CSCI 5105 Project 1: PubSub

Group members:
Shengping Wu wu000376
Fushan Wang wang9550

A. Instructions explaining

I. Registry server (Provided by TA)

1) Compile : go into Registry/ directory and run `make all`.
2) Run: `./super_server`.
3) User input interface:
   There are two command lines syntaxes on the registry server side: `list` and `exit`
   `list` : To show the information of all the group servers that are connected to this registry server.
   `exit`: shut down this registry server.

II. Group server

1) Compile : go into Server/ directory and run `make all`.
2) Run: `./super_server registry_server_address`
   The only one argument of this command should be the ip address of the registry server.
3) User input interface:
   There are three command line syntaxes on the group server side:`Deregister`,`listS` and `listC`
   `Deregister`:leave from the registry server
   `listC:` Show all clients that are connected to this group server.
   `listS:` Show all group servers that are connected to the same registry server as the current group server.

III. Client

1) Compile : go into Client/ directory and run `make all`.
2) Run:`./communicate_client registry_server_address`.
   The only one argument of this command should be the ip address of the **registry** server.
   When you succeed run this, you should see a list of available group server information.

3) User input interface:
   There are five command line syntaxes on the group server side:`join / publish / subs / unsubs / leave.`
   You should first run join to join, and then input the ip address of one group server to join in. After that you can run the other commands.
   `Join:` client chooses one of the group servers and gets connected to it.
   `Publish:`client publishes one article.
   `Subs:`client subscribes to one kind of article.
   Once some other clients(or itself) publish relative articles, it will receive the article from the group server.
   `Unsubs:`client unsubscribes one kind of article.
   `leave:` client leaves this group server.

B. Design details

Our PubSub system is comprised of: clients, group servers, and a registry-server.

Firstly, the group server begins by registering itself with a registry server, whose IP is supposed to be known, via UDP. During running, the registry server will send a ‘heartbeat’ string to the group servers via

UDP and wait 5 seconds for a response. If there is no response, then it will remove this group server from the existing server list.

The client then can contact the registry server to get a list of all existing group servers via UDP and manually call `join` command on one of the group servers with its IP address. When a client decides to leave the server, it should call the `leave` command to leave the current group server. After that the client side will shut down itself.

When a group server that a client has joined on goes offline, the client will notice this change via a thread that keeps pinging the group server every 5 seconds. If such a case happens, the client will wait until the registry server removes this offline group server information and then fetch a current available group server list via UDP. After that, the client will join the first server shown in the list and restore connection between client and server.

In normal state, the client can call `subs`,`unsubs` or `publish` commands on the server. Clients communicate to their server by means of RPC. The maximum length of communication is restricted to 120 bytes, according to the restriction of the article length.

On the client side, when calling the above command, it will check first whether the input of the article is valid or not. If it is not a valid article format, the client side will print an error message and go back to wait for the user's input.

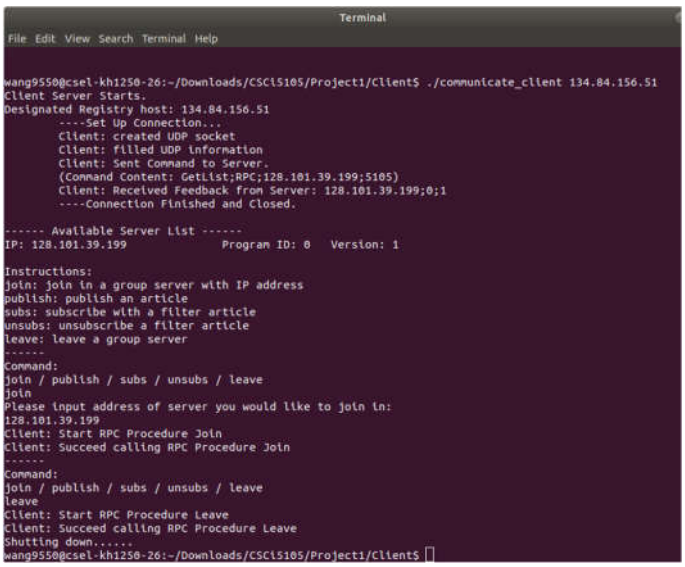
On the server side, we store all the clients information in a list `list<ClientInfo> g_clientList`.The class `ClientInfo` includes IP address, port and a list of all the subscriptions in a list `list<string> filter`. Once clients publish a new article, the group server determines the matching set of clients for a published article, and then propagates the article to each client via UDP. In order to wait for new articles, we devote a blocking thread in the client side to receive UDP information.

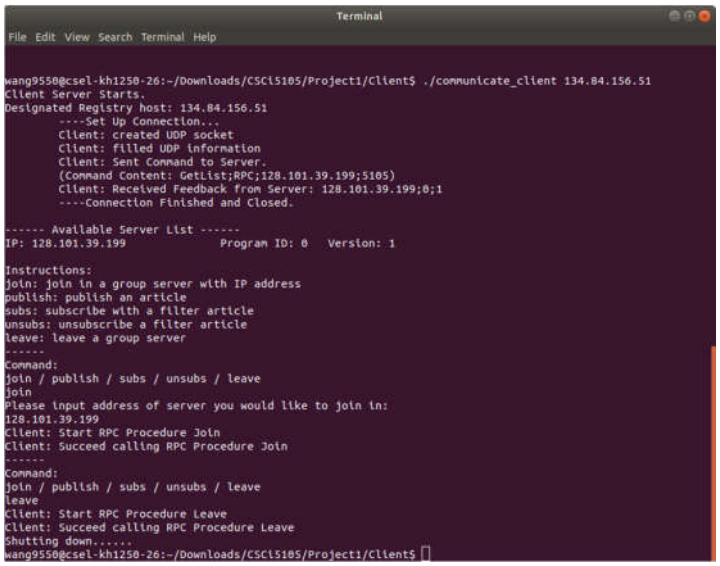
Files list:

1. Registry/ :
   stringTokenizer.h, stringTokenizer.cpp --tools
   super_server.cpp -- main function
   makefile
2. Server/ :
   client.h -- necessary data structure
   communicate.h,communicate_xdr.c -- generated by *rpcgen*
   communicate_server.c -- all the procedure stored in server for client calling
   communicate_svc.c -- main function
   makefile
3. Client/ :
   stringTokenizer.h,stringTokenizer.cpp --tools
   communicate.h,communicate_xdr.c,communicate_clnt.c --generated by *rpcgen*
   communicate_client.cpp - main function
   makefile

   The initial rpcgen file communicate.x is also included.

C. Test case



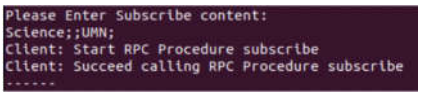


1. Join and leave (left) and Join + server offline case (right)

2. Subscribe, Unsubscribe and Publish.

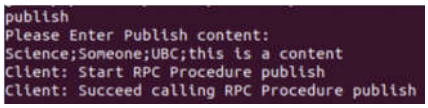
Subscribe - Client: Failed case

Subscribe - Client: Success case:



Publish - Client: Failed case

Publish - Client: Success case



Publish - Server sends and Client receives.

```
Instruction:
 listC / listS / Deregister
Client 128.101.39.220:5110 has already joined to the group server
Client 128.101.39.220:5110 add new subscription: Sports;;;
Client 128.101.39.199:5110 has already joined to the group server
Client 128.101.39.220:5110 add new subscription: Lifestyle;;;
Client 128.101.39.199:5110 is already registered to the server
Client 128.101.39.199:5110 add new subscription: Science;;UMN;
Read article content:
-Science--UMN-This is a content-
Send Article to Client 128.101.39.199:5110
Read article content:
-Lifestyle--UMN-this is also a content-
Send Article to Client 128.101.39.220:5110
Read article content:
-Science-Someone-UBC-this is a content-
Read article content:
-Sports---contents-
Send Article to Client 128.101.39.220:5110
```

```
receiving article ........
Lifestyle;;UMN;this is also a content
receiving article ........
Sports;;;contents
```

3. Error handle cases

Join - Invalid IP Address
Startup - Invalid IP Address

```
join
Please input address of server you would like to join in:
qqqqqqqqq
Error: not valid IP address
------
```

```
wang9550@csel-kh1250-26:~/Downloads/CSCi5105/Project1/Client$ ./communicate_clie
nt 128.34.34.34
Client Server Starts.
Designated Registry host: 128.34.34.34
        ----Set Up Connection...
        Client: created UDP socket
        Client: filled UDP information
        Client: Sent Command to Server.
        (Command Content: GetList;RPC;128.101.39.199;5105)
Error: Fail to find register server
: Resource temporarily unavailable
```

4. Known limitations.
   - We use the *checkip()* function provided by TA, we found that if we use ssh to different remote machines, it will return a dynamic IP address which is not the true static true IP of those remote machines, which will cause no response.
   - Part of our code used the stringTokenizer tool provided by TA.
   - The UDP port for communication is fixed in coded and will not be changed via command. Nevertheless it is sufficient to make this project run without Port Conflicts.
   - To make sure the whole communication system runs normal, it is required to start the registry server first, then start up the group server, and finally start up the client server.

D. Source code

Github: https://github.com/Inutto/CSCi5105.git (Files only included in “File list” above is useful)
Also available in the .zip file submitted.