Design Documentation of IRC Server (Final)

10302010023 Wang Xin

1. Brief Introduction

The standalone IRC server is a central server in the IRC network. It was designed to handle the instructions from the clients and transmit the appropriate messages in response. The project was mainly implemented in C language using library functions declared in “csapp.h”. It has been tested valid in Linux Debian 2.6.32(gcc 4.4.5) and Linux Ubuntu 3.5.0(gcc 4.4.7).

2. Data structure

Until now the server introduces three data structures: client, client pool and channel according to the need of the server, which are both defined in sircd.h. Here is the overview of the three structs(Figure 1).

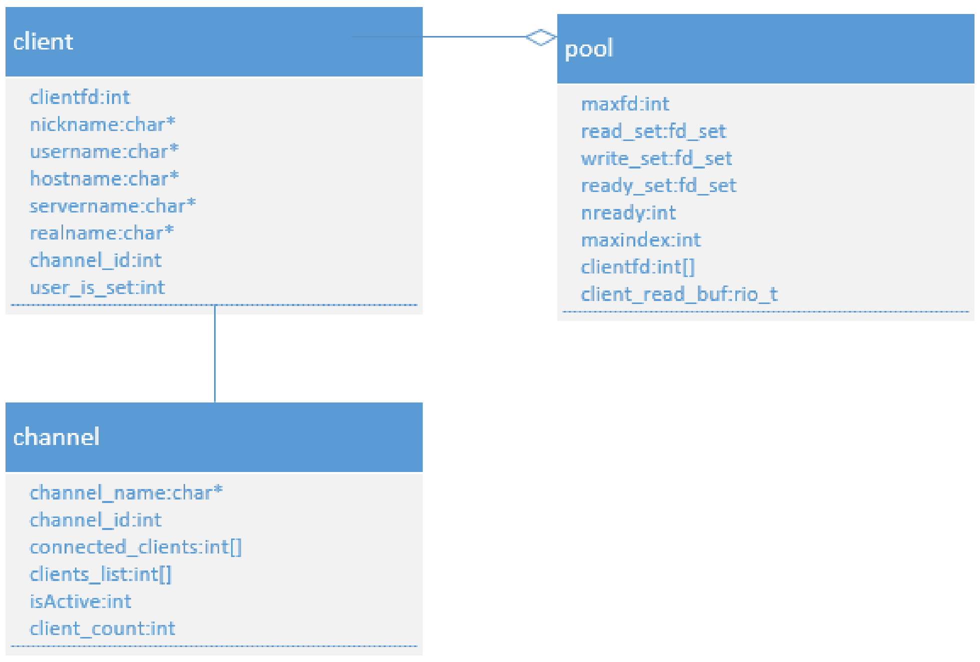
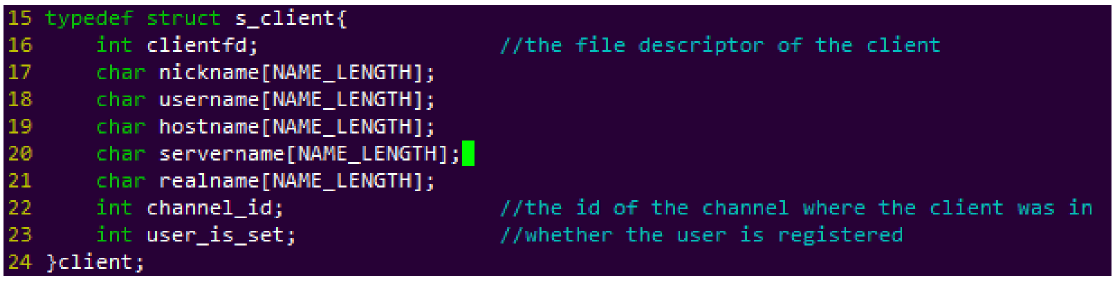


Figure 1 Structs

2.1. Client

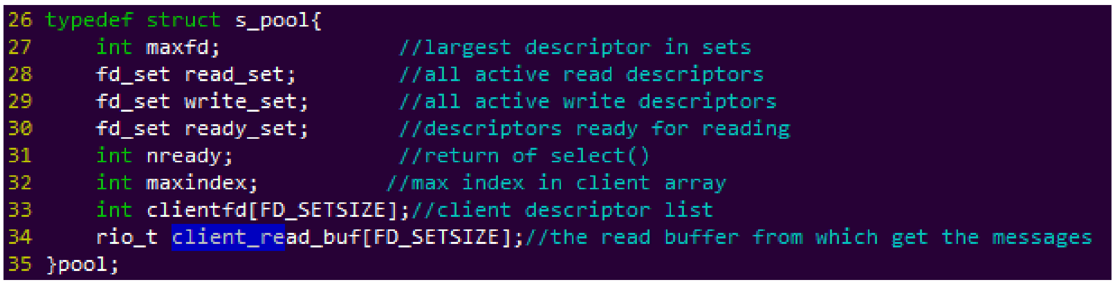


```
typedef struct s_client{
    int clientfd;                   //the file descriptor of the client
    char nickname[NAME_LENGTH];
    char username[NAME_LENGTH];
    char hostname[NAME_LENGTH];
    char servername[NAME_LENGTH];
    char realname[NAME_LENGTH];
    int channel_id;                 //the id of the channel where the client was in
    int user_is_set;                //whether the user is registered
}client;
```

Figure 2 Client

2.2. Pool

The client pool was used to store the client in this session.

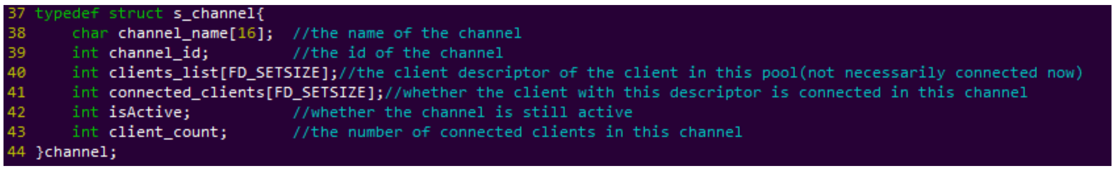


```
typedef struct s_pool{
    int maxfd;              //largest descriptor in sets
    fd_set read_set;        //all active read descriptors
    fd_set write_set;       //all active write descriptors
    fd_set ready_set;       //descriptors ready for reading
    int nready;             //return of select()
    int maxindex;          //max index in client array
    int clientfd[FD_SETSIZE];//client descriptor list
    rio_t client_read_buf[FD_SETSIZE];//the read buffer from which get the messages
}pool;
```

Figure 3 Pool

2.3. Channel

In this IRC server, a client can only join one channel. Joining in a new channel means departure from the old one. The channel uses a bit-vector to manage the connection state of each client. When the last client leaves the channel, the channel is no longer active and should be deleted from the global channel list.



```
typedef struct s_channel{
    char channel_name[16];  //the name of the channel
    int channel_id;         //the id of the channel
    int clients_list[FD_SETSIZE];//the client descriptor of the client in this pool(not necessarily connected now)
    int connected_clients[FD_SETSIZE];//whether the client with this descriptor is connected in this channel
    int isActive;           //whether the channel is still active
    int client_count;       //the number of connected clients in this channel
}channel;
```

Figure 4 Channel

3. Process

Here is the overall process of the server(Figure 5).

1) The server first create a socket descriptor (typically 3) and bind the socket descriptor with a local protocol. Then listen on the descriptor and get ready to accept connection requests.

These are compacted in the function Open_listenfd(). Some global variables such as client_pool, global_client_queue and global_channel_list was initialized.

2) Use function Select() to get the ready descriptor set. If the listening descriptor was ready for read, use function Accept() to create a socket descriptor for a new client (the first was typically 4).
3) Add the new client to the client pool.
4) Read the commands from each client, check the type and format of each commands and echo the appropriate message. The server will distinguish the commands according to the first several letters and check the parameter list, then dispatch the different commands to the corresponding handler.

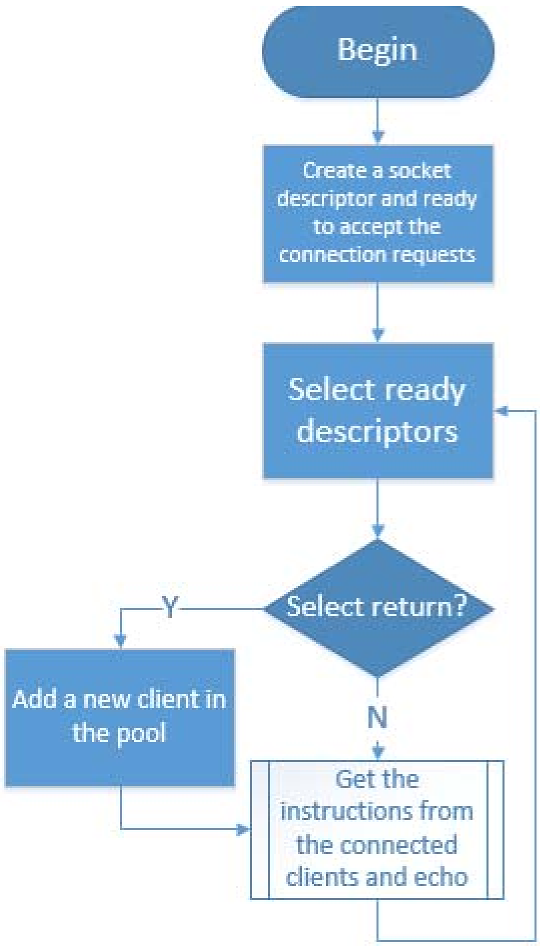

Begin
Create a socket descriptor and ready to accept the connection requests
Select ready descriptors
Select return?
Y
Add a new client in the pool
N
Get the instructions from the connected clients and echo

Figure 5 Overall Process

3.1. NICK

Command: NICK
Parameters: <nickname> [<hopcount>]

NICK message is used to give user a nickname or change the previous one. According to the protocol defined in RFC1459, the nickname should be unique and the operation should only be valid to registered users, however, to unregistered users, the server still set the nickname just without the success message (Figure 6). The length of all names (including nickname, hostname, servername, channel_name, etc.) was restricted less than 32.

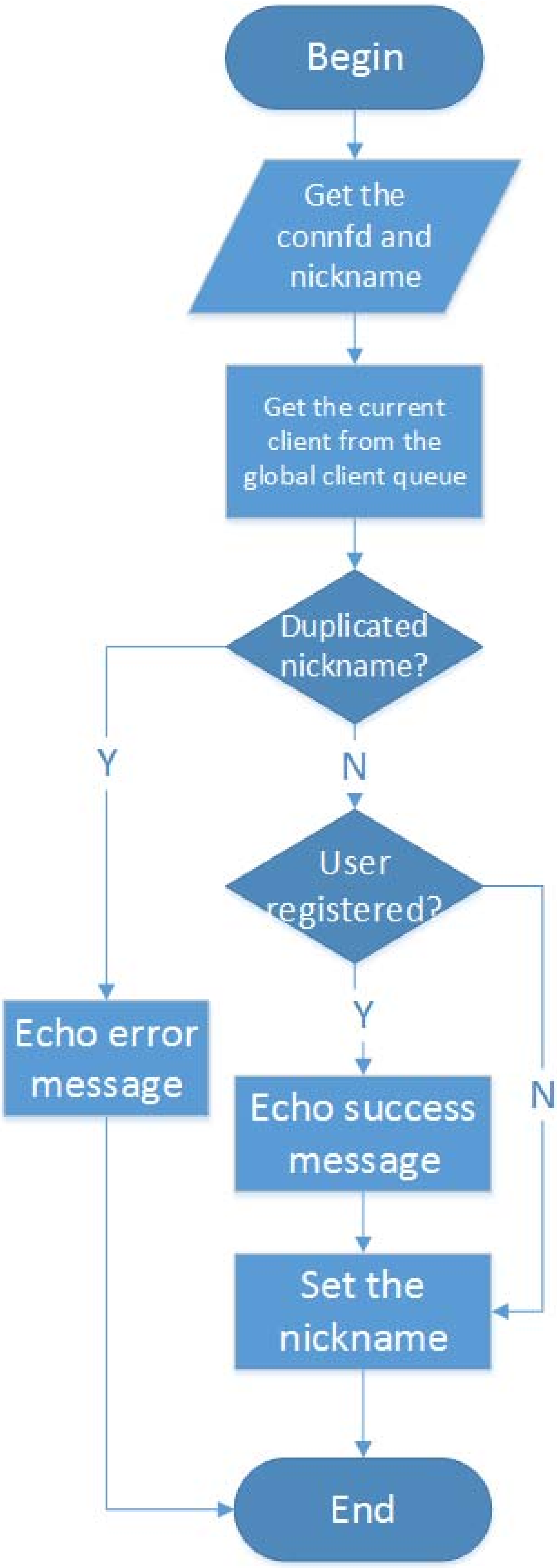


Figure 6 NICK

3.2. USER

Command: USER
Parameters: <username> <hostname> <servername> :<realname>

According to RFC1459, the USER message is used at the beginning of connection to specify the username, hostname, servername and realname of a new user. It is also used in communication between servers to indicate new user arriving on IRC, since only after both USER and NICK have been received from a client does a user become registered.

3.3. QUIT

Command: QUIT
Parameters: NULL

A client session is ended with a quit message. The server closes the connection to a client which sends a QUIT message. If a "Quit Message" is given, this will be sent instead of the default message, the nickname. The server must inform other users of the client's departure by echoing the QUIT command to all users sharing channels with the departing client, in the same manner as for a nickname change.

3.4. JOIN

Command: JOIN
Parameters: <channel>{,<channel>}

The JOIN command is used by client to start listening a specific channel. If the user is already on the channel, the message is silently ignored. Joining a new channel should implicitly cause the client to leave the current channel (Figure 7). If the channel does not exist, it is created automatically. The leaving step must precede the creation step, for the global channel list was a single-direction linked list.

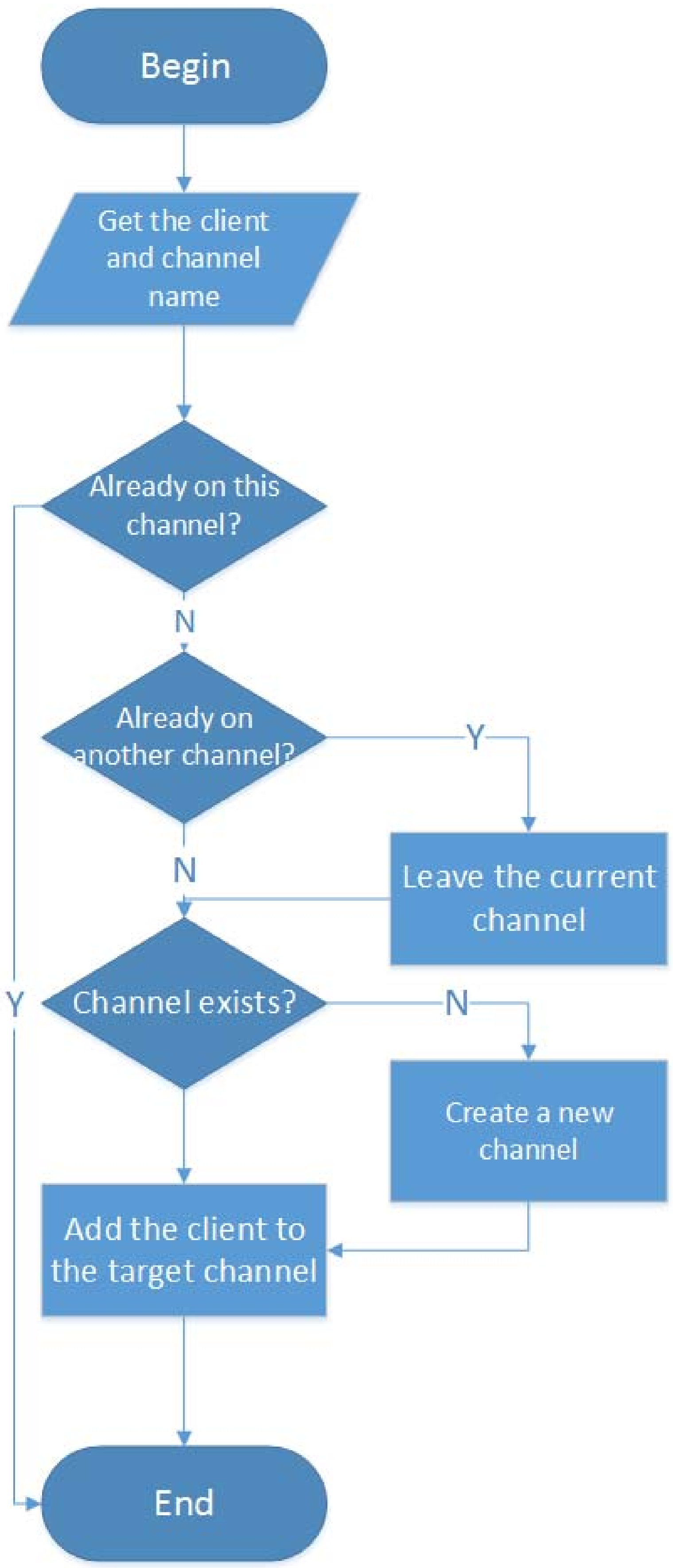

Begin
Get the client and channel name
Already on this channel?
N
Already on another channel?
Y
Leave the current channel
N
Channel exists?
N
Y
Create a new channel
Add the client to the target channel
End

Figure 7 JOIN

3.5. PART

Command:	PART
Parameters:	<channel>{,<channel>}

The PART message causes the client sending the message to be removed from the list of active users for all given channels listed in the parameter string (Figure 8).

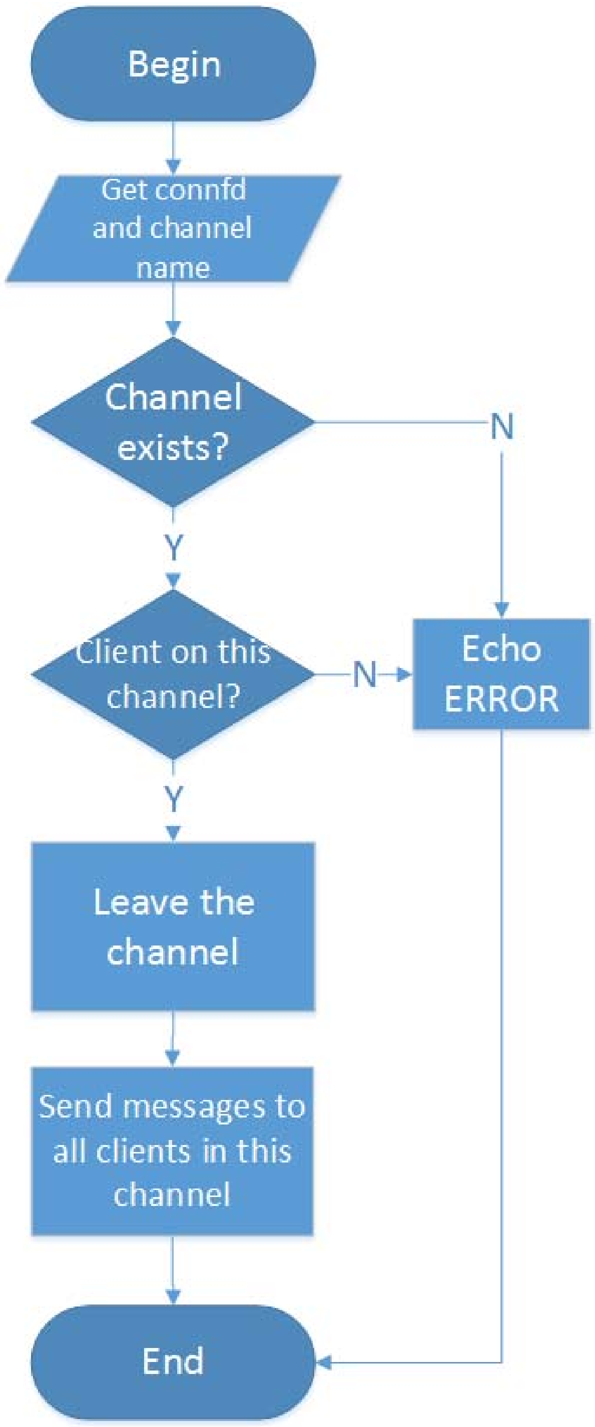


Figure 8 PART

3.6. LIST

Command: LIST
Parameters: NULL

According to RFC1459, the list message is used to list channels and their topics. If the <channel> parameter is used, only the status of that channel is displayed. Otherwise, list all channels. In this design, the server just simply ignore all parameters and list all channels to the client.

3.7. PRIVMSG

Command: PRIVMSG
Parameters: <target>{,<target>} <text to be sent>

PRIVMSG is used to send private messages between users. <target> is the nickname of the receiver of the message. <target> can also be a list of names or channels separated with commas (Figure 9). The length of messages was restricted less than 512.

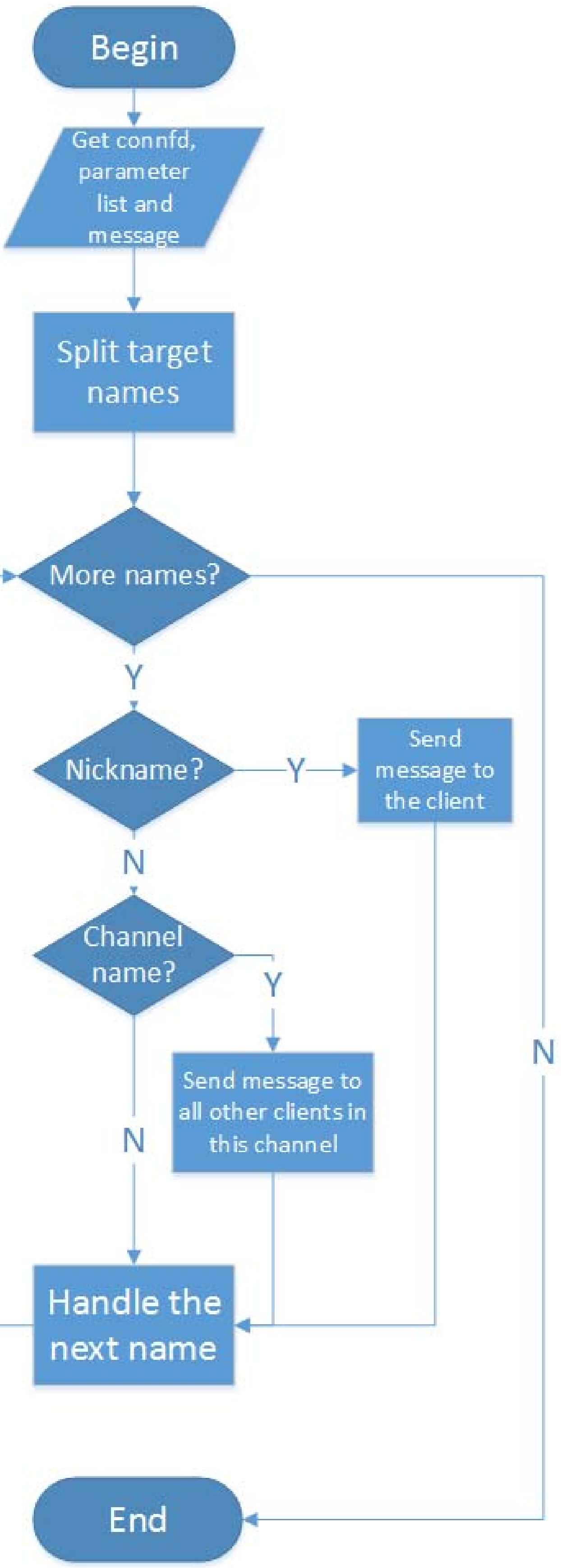

Begin
Get connfd, parameter list and message
Split target names
More names?
Y
Nickname?
Y
Send message to the client
N
Channel name?
Y
Send message to all other clients in this channel
N
N
Handle the next name
End

Figure 9 PRIVMSG

3.8. WHO

Command: WHO
Parameters: [<channel_name>]

The WHO message is used by a client to generate a query which returns a list of information which 'matches' the <name> parameter given by the client (Figure 10).

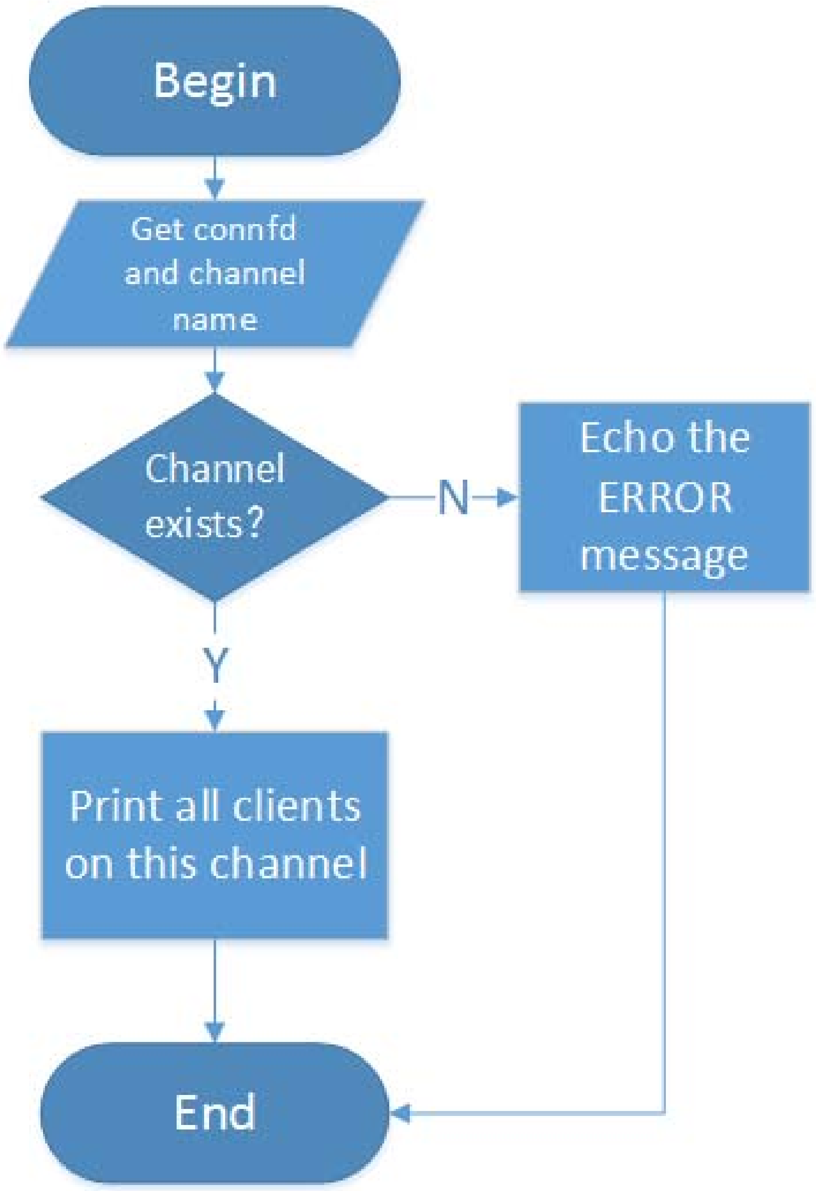

Begin
Get connfd and channel name
Channel exists?
N
Echo the ERROR message
Y
Print all clients on this channel
End

Figure 10 WHO

4. Verbosity

For debug use, I also set the verbosity level switch. “./sircd 1 node.conf –v <verbosity_level>” will print some vital debug information of the server, otherwise the server will run silently (Table 1).

VERBOSITY LEVEL	BEHAVIOR
0	Run silently
1	Print logging of users with signed on
2	Print logging of users, channels, messages

Table 1 verbosity level