

# Project 1 UDP socket programming

## 1. Use-base requirements:

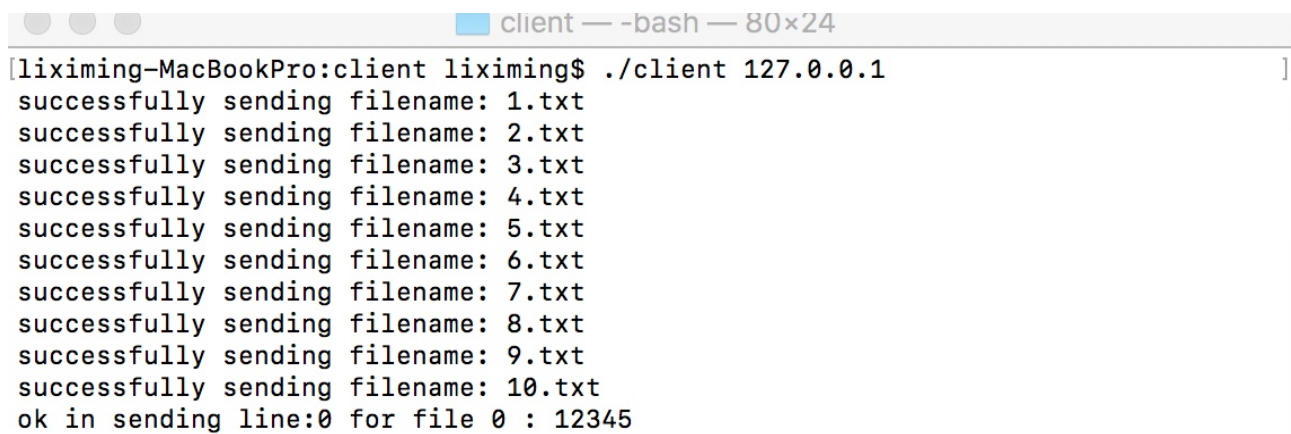
During normal transmission, the UDP server is listening the port 7777 and waiting for client to send data segments.

```
bzero(&server, sizeof(server));  
server.sin_family=AF_INET;  
server.sin_port=htons(7777);
```

And I also finished the one extra credit that the server registers itself to a register server listening at 8888. So the server will do the registration at start up as below:

```
address_sever.sin_port=htons(8888);  
A->cmd=REG;
```

In the use-case requirement, we can execute client with the ip address follow behind. During this process the client will read 10 file names form config.txt. As it already provide the ip address of server. It will not ask register center for register ip address.

A terminal window titled 'client — -bash — 80x24' showing the execution of a client program. The user runs './client 127.0.0.1' and the program outputs ten lines of 'successfully sending filename: X.txt' for X from 1 to 10, followed by 'ok in sending line:0 for file 0 : 12345'.

```
client — -bash — 80x24  
[lixi ming-MacBookPro:client lixi ming$ ./client 127.0.0.1  
successfully sending filename: 1.txt  
successfully sending filename: 2.txt  
successfully sending filename: 3.txt  
successfully sending filename: 4.txt  
successfully sending filename: 5.txt  
successfully sending filename: 6.txt  
successfully sending filename: 7.txt  
successfully sending filename: 8.txt  
successfully sending filename: 9.txt  
successfully sending filename: 10.txt  
ok in sending line:0 for file 0 : 12345]
```

A random file (from ten available ones) is selected, and, from this file, a random number of lines (from 1 - 3) are selected for transmission to the server as required in the project. Once the client finishes all transmissions, the server concatenates all files and sends them back to the client as one large file, with all lines in order.

```
ser — server — 80x24
matchline--file 4, line 7,len 10
get line 8 for file 4: vb0-0=8

matchline--file 4, line 8,len 8
get line 9 for file 4:

matchline--file 4, line 9,len 1
get line 10 for file 4:
matchline--file 4, line 10,len 0
file 4 is over.
get line 13 for file 5: cbv

matchline--file 5, line 13,len 4
get line 14 for file 5: cvb

matchline--file 5, line 14,len 4
get line 15 for file 5: xcv

matchline--file 5, line 15,len 4
get line 16 for file 5: asder

matchline--file 5, line 16,len 6
get line 17 for file 5: 5867
```

```
ser — server — 80x24
start to sending Concatenfile...
ok sending line 0: 12345

ok sending line 1: 12

ok sending line 2: abcdefg

ok sending line 3: abxx

ok sending line 4: zz

ok sending line 5: yy

ok sending line 6: kk

ok sending line 7: this is 3.txt

ok sending line 8: good

ok sending line 9: bad

ok sending line 10: asfsf

ok sending line 11: sxcg
```

After that the server will waiting for next client request.

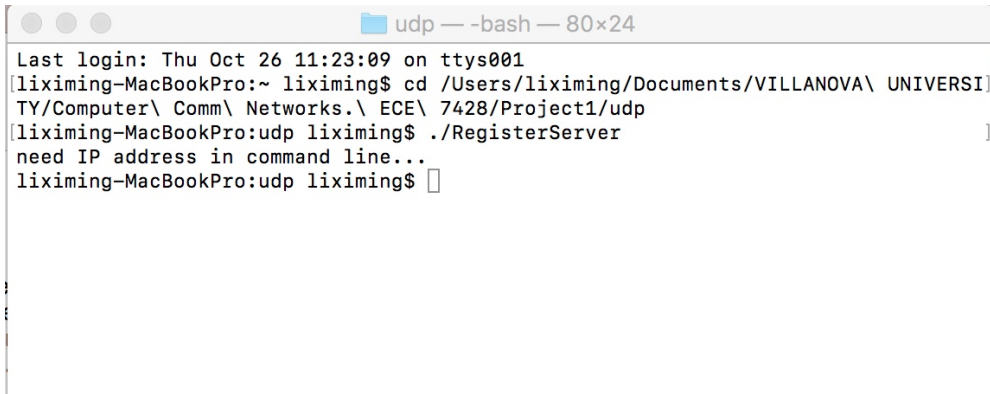
```
ser — server — 80x24
ok sending line 105: hnk
ok sending line 106: 890
ok sending line 107: k
ok sending line 108: gj
ok sending line 109: b
ok sending line 110: dser
ok sending line 111: fsr
ok sending line 112: 3f
ok sending line 113: dfg
ok sending line 114:

the concat file is sent over.
waiting for the next client
□
```

All above was test on my computer, I opened two terminal windows to finish the client and server process.

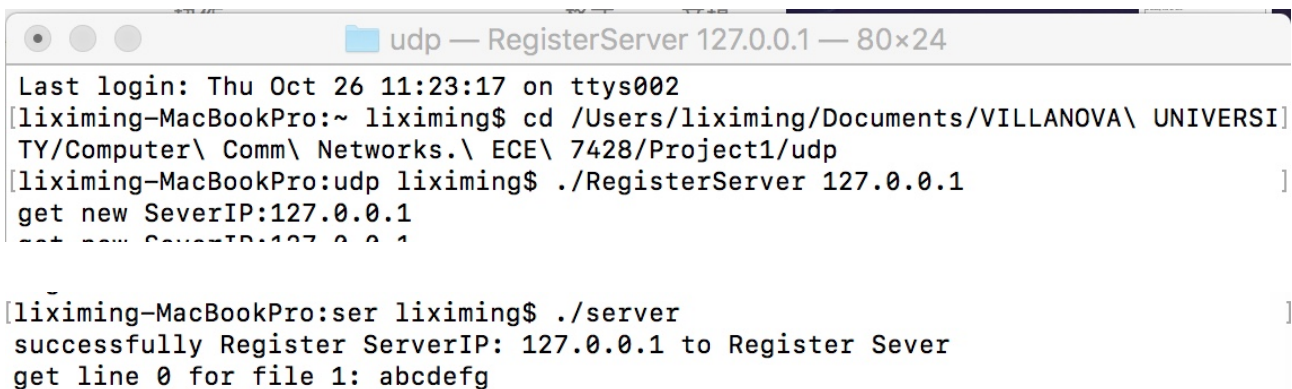
## 2. Extra credit requirement:

During this project, I only finished the first extra credit requirement-register server. When run the whole project, we will register server first.

A terminal window titled 'udp — -bash — 80x24'. The output shows the user navigating to the directory /Users/lixi ming/Documents/VILLANOVA\ UNIVERSITY/Computer\ Comm\ Networks.\ ECE\ 7428/Project1/udp and running the command ./RegisterServer. The program prompts for an IP address, and the user enters an empty line, resulting in the message 'need IP address in command line...'.

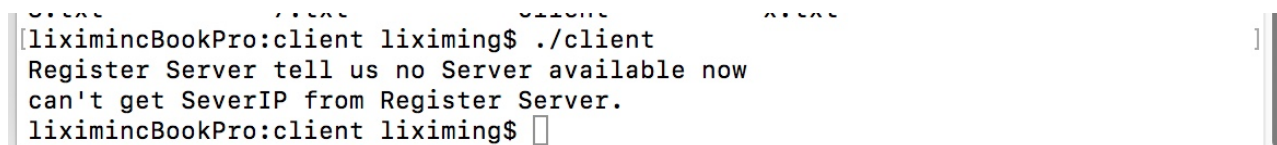
```
Last login: Thu Oct 26 11:23:09 on ttys001
[lixi ming-MacBookPro:~ lixi ming$ cd /Users/lixi ming/Documents/VILLANOVA\ UNIVERSITY/Computer\ Comm\ Networks.\ ECE\ 7428/Project1/udp
[lixi ming-MacBookPro:udp lixi ming$ ./RegisterServer
need IP address in command line...
lixi ming-MacBookPro:udp lixi ming$ ]
```

It will also give you some feedback if you type inappropriately. After that, when you run the server side, the server will read the registered IP address in config.txt to get it. There are also some interaction between Register server and data server.

A terminal window titled 'udp — RegisterServer 127.0.0.1 — 80x24'. The output shows the user running ./RegisterServer 127.0.0.1, which outputs 'get new SeverIP:127.0.0.1'. Then, the user runs ./server, which outputs 'successfully Register ServerIP: 127.0.0.1 to Register Server' and 'get line 0 for file 1: abcdefg'.

```
Last login: Thu Oct 26 11:23:17 on ttys002
[lixi ming-MacBookPro:~ lixi ming$ cd /Users/lixi ming/Documents/VILLANOVA\ UNIVERSITY/Computer\ Comm\ Networks.\ ECE\ 7428/Project1/udp
[lixi ming-MacBookPro:udp lixi ming$ ./RegisterServer 127.0.0.1
get new SeverIP:127.0.0.1
[lixi ming-MacBookPro:ser lixi ming$ ./server
successfully Register ServerIP: 127.0.0.1 to Register Server
get line 0 for file 1: abcdefg
]
```

A registry entry on the registry server is time out after an hour.

A terminal window showing the execution of the client command. The user runs ./client, and the program outputs 'Register Server tell us no Server available now' and 'can't get SeverIP from Register Server'.

```
[lixi ming-MacBookPro:client lixi ming$ ./client
Register Server tell us no Server available now
can't get SeverIP from Register Server.
lixi ming-MacBookPro:client lixi ming$ ]
```

## 3. Instruction

The programming language used in this project is C. The hardware platform is MAC OS Sierra 10.12.6. GCC was used to compile the code and executed on terminal.

First, you should change directory (cd) command to access the code file. Then you can use command: gcc xxx.c -o xxx to compile the code and generate the executive file.

Second, after the executive file was generated, use command “./ xxx” to run the program where “xxx” stands for the name of your executive file. If you run the register server first, be sure to add the IP address behind it or it will feedback some error messages.

Third, compile the server.c and “./” the server.

After the server side was established, you can do the same thing to execute the client and send files. There are some screenshots on how to run the program:  
Register server ip address:

```
udp — RegisterServer 127.0.0.1 — 80x24
Last login: Thu Oct 26 11:23:17 on ttys002
[lixi ming-MacBookPro:~ lixi ming$ cd /Users/lixi ming/Documents/VILLANOVA\ UNIVERSI
TY/Computer\ Comm\ Networks.\ ECE\ 7428/Project1/udp
[lixi ming-MacBookPro:udp lixi ming$ ./RegisterServer 127.0.0.1
get new SeverIP:127.0.0.1
set new SeverIP:127.0.0.1
```

After you run “./RegisterServe 127.0.0.1” it will not give you feedback until you tun the server side program, data server will read the IP address from Register server:

```
[lixi ming-MacBookPro:ser lixi ming$ ./server
successfully Register ServerIP: 127.0.0.1 to Register Sever
get line 0 for file 1: abcdefg
```

After the server provide IP address, we can execute client. The client will read the files’ names from config.txt file which is stored in the client folder.

```
client — -bash — 80x24
[lixi ming-MacBookPro:client lixi ming$ ./client
Register Server tells us serverip: 127.0.0.1
successfully sending filename: 1.txt
successfully sending filename: 2.txt
successfully sending filename: 3.txt
successfully sending filename: 4.txt
successfully sending filename: 5.txt
successfully sending filename: 6.txt
successfully sending filename: 7.txt
successfully sending filename: 8.txt
successfully sending filename: 9.txt
successfully sending filename: 10.txt
ok in sending line:0 for file 1 : abcdefg
```

“Register Serve tells us server ip: 127.0.0.1” will show up if you register server IP. If you don’t want to register server, you can provide the IP address of the server and run client:

```
client — -bash — 80x24
[lixi ming-MacBookPro:client lixi ming$ ./client 127.0.0.1
successfully sending filename: 1.txt
successfully sending filename: 2.txt
successfully sending filename: 3.txt
successfully sending filename: 4.txt
successfully sending filename: 5.txt
successfully sending filename: 6.txt
successfully sending filename: 7.txt
successfully sending filename: 8.txt
successfully sending filename: 9.txt
successfully sending filename: 10.txt
ok in sending line:0 for file 0 : 12345
```

After execute client, it will randomly send files as required in the project.

```
client — hash — 80x24
ser — server — 80x24

start to sending Concatfile...
ok sending line 0: 12345

ok sending line 1: 12

ok sending line 2: abcdefg

ok sending line 3: abxx

ok sending line 4: zz

ok sending line 5: yy

ok sending line 6: kk

ok sending line 7: this is 3.txt

ok sending line 8: good

ok sending line 9: bad

ok sending line 10: asfsf

ok sending line 11: sxcg
```

```
ser — server — 80x24

ok sending line 105: hnk

ok sending line 106: 890

ok sending line 107: k

ok sending line 108: gj

ok sending line 109: b

ok sending line 110: dser

ok sending line 111: fsr

ok sending line 112: 3f

ok sending line 113: dfg

ok sending line 114:

the concat file is sent over.
waiting for the next client
□
```

When the server receive them, server will concatenates all files and send a large file back to server.

## 4. User guide

The program is based on C language, and

- The folder ser/, the data server directory, it uses config.txt to initialize itself. config.txt contains the Register ServerIP, and data server IP.
- client/, the client directory, it reads config.txt to get the 10 files' name, and the Register ServerIP, if there is one line after the 10 files' name. you can also provide data server IP from command line, if so it will not ask Register Server for the data server IP. otherwise, it will ask Register Server for the data server IP. the concat file data will be saved as a file named concat.txt.