COMPUTER NETWORK
PROJECT 1 –STANDALONE IRC SERVER

11302010067 Zhou Yuwei

2014/2/3

INTRODUCTION

1. ABSTRACT

I develop a developing concurrent network application in this project 1. The standalone IRC server assumes that there is only one server and all the clients connect to this one, and it merely support a subset commands of the real IRC server. It is implemented in C language include ‘csapp.h’ provided by CSAPP on Ubuntu.

2. STRUCTURE

This concurrent event-driven server based on I/O multiplexing with three structures and two global arrays.

Consider the memory alignment, I set the MAX_NAME_LEN to 64, the multiple of word size.

```
user* user_list[FD_SETSIZE];
channel* channel_list[FD_SETSIZE];
```

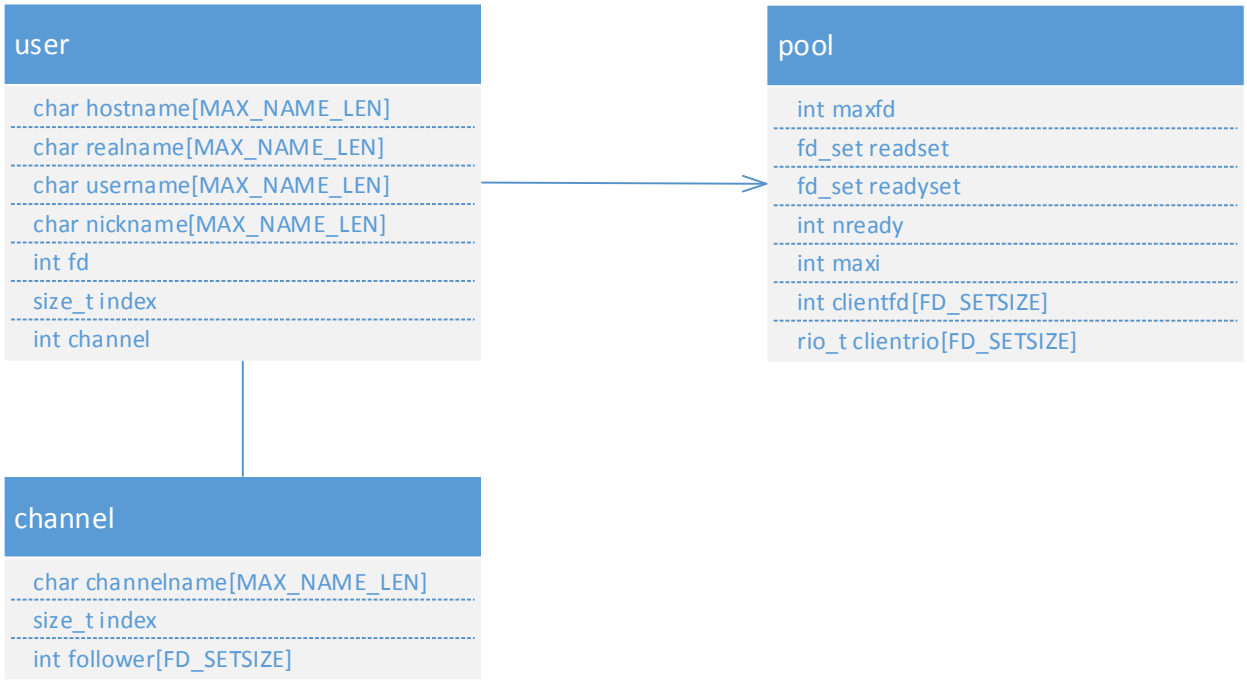

Figure 1 structure

a) Pool

```
typedef struct { /* represents a pool of connected descriptors */
    int maxfd;        /* largest descriptor in read_set */
    fd_set read_set;  /* set of all active descriptors */
    fd_set ready_set; /* subset of descriptors ready for reading  */
    int nready;       /* number of ready descriptors from select */
    int maxi;         /* highwater index into client array */
    int clientfd[FD_SETSIZE];    /* set of active descriptors */
    rio_t clientrio[FD_SETSIZE]; /* set of active read buffers */
} pool;
```

Figure 2 pool structure

b) User

For a better memory management, I dynamically allocate memory for a user. When a new client connect to the server, it allocates a user for this client and initializes its all name to ANONYMOUS, the certainly fd and channel to -1. Then it sets the index and add the user to user_list, where the index is same to the index to the certainly fd in clientfd array of pool structure. When a client disconnect to the server, the server set the fd in clientfd of pool structure to -1, and free the memory of user by invoking Free method.

```
typedef struct {
    char hostname[MAX_NAME_LEN];
    char realname[MAX_NAME_LEN];
    char username[MAX_NAME_LEN];
    char nickname[MAX_NAME_LEN];
    int fd;         /* the fd of the user */
    size_t index;   /* index in the user_list */
    int channel;    /* the user's following channel, -1 if null */
} user;
```

Figure 3 user structure

c) Channel

When a user ask to follow a channel, if the channel is not exist, the server will allocates a channel and initiates the new channel.When the last user parts the channel, the channel will be free.

```
typedef struct {
    char channelname[MAX_NAME_LEN];
    size_t index;   /* index in the channel_list */
    int follower[FD_SETSIZE];   /* a list of the follower's index */
} channel;
```

Figure 4 channel structure

3. PROCEDURE

The set of active clients is maintained in a pool structure. After initializing the pool by calling init_pool(), the server enters an infinite loop. During each iteration of this loop, the server calls the select function to detect two different kinds of input events:

- A connection request arriving from a new client
- A connected descriptor for an existing client being ready for reading.

When a connection request arrives, the server opens the connection and calls the add_client function to add the client to the pool. Finally, the server calls the check_clients function to handle the command.

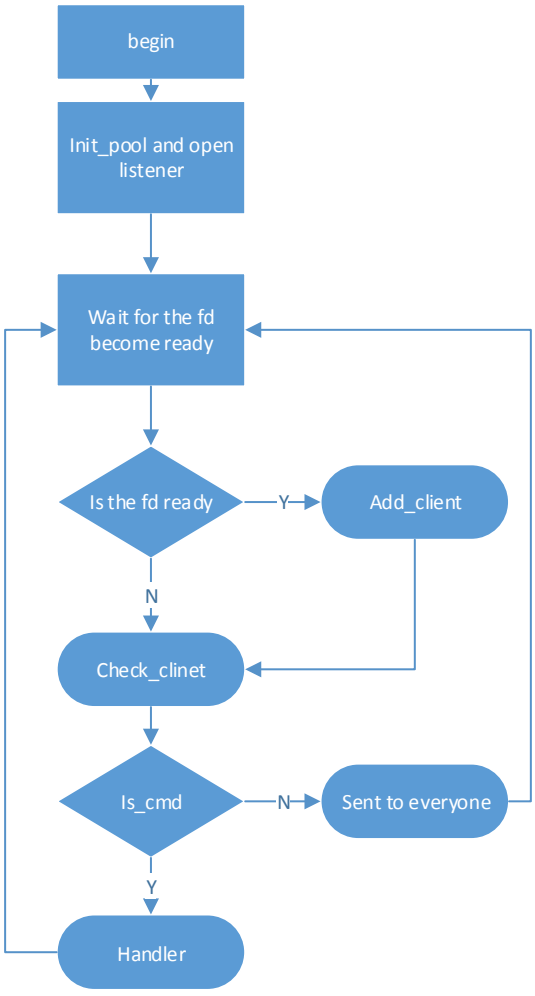

Figure 5 IRC server procedure

TEST AND GRADING

My implementation can pass the ruby script and the python one.