

Document for CS513 Course Project

TCP Network Storage

Team members: Pei Zhang, Ya Liu, Hao Zhou

Date: 04/15/2013

1. Files Introduction

There are 11 source code files:

1. **Client.c** > main file for Client
2. **Server.c** > main file for Server
3. **ClientFunc.c** > some important functions for Client wrapped here (Especially functions for commands)
4. **ClientFunc.h** > head file for ClientFunc.c
5. **ServerFunc.c** > some important functions for Server wrapped here (Especially functions for commands)
6. **ServerFunc.h** > head file for ServerFunc.c
7. **Common.c** > format check for commands (Both Client and Server use this file)
8. **Common.h** > head file for Common.c
9. **wrapsock.c** > wrap socket functions with error check inside, so when use socket function, we will no long do error check
10. **wrapsock.h** > head file for wrapsock.c
11. **protocol.h** > functions for messages, packets, frame, streams, and functions used to convert one to another. Data Link layer, Network Layer and Physical Layer are implemented here.

2. How to run it

1. Compile:

- Find the right directory and type "**make**" into Terminal, and then you can get the executable files named Client and Server.
- You can type "**make clean**" to remove the object files and executable files which are generated by command "make", so that you can re-compile this project.

2. Run:

- Run **Server**: Type [./Server] to run server
- Run **Client**: Type [./Client IP_Address] to run client, then you can input different command to do different things.

3. Commands Support

1. Register userName password
2. Login userName password
3. ModifyPwd oldPassword newPassword
4. Download fileName
5. Delete fileName
6. Synchronize
7. List
8. Help
9. Quit
10. Send fileName
11. Clear
12. clsRecords
13. show
14. Recv (*additional command*)

1. Commands (no file names) are **case insensitive.*

**2. Before logging in to the system, the user can only use "help", "register" and "login".*

Register: If you do not have your account, you can use "Register" command to create one for yourself (Be sure to remember your password).

When "Register" command works,

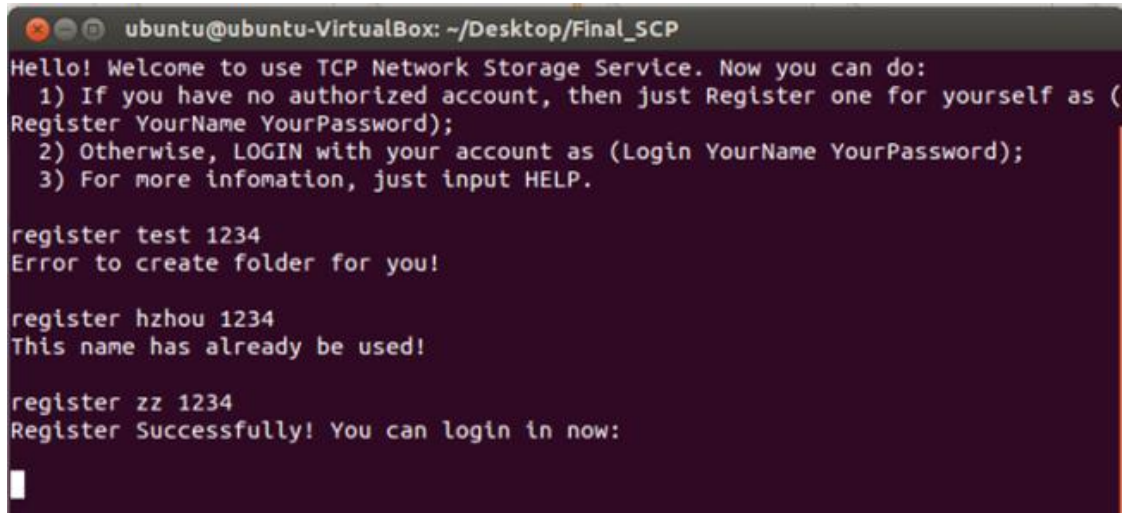
Client:

- 1). Creates a hidden file named .fileInfo.dat in the current folder.

Server:

- 1). Checks name to find whether it is registered. If it is, then returns message with "This name has already been used. Otherwise goes to step 2. (***Note**: "Error to create folder for you!" won't happen in normal use. As it only happens when there is a file whose name is the same with the name registered.)
- 2). Creates a folder for that user (folder name is the same as user name). And save the

password in the .passWord.dat (hidden file) into that folder created. At last, server returns "Register Successfully! You can login in now:" to client.



```
ubuntu@ubuntu-VirtualBox: ~/Desktop/Final_SCP
Hello! Welcome to use TCP Network Storage Service. Now you can do:
  1) If you have no authorized account, then just Register one for yourself as (
Register YourName YourPassword);
  2) Otherwise, LOGIN with your account as (Login YourName YourPassword);
  3) For more infomation, just input HELP.

register test 1234
Error to create folder for you!

register hzhou 1234
This name has already be used!

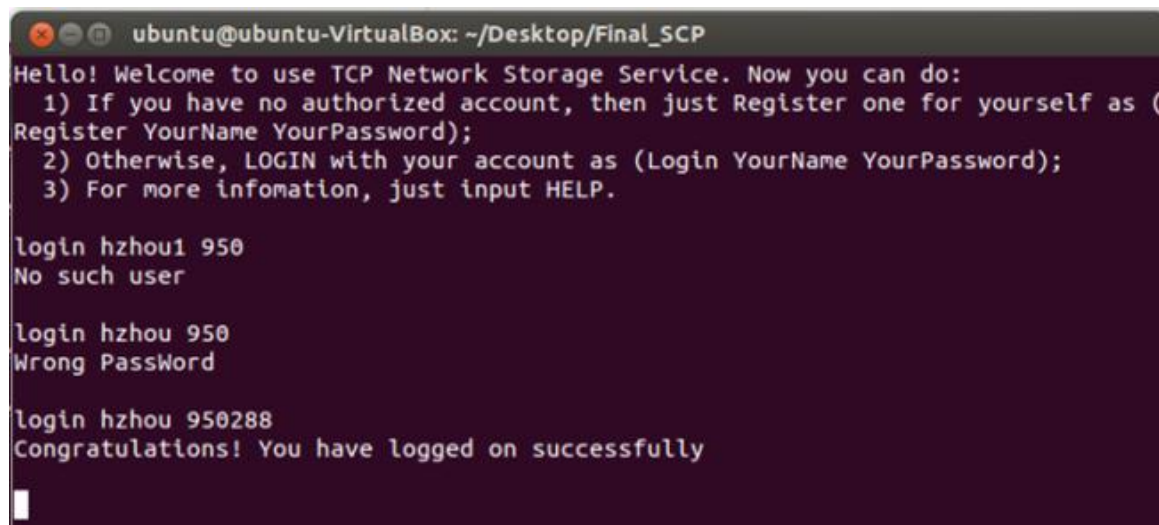
register zz 1234
Register Successfully! You can login in now:
█
```

Figure for Register

Login: If a user wants to enjoy the services provided by TNS, he must log on it with his account.

Server: Check the user name whether it is available (If there is a folder name same as the user name, it means the user name is available). If it is available, the server will get the password from .passWord.dat, so that the server will know whether the password from login command is correct or not. One of three messages will be returned to client as below:

1. *Congratulations! You have logged on successfully*
2. *Wrong Password*
3. *No such user*



```
ubuntu@ubuntu-VirtualBox: ~/Desktop/Final_SCP
Hello! Welcome to use TCP Network Storage Service. Now you can do:
  1) If you have no authorized account, then just Register one for yourself as (
Register YourName YourPassword);
  2) Otherwise, LOGIN with your account as (Login YourName YourPassword);
  3) For more infomation, just input HELP.

login hzhou1 950
No such user

login hzhou 950
Wrong PassWord

login hzhou 950288
Congratulations! You have logged on successfully
█
```

Figure for Login

ModifyPwd: You know, when you feel bored with your password, you can change it with this command. Possible messages returned as below:

1. *Modify successfully!*
2. *Your old password is wrong!*

```
modifypwd 950288 123456
Modify successfully!

modifypwd 950288 234567
Your old password is wrong!
```

Figure for Modifypwd

Download: Since Server will never send Client files automatically, so you need this command to download files from Server.

Delete: Delete the file in the server side, as you like. . Possible messages returned as below:

1. *Delete Successfully!*
2. *Delete failed!*

```
delete Server.o
Delete Successfully!

list

delete zzz
Delete failed!
```

Figure for delete

Synchronize: *[We recommend you re-login after this command executed]*

1. Client reads the information from .fileInfo.dat, which contains file name, file size and file last modified time. And then load the information into a linked list.
2. Client scans the files in the folder, if it finds one file in the folder doesn't have information in .fileInfo.dat, which means this file is newly added, so client uploads this file to server. If the client can find the information from .fileInfo.dat to the file in the folder, but with different last modified times, the client will also upload this file to server.
3. At last, the client will update the .fileInfo.dat.

List: See your files (file name and file size) in the server side.

```
list
1. test~          14
2. 2.jpg          176181
3. ClientFunc.c   9860
4. Server.o       15188
5. Client.o       15440
6. Client.c       5502
7. ClientFunc.h   648
8. ClientFunc.o   8820
-----
```

Figure for list

Help: If you don't this system well, you can use "help" command to help you know it better. Just try it.

```
help
----help information----
1. Register userName passWord
2. Login userName passWord
3. ModifyPwd oldPassword newPassword
4. Synchronize
5. List
6. Download fileName
7. Delete fileName
8. Help
9. Quit
10. Send fileName
11. clear(clear the screen)
12. clsRecords(clear local records, just try)
13. show(a list of frame statistics)
-----
```

Figure for help

Quit: After you finish everything, you can exit with "quit" command.

```
quit
Good Bye!
ubuntu@ubuntu-VirtualBox:~/Desktop/Final_SCPS
```

Figure for quit

Send: Send single file to Server directly, without checking .fileInfo.dat.

```
send zzx
No such File
send test
Send Successfully
```

Figure for send

Clear: Clear everything from the screen. --system("clear")

clsRecords: Remove records from .fileInfo.dat, so "Synchronize" command will upload *every* file to Server.

```
clsrecords
Your records are cleaned.
Now you can synchronize from scratch!
```

Figure for clsRecords

Show: Show the statistics of the frames

```
show
=====
# of frame sent:      4
# of frame recv:     47
# of frame erro:      0
=====
```

Figure for show

Recv: This is an additional command to clean the recv() buffer of socket. Because, when "Synchronize" is executed, maybe multiple files will be uploaded to the server in one time. As the server will reply to the client each time the server receives a file. So the client will leave a lot of information in the recv() buffer of socket, which will interfere the next command to be executed. Here, we recommend the user re-login after "Synchronize" command, but if the user won't do this, he can use "Recv" command to do some help. But this won't always work.

4. Design changes made since the original design report

We made a lot of changes, as when we met the professor for the second time, the professor said our project was so complex, and recommended us to ***delete the function of files sharing***. So, we did the changes as below:

1. We use txt file to replace MySQL. Since we won't implement files sharing, we made a decision that there was no need to use MySQL in our project any more.
2. "Synchronization" is implemented **by command**, which means the Client won't do it automatically every time interval set.
3. We add some commands, which we think is useful to users.

5. Accomplished Functions

1. We implemented all the commands described above. (*But sometimes "synchronize", "download" break down. [Partially work]*)
2. We implemented the process: message -> packet -> frame -> stream (string) -> frame -> packet -> message. During the process, we also did **stuffing** and added **Error Detection** field (*It is done in physical layer*) to frame. (No Go Back N in final program, but we give an extra demo for that.)

6. Unaccomplished Functions

1. "Go Back N" works partially. (I believe that it happened because our wait_for_event() function couldn't work well.)
2. We didn't add "Go Back N" to our final project, but we can show a demo about our "Go Back N".

7. Our work for Go Back N

We are so sorry that we couldn't add this feature into my final program, because it only partial works.

Ideas:

The application layer just cares when to send and when to receive, and doesn't tell the data link layer what to send or receive. So, when application layer needs to send something, it just calls Data_Link_Layer_Send(), then the data link layer will fetch what it needs actively. And when application layer needs to receive something, it will call Data_Link_Layer_Recv(), then the data link layer will get message from socket and deliver it to the application layer[In this demo, we take a file transmission as an example].


```

    if((stream = fopen("server.jpg","rb"))==NULL)
    {
        printf("The file was not opened! \n");
        exit(1);
    }
    Data Link Layer Send();
    printf("Server\n");
    fclose(stream);
    close(sockfd);
}

close(newsockfd);

```

Figure 7.1 Data_Link_Layer_Send()

```

    if((stream = fopen("fromServer.jpg","wb"))==NULL)
    {
        printf("The file was not opened! \n");
        exit(1);
    }
    Data Link Layer Recv();
    printf("Recieve File From Server[%s] Finished\n", argv[1]);
    fclose(stream);
    close(sockfd);

```

Figure 7.2 Data_Link_Layer_Recv()

Problems:

1. When I set MAX_SEQ=6, the sender will always lost the last 4 ACK frames from receiver, even I try to make receiver send the last ACK repeatedly (In figure 7.3, we can find the sender sends 937 frames, but just receives 933 ACK frames, and the last 4 ACK frames are lost. There are 13 frames timeout, because I set the program that if it receives 13 frames timeout, it will break, so the program won't wait forever.)
2. According to phenomenon 1, we believe that we have no efficient wait_for_event() function.

Pictures:

```

Even : TimeOut seq=2      ACK_Send=3      COUNT=928
Even : TimeOut seq=2      ACK_Send=4      COUNT=929
Even : TimeOut seq=2      ACK_Send=5      COUNT=930
Even : TimeOut seq=2      ACK_Send=6      COUNT=931
Even : TimeOut seq=2      ACK_Send=0      COUNT=932
Even : TimeOut seq=2      ACK_Send=1      COUNT=933
Even : TimeOut seq=2      ACK_Send=2      COUNT=934
Even : TimeOut seq=2      ACK_Send=3      COUNT=935
Even : TimeOut seq=2      ACK_Send=4      COUNT=936
Even : TimeOut seq=2      ACK_Send=5      COUNT=937
=====
# of frame Sent:          937
# of frame rcv:           933
# of frame erro:          0
# of frame TimeOut:       13
# of NWL disabled:        467
=====
Server
[
Recieve File From Server[127.0.0.1] Finished
ubuntu@ubuntu-VirtualBox:~/Desktop/File$

```

Figure 7.3 Go Back N

About Us

We are team 10.

If you have anything confused, do not hesitate to contact us:

hzhou@wpi.edu

yliu8@wpi.edu

pzhang@wpi.edu