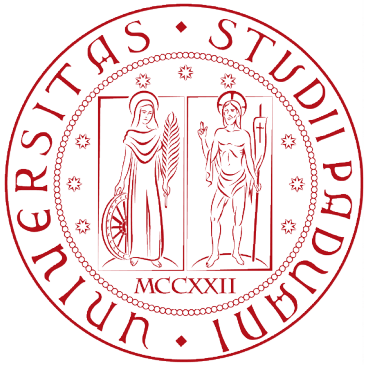
Univeristà degli studi di Padova

Dipartimento di ingegneria dell'informazione



IRC Server

Edoardo Vanin

6/07/2017

Obbiettivo

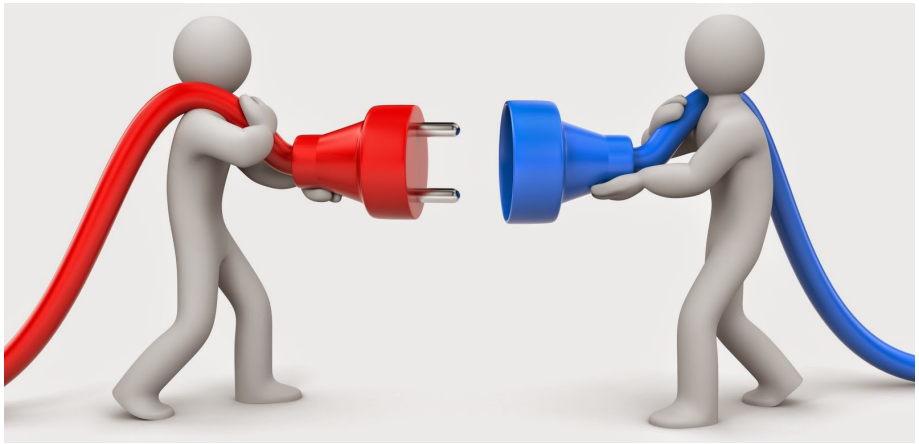
- JOIN
- PART
- PRIVMSG
- WHOIS
- QUIT
- NICK
- LIST

Implementazione

- JOIN
- PART
- PRIVMSG
- WHOIS
- QUIT
- NICK
- LIST
- WHO
- USER
- MODE
- PING

Architettura

Gestione connessione



Gestione utenti

Gestione connessione

```
91    while(1){
92      //accept non è all'interno dell'if di decisione perché è una funzione bloccante
93      if ((user_socket = accept(server_socket,
94                                (struct sockaddr *) &client_address,
95                                &client_address_size)) == -1 ) {
96        fprintf(stderr, "[Errore chiamando accept: %s]\n", strerror(errno));
97        exit(5);
98      }
99
100     // aggiunge l'utente alla lista se permesso
101     if(count < MAXUSER){
102       strcpy(username, "user"); //pulisco la stringa dal nome precendente
103       sprintf(temp,"%d", u_id);
104       strcat(username, temp);
105       new_user = create_user( username,
106                               inet_ntoa(client_address.sin_addr),
107                               u_id,
108                               user_socket);
109
110       u_id = u_id + 1; //incrementa il contatore degli id
111       pthread_mutex_lock(&main_user_list_mutex);
112       add_user(&main_user_list,new_user);
113       count = count + 1; //va a braccetto con la lista, deve andare qui
114       pthread_mutex_unlock(&main_user_list_mutex);
115       pthread_create(&(new_user -> thread), NULL, (void* (*)(void*))user_thread, new_user);
116     } else { //troppi utenti glielo dico
117       close(user_socket); // così rifiuto la connessione, chiudendo il socket
118     }
119   }
```

Gestinone utenti

- **user_lib** Libreria per la gestione degli utenti
- **channel_lib** Libreria per la gestione dei canali
- **utils_lib** Libreria di utilità
- **user_thread** Libreria per la gestine dei comandi

user_lib

```c
/*ADD USER*/
int add_user(User_list** list, User* user){
  // Utente nullo o già presente
  if(user == NULL || find_by_id(*list,user->id) != NULL){
    return -1;
  }

  User_list* temp = malloc(sizeof(User_list)); // allocazione memoria
  temp -> payload = user; // inserisco l'utene nel nodo di lista

  //gestisco il caso critico: il primo inserimento
  if(*list == NULL){
    temp -> prev = NULL;
    temp -> next = NULL;
  } else {
    //inserimento in testa
    temp -> prev = NULL;
    temp -> next = *list;
    (*list) -> prev = temp;
  }

  *list = temp; // riassegnazione della testa della lista

  return 1; // ritorno di successo
}
```

channel_lib

```
82  Channel* remove_channel(Channel_list** list, string name){
83    if(*list == NULL) return NULL;
84
85    Channel_list* temp = *list;
86
87    //scorro la lista per portarmi al canale scelto
88    while (strcmp(temp -> payload -> name, name) != 0) {
89      if(temp -> next == NULL) {
90        return NULL;
91      } else {
92        temp = temp -> next;
93      }
94    }
95
96    if(temp -> next == NULL && temp -> prev == NULL){ //c'è un solo nodo
97      (*list) = NULL;
98    } else if(temp -> prev == NULL){ //caso in cui il nodo da rimuovere è il primo
99      temp -> next -> prev = NULL;
100     *list = (*list) -> next;
101   } else if(temp -> next == NULL) { //devo rimuovere l'ultimo
102     temp -> prev -> next = NULL;
103   } else {
104     temp -> prev -> next = temp -> next;
105     temp -> next -> prev = temp -> prev;
106   }
107
108   Channel* chan = temp -> payload; //salvo il canale
109
110   free(temp); // libero la cella di memoria puntata
111
112   return chan; //ritoro il canale
113 }
```

utils_lib

```
19  ///@brief impone un numero massimo di utenti che si possono collegare al server
20  #define MAXUSER 150
21  #define MAXLINE 4096
22
23  ///@brief contatore per la lista utenti
24  int count;
25  ///@brief lista globale per contenente tutti gli utenti del presenti nel server
26  User_list* main_user_list;
27  ///@brief semaforo per la lista utenti, evita malfunzionamenti dalla programmazione concorrente
28  pthread_mutex_t main_user_list_mutex;
29
30  ///@brief lista globale per la gestione di tutti i canali
31  Channel_list* main_channel_list;
32  ///@brief semaforo per la lista cananli, evita malfunzionamenti dalla programmazione concorrente
33  pthread_mutex_t main_channel_list_mutex;
34
```

utils_lib

```
80  void send_channel_info(User* u, Channel* c) {
81    User_list* list = c -> users;
82    User* target;
83    char* send_line = malloc(MAXLINE + 1);
84    while (list != NULL) {
85      target = list -> payload;
86      send_user_info(u, target, c -> name);
87      list = list -> next;
88    }
89
90    strcpy(send_line, ":");
91    strcat(send_line, SERVER_NAME);
92    strcat(send_line, " ");
93    strcat(send_line, RPL_ENDOFWHO);
94    strcat(send_line, " ");
95    strcat(send_line, u -> name);
96    //strcat(send_line, " * ");
97    strcat(send_line, ENDOFWHO);
98    write(u -> socket, send_line, strlen(send_line));
99
100   free(send_line);
101 }
102
```

user_thread_lib

```
3   void user_thread(User* u){
4     char recvline[MAXLINE + 1];
5     char* line = malloc((MAXLINE + 1) * sizeof(char));
6     char* send_line = malloc((MAXLINE + 1) * sizeof(char));
7     char* command;
8     ssize_t n;
9
10    pthread_mutex_lock(&(u -> socket_mutex));
11    n = read(u -> socket, recvline, MAXLINE);
12    pthread_mutex_unlock(&(u -> socket_mutex));
13
14    while(n > 0){
15      recvline[n] = 0;
16      line = strcpy(line, recvline);
17      command = strtok(line, " \t\r\n/");
18 ›    while (command != NULL) {⋯
52
53      pthread_mutex_lock(&u->socket_mutex);
54      n = read(u->socket, recvline, MAXLINE);
55      pthread_mutex_unlock(&u->socket_mutex);
56    }
57  }
```

user_thread_lib

```
while (command != NULL) {
  printf("L'utente: %i ha inviato il comando: %s\n",u -> id, command);
  /*Qui parte lo switch di comparazaione con i comandi conosciuti*/
  if(strcmp(command,NICK) == 0){
    pthread_mutex_lock(&main_user_list_mutex);
    recieve_nick(u, main_user_list, strtok(NULL, " \t\r\n/"));
    pthread_mutex_unlock(&main_user_list_mutex);
  } else if(strcmp(command,USER) == 0){
    recieve_user(u);
  } else if(strcmp(command,JOIN) == 0){
    recieve_join(u, strtok(NULL, " \t\r\n/"));
  } else if(strcmp(command, MODE) == 0){
    strtok(NULL, " \t\r\n/");
    recieve_mode(u, strtok(NULL, " \t\r\n/"));
  } else if(strcmp(command, WHO) == 0){
    recieve_who(u, strtok(NULL, " \t\r\n/"));
  } else if(strcmp(command, WHOIS) == 0){
    recieve_whois(u, strtok(NULL, " \t\r\n/"));
  } else if(strcmp(command, PING) == 0){
    recieve_ping(u, strtok(NULL, " \t\r\n/"));
  } else if(strcmp(command, PRIVMSG) == 0){
    recieve_privmsg(u, strtok(NULL, "\n"));
  } else if(strcmp(command, PART) == 0){
    recieve_part(u, strtok(NULL, "\n"));
  } else if(strcmp(command, QUIT) == 0){
    recieve_quit(u, strtok(NULL, "\n"));
  } else if(strcmp(command, LIST) == 0){
    recieve_list(u);
  }
  command = strtok(NULL, " \t\r\n/");
}
```

user_thread_lib

```
345  void recieve_privmsg(User* u, char* message){
346    char* send_line = malloc(MAXLINE + 1);
347    char* target = strtok(message, " \t\r\n/");
348    char* send_message = strtok(NULL, "\n");
349    Channel* c;
350
351    pthread_mutex_lock(&main_channel_list_mutex);
352    c = find_channel(main_channel_list, target);
353    pthread_mutex_unlock(&main_channel_list_mutex);
354
355    //non faccio controlli sull'esistenza dell canale perché
356    //effettivamente il comando parte solo se l'utente è nel canale
357    strcpy(send_line, ":");
358    strcat(send_line, u -> name);
359    strcat(send_line, "!");
360    strcat(send_line, u -> hostname);
361    strcat(send_line, " ");
362    strcat(send_line, PRIVMSG);
363    strcat(send_line, " ");
364    strcat(send_line, target);
365    strcat(send_line, " ");
366    strcat(send_line, send_message);
367    strcat(send_line, "\n");
```

user_thread_lib

```
369     User_list* list = c -> users;
370     User* temp;
371     while (list != NULL) {
372       temp = list -> payload;
373       if(temp -> id != u -> id){
374         write(temp -> socket, send_line, strlen(send_line));
375       }
376       list = list -> next;
377     }
378     free(send_line);
379   }
```

Strumenti per lo sviluppo

- **Atom**
- **CMake**
- **Check**
- **Doxygen**
- **Git**
- **Lyx**

Conclusione

Possibili sviluppi del software potrebbero includere:

- Ampliare lo spettro dei comandi gestibili dal server
- Possibilià di scambio file
- Estensione del comando PRIVMSG al caso utente-utente
- Una migliore formattazione delle risposte al client
- Implementare tutti i casi identificati dalla specifica

Fine