

Project 3 Report

Name: [REDACTED] Student id: [REDACTED]

Abstract

This report outlines the development and implementation of a basic disk-storage system and a file system with multiple user support, which use socket for communication. The project consists of three main steps: designing a basic disk-storage system, implementing a basic file system, and incorporating multiple user functionalities.

1 Step 1: Design a Basic Disk-Storage System

1.1 Implementation Details

1.1.1 Server (BDS.c)

Initialization: The server initializes by obtaining the disk file name, number of cylinders, sectors per cylinder, track-to-track delay time, and port number from the command line arguments. It then opens the disk file, uses `lseek` and `write` to stretch the file, and maps the file into memory using `mmap`.

Pseudo-code for server initialization

```
// Pseudo-code for server initialization
function initializeServer(args):
    diskfname = args[1]
    ncyl = atoi(args[2])
    nsec = atoi(args[3])
    ttd = atoi(args[4])
    port = atoi(args[5])

    fd = open(diskfname, O_RDWR | O_CREAT, S_IRUSR | S_IWUSR)
    FILESIZE = ncyl * nsec * BLOCKSIZE
    lseek(fd, FILESIZE - 1, SEEK_SET)
    write(fd, "", 1)
    disk_data = mmap(NULL, FILESIZE, PROT_READ | PROT_WRITE,
        MAP_SHARED, fd, 0)
```

Command Handling:

- `cmd_i`: Returns the disk parameters (number of cylinders and sectors per cylinder).
- `cmd_r`: Reads the data from the specified cylinder and sector. If the address is valid, it sends the data along with a "Yes" response to the client.
- `cmd_w`: Writes data to the specified cylinder and sector. If the address and data length are valid, it writes the data and sends a "Yes" response.
- `cmd_e`: Sends a "Bye!" response and terminates the connection.

Pseudo-code for command handling

```
// Pseudo-code for command handling
function handleCommand(cmd, args):
    if cmd == "I":
        sendToClient(ncyl, nsec)
    elif cmd == "R":
        c, s = parseArgs(args)
        if validAddress(c, s):
            data = readFromDisk(c, s)
            sendToClient("Yes", data)
        else:
            sendToClient("No")
    elif cmd == "W":
        c, s, l, data = parseArgs(args)
        if validAddress(c, s) and validLength(l):
            writeToDisk(c, s, data)
            sendToClient("Yes")
        else:
            sendToClient("No")
    elif cmd == "E":
        sendToClient("Bye!")
        terminateConnection()
```

Client Handling: The server parses client messages using `strtok` and calls the corresponding command handling function. If the command is unknown, it returns an "Unknown command" response.

Pseudo-code for client handling

```
// Pseudo-code for client handling
function handleClientMessage(msg):
    cmd, args = parseMessage(msg)
    if validCommand(cmd):
        handleCommand(cmd, args)
    else:
        sendToClient("Unknown command")
```

1.1.2 Client (BDC.c)

Initialization: The client initializes by obtaining the port number from the command line arguments and sets up a TCP client connection to the local server.

Pseudo-code for client initialization

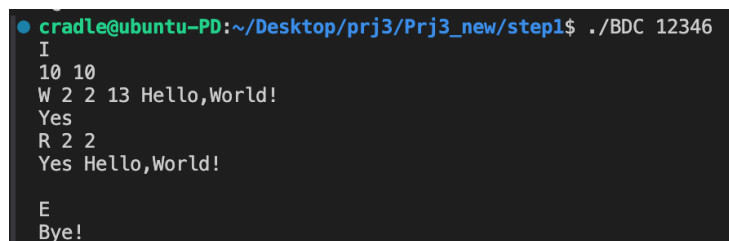
```
// Pseudo-code for client initialization
function initializeClient(args):
    port = atoi(args[1])
    client = tcpClient("localhost", port)
```

Interaction Loop: The client reads user input, sends it to the server, receives the server's response, and prints it. If the response is "Bye!", the client terminates the loop and closes the connection.

Pseudo-code for client interaction loop

```
// Pseudo-code for client interaction loop
function clientLoop(client):
    while true:
        input = readUserInput()
        sendToServer(client, input)
        response = receiveFromServer(client)
        print(response)
        if response == "Bye!":
            break
    closeClient(client)
```

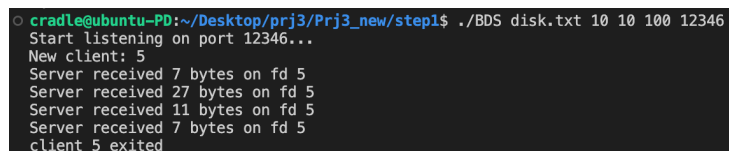
1.2 Testing Results



```
cradle@ubuntu-PD:~/Desktop/prj3/Prj3_new/step1$ ./BDC 12346
I
10 10
W 2 2 13 Hello,World!
Yes
R 2 2
Yes Hello,World!

E
Bye!
```

Figure 1: step1 test: disk server



```
cradle@ubuntu-PD:~/Desktop/prj3/Prj3_new/step1$ ./BDS disk.txt 10 10 100 12346
Start listening on port 12346...
New client: 5
Server received 7 bytes on fd 5
Server received 27 bytes on fd 5
Server received 11 bytes on fd 5
Server received 7 bytes on fd 5
client 5 exited
```

Figure 2: step1 test: client

2 Step2: Design a basic file system

2.1 INode Design

structure of INode

```

+-----+
|           INode           |
+-----+
| filename (100 chars)      |
+-----+
| type (char)               |
+-----+
| size (int)                |
+-----+
| last\_modified (time\_t) |
+-----+
| direct\_blocks[10]        |
| [int, int, ..., int]     |
+-----+
| single\_indirect\_block |
| (int)                    |
+-----+
| double\_indirect\_block |
| (int)                    |
+-----+

```

2.2 Implementation Details

2.2.1 Server Setup and Configuration

- Use `socket()` to create a socket and configure the network endpoint using `bind()` and `listen()`.
- The server accepts client connection requests and generates a new socket for communication using the `accept()` method.

2.2.2 Disk Storage Initialization

- The server connects to the disk server, exchanges data via socket, and sets disk parameters such as the number of cylinders and sectors.
- Data is loaded or updated from the disk using specific protocol commands (R for read, W for write).

2.2.3 Directory and File Management

- The file system maintains a root directory table, `root_directory`, with each entry containing file/directory name, type, starting block, file size, and last modification time.

- Basic file operations such as `create_file`, `delete_file`, `change_dir`, `list_file`, etc., are provided.
- Creation and deletion of files and directories are achieved by updating the root directory table and disk blocks. For directories, it checks if the directory is empty; for files, it releases disk blocks and updates the disk.

2.2.4 Data Handling

- File data is stored in disk blocks and accessed directly via indexing. Modifications to file content (e.g., write, insert, delete) directly affect the disk block data.
- File operations update the file size and last modification time, ensuring consistency of file metadata.

2.3 Command Handling

In this section, we describe the various command handling functionalities implemented in the file system. Each function serves a specific purpose, from initializing the file system to handling file operations like creation, deletion, reading, writing, and directory management.

2.3.1 Initialize the File System

To initialize the file system, we need to set up the root directory and prepare the data blocks for storing file contents. This involves clearing any pre-existing data and ensuring the system is ready for new operations.

Function to initialize the file system

```
initialize_file_system()  
// Initialize root directory and disk structures  
root_directory = []  
disk = []  
file_count = 0
```

2.3.2 Create a File

The `create_file` function is responsible for creating a new file or directory. It first checks if a file with the same name already exists. If not, it finds a free directory entry and assigns the filename, type, and initializes the start block and file size. For directories, it sets the start block to 0.

Function to create a new file

```
create_file(name, type)
filename = concatenate(current_directory, name)

if file_exists(filename)
    print("File_already_exists")
    return

entry_index = find_free_entry()
if entry_index == -1
    print("Directory_is_full")
    return

root_directory[entry_index] = create_directory_entry(filename, type)
if type == 'D'
    root_directory[entry_index].start_block = 0
    root_directory[entry_index].file_size = 0
    print("Directory_created")
else
    root_directory[entry_index].start_block = find_free_block()
    write_to_disk(root_directory[entry_index].start_block, "0")
    root_directory[entry_index].file_size = 1

file_count += 1
update_last_modified(root_directory[entry_index])
```

2.3.3 Read Data from a File

The `read_file` function allows reading data from a specified file. It locates the file in the directory, reads the data from the disk, and prints it.

Function to read data from a file

```
read_file(name)
filename = concatenate(current_directory, name)
file_entry = find_file_entry(filename)

if file_entry is not null and file_entry.type == 'F'
    start_block = file_entry.start_block
    file_size = file_entry.file_size
    data = read_from_disk(start_block, file_size)
    print("File_data:", data)
else
    print("File_does_not_exist_or_is_not_a_file")
```

2.4 Write Data to a File

The `write_file` function writes the given data to a specified file. It updates the file size and modifies the disk content accordingly.

Function to write data to a file

```
write_file(name, len, data)
filename = concatenate(current_directory, name)
file_entry = find_file_entry(filename)
if file_entry is not null and file_entry.type == 'F'
    start_block = file_entry.start_block
    write_to_disk(start_block, data, len)
    file_entry.file_size = len
    update_last_modified(file_entry)
else
    print("File_does_not_exist_or_is_not_a_file")
```

2.5 Append Data to a File

The `insert_file` function appends data at a specific position within a file. It adjusts the file size and content accordingly.

Function to append data to a file

```
append_file(name, data)
filename = concatenate(current_directory, name)
file_entry = find_file_entry(filename)

if file_entry is not null and file_entry.type == 'F'
    start_block = file_entry.start_block
    append_to_disk(start_block, data)
    file_entry.file_size += length(data)
    update_last_modified(file_entry)
else
    print("File_does_not_exist_or_is_not_a_file")
```

2.5.1 Remove a File

The `delete_file` function deletes a file or directory. If it's a directory, it ensures the directory is empty or recursively deletes its contents.

Function to remove a file

```

delete_file(name)
filename = concatenate(current_directory, name)
file_entry = find_file_entry(filename)

scss

if file_entry is not null
    if file_entry.type == 'F'
        clear_directory_entry(file_entry)
        clear_disk(file_entry.start_block)
        file_count -= 1
        print("File_deleted")
    else if file_entry.type == 'D'
        if is_directory_empty(file_entry)
            clear_directory_entry(file_entry)
            print("Directory_deleted")
        else
            recursively_delete_directory(file_entry)
            clear_directory_entry(file_entry)
            print("Directory_and_contents_deleted")
    else
        print("Unknown_type")
else
    print("File_or_directory_does_not_exist")

```

2.5.2 Create Directories

Directory creation is handled within the `create_file` function when the type is specified as 'D'. It follows similar steps as file creation but sets the start block to 0.

Function to create directories

```

create_directory(name)
create_file(name, 'D')

```

2.6 Test Results



Figure 3: step2 test:disk server file system server


```
cradle@ubuntu-PD:~/Desktop/prj3/Prj3_new/step3$ ./FC 10.211.55.10 10367
pwd
/
ls

f
Done
create_user yjh
No: Missing arguments. Usage: create_user <username> <password>
create_user yjh 123
Yes: User created
create_user ljj 123
Yes: User created
login yjh 123
Yes: Logged in as yjh
pwd
/yjh/
mkdir a1
Invalid command.
mkdir a1
Yes
cd a1
Yes: /yjh/a1/
mk f.c
Yes
logout
Yes: Logged out
pwd
/
ls
/user/ljj.txt, Size:0, Last Modified Time:Thu May 30 23:55:47 2024 /user/yjh.txt, Size:0,
Last Modified Time:Thu May 30 23:55:47 2024 /yjh/a1/f.c, Size:13, Last Modified Time:Thu
May 30 23:55:47 2024
&
/ljj /user /yjh /yjh/a1
delete_user yjh
No: Missing arguments. Usage: delete_user <username> <password>
delete_user yjh 123
Yes
Yes: User deleted
ls
/user/ljj.txt, Size:0, Last Modified Time:Thu May 30 23:55:47 2024
cd ..
Yes: /
ls
/user/ljj.txt, Size:0, Last Modified Time:Thu May 30 23:55:47 2024
&
/ljj /user
delete_user ljj 123
Yes
Yes: User deleted
cd ..
Yes: /
ls
&
/user
cd .
Yes: /
ls
&
/user
cd user
Yes: /user/
ls
e
Client closed
```

Figure 4: step1 test: file system client

3 Multiple Users

This section describes the implementation of multiple user support in our file system, including user creation, deletion, login, and file operations associated with users.

3.1 User Structure and INode

```
// Define the User structure
User {
    username: string;
    password: string;
    home_directory: string;
}

user_list = array of User;
user_count = 0;
current_user = null;
// Define the DirectoryEntry structure
DirectoryEntry {
    filename: string;
    type: char; // "F" indicates a file, "D" indicates a directory
    start_block: int;
    file_size: int;
    owner: string; // File owner
    permissions: char; // 'r': read-only, 'w': writable
    last_modified: string;
}
```

3.2 Class View

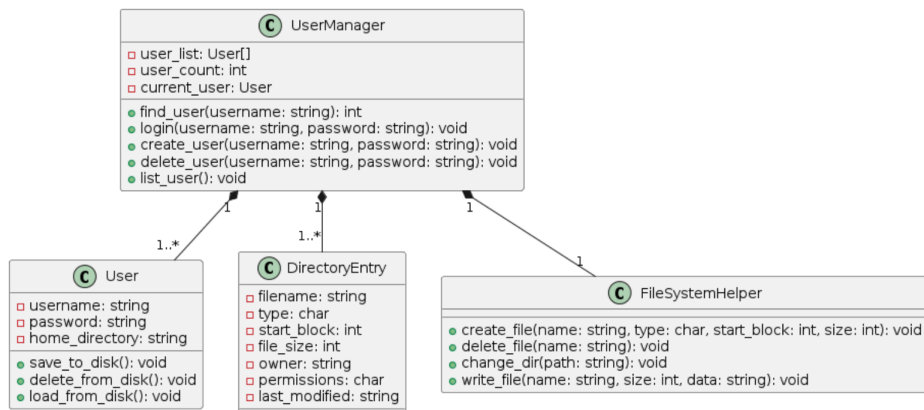


Figure 5: Class view of multiple users

3.3 Test Results

```
cradle@ubuntu-PD:~/Desktop/prj3/Prj3_new/tmp/4$ ./client 10.211.55.10 12437
ls
f
Done
pwd
/
create_user yjh 123
Yes: User created
create_user wzy 123
Yes: User created
login yjh 111
No: Invalid username or password
login yjh 123
Yes: Logged in as yjh
pwd
/yjh/
mkdir a1
Yes
cd a1
Yes: /yjh/a1/
mk f.c
Yes
w f.c 3 abc
Yes
cat f.c 3 abc
Yes: abc
owner: yjh, pms: r
rm f.c
Yes
ls
cd ..
Yes: /yjh/
logout
Yes: Logged out
login wzy 123
Yes: Logged in as wzy
pwd
/wzy/
mk WZY.c
Yes
w WZY.c 3 abc
Yes
cat WZY.c
Yes: abc
owner: wzy, pms: r
logout
Yes: Logged out
login yjh 123
Yes: Logged in as yjh
cd ..
Yes: /
cd wzy
Yes: /wzy/
ls
/wzy/WZY.c, Size:0, Last Modified Time:Thu May 30 00:20:22 2024
cat WZY.c
Yes: abc
owner: wzy, pms: r
rm WZY.c
owner: wzy, pms: r
No: Permission denied
e
Client closed
cradle@ubuntu-PD:~/Desktop/prj3/Prj3_new/tmp/4$
```

Figure 6: Class view of multiple users