

**课程设计报告**

**学生成绩管理系统**

|  |  |
| --- | --- |
| **课程名称：** | **操作系统** |
| **学生姓名：** | **杨家瑞、陈硕、李晟旸、张驰** |
| **学生学号：** | **202130381015** |
| **学生专业：** | **数据科学与大数据技术** |
| **开课学期：** | **2023-2024学年第1学期** |

**未来技术学院**

**2023年10月**

目录

[1 Experimental purpose 2](#_Toc155546334)

[2 Experimental design ideas and processes 4](#_Toc155546335)

[2.1 General Idea 4](#_Toc155546336)

[2.2 Socket 5](#_Toc155546337)

[2.2.1 Class TcpServer 6](#_Toc155546338)

[2.2.2 Client-Server Connection 7](#_Toc155546339)

[2.2.3 File Transfer 8](#_Toc155546340)

[2.3 Server 9](#_Toc155546341)

[2.3.1 Server Initialization 9](#_Toc155546342)

[2.3.2 Server Login 10](#_Toc155546343)

[2.3.3 Server’s Workflow 10](#_Toc155546344)

[2.3.4 Server’s Function 11](#_Toc155546345)

[2.3.5 Mutex Lock 13](#_Toc155546346)

[2.3.6 Server’s Child Thread killed 14](#_Toc155546347)

[2.3.7 Server Shutdown 14](#_Toc155546348)

[2.4 Client 15](#_Toc155546349)

[2.4.1 Client Initialization 15](#_Toc155546350)

[2.4.2 Client Login 15](#_Toc155546351)

[2.4.3 Three Kinds of Clients 16](#_Toc155546352)

[2.4.4 Client’s Workflow 17](#_Toc155546353)

[2.4.5 Client’s Function 18](#_Toc155546354)

[2.4.6 Client Shutdown 18](#_Toc155546355)

[2.5 OS 19](#_Toc155546356)

[2.5.1 Inode design 19](#_Toc155546357)

[2.5.2 block design 20](#_Toc155546358)

[2.5.3 File Structure in a File System 22](#_Toc155546359)

[2.5.4 Functions of the File system 22](#_Toc155546360)

[2.6 Extra Fuction 36](#_Toc155546361)

[3 Instruction of experiment operation 45](#_Toc155546362)

[3.1 Project Compilation 46](#_Toc155546363)

[3.2 Project Operation 48](#_Toc155546364)

[4 Experimental results show 50](#_Toc155546365)

[5 Team member contribution description 56](#_Toc155546366)

## 1 Experimental purpose

This experiment is to create a student score management system, the system adopts server-client architecture, that is, data is stored on the server, users access through the client, and multiple users are allowed to access the system, that is, multiple terminals can access. Among them, we need to simulate the basic file management and process communication management of the operating system, use the I-Node file system to simulate disk storage, and according to the actual and thinking, complete the creation of more functions, and according to the content of this experiment, we want to achieve the purpose of this experiment as follows:

The first is to understand the practical application of the operating system concept, through the development of student achievement management system, we will deeply understand the core concepts of the operating system, such as file management, process communication, mutual exclusion and so on. This allows us to apply the theoretical knowledge learned in the classroom to real projects, and to better understand how simple operating systems are stored and how communication is achieved on the front and back ends.

The second is to exercise teamwork and communication skills, to complete this big assignment in a group form, which will exercise our ability to coordinate and communicate in a team. Learning how to divide this operating system segment and how to assign tasks is also very helpful for future cooperation after employment, and by solving problems and cooperating with the team members to develop the system, we will learn how to effectively cooperate with others to write code to achieve a common goal.

In addition, we can have a deep understanding of the server-client architecture, and the big job requires us to develop an application based on the server-client architecture, which will give us a deep understanding of the basic principles of distributed system design. We will learn how to handle multi-user concurrent access to ensure system stability and performance.

There is also to strengthen programming skills, to create users, folders, files and other basic operations, not only requires us to master the principle of the operating system, but also need to flexibly use programming skills, improve our use of c++ language programming proficiency. This will also consolidate our coding and debugging capabilities and develop the ability to solve real problems.

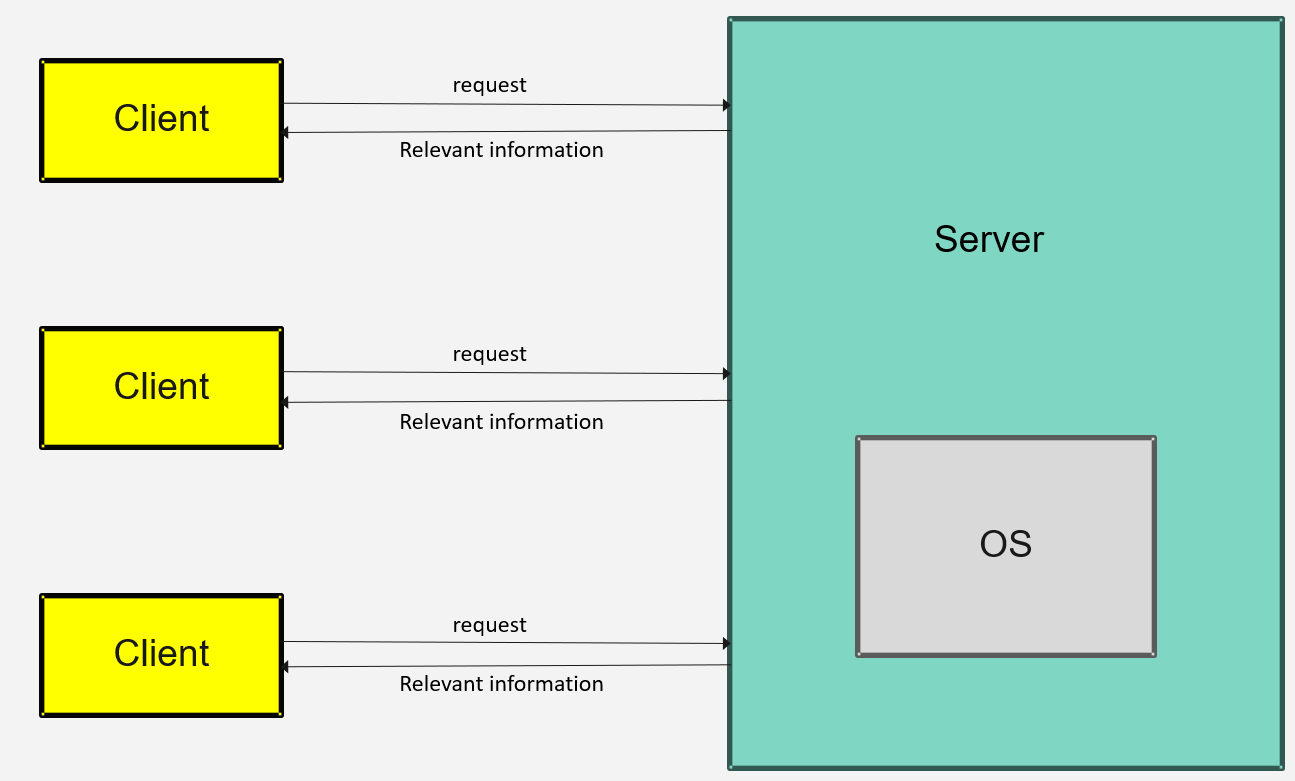
In addition to the basic functions, we have the opportunity to implement additional functions, such as backup system, recovery system, file snapshot, etc. This will stimulate our innovative thinking about system design and improve our independent thinking ability to solve problems.

The last is to face the challenges of the actual scenario: the experiment may involve challenges such as excessive file size and process communication. By facing the challenges of these practical scenarios, we will improve our practical ability to solve problems and prepare for future career development. In the future, if you are engaged in software development, the experience of this experiment can also provide a lot of help.

## Experimental design ideas and processes

### General Idea

In this experiment, we strictly follow the requirements of the document, and at the same time, we add the functions and concepts that should be in the operating system and expand the functions to the student score management system, and design it as a mature management system. In the following figure, we can see our overall design train of thought, will test a total divided into four parts, respectively is the Client, Server, OS, communicate four parts of the plate, the Client as a Client, is the front end, need to own request, input command in the command line, Then responsible for communicating part will use the socket socket interface to complete the communication between processes, pass the requirements to the Server side, that is, the back-end, the back-end will call the OS file system interface, to achieve their own functions, for example, we in the release of the job function, we receive information after receiving, Will call the OS file system to check whether the path exists this interface, if it does, will continue to call the view permissions, create directories, create files and other interfaces, thus forming a successful function, similarly, when we need to send information from the Server side, We can store the basic storage unit block in the file system to extract the text information into the string, and then send the socket socket interface to the front end to show the user, so as to complete the implementation of the overall file system.



The following we will introduce how the design of each part is composed, and it is divided into Socket, Server, Client, OS, Extra function five plates to expand.

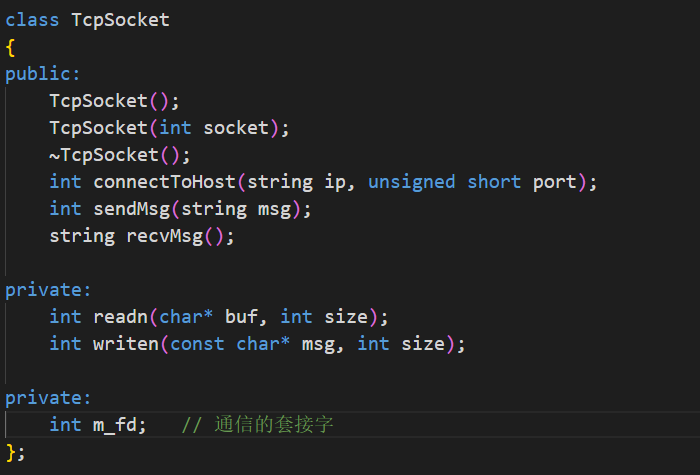
## 2.2 **Socket**

The server and the client use Socket to communicate with each other. Two classes TcpSocket and TcpServer need to be implemented first.

#### Class TcpSocket

The TcpSocket class is a socket communication class that can be used both on the client side and on the server side. Its duty is to receive and send packets.

The following figure shows the class definition.



The functions in the class are as follows：

* TcpSokcet(): Create socket for communication.
* int connectToHost(): Connect to a certain IP and port.
* int sendMsg(): Send message.
* string recvMsg(): Receive message.

### Class TcpServer

The TcpServer class is mainly used for the server side of socket communication, and there is no communication ability. When the new connection between the server and the client is established, the communication descriptor needs to be wrapped into a communication object through the parameter construction of the TcpSocket class, so that the object can be used to communicate with the client.

The following figure shows the class definition.



The functions in the class are as follows：

* TcpServer(): Create socket for listening.
* int setListen(): Setting the listening port
* int acceptConn(): Accept connection from the clients.

### Client-Server Connection

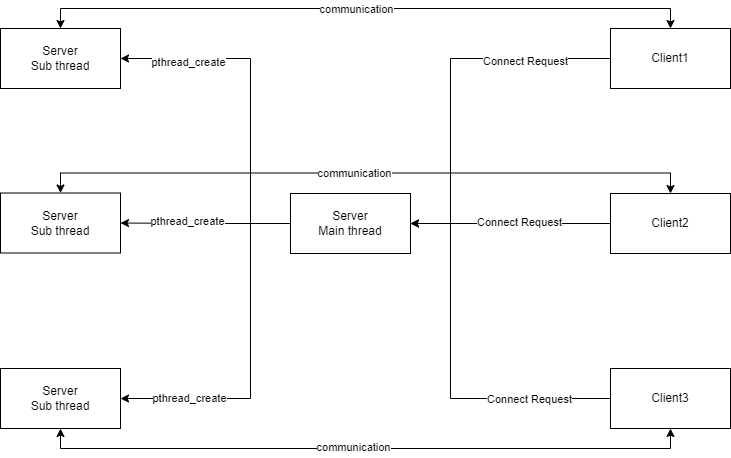
The server workflow is as follows:

1. Create a listening socket.
2. Bind local IP and port and set up listening.
3. Block and wait for client connections.
4. Accept connections from clients and create child threads to communicate with them.

The clinet workflow is as follows:

1. Create a socket for communication.
2. Connect to server’s IP and port.

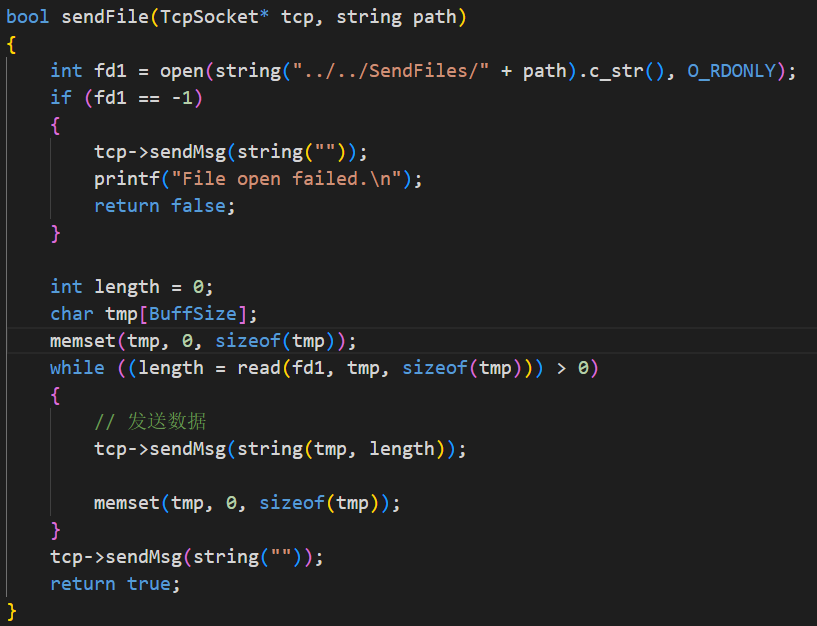
The following flowchart shows the connection between the client and server.



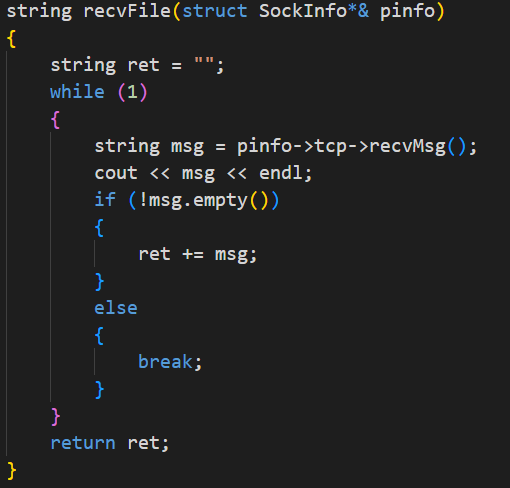
### File Transfer

Two functions for transferring files are implemented.

* Send File



* Receive File



## Server

The server manages the platform using the interfaces provided by the operating system, and also uses sockets to communicate with the clients and provide them with the appropriate interfaces.

### Server Initialization

The server is initialized as follows:

1. Instantiate an object of the OperatingSystem class and writes operating system files to memory.



1. Initialize mutual exclusion locks.



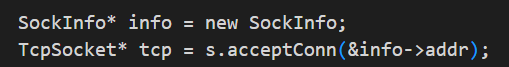
1. Create a listening socket.



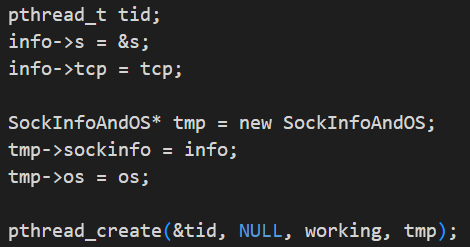
1. Bind to a local IP and port and set up listening.



1. Blocking and wait for client connections.



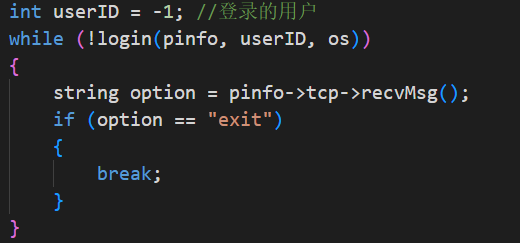
1. Create child threads to communicate with the client.



### Server Login

After the server connects successfully with the client, it prints its IP address and port of the connection. Then the server receives the username and password from the client and try to log in the student management system.

* If the login succeeds, the login user is recorded.
* If the login fails, the user is required to log in again.

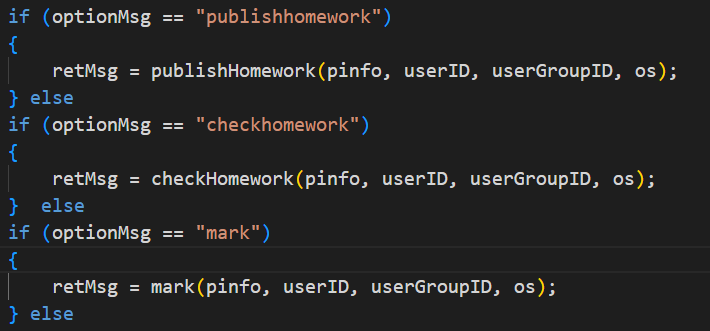


### Server’s Workflow

After successful login, the server child thread is blocked and receives instructions from the client.



Then according to different instructions, different functions are called as shown in the figure below.



Finally, the information generated by the server is transmitted to the client.



### Server’s Function

All server functions are shown below:

* string publishHomework(): Assign an assignment and return the status.
* string checkHomework(): Check student’s homework and return it to the client.
* string mark(): Mark for student’s homework and return the status.
* string writeComment(): Write comment for student’s homework and return the status.
* string checkScoreAndComment(): Check the mark and comment of student’s homework and return it to the client.
* string checkAssignment(): Check the statement of an assignment and return it to the client.
* string submitHomework(): Submit homework to an assignment and return the status.
* string deleteHomework(): Delete an assignment and return the status.
* string listCourse(): Check all courses that a certain user can access to and return it to the client.
* string listHomework(): Check all the assignment of a certain course and return it to the client.
* string createCourse(): Create a course and return the status.
* string deleteCourse(): Delete a course and return the status.
* string addToCourse(): Add a certain user to a course and return the status.
* string listCourseUser(): Check the teacher and all the student of a certain course and return it to the client.
* string listGroupUser(): Check all the users of a certain group and return it to the client.
* string createuser(): Create an user and return the status.
* string deleteuser(): Delete an user and return the status.
* string modifyPassword(): Modify a certain user’s password and return the status.
* string backup(): Backup the system and return the status.
* string recover(): Recover the system and return the status.
* string revoke(): Rollback the system and return the status.
* string clearsnapshot(): Initialize the snapshot file and the log file and return the status.
* string checkLog(): Check the log file and return it to the client.

### Mutex Lock

The system is designed with reference to the readers and writers problem models.

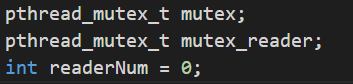
The following roles must be followed in this model:

* Multiple readers can read the data simultaneously.
* Only one writer can write the data at any time.
* A reader and a writer cannot in critical section together.

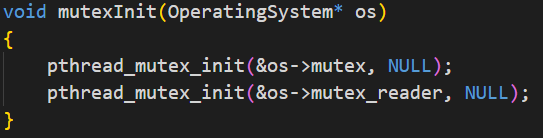
The following solutions are used:

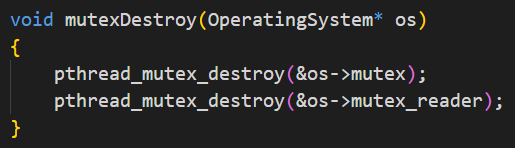
* Suppose new reader come: If writer is waiting and some reader is reading, then read ok.
* Suppose new writer come: If no reader or writer, it can write, else wait.

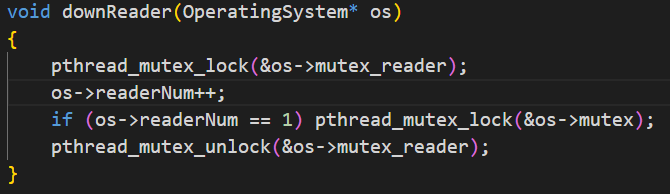
Two semaphores and a counter for reader need to be prepared.

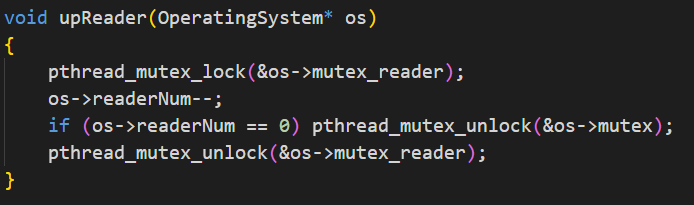


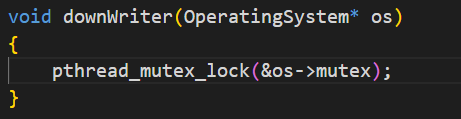
The related functions are shown in the following graphs.

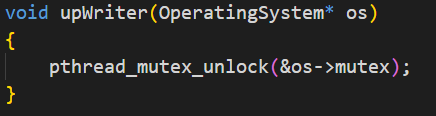






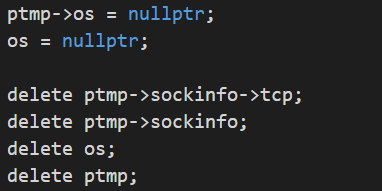






### Server’s Child Thread killed

When the client disconnects, the server child thread needs to be destroyed and the associated resources needs to be released.





### Server Shutdown

When the server shutdown, mutual exclusion locks need to be destroyed and resources need to be released.



## Client

The client interacts with the user, transmits user instructions to the server, receives information from the server and outputs it to the user's terminal.

### Client Initialization

The client is initialized as follows:

1. Create socket for communication.



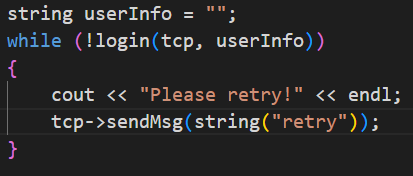
1. Connect to the server.



### Client Login

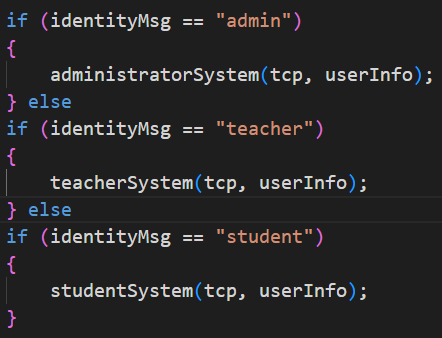
The client transmits the username and password entered by the user to the server. The server attempts to log in to the system and transmits the login information to the client.

* If the login succeeds, the login user is recorded.
* If the login fails, the user is required to log in again.



### Three Kinds of Clients

After a successful login, the client will launch the appropriate client based on the user's identity.



There are three clients in total:

* Administrator Client:



* Teacher Client:

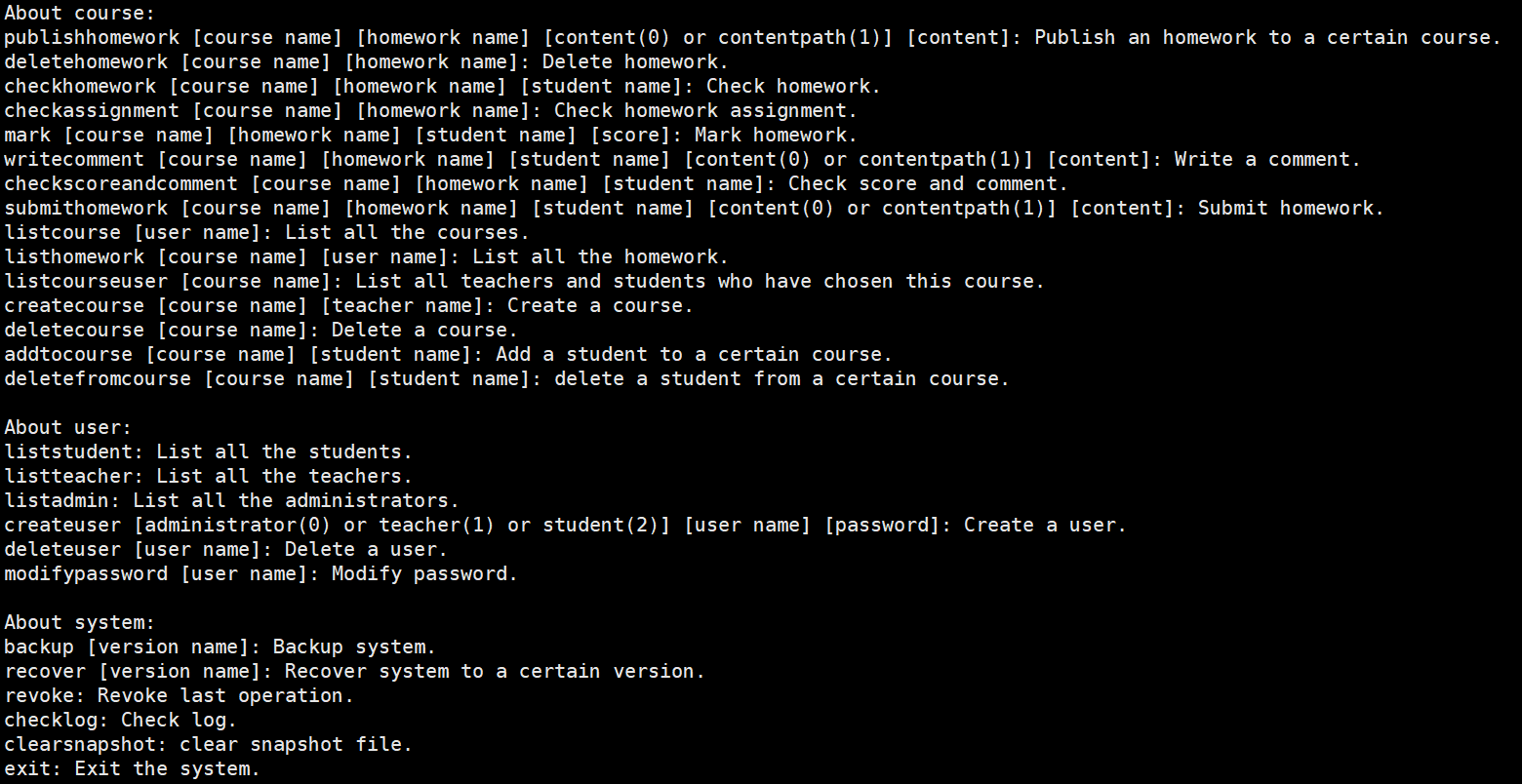


* Student Client:

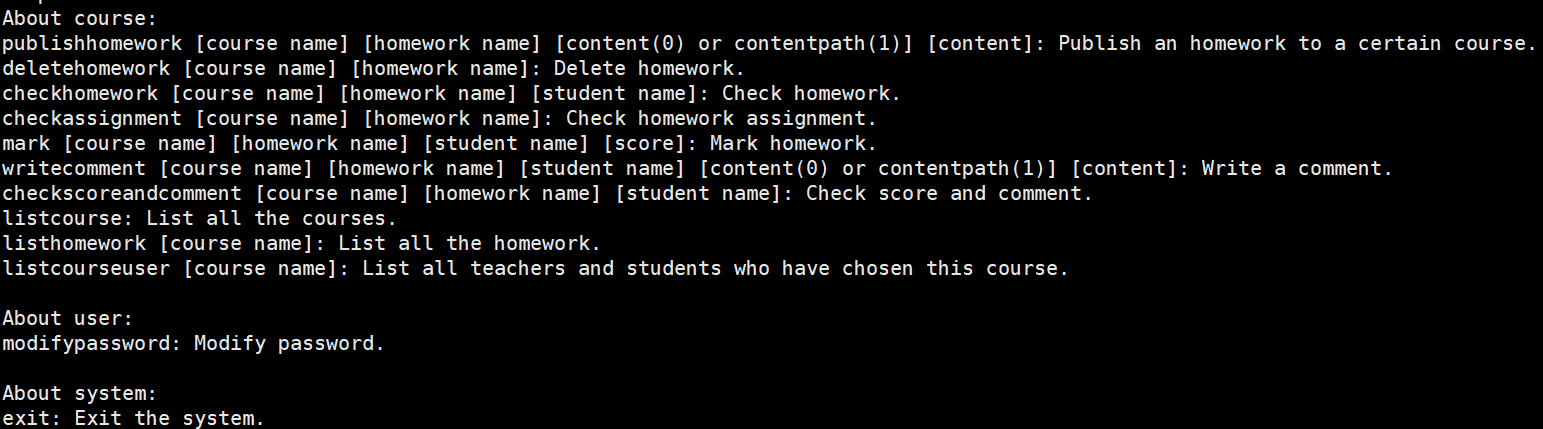


Their functions are not the same as each other.

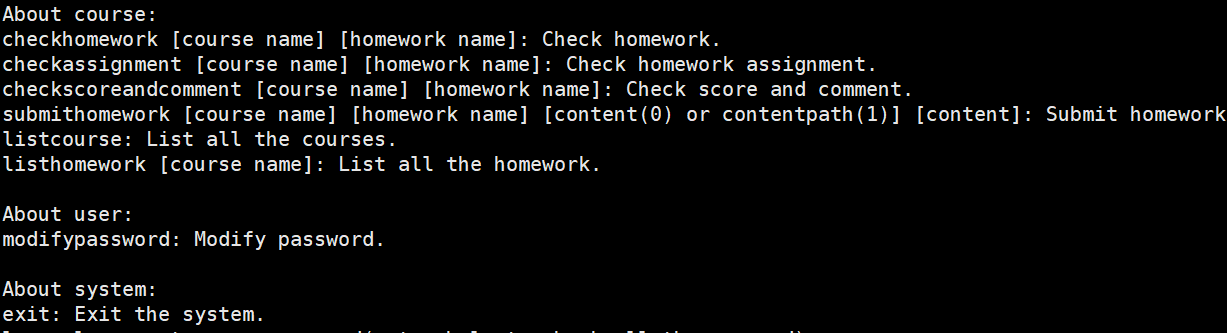
* Administrator Client:



* Teacher Client:



* Student Client:

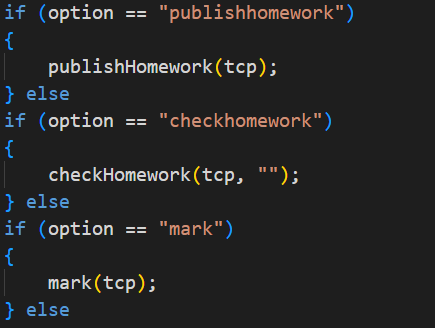


### Client’s Workflow

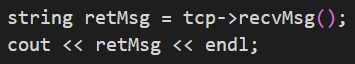
User input instructions to the client.



Then according to different instructions, different functions are called as shown in the figure below.



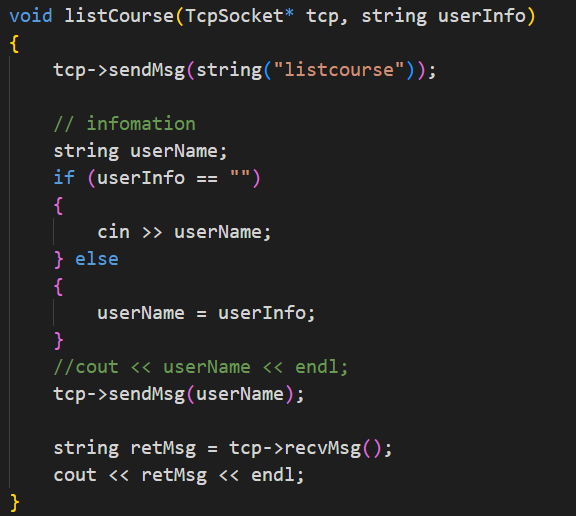
The client receives the information sent by the server and prints it to the terminal.



### Client’s Function

Client-side functions correspond to the server. Its main role is to transmit instructions and the related information to the server.

For example, function listCourse() is shown below.



### Client Shutdown

When the client shutdown, the connection will be broken and the socket must be released.



## OS

I-Node file system implementation roadmap (excluding snapshots and extensions)

Our file system is stored by dividing the storage space into blocks, and each file has its own corresponding inode number, which contains the meta information of the file. We design the file system as a class to facilitate subsequent server side calls.

### Inode design

To introduce inodes, start with file storage. Files are stored on a hard disk, and the smallest unit of storage on a hard disk is called a Sector. Each sector stores 512 bytes (equivalent to 0.5KB). When the operating system reads the hard disk, it does not read sectors one by one, which is too inefficient, but continuously reads multiple sectors at a time, that is, read a "block" at a time. This "block" consisting of multiple sectors is the smallest unit of file access. The size of a block, most commonly 4KB, is a block made up of eight consecutive sectors. The file data is stored in "blocks", so obviously we must also find a place to store the file's meta-information, such as the creator of the file, the date it was created, the size of the file, and so on. This area where meta-information is stored is called the inode, which translates to "index node" in Chinese. Each file has its own inode, which contains information about the file.

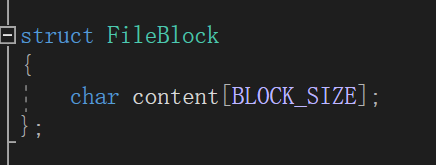
In our file system, the inode section, we contain the following file meta information:

* InodeNumber:inode number，Tag each inode
* Filetype: 0 represents directory, 1 represents normal file, 2 represents indirect block
* blockID,userID,groupID：Tag the owning group, block, and user
* ownerPermission,groupPermission,otherPermission:Assign authority

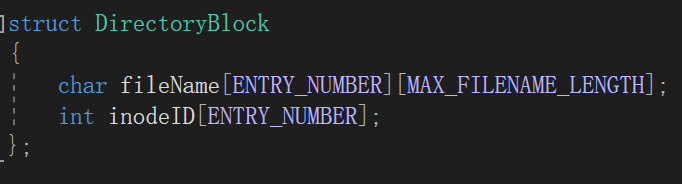
### block design

Our block design consists of five types of blocks:：

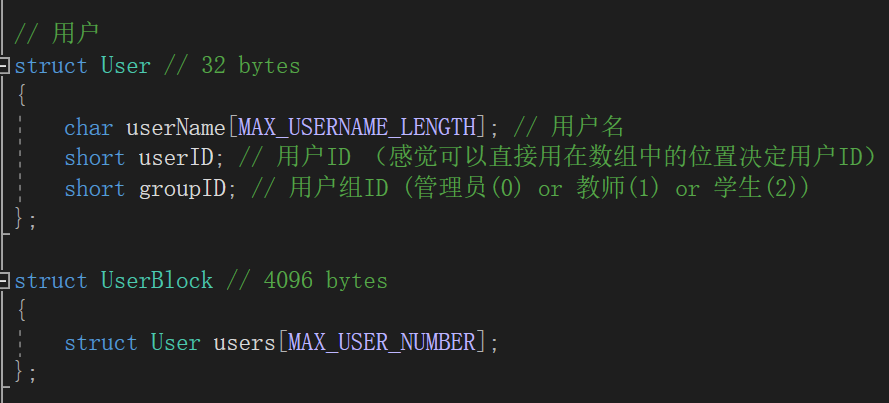
* FileBlock:Records the content of a file.



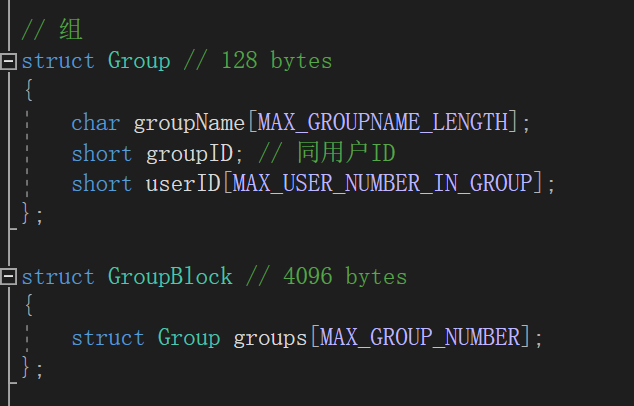
* DirectoryBlock:file names and corresponding inode IDs inside a directory.



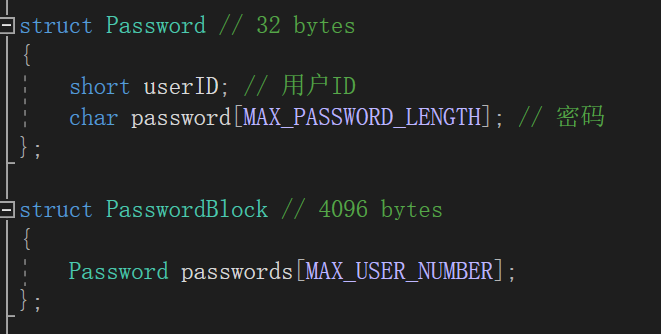
* UserBlock:Records user information such as username, inode ID, and group ID.



* GroupBlock:Records information about a group, including group name, group ID, and user IDs within the group.

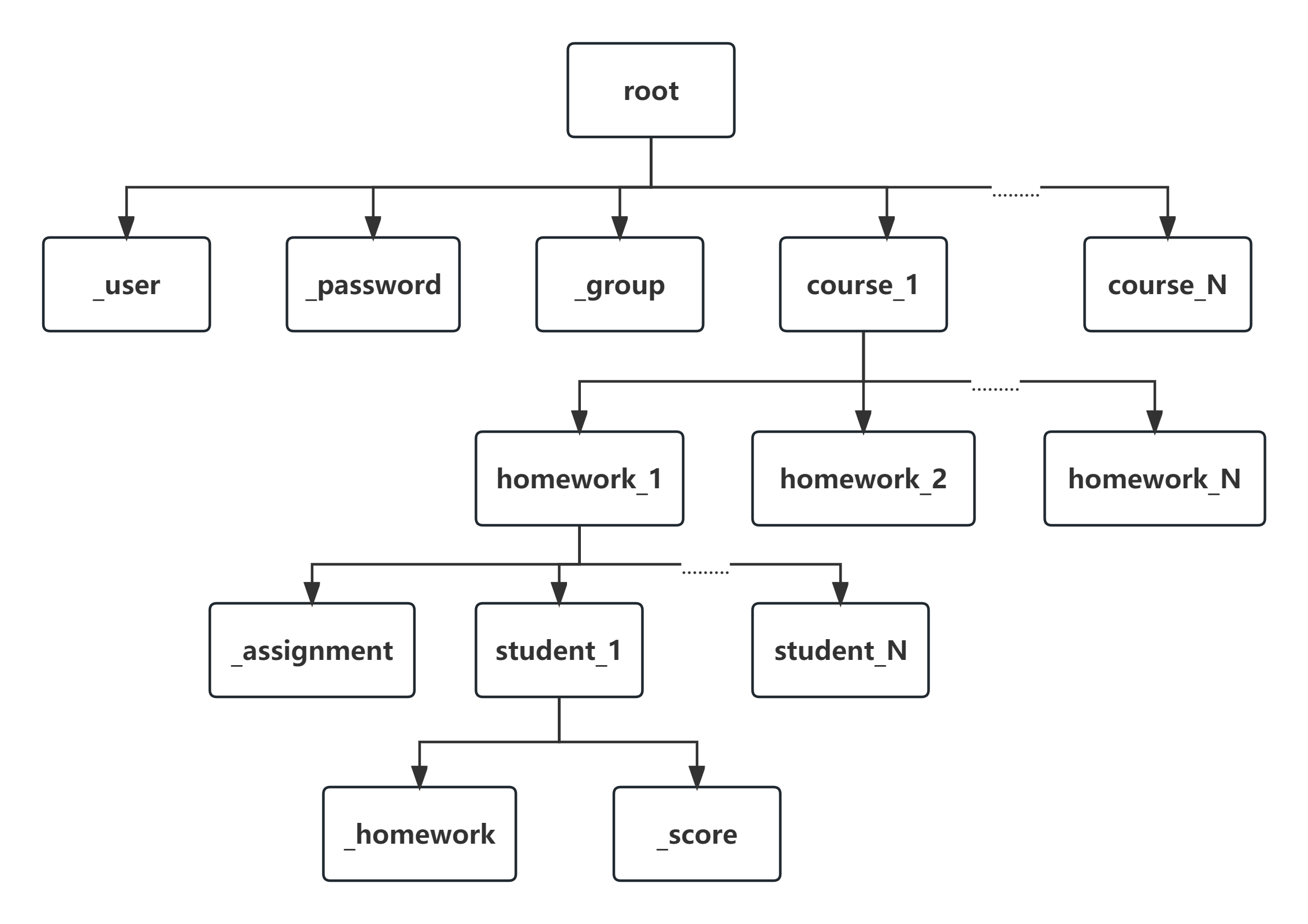


* PasswordBlock:Records user passwords.



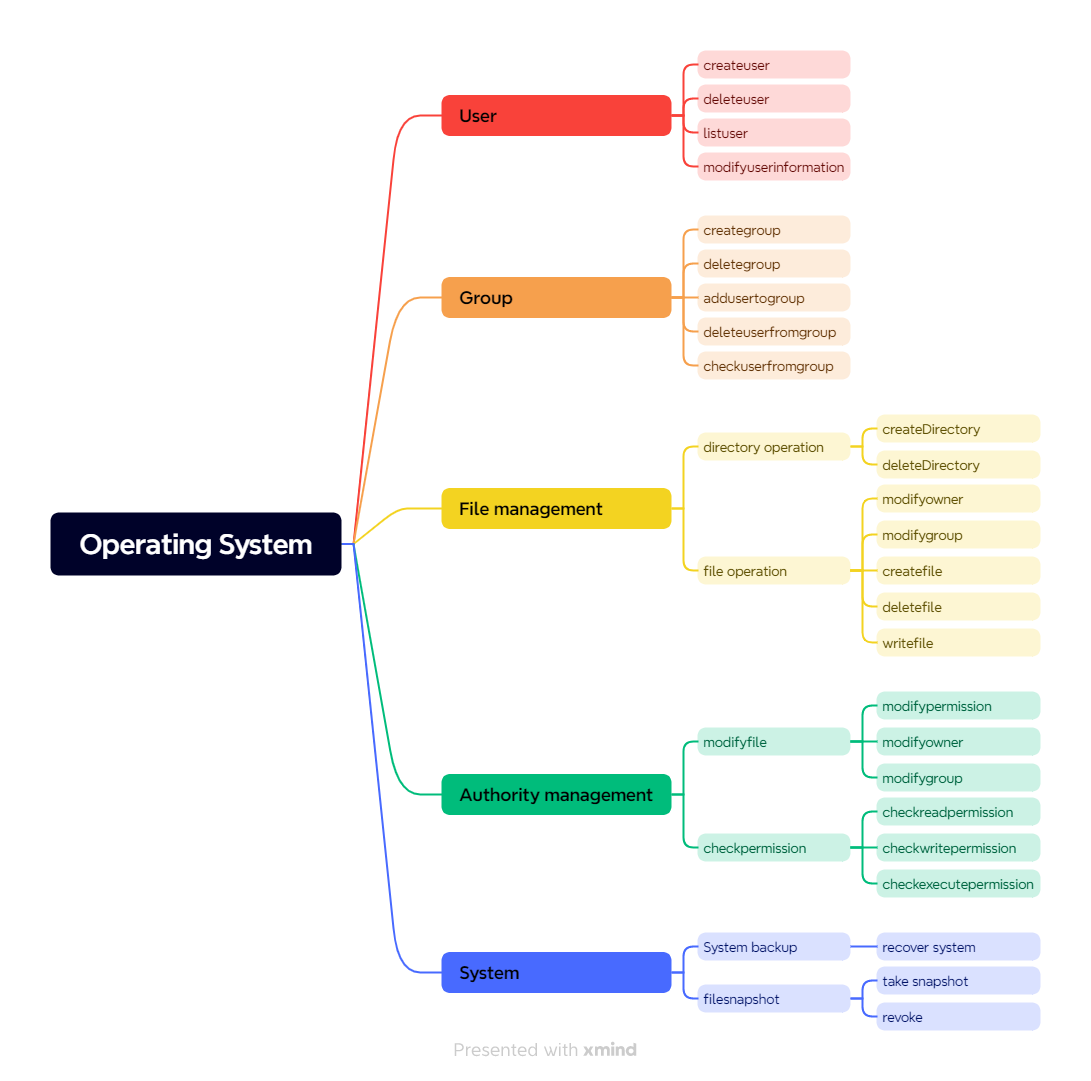
### File Structure in a File System

In the root directory, we store the basic user and group information of the file system in \_user, \_password,\_group. At the same time, each course corresponds to a directory block, which stores the relevant information contained in the course. A course folder contains multiple homework folders, which contain the specific content of the homework, the homework submission of each student and the score.



### Functions of the File system

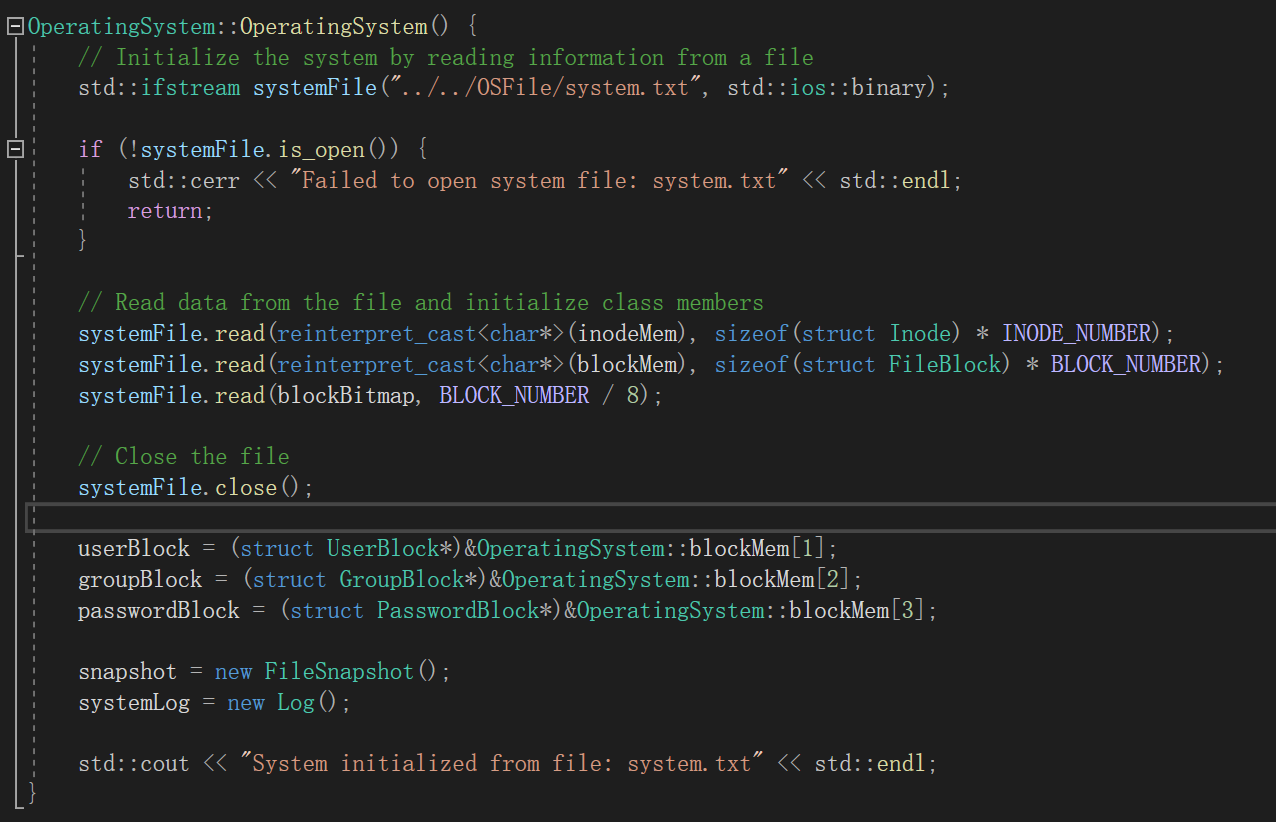
In general, the functions of our file system include: user, group, file management, permission management, system backup and snapshot five functions.



#### File system initialization - constructors and destructors

* Constructors

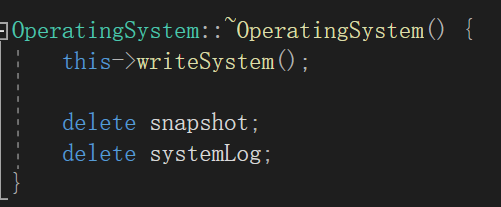
The system initializes users, groups, directories, passwords, and files in the file system by reading the contents of system.txt.



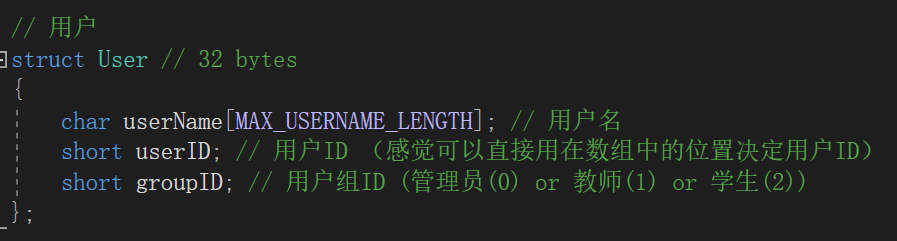
* Destructors

In the destructor, the effect of the operation on the system is written to the system.txt file.





#### **User-Related Operations**



In the user setting, there are three types of users: administrators, teachers and students. This information is stored as a primary group of users. Users also have their own user name and ID.

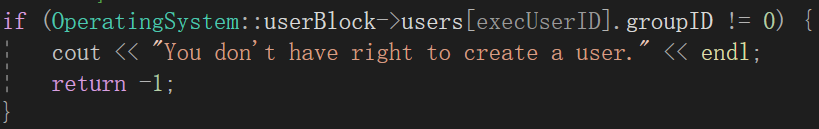
User operations include creating, deleting, and displaying users, and obtaining and modifying user information (ID, Name, MainGroup, and Password). Here we focus on the specifics of creating and deleting functions:

* Creates and deletes functions

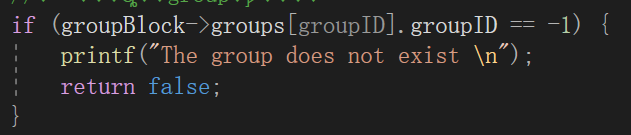
It mainly includes three parts: the rationality judgment of program execution, the search of empty position in userBlock and the insertion of information.

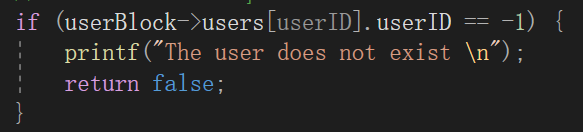
* + The judgment of the rationality of program execution includes the judgment of the current user permission, the judgment of whether the input of user information is reasonable, and the judgment of whether the user information is repeated.

Only administrators can create and delete.

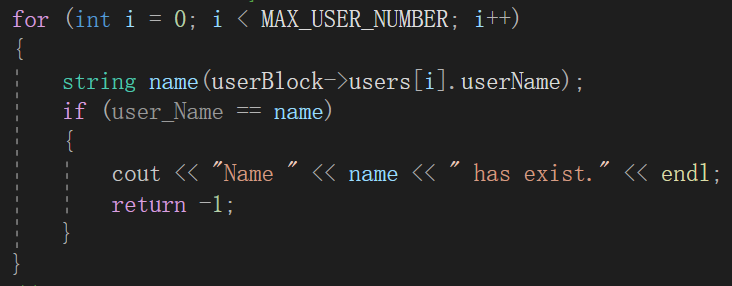


Make sure the information you enter is OK

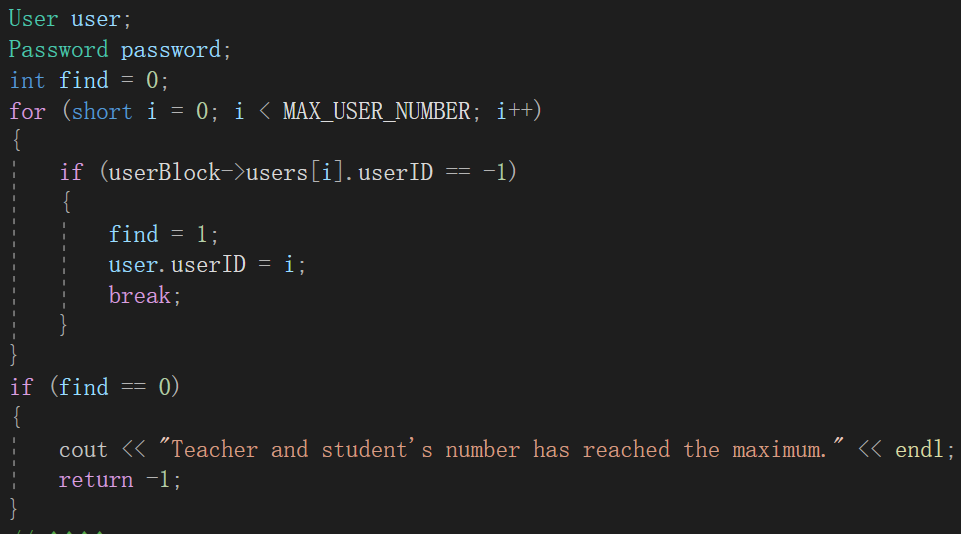




Ensure that user information does not duplicate (create)

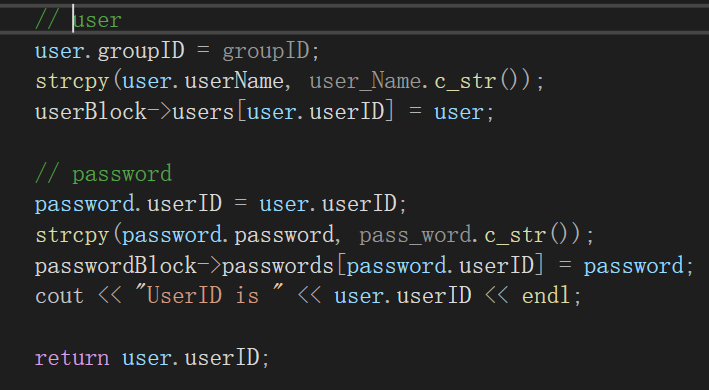


* + Find (create) free space in userBlock

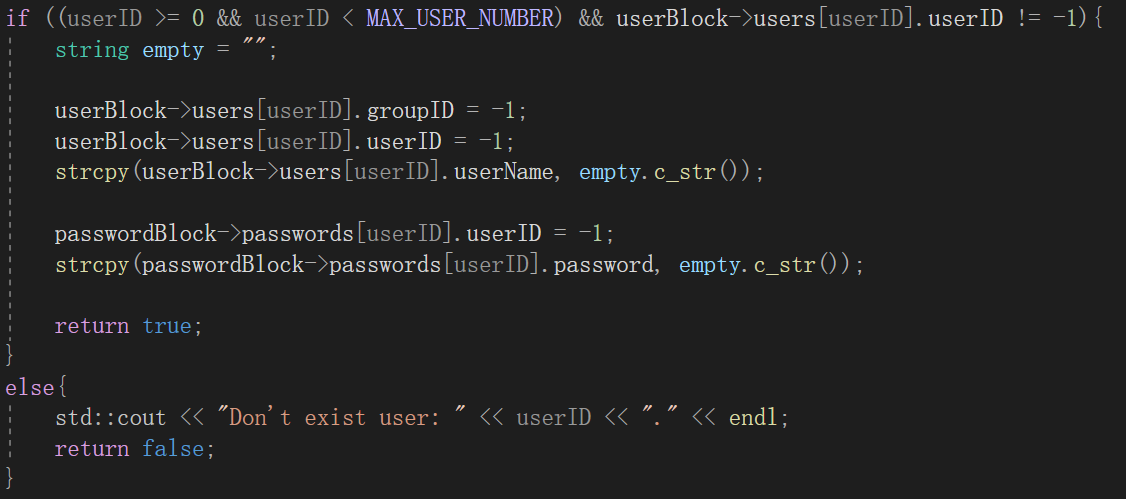


* + Insert and delete information

Insert



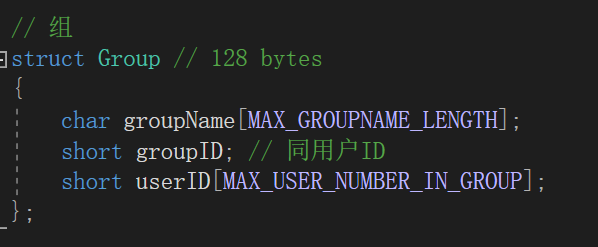
delete



* Other functions

It mainly includes two parts: one is permission determination, and the other is operation execution. You can see the code details in the source code file provided (OperaringSystem.cpp)

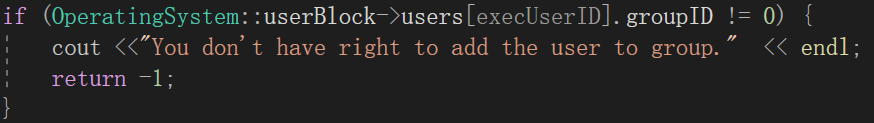
4.3 Group dependent operation



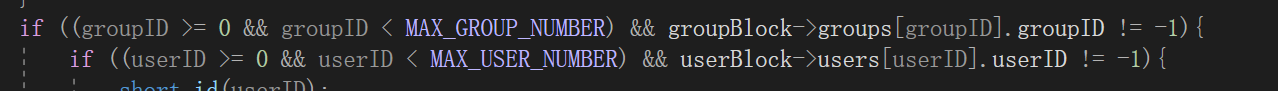
In addition to the group name and ID, the group definition contains an array of user ids. A user has multiple groups, including a primary group (administrators, teachers, students, must have) and multiple sub-groups (courses).

The operation of a group mainly includes the creation and deletion of a group, the query of information in a group, and the addition and deletion of users in a group. The creation and deletion of a group is similar to the creation and deletion of a user. This section focuses on adding and deleting users from a group.

* Add and remove users from the group
  + Permission determination

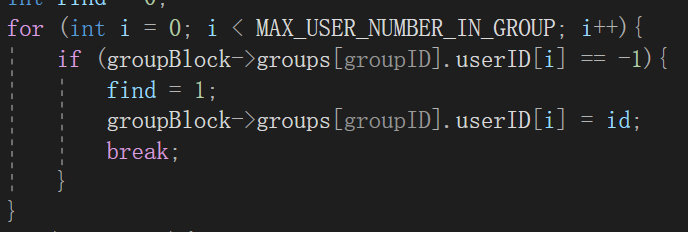


* + Check whether the input is reasonable

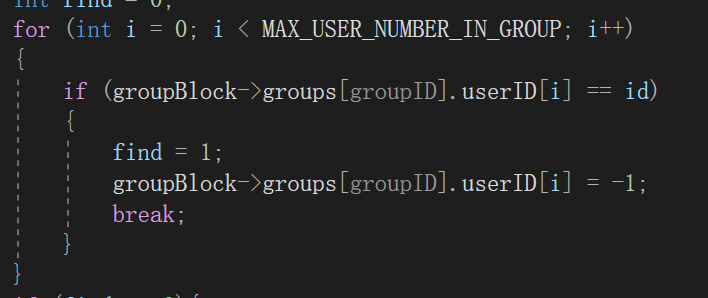


* + Perform the add or remove operation

Add



Delete



* Other functions

It mainly includes two parts: one is permission determination, and the other is operation execution. You can see the code details in the source code file provided (OperaringSystem.cpp).

#### **Operations Related to File Management**

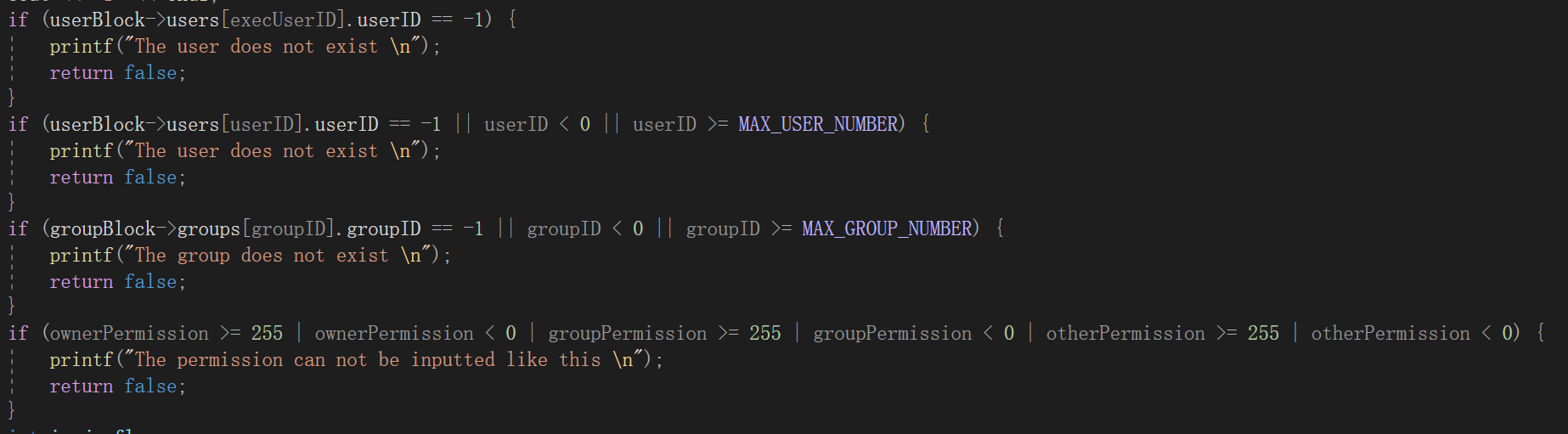
In file management, it mainly includes some basic operations of files and directories: creating and deleting files and folders, displaying files in folders, and reading and writing files. Here we focus on creating and deleting files and folders as well as reading and writing files.

* file and folder creation

The creation operation mainly includes determining permission, determining input rationality, searching for the parent directory, searching for the block and inode to be inserted, and creating files or directories

* + Enter the rationality judgment

Here you need to check whether the entered user, group, and permission are reasonable values.

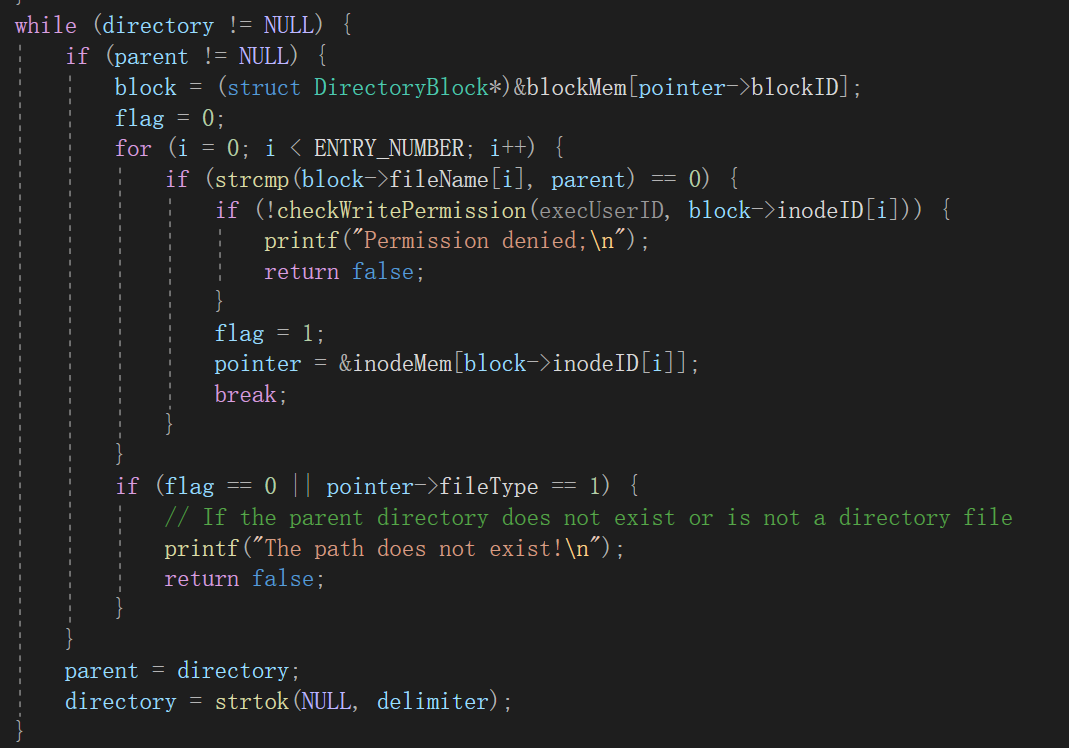


At the same time, when recursively finding the parent directory, check whether the path is reasonable.

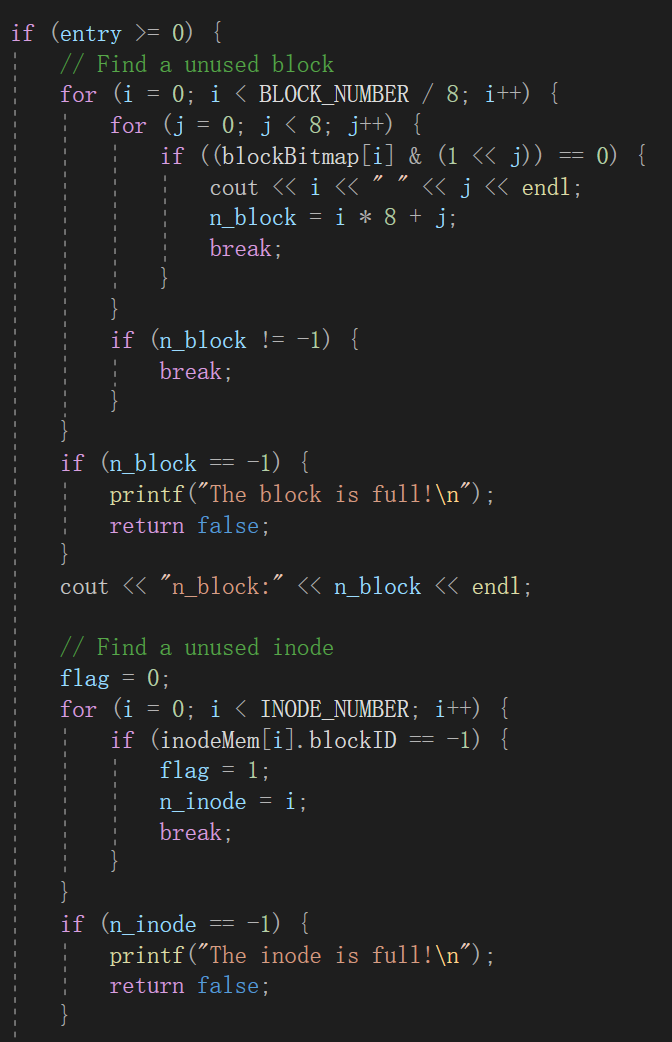


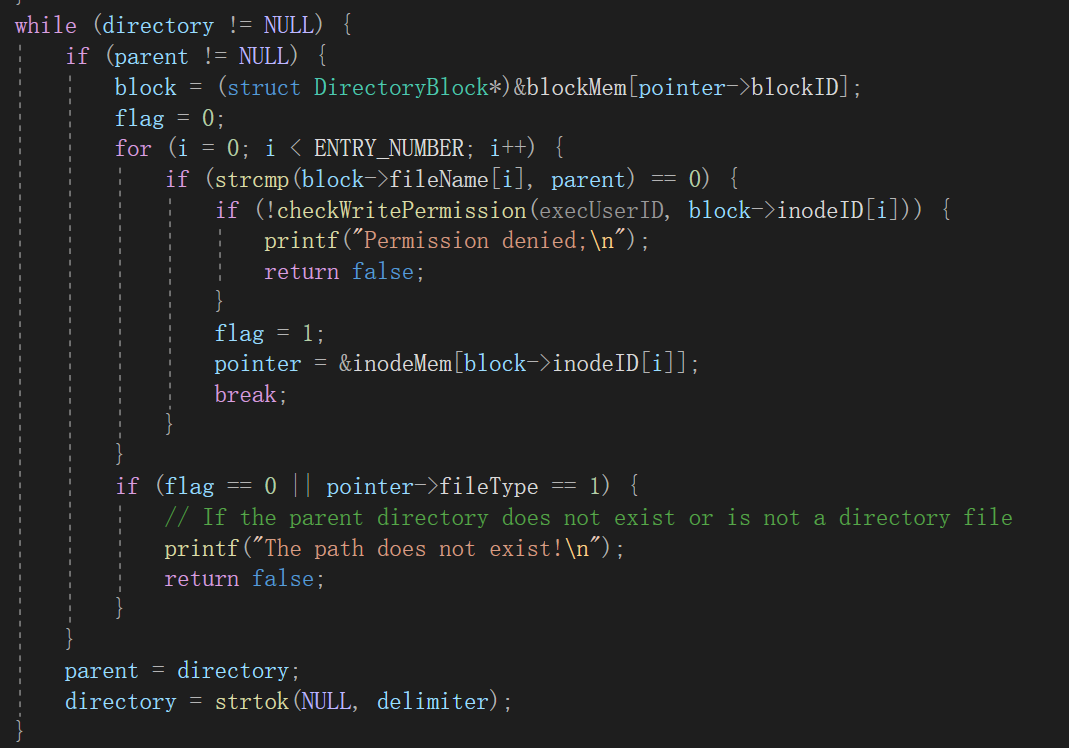
* + Permission determination

The permission determination here is different from the permission determination for user and group operations. This is based on the permission recorded in the inode file. Here, when we find recursion in the specified directory, we must first check whether there is write permission. (Refer to 4.5 for details)

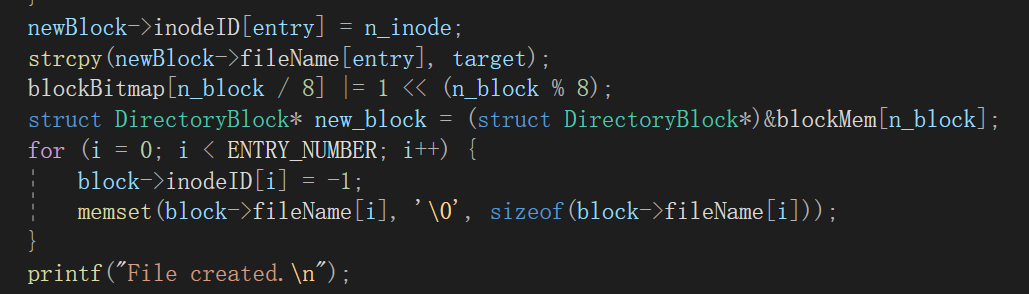


* + Look for inode numbers and block numbers to insert



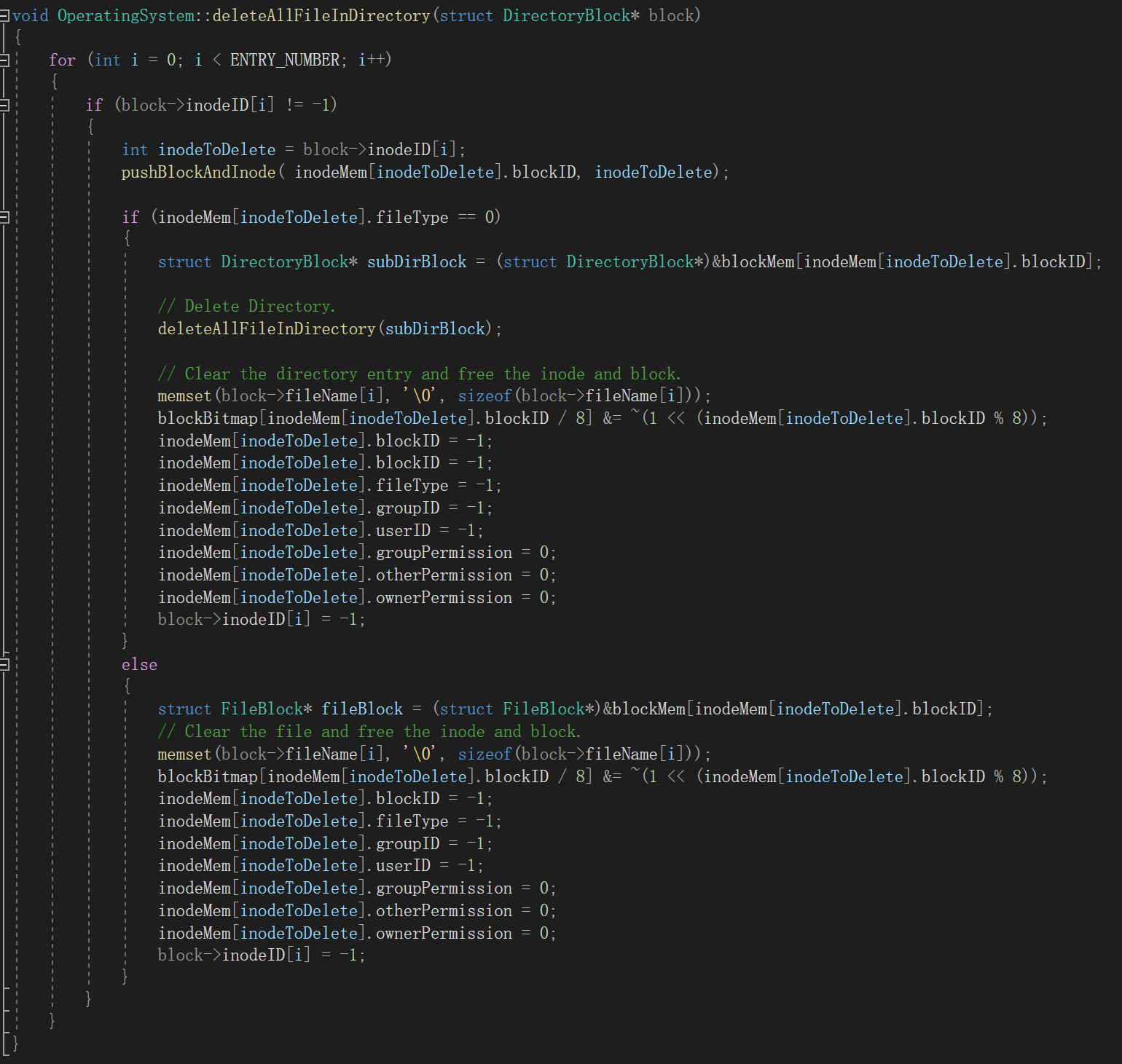


* + Creates a file or directory

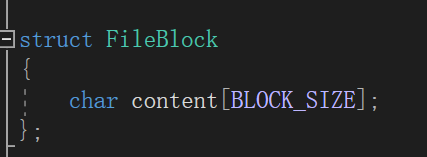


* Delete files and folders

It mainly includes the path search of deleted files and the recursive deletion of two operations. Here, when we delete, we restore the location of the inode and block to the original state. The following function can recursively delete all files in the folder/or this file.



* Read and write files

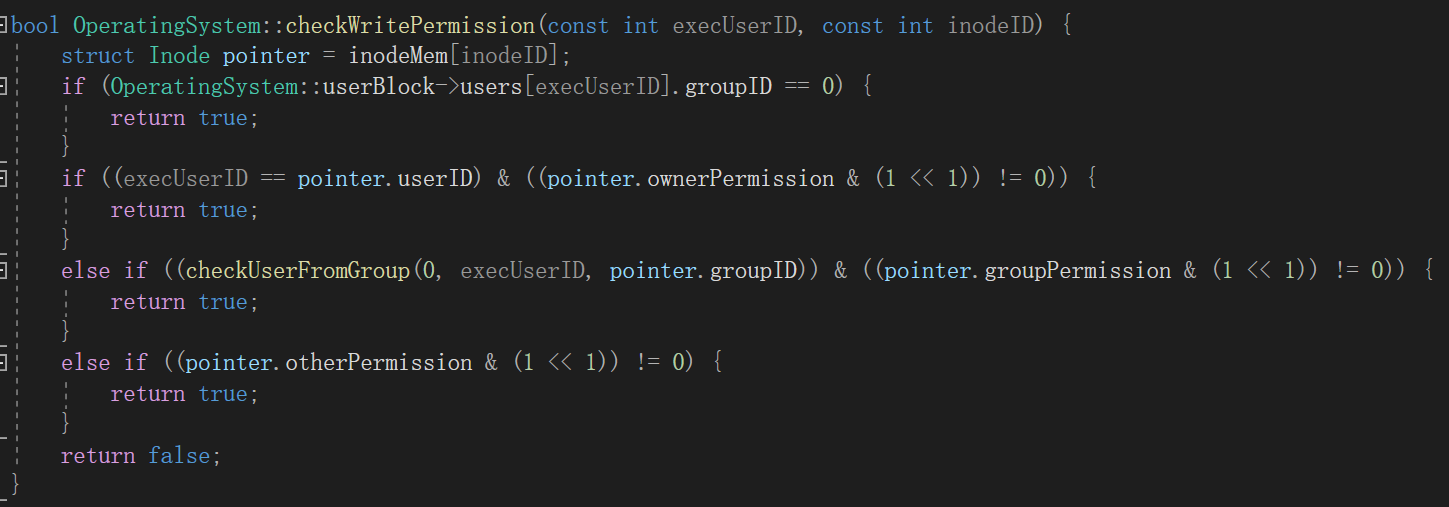


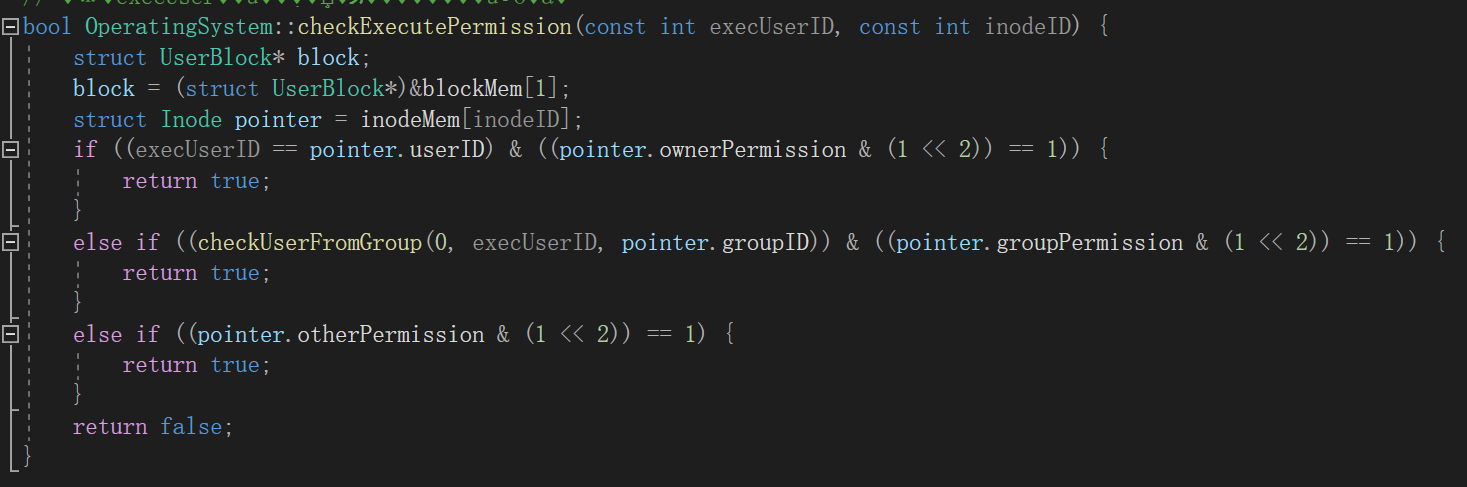
It mainly includes path search, rationality judgment, permission judgment, and content reading and writing. The file path search and rationality judgment are similar to the above operations. content read and write is used to read and write content in fileBlock.

#### **Rights-Related Operations**

If the operating user is the same as the file owner user, it can be executed. Other If the user is in the same group as the file, the groupPermission of the file is displayed. Look at the otherPermission of the file. Here a permission is a string, 0-2 characters record read and write execution of three kinds of permission

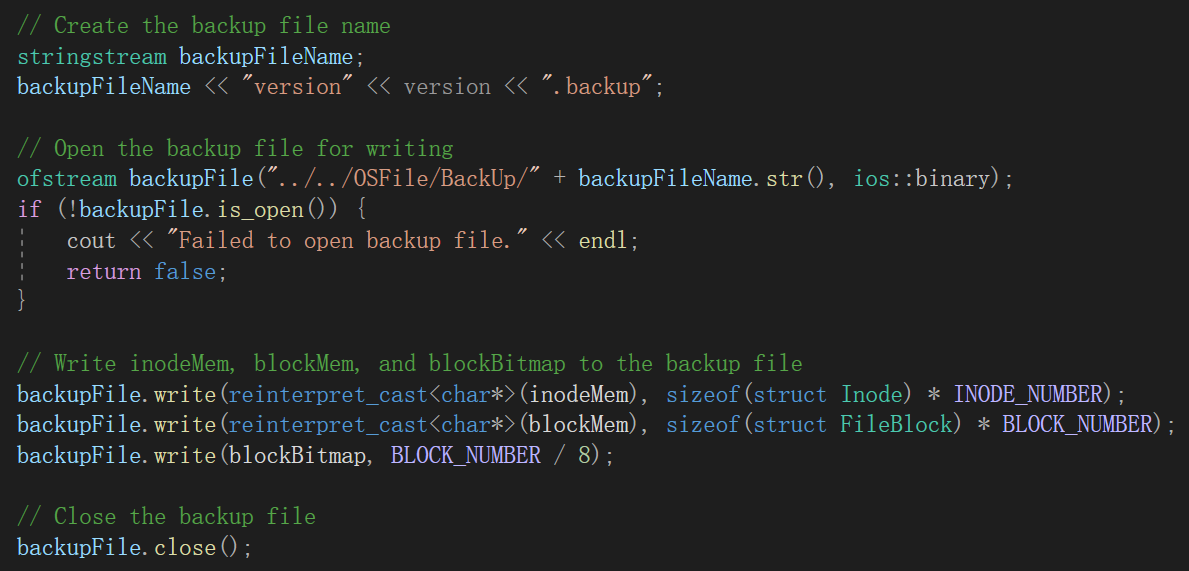






#### Backing Up and Restoring File Systems

Here the operation mainly includes permission determination and operation execution. The idea of backup and recovery is that we backup the block and inode in the file to a file, read the file content can be restored.



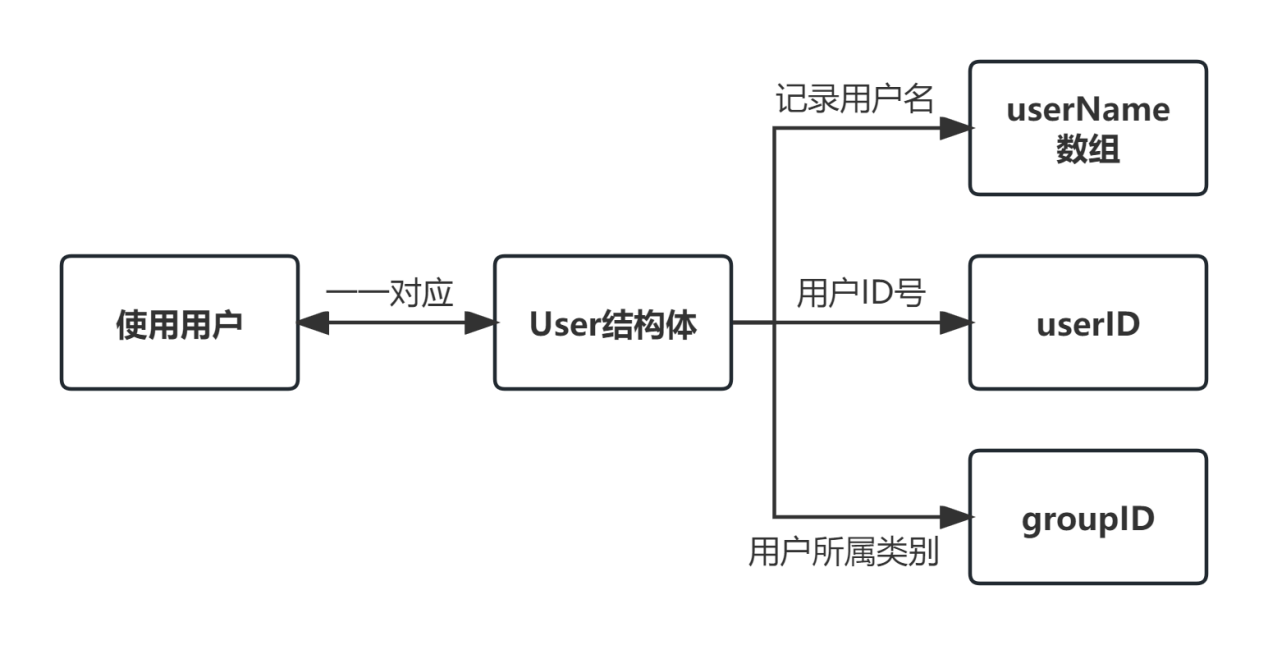
### Extra Fuction

#### Extension block

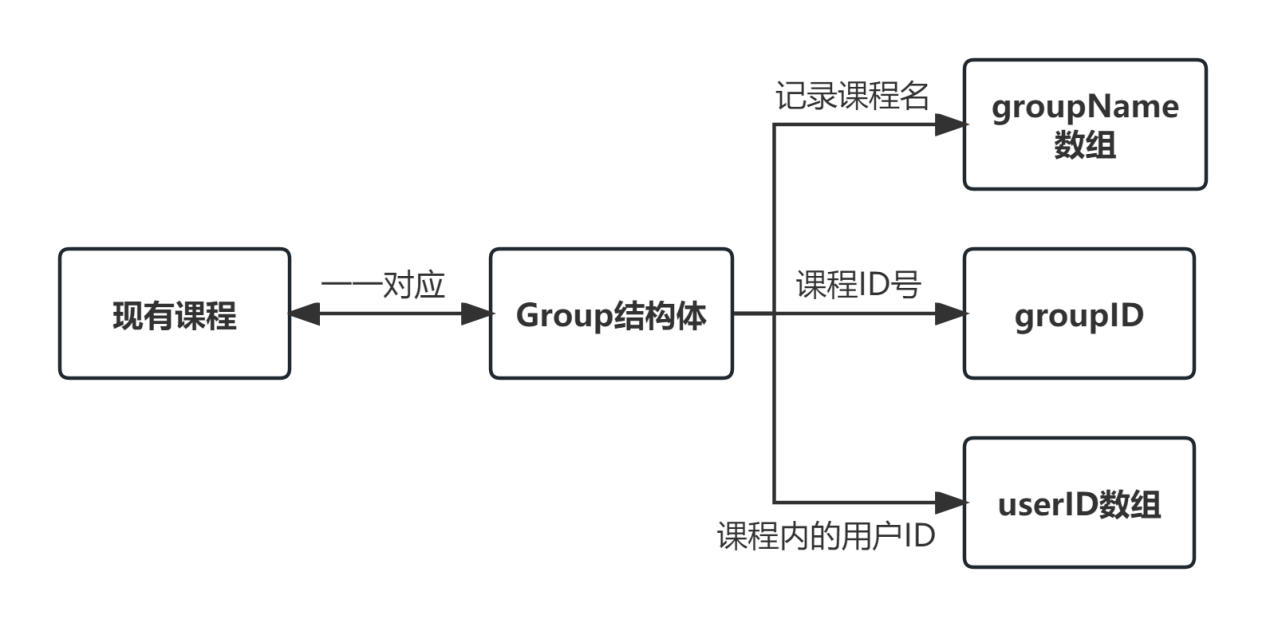
##### Prerequisites

For the concept of User and Group, our group has set up two structures for it, namely user and group. In the actual use process, the User is our user, and the group is the course.

Among users, we define the char array userName to hold the user name and limit its length; The userID and groupID variables of type short are defined to record the ID of the user and the type of user they belong to, respectively.



In the group, we define the array groupName of type char to record the name of the current course with a length limit; The variable groupID of type short is used to record the group number; The array userID of type short is used to record all student numbers within the current course.

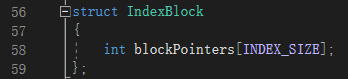


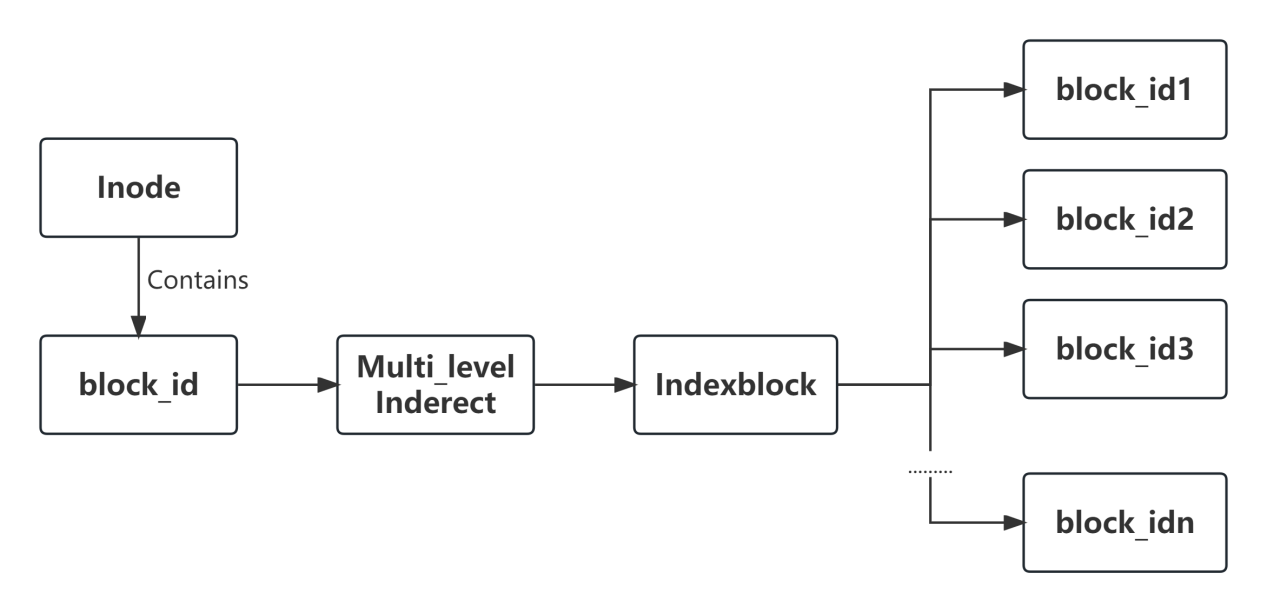
For users and groups, we set up a number of functions. The user has the login function loginSystems(); Create the user function createUser(); Delete user function deleteUser(); The user information query function and the user information modification function. Group displays the current group function listgroup(); createGroup() and deleteGroup(); Add a user to the group addUserToGroup() and delete a user from the group deleteUserFromGroup(); Course information query function and so on.

##### Design Ideas

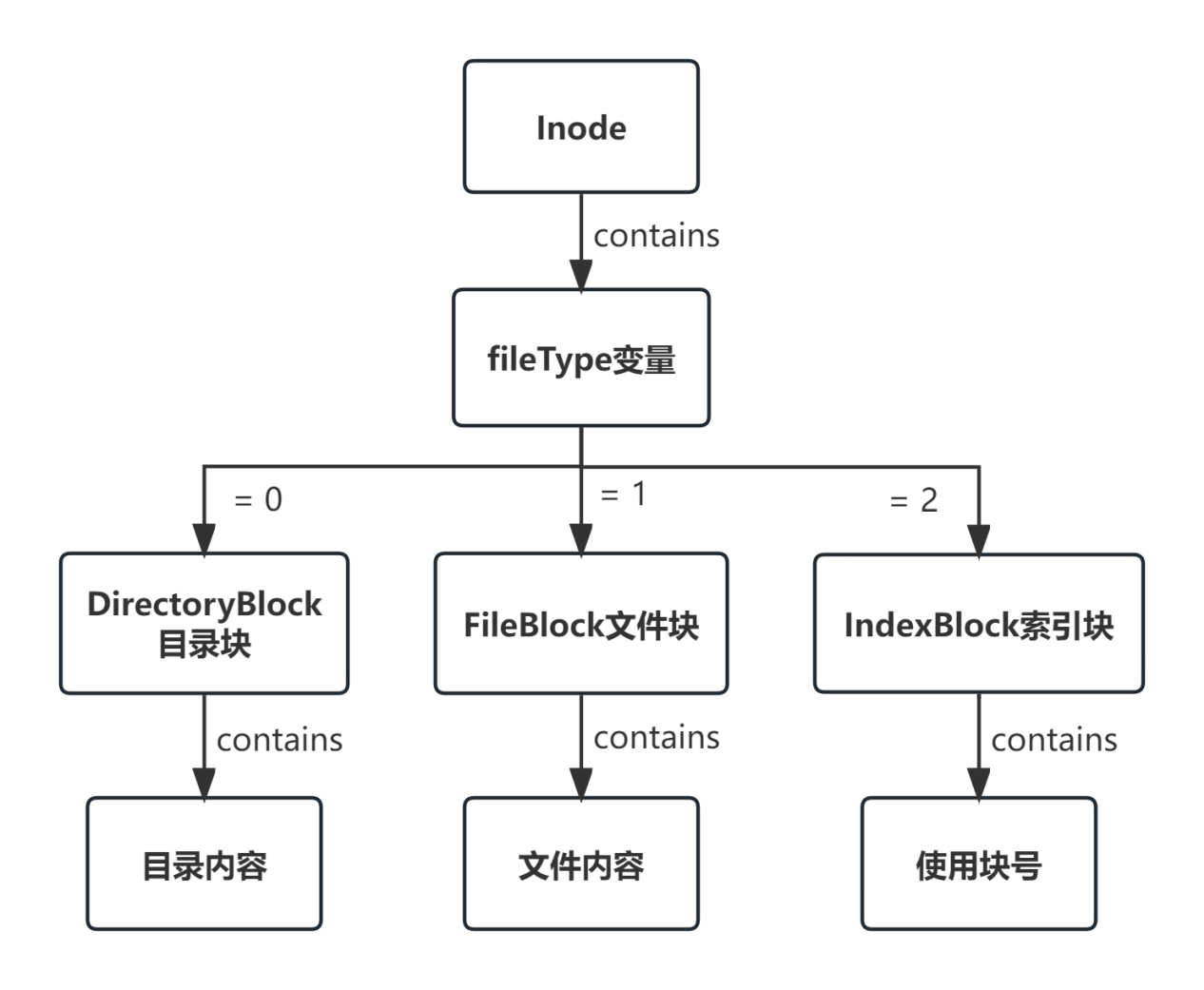
Normally, our file content is stored in a block, which has a certain size limit, our group set the block size is 2048, when we want to input more than the block size limit, that is, can not fit in a block, we will lose some content.

To solve this problem, our group adopts the method of index block to realize the expansion of block. In this regard, our group defines a new structure IndexBlock, in this structure, it contains an int type array blockPointers, because it is an int type element and the block size is 2048, so the internal size of the array is 512, the element in the array is the block number that stores the file content. The code and internal structure diagram are shown below.

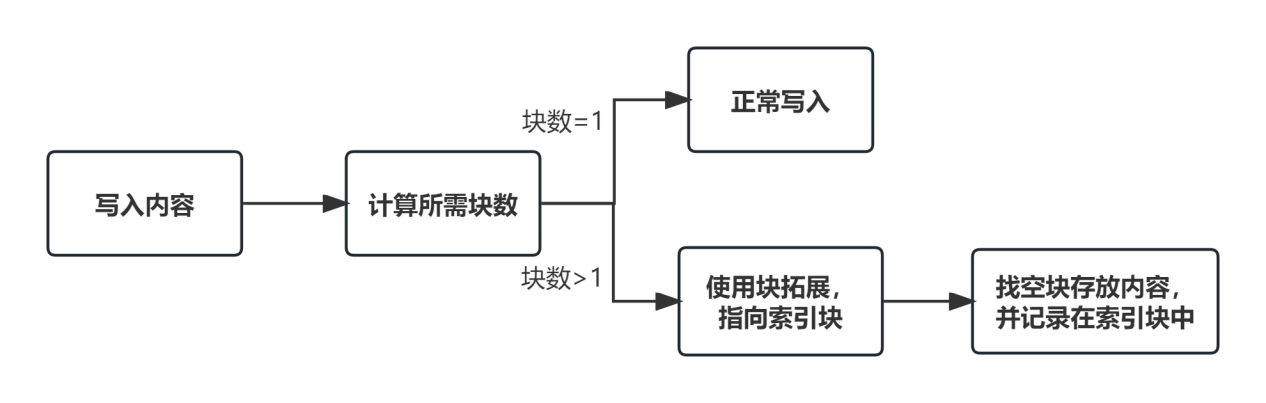


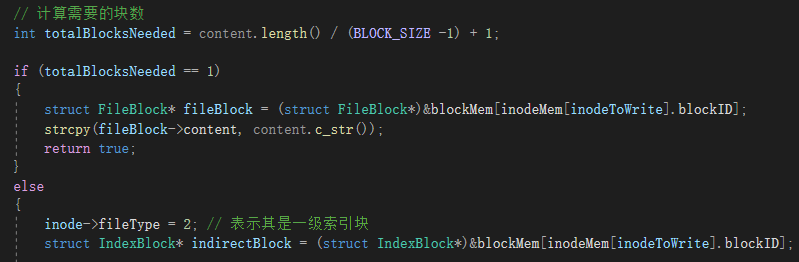


There is only one block number in the Inode set by our group, and its fast category is distinguished by the fileType variable in the Inode. When fileType is 0, the block directed by the current block\_id is the DirectoryBlock type. When fileType is 1, the block pointed by block\_id is FileBlock type. When fileType is 2, the block pointed by the current block\_id is IndexBlock. With this variable we realize the differentiation of blocks.

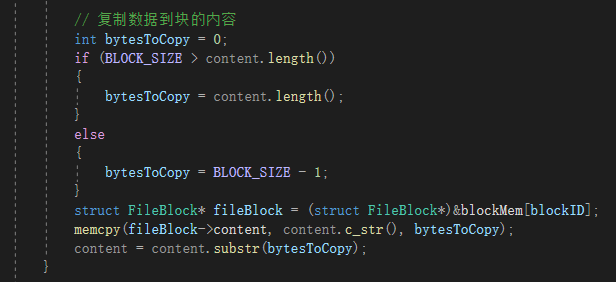


When we need to input a file, we will calculate the number of blocks required for the input content in advance. If the number of blocks is 1, it means that the input size does not exceed the block limit, and block\_id points to a file block and is directly input. When the number of blocks is greater than 1, it means that block expansion is required. block\_id indicates an index block, and the index block records the block number where the file content resides.

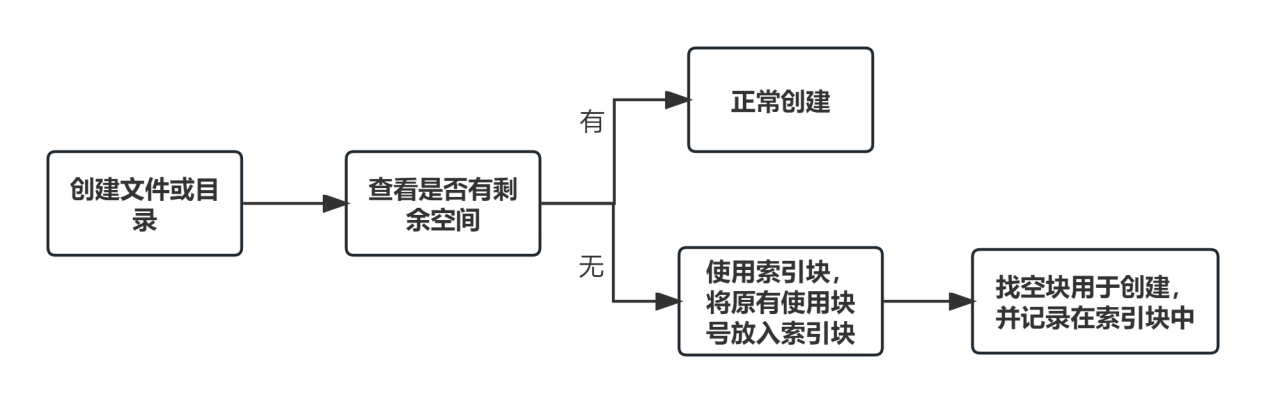


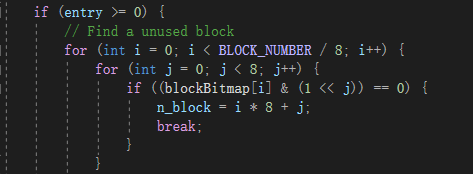


When the files exceeding the size limit are written into the block, our group assigns the value of the first block size character to the target block each time through the memcpy() function, and then substr() function subtracts the input content by the first block size character, so as to achieve the writing of the large file.

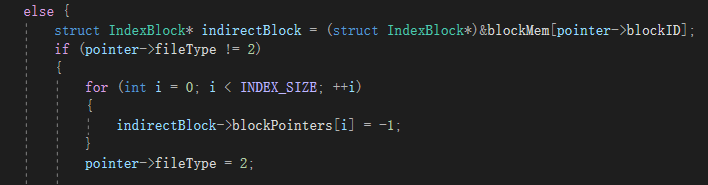


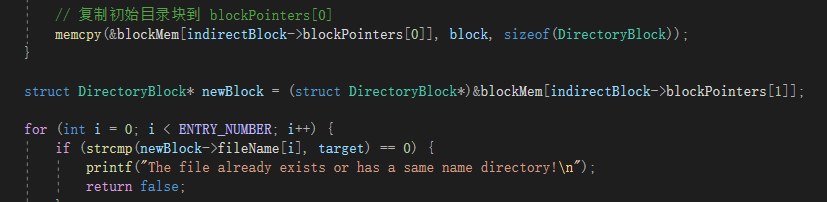
When we need to create a new file or directory, we check to see if the current directory is full. If there is any space left, we will create it directly; If it is full, we create a new index block, transfer the block number from the original directory block to the index block, and then create a new block in the index block.





When there is no place under the directory block, we point the block number contained in the current directory block to an index block. After transferring the contents of the initial block, we start to create the second element position under the index block to achieve expansion.

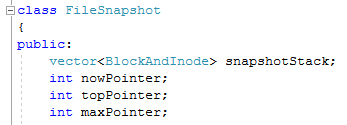


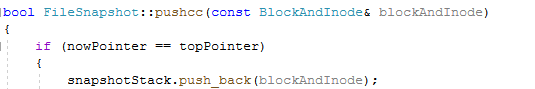


#### File snapshot

The specific purpose and idea of file snapshot is described as follows. Considering that the backup function takes too long, writing all inodes, blocks and blockbitmaps into txt files during the operation of a large system will waste resources and time, and it will be difficult to archive when users accidentally close occasionally. Therefore, after reading the relevant materials, we decided to provide the file snapshot function.

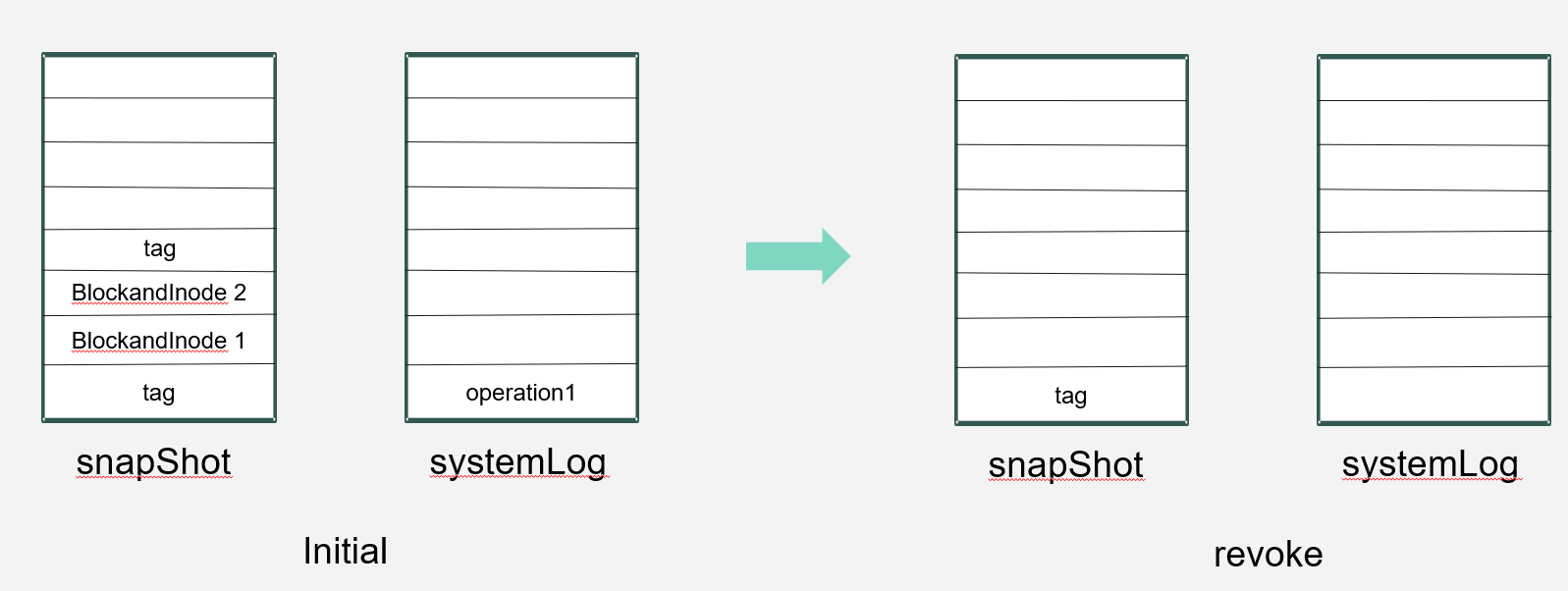
Theoretically, file snapshot can be divided into global snapshot and incremental snapshot operations. The global snapshot has an easier sight line, but there are too many similarities between it and backup, which is also more time-consuming and resource consuming. The incremental snapshot mainly stores two kinds of logs, one is the changed Inode data, and the other is the operation log, which needs to store each step of operations for easy backtracking. In order to achieve this function, the first data model we think of using is the stack model, using the stack, we can realize the last read operation to be the first recovery, the theory of the stack is in line with the actual situation, so we introduce the stack in the concrete code, but if we want to look at the log file, We have to take out all the elements in the stack to read, for the stack, except the top of the stack, the rest are invisible, so in order to facilitate our implementation, we use vector to simulate the stack, because vector has the same push\_back function, and vector can iterate through. So we store logs and so on through vector, as shown in the figure below.



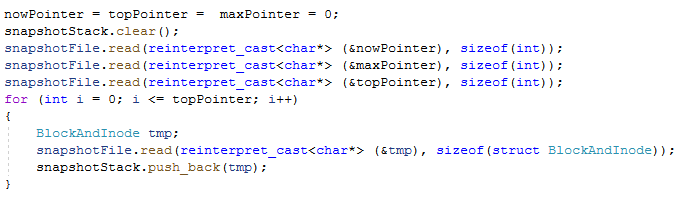


The specific design idea and diagram are below. From the design idea, we set three Pointers and two stacks. The three Pointers respectively represent the topPointer of the stack, the nowPointer of the operation pointer, and the maxPointer of the length limit pointer. The two stacks are the snapshot stack, where data is stored, and the systemLog stack, where logs are stored. In the data stack, where maxPointer can be used as a prerequisite as the upper limit on the number of operations stored in the operation log, nowPointer faces conditions that point to the current operating system, if faced with backtracking, the top pointer remains the same, and nowPointer will point to a previous operation and change. The log written will be overwritten starting with the operation that nowPointer points to.

At the same time, since we call related interfaces, we may trace the block impact caused by multiple file system interfaces at a time. Therefore, we will insert tag values before and after each Client operation to mark the impact of each operation on the block and Inode. Each file snapshot will delete the tag at the top of the stack and recover the data at the top of the stack, thus completing the function of file snapshot according to log operations, as shown in the following figure:



As for how to store the block and Inode into the file, we use the function of casting to char\* variables, which can successfully read and write from the file. In summary, our file snapshot function is complete.



# **Instruction of experiment operation**

In order to compile and run on different platforms, we use cmake to build the project, which also brings some benefits:

Keep the source directory clean: Keep the source directory relatively clean by separating the generated build and object files from the source code. This makes it easier to version control, publish, and share the source code without mixing the build files in it.

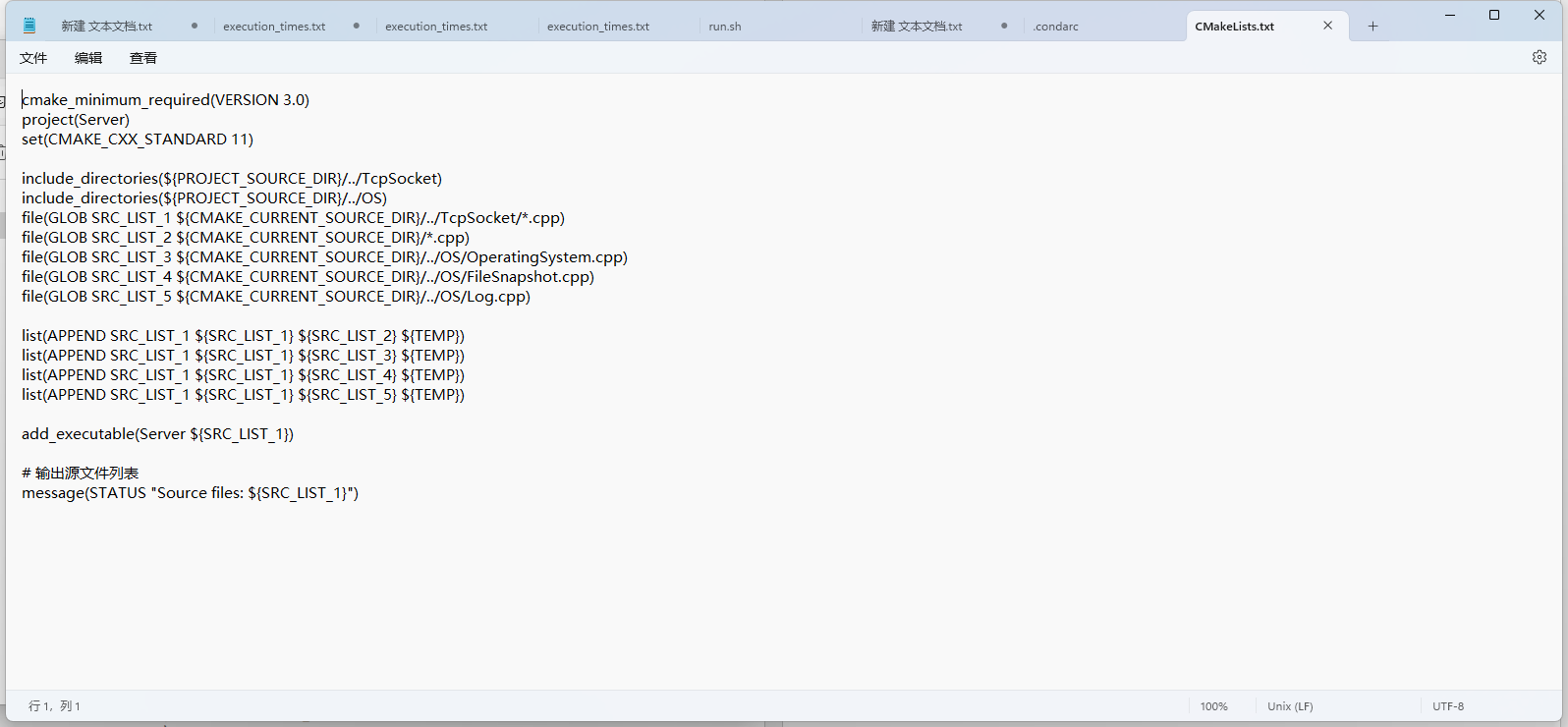
Support for multiple platforms: Separating source code and build files makes it easier to build projects on different platforms, since build files are often platform-independent.

Avoid introducing unnecessary files into the source code: Keep build files in a separate directory to prevent the temporary and intermediate files generated by the build process from polluting the source code directory.

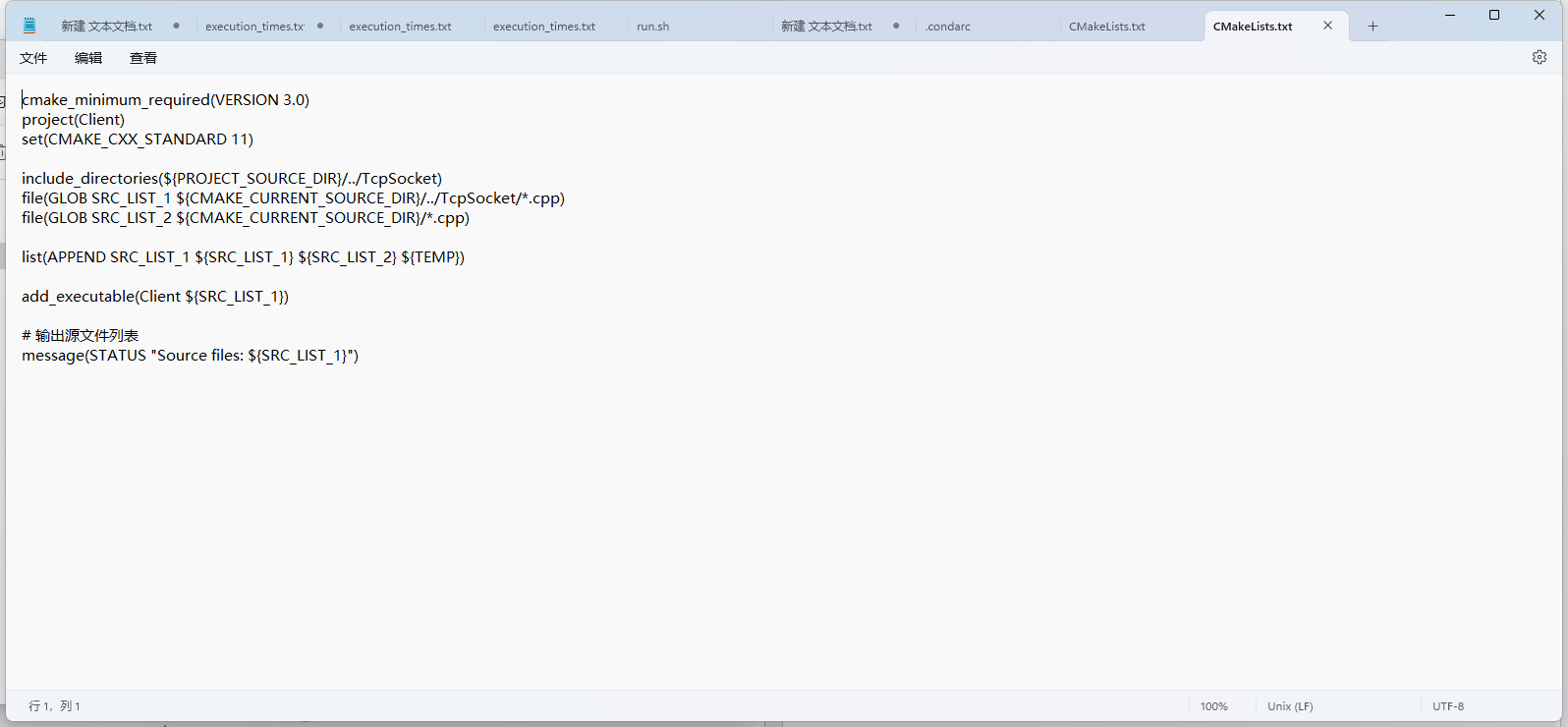
Therefore, we provide the cmakelist.txt file, which can be compiled and run by the user only by entering instructions.

### Project Compilation

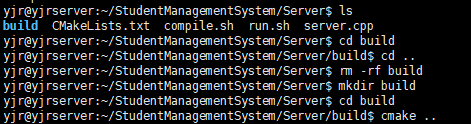
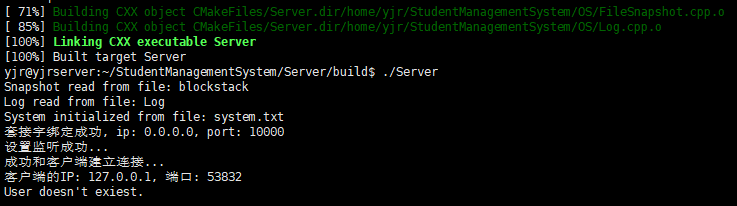
The Cmakelists file on the Server is as follows:



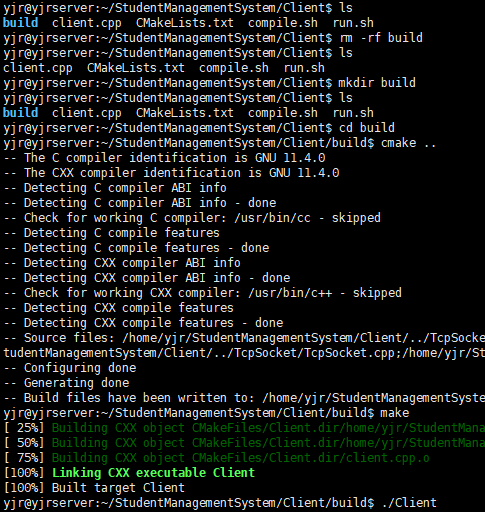
The Cmakelists file in the Client is as follows:

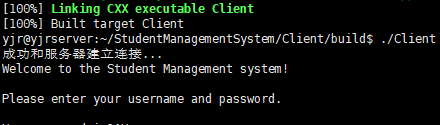


Since we need to open the Server user before we can access it, we need to enter the server folder first before we can operate. First, we need to check whether the folder contains an empty build folder. If it does not contain an empty build folder or is not empty, we need to delete it and add a new one. At this time, we can run ./Server, the following figure is the compilation of the Server, the figure below is the successful operation of the Server screen:

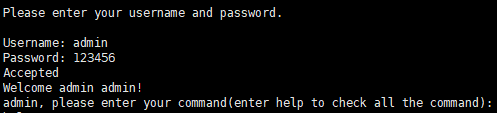
The compilation and running of the Client side is similar to that of the Server side. We only need to remember that the running of the Client needs to be done after the running of the Server. At the same time, we can open multiple terminals for access at the same time, and the Server side will modify it in real time, that is, multi-user access:

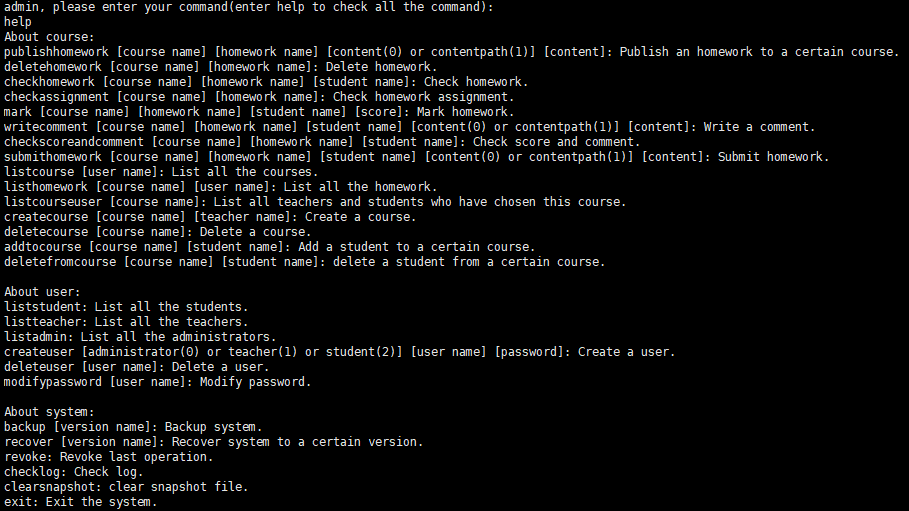




### Project Operation

After the initial login, we can log in as the administrator. After entering the account and password, we can input help to view the available instructions as well as the format and instructions of each instruction. Finally, if we need to exit, we can input exit on the client side to complete the exit. On the Server side, you need to enter Ctrl+C to terminate the run.





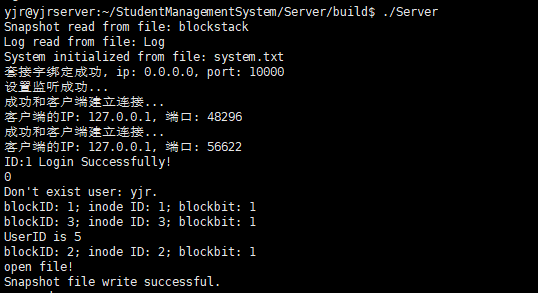




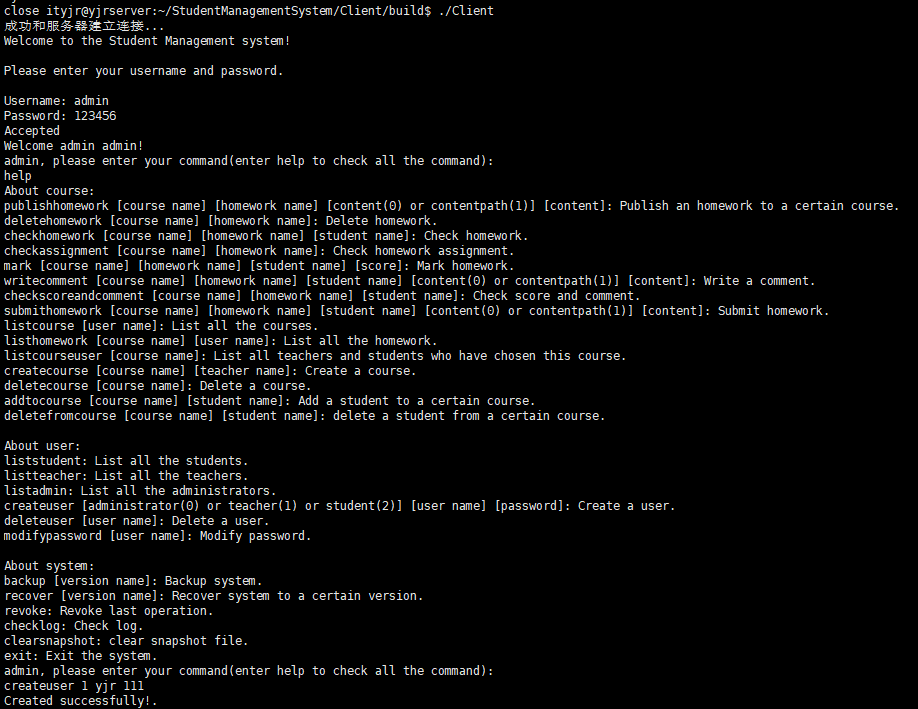
At the same time, we need to remember that a Server can only run one Server, if multiple users access the Server at the same time to start their own server will produce an error, so it is best to run on the local server or isolated server, and exit the server after each use

## Experimental results show

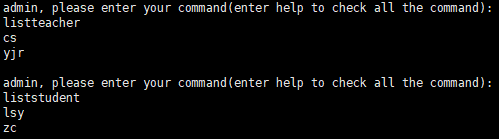
First we need to start the Server side, which will output some information about the operating system after the back-end operation, but it is not important.

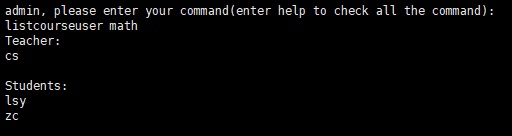


Then we open a terminal, enter the administrator account to log in, type help to see what commands are available, and create the teacher and student users.

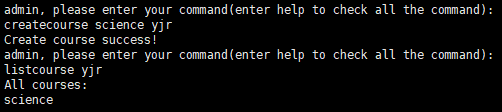


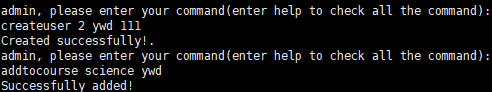
After creation we can view the information of all teachers and students, here we can see that there are two students and two teachers, as well as the teacher cs courses math and math courses have students under:



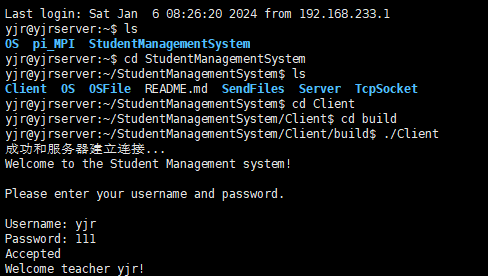
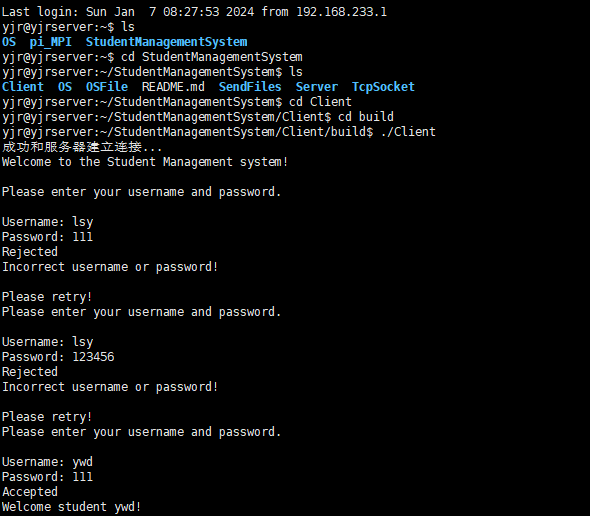


We can also create the science course for the new teacher yjr, at this time we can create a student ywd, and add the student ywd to the science course:

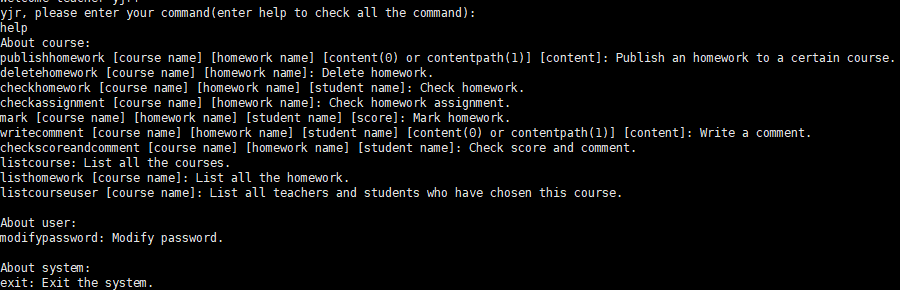
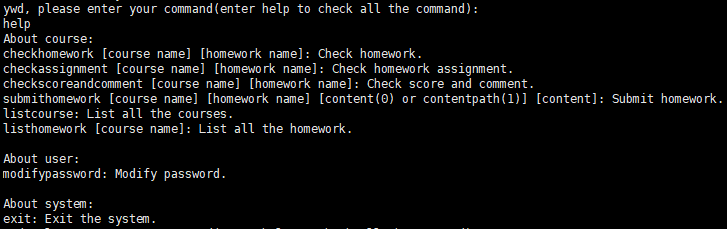




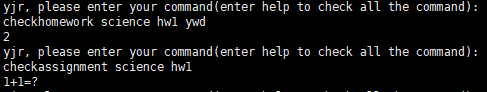
At this time, we can create two terminals, enter the build folder of the client in the same way, and run the client to log in as teacher yjr and student ywd respectively.

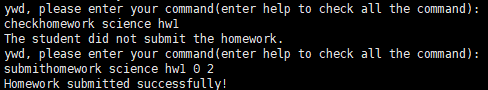
 

At this time, we can view the available commands at both ends. Due to permission restrictions, we have limited the commands that can be used by both teacher users and student users:

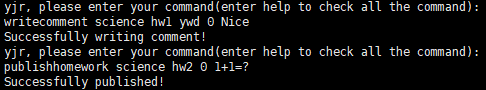
 

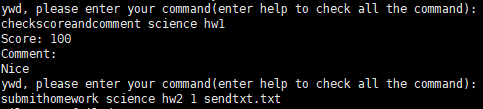
Then we can publish the homework on the teacher side. The content of the homework can be input by ourselves or written in the file and then uploaded. Since the Server is modified in real time, students can submit their homework after the publication of the homework, and the teacher will grade and comment, and the students can check their scores at last.



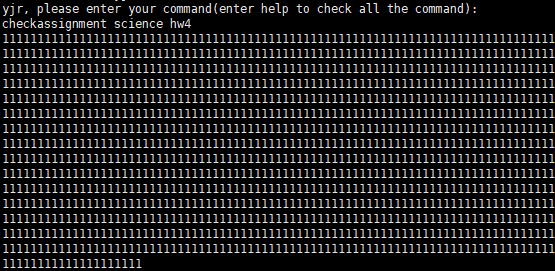




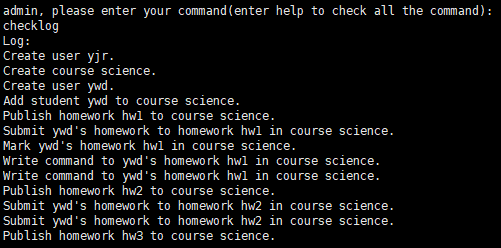




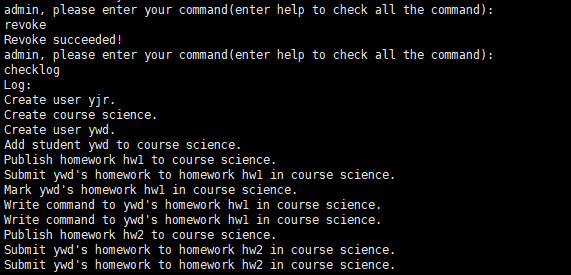


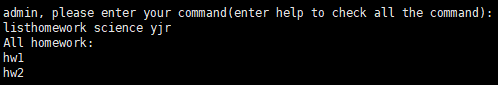


After completing the above operations, we can return to the administrator side to view the file snapshot, first view the operation log to see what operations have been performed:



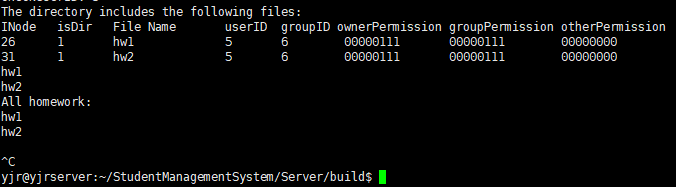
As you can see, we have done the above operations, and then we backtrack, we found that a log has been cleared, and the hw3 item is missing from the published job:





Now that we have demonstrated most of our functionality, we can enter exit to exit the Client and Ctrl+C to exit the Server.





## Team member contribution description

The specific division of labor of our team members is as follows:

Chen Shuo: Responsible for writing the Server side code、complete the communication between the Socket socket implementation processes and complete the mutex lock, and also wrote part of the code to help the Client side.

Yang Jiarui: Responsible for part of the work of file system construction and the implementation of some interfaces, and the completion of file snapshots.

Li Shengyang: Responsible for part of the construction of the file system and the construction of the test code, and participated in the writing of the Client side code.

Zhang Chi: Responsible for part of the construction of the file system and the implementation of some of the interfaces, and the realization of the extension block function.