TypeScript of Project3: Design and Implementation of File System

Zhou Zihan 522031910214

Contents

TypeScript of Project3: Design and Implementation of File System **1**

1. Step 1 ..... **1**

2. Step 2 ..... **3**

2.1. `cmd1.txt` : Basic Commands 4

2.2. `cmd2.txt` : Multi-level Paths to Target 5

2.3. `cmd3.txt` : Additional Features 7

2.4. `cmd4.txt` : Error Handling 9

2.5. `cmd5.txt` : Stress Test 12

3. Step 3 ..... **15**

3.1. `cmd1.txt` : Basic User Operations 16

3.2. `cmd2.txt` : Premission Control 17

3.3. `cmd3.txt` : Error Handling 19

3.4. Consistency Check 21

1. Step 1

Setting up:

After executing the makefile in the `Project3` directory, you can run the Basic Disk-like Server with the following command:

```
./BDS <DiskFileName> <#Cylinders> <#SectorPerCylinder> <#Track-to-trackDelay> <#port>
```

e.g., a disk file named `test.disk` with 64 cylinders, 8 sectors per cylinder, 100us track-to-track delay, and listen on port 2333:

```
./BDS test.disk 64 8 100 2333
```

Then we provide two types of clients, the **Command-line driven client** (`BDC_command`) and **Random data client** (`BDC_random`). You can run the clients with the following commands:

```
./BDC_command <DiskServerAddress> <#port>
```

```
./BDC_random <DiskServerAddress> <#port> <#numRequests>
```

For exmaple, the following results show the output of the random data client with 20 requests:

```
./BDC_random localhost 2333 20
```

The output is randomly generated, so the results may vary:

```
I
64 8

W 31 3
Yes

W 12 0
Yes

R 20 7
Yes

R 21 3
Yes

R 39 2
Yes

R 8 5
Yes

R 58 6
Yes

W 40 5
Yes

W 56 3
Yes

W 2 5
Yes
```

```
R 9 0
Yes

W 40 2
Yes

W 39 1
Yes

W 56 2
Yes

R 17 3
Yes

W 32 6
Yes

W 2 7
Yes

R 52 4
Yes

W 57 4
Yes

R 30 7
Yes

E
Exiting...
Time used: 51575 microseconds.
```

The command-line driven client will prompt you to input the commands:

```
./BDC_command localhost 2333
```

When reading the sector data that had been write in the previous step, the client will print the data which shows the read and write operations are successful:

```
R 12 0
Yes DW|dKaJMoz,xfhqm2nQ0@uFm&X r)7aMn[27=Yd]Qm#5Wt$f`Ut"K;oNslA|$
Go[W'u.h ]W$r,uvrWIdYu FDs3cm4`3$=g(0rm1PB2AN(5?__$6R%\sulTa!6t"PY.`Luq}8$<g_Q$i1(!
`*ZU}Hx_F=QIm+POWF>R_@nGJAH4OP21XjcV0m6V)g
voQCGwbwW#f{J%D[RrkcKVD~f/5:8y;0iiS.^3&43m0]oR:BB&#jYGi=W{Wluo|\YNj6

W 0 0 10 helloworld
Yes

R 0 0
Yes helloworld
```

It could also correctly handle the invalid commands:

```
ABC
Invalid command

R 0 0 ABC
Invalid command

W 0 0 ABC
Invalid command

R -1 -1
No - Invalid cylinder or sector
```

Both the clients and server support exit command. Enter `E` in the client or `exit` in the server to exit the program:

```
$ ./BDC_command localhost 2333
E
Server disconnected
```

```
$ ./BDS test.disk 64 8 100 2333
Received command: E
Child process terminated, status = 0
[Host disconnected]
exit
Exiting...
```

2. Step 2

Setting up:

After making the executable files, you can run the Basic Disk-like Server, the File-system Server and Client with the following commands:

- **Run the Basic Disk-like Server:**

  ```
  ./BDS <DiskFileName> <#Cylinders> <#SectorPerCylinder> <#Track-to-trackDelay> <#port>
  ```

  similar to the previous step, an example command is as follows:

  ```
  ./BDS test.disk 64 8 100 2333
  ```

- **Run the File-system Server:**

  ```
  ./FS <DiskServerAddress> <#BDS_port> <#FS_port>
  ```

  for example, the following command will connect to the Disk Server on port 2333:

  ```
  ./FS localhost 2333 2560
  ```

- **Run the File-system Client:**

  ```
  ./FC <File-systemServerAddress> <#port> [cmdFile]
  ```

for example, the following command will connect to the File-system Server on port 2560:

```
./FC localhost 2560
```

If it is the first time you run the File-system Server, you should **format the disk** first by typing `f` in the client:

```
$ ./FC localhost 2560
Connecting to localhost:2560
File-system Server Connected
Please format the disk
FileServer:$ f

FileServer:/$
```

Then you can run the commands to test the file system.

We provide a **integrated automated verification tool** to test the file system. It could import commands from a file and execute them automatically. Simply run the File-system Client with the **4th argument** as the file name, e.g. run the following command to test the file system with basic commands:

```
./FC localhost 2560 test/cmd1.txt
```

The client will read the commands from the file and send them to the server to execute. The preset commands are as follows:

2.1. `cmd1.txt` : Basic Commands

This file contains the basic commands listed in the problem description.

```
f                       # f                 Format the disk
ls                      # ls                Directory listing
mkdir hello             # mkdir d           Create directory
mkdir world
mk test                 # mk f              Create file
ls -l                   # ls -l             Listing with details
w test 10 helloworld    # w f l data        Write data to a file
cat test                # cat f             Catch file
i test 5 5 _zzh_        # i f pos l data    Insert to a file
cat test
d test 10 5             # d f pos l         Delete from a file
cat test
rm test                 # rm f              Remove file
ls -l
cd hello                # cd path           Change directory
cd ..
ls -l
rmdir world             # rmdir d           Remove directory
ls -l
e                       # e                 Exit
```

The results should be as follows:

```
FileServer:/$ f

FileServer:/$ ls

FileServer:/$ mkdir hello

FileServer:/$ mkdir world

FileServer:/$ mk test

FileServer:/$ ls -l
d     17     Mon May 27 10:53:22 2024   hello
-      0     Mon May 27 10:53:23 2024   test
d     17     Mon May 27 10:53:22 2024   world

FileServer:/$ w test 10 helloworld

FileServer:/$ cat test
helloworld

FileServer:/$ i test 5 5 _zzh_

FileServer:/$ cat test
hello_zzh_world

FileServer:/$ d test 10 5

FileServer:/$ cat test
hello_zzh_

FileServer:/$ rm test

FileServer:/$ ls -l
d     17     Mon May 27 10:53:22 2024   hello
d     17     Mon May 27 10:53:22 2024   world

FileServer:/$ cd hello

FileServer:/hello$ cd ..

FileServer:/$ ls -l
d     17     Mon May 27 10:53:22 2024   hello
d     17     Mon May 27 10:53:22 2024   world

FileServer:/$ rmdir world

FileServer:/$ ls -l
d     17     Mon May 27 10:53:22 2024   hello

FileServer:/$ e
```

2.2. `cmd2.txt` : Multi-level Paths to Target

This file contains commands that locate directories and files with multi-level paths, both absolute and relative, and perform various operations including `mkdir` , `mk` , `w` , `cat` , `i` , `d` , `cd` , `rm` , and `rmdir` .

```
f
ls
mkdir hello                              # 1-level, relative
mkdir hello/world                        # 2-level, relative
mkdir hello/world/.././../world2         # Multi-level, relative
```

```
ls
mk /hello/world/test                        # Multi-level, absolute
w ./../hello/../hello/world/test 10 helloworld
cat hello/world/test
i /hello/world/test 5000 5 _zzh_            # Position larger than size
cat hello/world/test
d hello/./world/./test 5 5000               # Delete till the end
cat hello/world/test
cd hello/world
ls
cd ../../world2
ls
cd /
ls
rm hello/world/test
rmdir hello/world
rmdir hello
ls
e
```

The results should be as follows:

```
FileServer:/$ f

FileServer:/$ ls

FileServer:/$ mkdir hello

FileServer:/$ mkdir hello/world

FileServer:/$ mkdir hello/world/.././../world2

FileServer:/$ ls
hello  world2

FileServer:/$ mk /hello/world/test

FileServer:/$ w ./../hello/../hello/world/test 10 helloworld

FileServer:/$ cat hello/world/test
helloworld

FileServer:/$ i /hello/world/test 5000 5 _zzh_

FileServer:/$ cat hello/world/test
helloworld_zzh_

FileServer:/$ d hello/./world/./test 5 5000

FileServer:/$ cat hello/world/test
hello

FileServer:/$ cd hello/world

FileServer:/hello/world$ ls
test

FileServer:/hello/world$ cd ../../world2

FileServer:/world2$ ls

FileServer:/world2$ cd /
```

```
FileServer:/$ ls
hello  world2
FileServer:/$ rm hello/world/test
FileServer:/$ rmdir hello/world
FileServer:/$ rmdir hello
FileServer:/$ ls
world2
FileServer:/$ e
```

2.3. `cmd3.txt` : Additional Features

We implement three additional commands: `stat` , `du` and `pwd` :

- `stat <filename>` : Show the information of a file / directory, with its data block indexes.
- `du` : Show the disk usage and bitmaps of the disk.
- `pwd` : Show the current working directory.

We also add the ability to parse escaped characters in the data, starting with `\` then followed by two hexadecimal digits. For example, `\0a` represents a newline character `\n` .

```
f
du                      # du                Disk usage
mk test
w test 5000 asdfghjklqwerty...(4986 bytes more)
mkdir hello
mk hello/test2
w hello/test2 11 hello\0aworld            # Escaped characters
cat hello/test2
mkdir hello/world
pwd                     # pwd               Print working directory
cd hello/world
pwd
cd ../..
ls -l
stat test               # stat f            File information
stat hello
du
rm test
du
e
```

These features should work as expected:

```
FileServer:/$ f
FileServer:/$ du
Volume information:
    Block size:      256    Inode size:       64
  Blocks count:      445  Inodes count:      256
```

```
   Free blocks:      444   Free inodes:      255
   Total space:   131072    Used space:    0.83%

Inode bitmap:
0x0000 - 0x00ff: .

Block bitmap:
0x0000 - 0x00ff: .
0x0100 - 0x01ff:

FileServer:/$ mk test

FileServer:/$ w test 5000 asdfghjklqwertyuiopzxcvbnm1234567890...(4963 bytes more)

FileServer:/$ mkdir hello

FileServer:/$ mk hello/test2

FileServer:/$ w hello/test2 11 hello\0aworld

FileServer:/$ cat hello/test2
hello
world

FileServer:/$ mkdir hello/world

FileServer:/$ pwd
/

FileServer:/$ cd hello/world

FileServer:/hello/world$ pwd
/hello/world

FileServer:/hello/world$ cd ../..

FileServer:/$ ls -l
d     39     Mon May 27 16:39:17 2024   hello
-   5000     Mon May 27 16:39:15 2024   test

FileServer:/$ stat test
  File: test
  Size: 5000            Blocks: 20                File
 Inode: 1               Parent: 0               Links: 1
Access: Mon May 27 16:39:15 2024
Modify: Mon May 27 16:39:15 2024
Create: Mon May 27 16:39:15 2024

Direct blocks:         1       2       3       4       5       6       7
Indirect blocks:        9
               8      10      11      12      13      14      15      16
              17      18      19      20      21

FileServer:/$ stat hello
  File: hello
  Size: 39              Blocks: 1              Directory
 Inode: 2               Parent: 0               Links: 1
Access: Mon May 27 16:39:18 2024
Modify: Mon May 27 16:39:17 2024
Create: Mon May 27 16:39:15 2024

Direct blocks:        22

FileServer:/$ du
Volume information:
    Block size:      256    Inode size:       64
```

```
  Blocks count:      445  Inodes count:      256
   Free blocks:      420   Free inodes:      251
   Total space:   131072    Used space:    5.71%

Inode bitmap:
0x0000 - 0x00ff: ⣿

Block bitmap:
0x0000 - 0x00ff: ⣿⣿⣿⣿⣿⣿⣿⣿
0x0100 - 0x01ff:

FileServer:/$ rm test

FileServer:/$ du
Volume information:
    Block size:      256    Inode size:       64
  Blocks count:      445  Inodes count:      256
   Free blocks:      441   Free inodes:      252
   Total space:   131072    Used space:    1.56%

Inode bitmap:
0x0000 - 0x00ff: ⣿

Block bitmap:
0x0000 - 0x00ff: ⠁ ⣿
0x0100 - 0x01ff:

FileServer:/$ e
```

2.4. `cmd4.txt` : Error Handling

If everything goes well, you shouldn't see any error messages in previous tests. This file contains commands that may cause errors, such as creating a file that already exists, deleting a file that doesn't exist, etc.

```
f
invalidCmd                      # Invalid command
mk more args                    # More arguments than expected
mkdir                           # Less arguments than expected
mk test
mk test/notDir                  # Item in path is not a directory
mkdir dir
mkdir dir                       # Directory already exists
mkdir removed
cd removed
rmdir ../removed                # Try to remove current dir (Warning)
ls
w dir 10 writeToDir             # Write to a directory
w test -10 invalidLen           # Invalid data length
w test 11 normalWrite
cat test
w test 0 emptyWrite
cat test
w test 2000 dataShorterThanLen  # Data shorter than length (Warning)
cat test
stat test
w test 4 dataLongerThanLen      # Data longer than length (Warning)
```

```
cat test
i test -1 10 invalidOffset      # Invalid position
i test 0 -10 invalidLen
d test -1 10
d test 0 -10
d test 5000 10                  # Position out of range
cat test
cd ..                           # Change to parent of root
cd notExist                     # Change to directory not exist
mk dir/notEmpty
rmdir dir                       # Remove non-empty directory
rmdir test                      # Use rmdir to remove a file
rm dir                          # Use rm to remove a directory
ls
e
```

The results you would see should be like:

```
FileServer:/$ f

FileServer:/$ invalidCmd
Error: Invalid command: invalidCmd

FileServer:/$ mk more args
Error: Invalid argument

FileServer:/$ mkdir
Error: Invalid argument

FileServer:/$ mk test

FileServer:/$ mk test/notDir
Error: Invalid path

FileServer:/$ mkdir dir

FileServer:/$ mkdir dir
Error: Target already exists

FileServer:/$ mkdir removed

FileServer:/$ cd removed

FileServer:/removed$ rmdir ../removed
Warning: Current path no longer exists

FileServer:/$ ls
dir  test

FileServer:/$ w dir 10 writeToDir
Error: Invalid path

FileServer:/$ w test -10 invalidLen
Error: Invalid argument

FileServer:/$ w test 11 normalWrite

FileServer:/$ cat test
normalWrite

FileServer:/$ w test 0 emptyWrite
Warning: Input data truncated by data length
```

```
FileServer:/$ cat test

FileServer:/$ w test 2000 dataShorterThanLen
Warning: Input data shorter than data length, padding with zeros

FileServer:/$ cat test
dataShorterThanLen

FileServer:/$ stat test
  File: test
  Size: 2000            Blocks: 8                  File
 Inode: 1               Parent: 0                Links: 1
Access: Mon May 27 11:40:32 2024
Modify: Mon May 27 11:40:32 2024
Create: Mon May 27 11:40:29 2024

Direct blocks:         2       3       4       5       6       7       8
Indirect blocks:        10
               9

FileServer:/$ w test 4 dataLongerThanLen
Warning: Input data truncated by data length

FileServer:/$ cat test
data

FileServer:/$ i test -1 10 invalidOffset
Error: Invalid argument

FileServer:/$ i test 0 -10 invalidLen
Error: Invalid argument

FileServer:/$ d test -1 10
Error: Invalid argument

FileServer:/$ d test 0 -10
Error: Invalid argument

FileServer:/$ d test 5000 10
Error: Invalid position

FileServer:/$ cat test
data

FileServer:/$ cd ..
Error: No such file or directory

FileServer:/$ cd notExist
Error: No such file or directory

FileServer:/$ mk dir/notEmpty

FileServer:/$ rmdir dir
Error: Directory not empty

FileServer:/$ rmdir test
Error: Invalid path

FileServer:/$ rm dir
Error: Invalid path

FileServer:/$ ls
dir  test

FileServer:/$ e
```

2.5. `cmd5.txt` : Stress Test

This file contains commands that create a super large file, write, inset and delete data from it. Please make sure you have created the disk with enough data blocks.

```
f
du
ls
mk largeFile
w largeFile 5000 asdfghjklqwerty...(4986 bytes more)    # 5000 bytes
i largeFile 5000 5000 asdfghjklq...(4991 bytes more)    # 10000 bytes
stat largeFile
i largeFile 10000 5000 asdfghjkl...(4992 bytes more)    # 15000 bytes
i largeFile 15000 5000 asdfghjkl...(4992 bytes more)    # 20000 bytes
i largeFile 20000 5000 asdfghjkl...(4992 bytes more)    # 25000 bytes
du
i largeFile 25000 5000 asdfghjkl...(4992 bytes more)    # 30000 bytes
i largeFile 30000 5000 asdfghjkl...(4992 bytes more)    # 35000 bytes
i largeFile 35000 5000 asdfghjkl...(4992 bytes more)    # 40000 bytes
du
stat largeFile
d largeFile 35000 5000                                  # 35000 bytes
d largeFile 30000 5000                                  # 30000 bytes
d largeFile 25000 5000                                  # 25000 bytes
stat largeFile
d largeFile 20000 5000                                  # 20000 bytes
d largeFile 15000 5000                                  # 15000 bytes
d largeFile 10000 5000                                  # 10000 bytes
du
d largeFile 5000 5000                                   # 5000 bytes
d largeFile 0 5000                                      # 0 byte
stat largeFile
e
```

The results could show that the file system could handle large files and operations correctly:

```
FileServer:/$ f

FileServer:/$ du
Volume information:
    Block size:      256    Inode size:       64
  Blocks count:      445  Inodes count:      256
   Free blocks:      444   Free inodes:      255
   Total space:   131072    Used space:    0.83%

Inode bitmap:
0x0000 - 0x00ff: .

Block bitmap:
0x0000 - 0x00ff: .
0x0100 - 0x01ff:

FileServer:/$ ls

FileServer:/$ mk largeFile
```

```
FileServer:/$ w largeFile 5000 asdfghjklqwertyuiopzxcvbnm12345...(4968 bytes more)

FileServer:/$ i largeFile 5000 5000 asdfghjklqwertyuiopzxcvbnm...(4973 bytes more)

FileServer:/$ stat largeFile
  File: largeFile
  Size: 10000           Blocks: 40                File
 Inode: 1               Parent: 0               Links: 1
Access: Thu May 30 10:36:48 2024
Modify: Thu May 30 10:36:48 2024
Create: Thu May 30 10:36:47 2024

Direct blocks:         1       2       3       4       5       6       7
Indirect blocks:        9
               8      10      11      12      13      14      15      16
              17      18      19      20      21      22      23      24
              25      26      27      28      29      30      31      32
              33      34      35      36      37      38      39      40
              41

FileServer:/$ i largeFile 10000 5000 asdfghjklqwertyuiopzxcvbn...(4974 bytes more)

FileServer:/$ i largeFile 15000 5000 asdfghjklqwertyuiopzxcvbn...(4974 bytes more)

FileServer:/$ i largeFile 20000 5000 asdfghjklqwertyuiopzxcvbn...(4974 bytes more)

FileServer:/$ du
Volume information:
    Block size:      256    Inode size:       64
  Blocks count:      445  Inodes count:      256
   Free blocks:      343   Free inodes:      254
   Total space:   131072    Used space:   20.61%

Inode bitmap:
0x0000 - 0x00ff: ⁝

Block bitmap:
0x0000 - 0x00ff: ⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
0x0100 - 0x01ff:

FileServer:/$ i largeFile 25000 5000 asdfghjklqwertyuiopzxcvbn...(4974 bytes more)

FileServer:/$ i largeFile 30000 5000 asdfghjklqwertyuiopzxcvbn...(4974 bytes more)

FileServer:/$ i largeFile 35000 5000 asdfghjklqwertyuiopzxcvbn...(4974 bytes more)

FileServer:/$ du
Volume information:
    Block size:      256    Inode size:       64
  Blocks count:      445  Inodes count:      256
   Free blocks:      283   Free inodes:      254
   Total space:   131072    Used space:   32.32%

Inode bitmap:
0x0000 - 0x00ff: ⁝

Block bitmap:
0x0000 - 0x00ff: ⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
0x0100 - 0x01ff:

FileServer:/$ stat largeFile
  File: largeFile
  Size: 40000           Blocks: 157               File
 Inode: 1               Parent: 0               Links: 1
```

```
Access: Thu May 30 10:36:50 2024
Modify: Thu May 30 10:36:50 2024
Create: Thu May 30 10:36:47 2024

Direct blocks:         1       2       3       4       5       6       7
Indirect blocks:        9
               8      10      11      12      13      14      15      16
              17      18      19      20      21      22      23      24
              25      26      27      28      29      30      31      32
              33      34      35      36      37      38      39      40
              41      42      43      44      45      46      47      48
              49      50      51      52      53      54      55      56
              57      58      59      60      61      62      63      64
              65      66      67      68      69      70      71      72
Double indirect:        74
        Indirect blocks:        75
                      73      76      77      78      79      80      81      82
                      83      84      85      86      87      88      89      90
                      91      92      93      94      95      96      97      98
                      99     100     101     102     103     104     105     106
                     107     108     109     110     111     112     113     114
                     115     116     117     118     119     120     121     122
                     123     124     125     126     127     128     129     130
                     131     132     133     134     135     136     137     138
        Indirect blocks:        140
                     139     141     142     143     144     145     146     147
                     148     149     150     151     152     153     154     155
                     156     157     158     159     160     161

FileServer:/$ d largeFile 35000 5000

FileServer:/$ d largeFile 30000 5000

FileServer:/$ d largeFile 25000 5000

FileServer:/$ stat largeFile
  File: largeFile
  Size: 25000           Blocks: 98                File
 Inode: 1               Parent: 0                Links: 1
Access: Thu May 30 10:36:52 2024
Modify: Thu May 30 10:36:52 2024
Create: Thu May 30 10:36:47 2024

Direct blocks:         1       2       3       4       5       6       7
Indirect blocks:        9
               8      10      11      12      13      14      15      16
              17      18      19      20      21      22      23      24
              25      26      27      28      29      30      31      32
              33      34      35      36      37      38      39      40
              41      42      43      44      45      46      47      48
              49      50      51      52      53      54      55      56
              57      58      59      60      61      62      63      64
              65      66      67      68      69      70      71      72
Double indirect:        74
        Indirect blocks:        75
                      73      76      77      78      79      80      81      82
                      83      84      85      86      87      88      89      90
                      91      92      93      94      95      96      97      98
                      99     100     101

FileServer:/$ d largeFile 20000 5000
```

```
FileServer:/$ d largeFile 15000 5000

FileServer:/$ d largeFile 10000 5000

FileServer:/$ du
Volume information:
    Block size:      256    Inode size:       64
  Blocks count:      445  Inodes count:      256
   Free blocks:      403   Free inodes:      254
   Total space:   131072    Used space:    8.89%

Inode bitmap:
0x0000 - 0x00ff: ⠁

Block bitmap:
0x0000 - 0x00ff: ⣿⣿⣿⣿⣿⣿⣿⣿
0x0100 - 0x01ff:

FileServer:/$ d largeFile 5000 5000

FileServer:/$ d largeFile 0 5000

FileServer:/$ stat largeFile
  File: largeFile
  Size: 0               Blocks: 0                File
 Inode: 1               Parent: 0               Links: 1
Access: Thu May 30 10:36:54 2024
Modify: Thu May 30 10:36:54 2024
Create: Thu May 30 10:36:47 2024

Direct blocks:

FileServer:/$ e
```

You can also add a `cat largeFile` command after the final insertation to test the robustness and performance of the socket communication.

(This is not included defaultly just because it's difficult to scroll back to the top of the output to check the results. But through my tests, the file system was able to return this 40,000 bytes data intact and correctly)

3. Step 3

Setting up:

Similar to the previous step, after compiling the code, you can run the server and client with the following commands:

```
./BDS <DiskFileName> <#Cylinders> <#SectorPerCylinder> <#Track-to-trackDelay> <#port>
```

```
./FS <DiskServerAddress> <#BDS_port> <#FS_port>
```

```
./FC <File-systemServerAddress> <#port> <Username> [Password] [cmdFile]
```

For example, run the following commands in three different terminals:

```
./BDS test.disk 64 8 100 2333
```

```
./FS localhost 2333 2560
```

```
./FC localhost 2560 root
```

The first-run results should be as follows:

```
$ ./FC localhost 2560 root
Connecting to localhost:2560
File-system Server Connected
Please format the disk
:$ f

root:/home$
```

You could type in any username in its first run, after formatting the user will be the root user automatically.

The **integrated automated verification tool** is also available for this step. You can place the test file in the **6th argument** to run the commands. Note that if the user has not set a password, fill in the fifth parameter with `-` . For example, run the following command to test the file system with the `cmd1.txt` file:

```
./FC localhost 2560 root - test/cmd1.txt
```

3.1. `cmd1.txt` : Basic User Operations

In this file, we will test the basic user operations, such as adding a new user, setting a password, switching between users, and deleting a user. The commands including:

- `useradd <username>` : Add a new user.
- `userdel <username>` : Delete a user.
- `passwd [username]` : Set a password for the user. If no username is provided, set the password for the current user.
- `su [username]` : Switch to another user. If no username is provided, switch to root.

By design, only the root user can add or delete users.

```
f
ls -l
useradd user1           # useradd u         Add user
ls -l
passwd user1            # passwd u          Set password
123
123
su user1                # su u              Switch user
123                                       # Enter password for user1
ls -l
mk hello
w hello 10 helloworld
```

```
su                      # su                Switch to root
                                           # No password for root
ls -l
cat user1/hello
userdel user1
ls -l
e
```

After a new user is added, a directory with the same name as the user should be created in the `/home` directory. The user should be able to write and read files in their home directory. However, this directory will not be removed when the user is deleted to keep the data.

The results should be as follows:

```
root:/home$ f

root:/home$ ls -l
drwxrwx            root     17     Wed May 29 21:05:34 2024     public

root:/home$ useradd user1

root:/home$ ls -l
drwxrwx            root     17     Wed May 29 21:05:34 2024     public
drwx---           user1     17     Wed May 29 21:05:35 2024     user1

root:/home$ passwd user1
Enter password: 123
Confirm password: 123

root:/home$ su user1
Enter password: 123

user1:/home/user1$ ls -l

user1:/home/user1$ mk hello

user1:/home/user1$ w hello 10 helloworld

user1:/home/user1$ su
Enter password:

root:/home$ ls -l
drwxrwx            root     17     Wed May 29 21:05:34 2024     public
drwx---           user1     28     Wed May 29 21:05:36 2024     user1

root:/home$ cat user1/hello
helloworld

root:/home$ userdel user1

root:/home$ ls -l
drwxrwx            root     17     Wed May 29 21:28:23 2024     public
drwx---         unknown     28     Wed May 29 21:28:25 2024     user1

root:/home$ e
```

3.2. `cmd2.txt` : Premission Control

With the user system, we can now implement permission control. We implement Unix-like permission control, including read, write, and execute permissions for the owner and other users (no group permission). The commands include:

- `chmod <filename> <permission>` : Change the permission of a file.
- `chown <filename> <username>` : Change the owner of a file.

Without premission, users should only be able to read and write files in their home and the public directory. They should not be able to read or write files in other users' directories. The root user should have full access to all files.

The following commands will test the permission control:

```
f
useradd user1
useradd user2
ls -l
mk hello                # root: create a file in home
ls -l
su user1

mk hello                # user1: create a file in user1
ls -l
chown hello user2       # chown f u          Change owner
ls -l
cd ..
w hello 10 helloworld   # user1: cannot create/modify file in home
mk world
cd user2                # user1: cannot access user2 directory
chmod user1 77          # chmod u p           Change permission
ls -l
su user2

cd ../user1             # user2: access user1 directory after chmod
mk world
w hello 10 helloworld   # user2: write to file after chown
ls -l
cat hello
e
```

You will see that, `user1` do not have permission to create a file in `home` directory or even change to `user2` directory. But after changing the permission of `user1` directory, `user2` can create a file in `user1` directory.

Also, `ls -l` command is more powerful now, it should show the permission and owner of the files and directories. The results should be as follows:

```
root:/home$ f
root:/home$ useradd user1
root:/home$ useradd user2
root:/home$ ls -l
drwxrwx            root     17     Wed May 29 21:37:15 2024     public
drwx---           user1     17     Wed May 29 21:37:15 2024     user1
drwx---           user2     17     Wed May 29 21:37:16 2024     user2
```

```
root:/home$ mk hello

root:/home$ ls -l
-rw-r--            root      0     Wed May 29 21:37:16 2024     hello
drwxrwx            root     17     Wed May 29 21:37:15 2024     public
drwx---           user1     17     Wed May 29 21:37:15 2024     user1
drwx---           user2     17     Wed May 29 21:37:16 2024     user2

root:/home$ su user1
Enter password:

user1:/home/user1$ mk hello

user1:/home/user1$ ls -l
-rw-r--           user1      0     Wed May 29 21:40:57 2024     hello

user1:/home/user1$ chown hello user2

user1:/home/user1$ ls -l
-rw-r--           user2      0     Wed May 29 21:40:57 2024     hello

user1:/home/user1$ cd ..

user1:/home$ w hello 10 helloworld
Error: Permission denied

user1:/home$ mk world
Error: Permission denied

user1:/home$ cd user2
Error: Permission denied

user1:/home$ chmod user1 77

user1:/home$ ls -l
-rw-r--            root      0     Wed May 29 21:37:16 2024     hello
drwxrwx            root     17     Wed May 29 21:37:15 2024     public
drwxrwx           user1     17     Wed May 29 21:37:15 2024     user1
drwx---           user2     17     Wed May 29 21:37:16 2024     user2

user1:/home$ su user2
Enter password:

user2:/home/user2$ cd ../user1

user2:/home/user1$ mk world

user2:/home/user1$ w hello 10 helloworld

user2:/home/user1$ ls -l
-rw-r--           user2     10     Wed May 29 21:42:24 2024     hello
-rw-r--           user2      0     Wed May 29 21:42:24 2024     world

user2:/home/user1$ cat hello
helloworld

user2:/home/user1$ e
```

3.3. `cmd3.txt` : Error Handling

We have experienced some error messages like premission denied in the previous tests. This file contains more cases that may cause errors for the user and permission control system.

```
mk hello                # root: create a file in home
f
useradd root
userdel root            # Try to delete root
userdel notExist        # Try to delete a non-exist user
useradd user1
useradd user2
su user1
123                     # Wrong password
su user1

cd ../public
mk test
chown test notExist     # Change owner to a non-exist user
chmod test 88           # Invalid permission
chown test user2
chmod test 77           # Modify other user's file
rm test                 # Remove without permission
e
```

The results should be as follows:

```
root:/home$ f

root:/home$ useradd root
Error: Invalid user

root:/home$ userdel root
Error: Permission denied

root:/home$ userdel notExist
Error: Invalid user

root:/home$ useradd user1

root:/home$ useradd user2

root:/home$ su user1
Enter password: 123
Error: Password mismatch

root:/home$ su user1
Enter password:

user1:/home/user1$ cd ../public

user1:/home/public$ mk test

user1:/home/public$ chown test notExist
Error: Invalid user

user1:/home/public$ chmod test 88
Error: Invalid argument

user1:/home/public$ chown test user2

user1:/home/public$ chmod test 77
Error: Permission denied

user1:/home/public$ rm test
Error: Permission denied
```

```
user1:/home/public$ e
```

3.4. Consistency Check

The File System should be able to handle multiple users correctly.

When multiple clients are connected to the server, the file system should be able to handle the requests from different users correctly.

However, our integrated automated verification tool does not support sending commands from multiple clients at the same time. You can run the following commands in two different terminals to test the consistency of the file system:

- **Format when other users are connected:**

```
# Terminal 1:
# ./FC localhost 2560 root
f
useradd user1
```

```
# Terminal 2:


# ./FC localhost 2560 user1
f
```

The results should be as follows, the first client should be disconnected when formatting:

```
root:/home$ f
root:/home$ useradd user1
root:/home$ Warning: Disk formatted
Server disconnected
```

```


user1:/home/user1$ f
root:/home$
```

- **User operations when other users are connected:**

```
# Terminal 1:
# ./FC localhost 2560 root
f
useradd user1

userdel user1
```

```
# Terminal 2:



# ./FC localhost 2560 user1
```

User1 should be disconnected when deleted by the root user:

```
root:/home$ f
root:/home$ useradd user1
root:/home$ userdel user1
root:/home$
```

```


user1:/home/user1$
Warning: User deleted
Server disconnected
```

- **File operations when other users are connected:**

```
# Terminal 1:
# ./FC localhost 2560 root
f
useradd user1
cd public
mk test


ls

rmdir hello
```

```
# Terminal 2:


# ./FC localhost 2560 user1
cd ../public

ls
mkdir hello

cd hello

ls
```

They should be able to see the files created by the other user, also the file system should handle the case when the current path is removed:

```
root:/home$ f
root:/home$ useradd user1
root:/home$ cd public
root:/home/public$ mk test




root:/home/public$ ls
hello  test

root:/home/public$ rmdir hello



root:/home/public$
```

```



user1:/home/user1$ cd ../public

user1:/home/public$ ls
test
user1:/home/public$ mkdir hello

user1:/home/public$ cd hello

user1:/home/public/hello$ ls
Warning: Current path no longer exist
user1:/home$
```