

OS LAB 12 - ASSIGNMENT

Client Server Application

HAMIZ ALI – 260251

MAIRA TARIQ – 263021

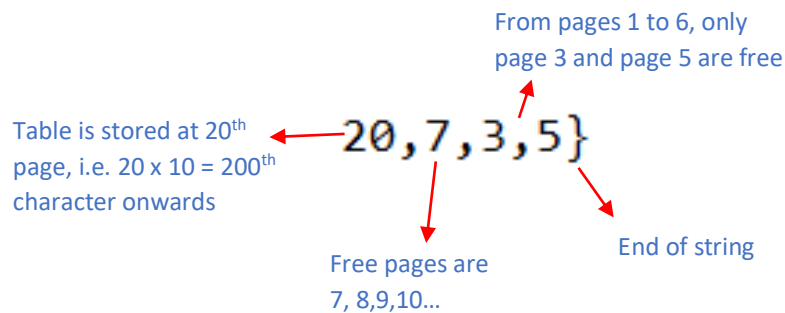
UME HANI - 271976

# User Guide

All the data of our file structure is stored in [SAMPLE.txt](#) file. The structure of the directories and files (i.e. the table) is stored in the form of a string which will be explained later in this guide.

## Top part of the file (free locations)

We have divided our memory into pages of 100 characters each. The first hundred characters, i.e. the 0<sup>th</sup> page, is reserved to store location of table, followed by last empty page from which onwards, all pages are empty. After these two elements, rest are pages freed by truncation of files and they are separated by commas. This is explained below in snapshot.



When the free pages reach one less than the table location, e.g. 19 in above example, the table is moved ahead by 10 pages. So new location will be 30<sup>th</sup> page and so on.

Note that the table itself is stored in the end and only its location is stored in the beginning. This is to make sure in case pages are filled up, only table needs to be moved and not the other data which is larger than the table. We need to update the first line only in this case.

## Middle part of the file (pages)

After the 0<sup>th</sup> page, the other pages are simply filled with contents of the file. If the page is not filled fully, e.g. only 30 characters are to be added, there is marker in file which stores this value. So next time the same file is written to, it first writes in this last page from 31<sup>st</sup> index.

## Last part of the file (table)

### **How the directories and file structure is stored in the table:**

We have used special characters to denote hierarchy in our structure. They are follows:

- | - This character is used to denote that we are stepping inside a directory, i.e. going from a parent directory to child directory
- ] - This character denotes moving up one step in the hierarchy, i.e. going from child directory to parent directory

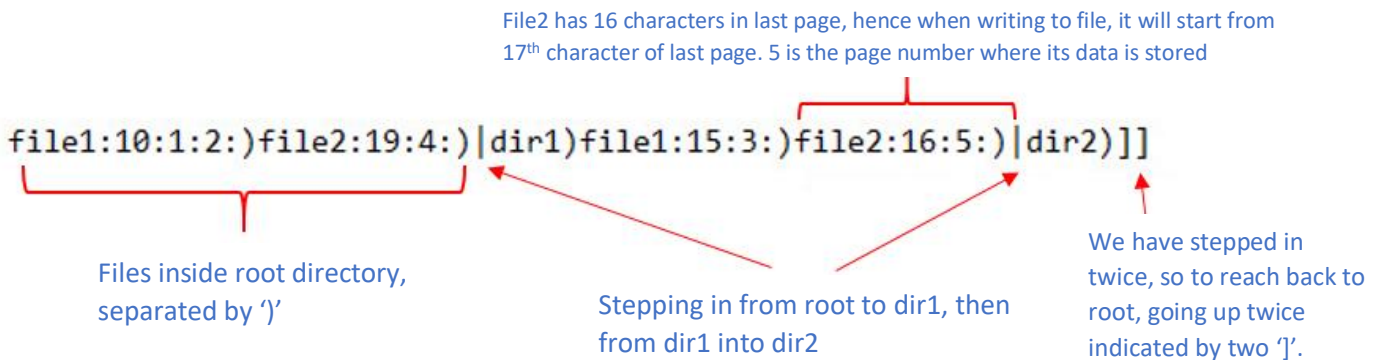
- ) - Closing parenthesis indicates files inside the given directory. The first occurrence of this character is after the directory name, which is followed by all the files inside it, separated by ')'
- : - Colon is used to separate the structure of files. First occurrence is after the file name, followed by the number of characters of entered in this file's last page, and followed by page numbers where the content of this file is stored. They are separated by ':'.
- The [ symbol donates the end of the table.

The snapshot below best explains usage of these characters. The structure is as follows: *(note the duplicate file names inside different directories, this is allowed)*

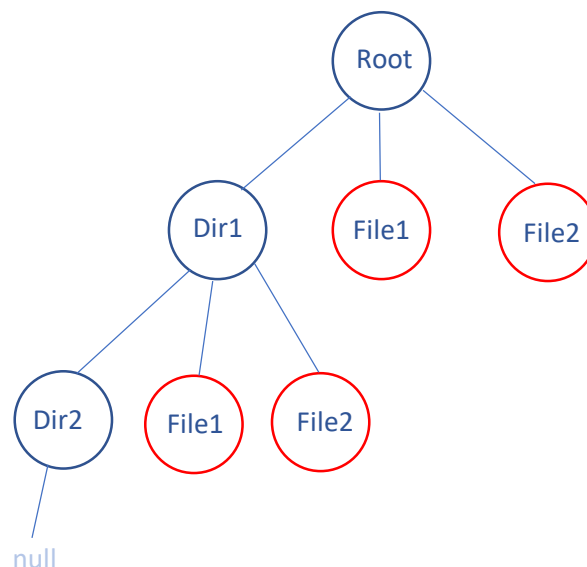
**Root** -> file1, file2, dir1

**dir1** -> file1, file2, dir2

**dir2** -> (empty directory)



In order to read the table, we only read this string from the file and there is no need to —read the entire file. The string is de-serialized inside our code and stored in a **tree**, whose root node is our 'root' directory. The tree made from above example will look like this: *(note: red nodes are file nodes and blue nodes are directory nodes)*



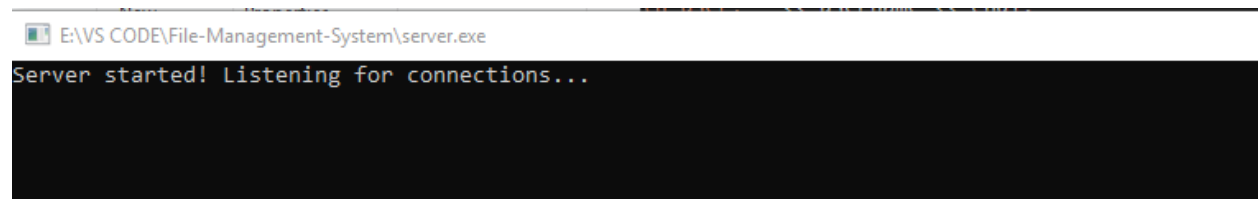
## Compiling Files:

- Server:
  - `g++ Server.cpp -o server -lws2_32`
- Client:
  - `g++ client.cpp -o client -lws2_32`
  - `./client.exe "server ip address"` as argument 2.

## Usage of our system

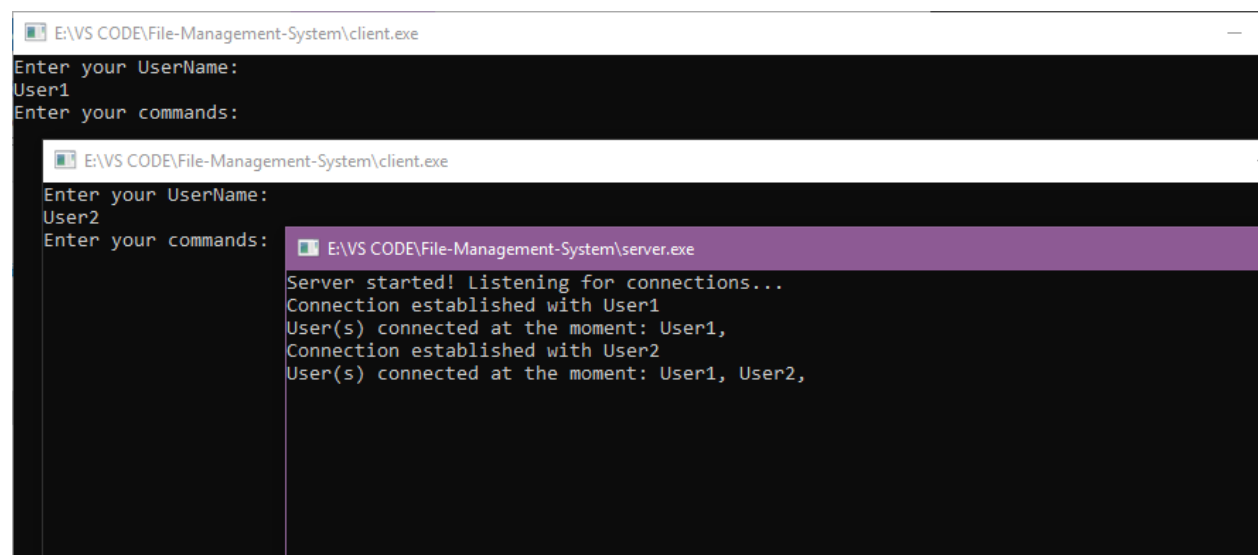
All the commands supported by our system are given below, along with their purpose and usage.

This part of our project is a client server application. Firstly, the server.cpp file is executed on the server which sets up our file management system.



```
E:\VS CODE\File-Management-System\server.exe
Server started! Listening for connections...
```

Then multiple clients can connect to this server and use the file management system simultaneously. In order to connect a client, run the .exe file with the ip address of the server as a parameter. In case, connection is not established, an error message will be displayed to the client. Otherwise, the connection will be established and the client then has to provide his/her username which will also be displayed on the server end.



```
E:\VS CODE\File-Management-System\client.exe
Enter your UserName:
User1
Enter your commands:

E:\VS CODE\File-Management-System\client.exe
Enter your UserName:
User2
Enter your commands:

E:\VS CODE\File-Management-System\server.exe
Server started! Listening for connections...
Connection established with User1
User(s) connected at the moment: User1,
Connection established with User2
User(s) connected at the moment: User1, User2,
```

Based on the input given by the clients, the actions are carried out until our clients exit the system using "exit;" command.

The server user can exit the server any time by pressing "Ctrl+C". If there are no clients connected at that time, server will close down. Else, it will not stop the server and show the user how many clients are

connected at the moment. Hence, server can be closed only if there are no clients connected at that very moment.

Moreover, server will automatically close down after maximum number of clients are served, the number is specified in MAX\_CLIENT.

- Writing commands by clients:
  - Make sure you end every command with a semicolon
  - Press enter to send the command to the server
- createDir
  - Purpose: This command creates a new directory in the current directory with the given name. Duplicate names in one location are not allowed.
  - Usage: *createDir <directory name>;*

```
Enter your commands:
createDir dir3;
root/:
Directory created.
#
```

- createFile
  - Purpose: This command creates a new file with the given name in current directory. After the command is run, user is prompted to enter the data to be entered in the file. Duplicate file names in same directory are not allowed.
  - Usage: *createFile <filename> "<file data>";*

```
Enter your commands:
createFile fileTest "This is test file";
root/:
File created.
#
```

- ls
  - Purpose: lists the content of the current directory
  - Usage:
    - *Ls;*

```
Enter your commands:
ls;
root/:
dir2  dir1  dir3
```

- Cd
  - Purpose: This command changes the current directory. If used with '~', it takes the user one step above, i.e. to the parent directory. If directory name is specified, it takes the user below in the specified child directory. If an absolute or relative path is given, it works accordingly. If a said directory doesn't exist, you will be directed to the last possible directory stated.
  - Usage:
    - *cd ~;*
    - *cd <directory name/ path and directory name>;*

- **openFile**

- Purpose: This command opens a file from current directory so that operations can be performed on it. The file can be opened in two modes: read or write. For read mode, enter mode as 'rf', and for write mode, enter mode as 'wf'. A file open in write mode can only perform write, delete and truncate operations and a file open in read mode can only perform read and read from specific position operation.
- Usage: *openFile <filename> <mode>;*

```
Enter your commands:
openFile test1 rf;
root/:
The file has been opened.
#
```

```
Enter your commands:
openFile test1 wf;
root/:
The file has been opened.
#
```

- **readFile**

- Purpose: This command is used to read the content of the currently opened file. The *openFile* needs to be called before this and mode should be set as 'rf'.
- Usage: *readFile;*

```
Enter your commands:
openFile test1 rf;
root/:
The file has been opened.
#
Enter your commands:
readFile;
root/:
This is test file number 1
#
```

- **readFromSpecificPosition**

- Purpose: This command is used to read content from a specific position for a specific number of characters. The *openFile* needs to be called before this and mode should be set as 'rf'.
- Usage: *readFromSpecificPosition start characters;*

```
Enter your commands:
readFromSpecificPosition 4 6;
root/:
s is t
#
```

- **writeFile**

- Purpose: This command is used to write to currently existing file. After the command is run, user is prompted to enter the data to be entered. The data is appended at end of existing content. The *openFile* needs to be called before this and mode needs to be set as 'wf'.
- Usage: *writeFile "<data>;"*

```
Enter your commands:
openFile test1 wf;
root/:
The file has been opened.
#
Enter your commands:
readFile;
root/:
Command denied: the file is opened in write mode. Open file in read mode to perform command.
#
Enter your commands:
writeFile "append";
root/:
#
```



- `truncateFile`

- Purpose: This command is used to truncate data from opened file from the specified size onwards. The *openFile* needs to be called before this and mode should be set as 'w+'.
- Usage: *truncateFile <size integer>;*

```
Enter your commands:
truncateFile 12;
root/:
The file being truncated: fileTest
#
```

- `deleteFile`

- Purpose: This command deletes the opened file from the directory along with all its data. The file has to be opened in 'w+' mode for this operation.
- Usage: *deleteFile;*

```
Enter your commands:
openFile test1 w+;
root/dir1/:
The file has been opened.
#
Enter your commands:
deleteFile;
root/dir1/:
File being deleted: test1

The file has been deleted.
#
```

- `closeFile`

- Purpose: This command closes a file that is opened, and no further read/write operations can be performed.
- Usage: *closeFile;*

```
Enter your commands:
closeFile;
root/:
File closed.
#
Enter your commands:
```

- `moveFile`

- Purpose: The purpose of this command is to change the directory of a file. If the said directory is the destination, write current. If the said directory doesn't exist, it will go to the previous directory for move operation. Moving file with same name in destination is not allowed.
- Usage: `moveFile <file> <destination directory>;`

```
Enter your commands:
moveFile file1 dir3;
root/:
The source directory found for the file: root

Directory where file is being moved: dir3
File moved successfully.
#
Enter your commands:
memoryMap;
root/:

The entire file system has the following heirarchy:
root/: dir2 dir1 dir3

|> dir2:
|> dir1:
|> dir3:  file1.txt
```

- `memoryMap`

- Purpose: This command displays the entire structure of directories and files.
- Usage: `memoryMap;`

```
Enter your commands:
memoryMap;
root/:

The entire file system has the following heirarchy:
root/: dir2 dir1 dir3

|> dir2:
|> dir1:
|> dir3:  file1.txt

#
```