	6 double direct)
	(1 tripple direct)

My inode prototype

¬ The file system structure

Super block: to store the overall information of the file system, like inode number, size of a inode and so on.

Inode-bitmap: (for convenience we use a byte to represents an inode so actually it's a byte-map :)) In this file system. Use "y" represents available and use "n" represents occupied.

Block-bitmap: same as inode-bitmap, use one byte fer file operations to block operation, and send execueach to represents block state in disk.

Inode-list: we define each inode is 127bytes long, the structure of inode have been shown before.

Block-contents: blocks to save file information

Here I use some global variables to calculate block nums , to ensure that the blocks are always sufficient.

```
int CYLINDER; // total cylinder
int SECTOR; //sector per cylinder
int INBIT; // total inode bitmap  block numbers
int BLBIT; // total block bitmap  block numbers
int IMAP; // total inode map block number

INBIT = (SECTOR*CYLINDER/512+1);
BLBIT = (INBIT/2*3);
IMAP = (INBIT*256);
```

Warning:

Due to my weakness , I didn't achieve the part that save the inode system in the disk , though I tried many times , it always turned out to be segment fault … so every time excute the programm, you need to format it first , or it will turn to wrong part.

¬ **Program Logic:**

f: Init the system, including:init block superblock, inode-bitmap, block-bitmap, inode-list. Then write the root directory to the system, which is the first inode. After all that, refresh the block-bitmap.

mk f: Create new file, including: Read the current directory(which in fact is a file too), search for the file with same name. If not exists, looking for available inode and available block, then distribute them to the file by write the inode number and current_block number to the inode. At last modify the root directory to assign the new file.

mkdir: Create new directory, just like mk , but we should modify the file_or_dir flag to be 'd' to show it is a directory.

rm f: remove file, including : Searching inode in the father directory, if exist, clean all the bitmap of the file(inode bitmap and block bitmap) , remove the information of the file in the father directory.

rmdir d: remove directory, including: Searching inode in the father directory, if exist, clean all the bitmap of the directory, read the information of the directory ,and excute rm f and rmdir d for its lower files and directories.

cd path: change current directory, divide the path by '/' ,move to the first directory first and then find the next directory and move to it , finally it will move to target directory. When handle '.', it just do nothing, and when handle '..' ,move to the father directory of current directory.

ls b: list all the files. Just read the current directory and print out the informations.

w f l data: write file . Clean all the block bit first , and then deal with the single direct, write data to the avaliable block, when single direct is not enough, we use double direct, every double direct map to a new inode , the new inode will provide next 27 fresh blocks, when double is not enough ,we got tripple ~

a f l data: append to the file, it's just the same as cat and write .

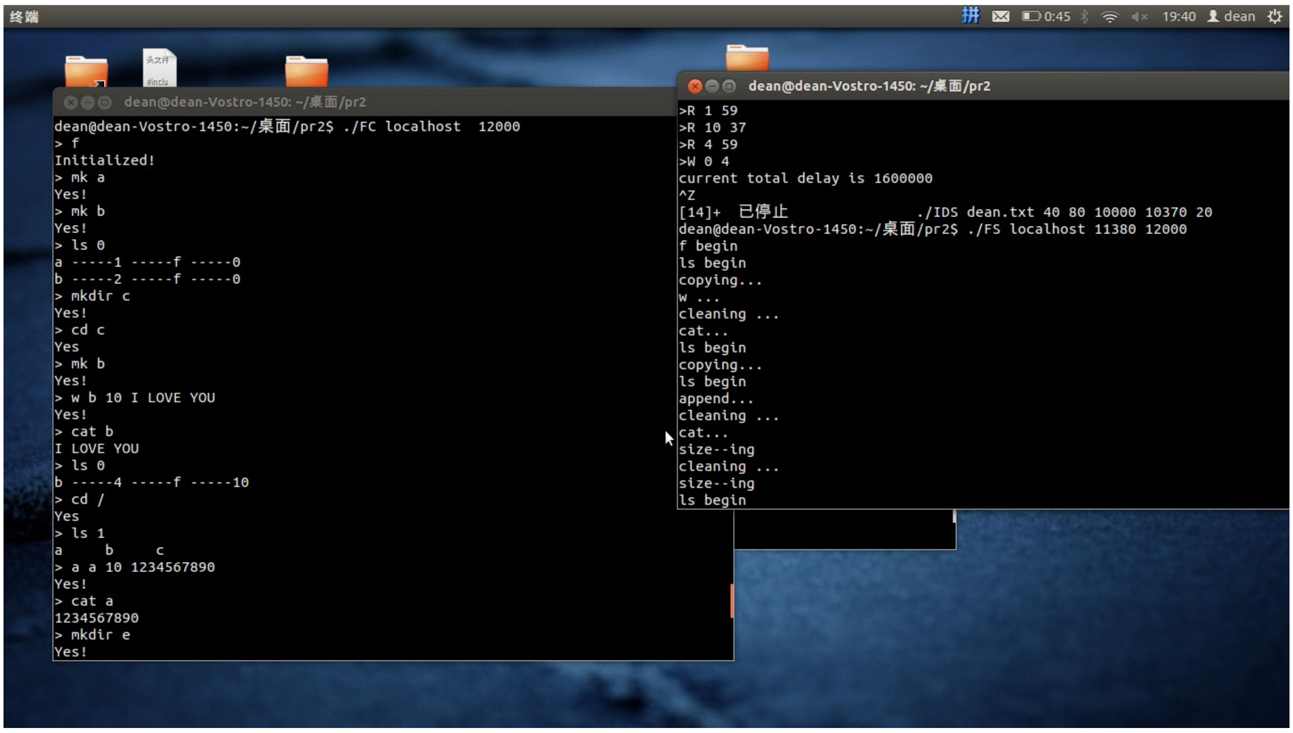
For specific function realization , please check my source codes.

¬ **Some Functions:**

```
int parse_line(char *, char *[], int); // divide command
void get_disk_info(int); // get cylinder and sector
void write_to_inode_bitmap(int); // save inode map
void read_from_disk(int, char[], int, int); // disk reading function
void write_to_block_bitmap(int);// save blockbit map
void write_to_disk(int, char[], int, int); // disk writing function
int find_inode_by_name(int, char[], int);
```

```
void read_from_inode_bitmap(int );
int find_next_inode();
int find_next_block();
char write_to_file(int, int, char *, int); // write data to file
char clean_file(int); // clean the bitmap of file
void remove_name(int,int, char[]); // remove name from high dir
void remove_dir(int , char[] , int );
char* i2a(int ,char* ); // change int to char*
int a2i(char*);          // change char * to int
```

¬ **Sample Run**



```
dean@dean-Vostro-1450: ~/桌面/pr2
dean@dean-Vostro-1450:~/桌面/pr2$ ./FC localhost  12000
> f
Initialized!
> mk a
Yes!
> mk b
Yes!
> ls 0
a -----1 -----f -----0
b -----2 -----f -----0
> mkdir c
Yes!
> cd c
Yes
> mk b
Yes!
> w b 10 I LOVE YOU
Yes!
> cat b
I LOVE YOU
> ls 0
b -----4 -----f -----10
> cd /
Yes
> ls 1
a      b      c
> a a 10 1234567890
Yes!
> cat a
1234567890
> mkdir e
Yes!
```

```
dean@dean-Vostro-1450: ~/桌面/pr2
>R 1 59
>R 10 37
>R 4 59
>W 0 4
current total delay is 1600000
^Z
[14]+  已停止                 ./IDS dean.txt 40 80 10000 10370 20
dean@dean-Vostro-1450:~/桌面/pr2$ ./FS localhost 11380 12000
f begin
ls begin
copying...
w ...
cleaning ...
cat...
ls begin
copying...
ls begin
append...
cleaning ...
cat...
size--ing
cleaning ...
size--ing
ls begin
```

5. Self-impressions

It was my first time to write such a big project alone, and it was really a big test for my coding ability. During dealing with the project, I need to study the linux inode system by myself ,and I need to deeply consider my program API . At the beginning, my efficiency is very low as I can't understand the inode system, so I have to search it in the internet, and ask classmates for help. When it comes to the coding part, I need to handle the whole system, I cannot turn to internet for help, I have to rely on myself. And at that time I had never thought that I can made it, though there are still some bugs in my program. I'm very content with it. And it really improve my patience and coding abilities.**At last thanks Dr.Wu for teaching me , and thanks for TA for guiding and writing the PDF.**