Networks and Systems Security
Assignment 0

By Mayank Mohindra 2015056

System Description

The assignment presents a simulation of a well defined file system supporting the creation and manipulation of files according to certain permissions and privileges.

The server is a multithreaded TCP server that listens on user defined port and is responsible for listening multiple clients at the same time catering to their requests.
It is capable of understanding a set of commands that the client is sending, and generate suitable response for the same.

General Assumptions:

A. Any file name has a single '.'
B. Directory names cannot have any '.'
C. If a user does not exists then the server prompts for the creation of a new one
D. Each of the user's username has to be unique (thus avoiding the use of userID).
E. No file or directory can have names ending with '.m' or '.d'
F. Arguments cannot have size more than 294
G. / and simple_home have group name associated with all which allows all users to read the directory
H. Check for extra spaces and tabs in the input from client and remove them

---

1. **`"ls"`**

"ls" takes a single argument which is a absolute/relative path of the directory that we want to list.

Assumptions:

- ls takes exactly one argument (the directory path absolute or relative)
- ls command runs on directory only with read permissions

Errors:

- Error: Could not find directory
- Error: Unauthorised Access
- Error: No permissions to read this directory

Examples:

- ➔ `ls .`
- ➔ `ls /simple_home/u1/f1/`

2. **“fput”**

“fput” takes a single argument of the absolute/relative path of the file that need to be created/appended.
In case of a new file, fput asks the client to type in the owner and group information and thus create a metadata file, along with providing the creator of the file with default options inherited from the parent directory the file is being created in. The user need certain permissions of the file while appending to the created file and certain permission for the directory when creating a new one.

Assumptions:
- ➔ fput quit from file input output using 'Q\n'
- ➔ fput to directory only when we have write permissions
- ➔ fput to a new file takes owner and group first time and no other time
- ➔ fput to a new file takes input from the user creating the file regardless of the owner of the file.

Errors:
- ➔ Error: Incorrect path to file
- ➔ Error: Unauthorised Access
- ➔ Error: File cannot have .m or .d extension
- ➔ Error: No permissions to write to this file
- ➔ Error: User/Group typed does not exist
- ➔ Error: No write permissions for the directory

Examples:
- ➔ `fput f1.txt`
- ➔ `fput d1/f1.txt`

3. **“fget”**

“fget” takes a single argument of the absolute/relative path of the file that need to be sent to the client console. This is possible only when the requesting client has the read privileges of the file.

Errors:
- ➔ Error: Incorrect path to file
- ➔ Error: Unauthorised Access
- ➔ Error: No permissions to read this file
- ➔ Error: No such file exists

Examples:
- ➔ `fget f1.txt`
- ➔ `fget d1/f1.txt`

4. **“mkdir”**

Just like fput for new files, “mkdir” can be used for creating new directories. In a similar manner the client gets to type in the owner and group for the directory with the default options derived from the parent directory.
One can make a directory only when one has the write permissions for the parent directory.
Assumptions:

- ➔ *The parent directory must exist.*

Errors:

- ➔ Error: Could not find directory
- ➔ Error: Unauthorised Access
- ➔ Error: Invalid directory name
- ➔ Error: Unable to make directory
- ➔ Error: User/Group typed does not exist

Examples:

- ➔ `mkdir d1`
- ➔ `mkdir /simple_home/u1/d2/`
- ➔ `mkdir d1/d2`

5. **“cd”**

“cd” provides the clients with a simulation of changing the directory. This means that the client on a particular thread handler appears to be changed to some other directory with relative paths from that directory only.
One can change to a directory of which one has the read permissions.

Assumptions:

- ➔ cd to a directory if we have read permissions

Errors:

- ➔ Error: Could not find directory
- ➔ Error: Unauthorised Access
- ➔ Error: No permissions to change to this directory

:

- ➔ `cd ..`
- ➔ `cd ../../..`
- ➔ `cd /simple_home/u2/d4`

Vulnerabilities Handled

Multiple Logins from the same user:
Any user must not be allowed to login to the system at the same time. This means that a user once logged in cannot login again unless logged out.
For this a list is used
Every time a user logs in, we insert the user in the list.
Before logging in, one has to check the username existing in the loggedUsers.
After logging out, one has to remove the username from the list.

```
static list<string> loggedUsers;
```

Buffer Overflow Attack
Limiting the client and server side buffers to 300, because of input validation on the size, constraint on maximum number of bytes to be taken in input.

Login brute force Attack
One can launch a brute force on the server by testing different usernames.
To prevent from this attack, we have made an interactive session, so that a non existent user has the option to create user and thus enter the system. This is because of the assumption to create the user.

Directory Enumeration
Our assumption is that we are not allowed to list a directory if we don't have read access to the directory.
If a file is does not exist, then the error message will be file not found and if it exists, then the error message will be 'no read permissions'. This means that, one can brute force the contents of the directory, and thus list it without the read permissions.
To solve this we suggest to change the error messages.