Università degli Studi di Padova

Laboratorio di Ingegneria Informatica
A.A. 2015/16

C-chat v1.0
Chat multiutente con architettura client/server

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Enrico Vianello [mat. 1068326]
30/06/2016

Introduzione

- Eseguibile da terminale
- Numerosi gruppi di persone
- Veloce e minimale
- Facilità d’uso
- Modulare ed espandibile

Introduzione

Perchè il linguaggio C?

- Prestazioni ottime
- Pieno controllo sul protocollo di comunicazione

Perchè client/server?

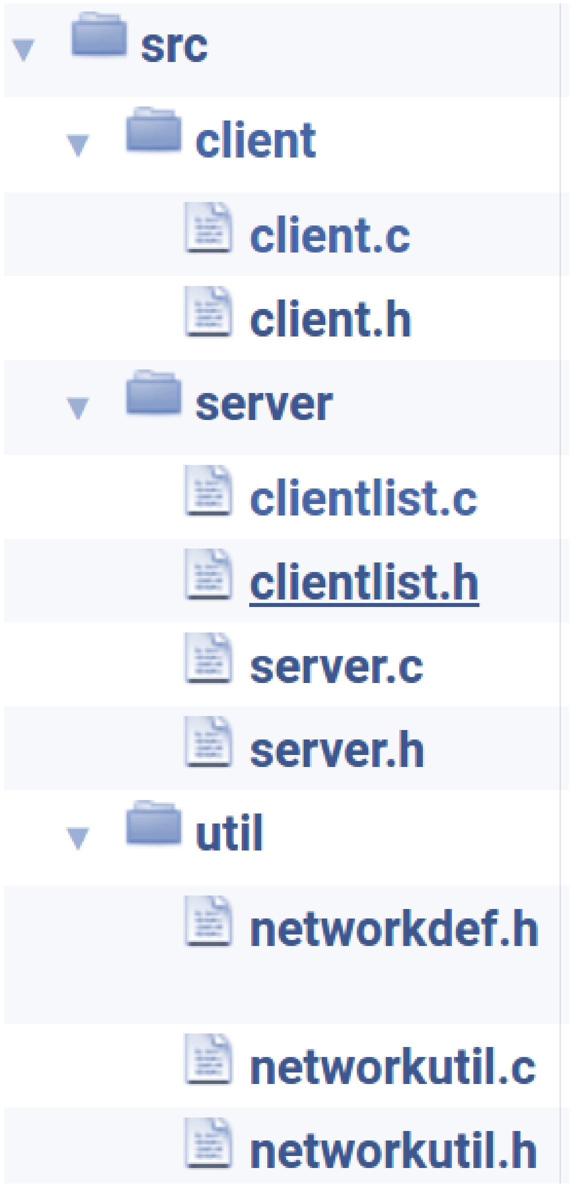
- Risorse centralizzate
- Sicurezza
- Facilità di amministrazione
- Versatilità della rete

Introduzione

Struttura delle directory

2 file eseguibili:

- Client
- Server



```
src
  client
    client.c
    client.h
  server
    clientlist.c
    clientlist.h
    server.c
    server.h
  util
    networkdef.h
    networkutil.c
    networkutil.h
```

Introduzione
User Interaction

- Comandi nella forma **/[COMANDO]**

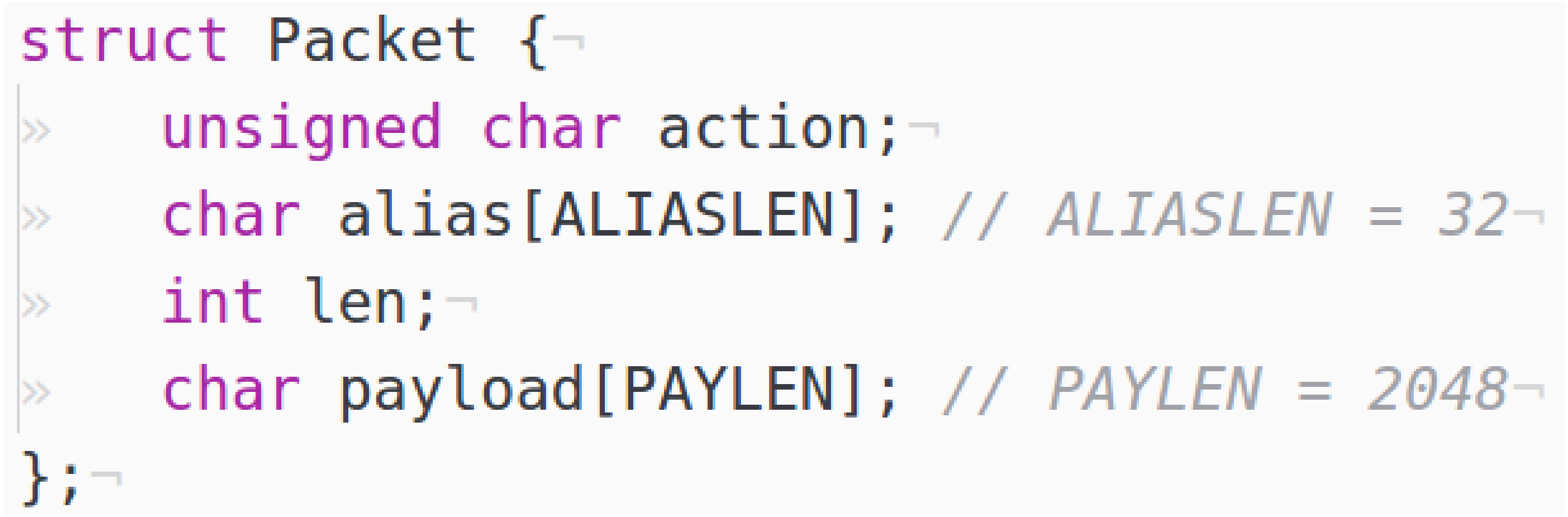
- Altri input interpretati come messaggi di chat

- Lista di comandi disponibile al comando **/help**

Protocollo di comunicazione

- Utilizzo TCP/IP
- Supporto a IPv4 e IPv6

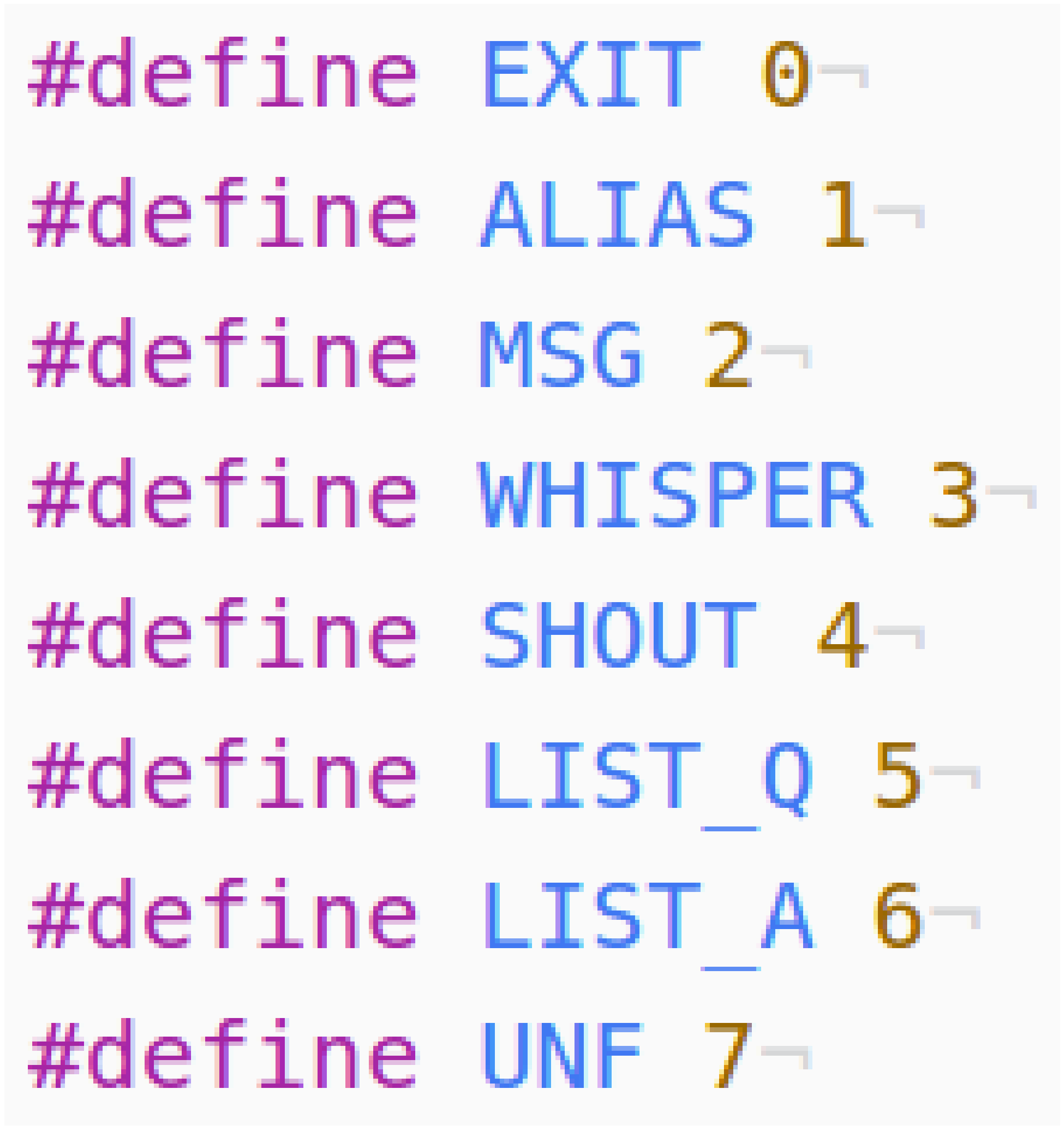
- Pacchetti standard da circa 2KB (diminuzione tempo di decodifica)



```c
struct Packet {
	unsigned char action;
	char alias[ALIASLEN]; // ALIASLEN = 32
	int len;
	char payload[PAYLEN]; // PAYLEN = 2048
};
```

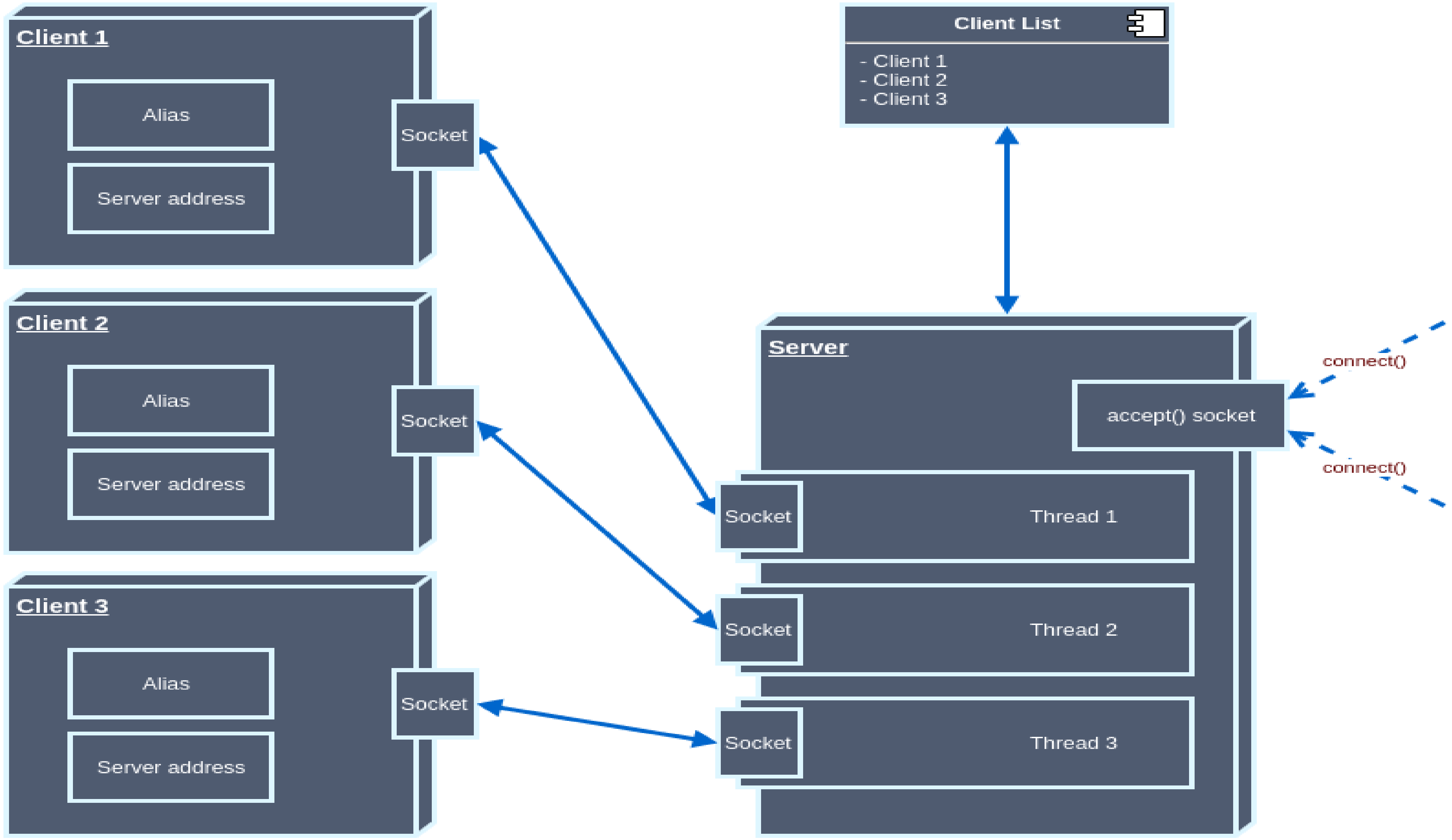
Protocollo di comunicazione

Tipologie di pacchetti



```c
#define EXIT 0
#define ALIAS 1
#define MSG 2
#define WHISPER 3
#define SHOUT 4
#define LIST_Q 5
#define LIST_A 6
#define UNF 7
```

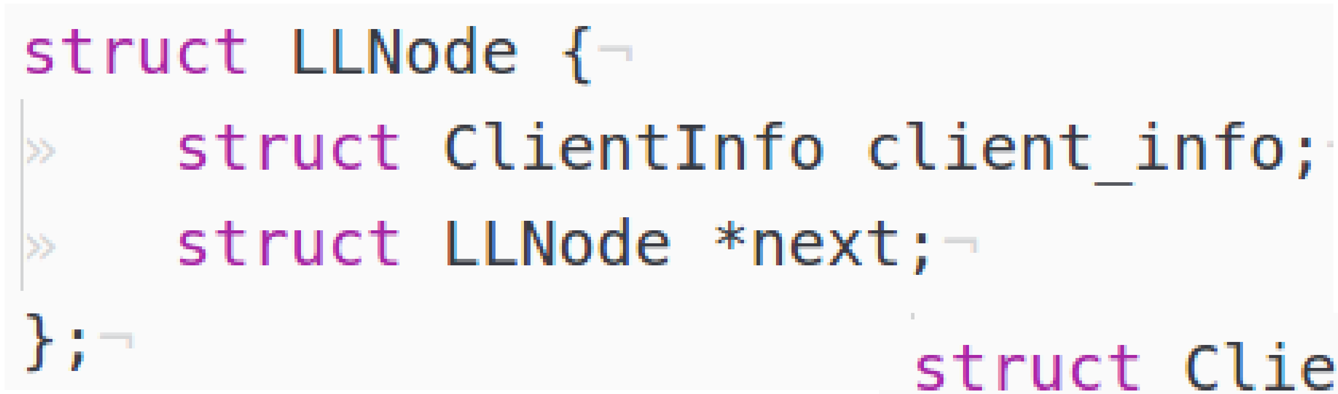
Architettura



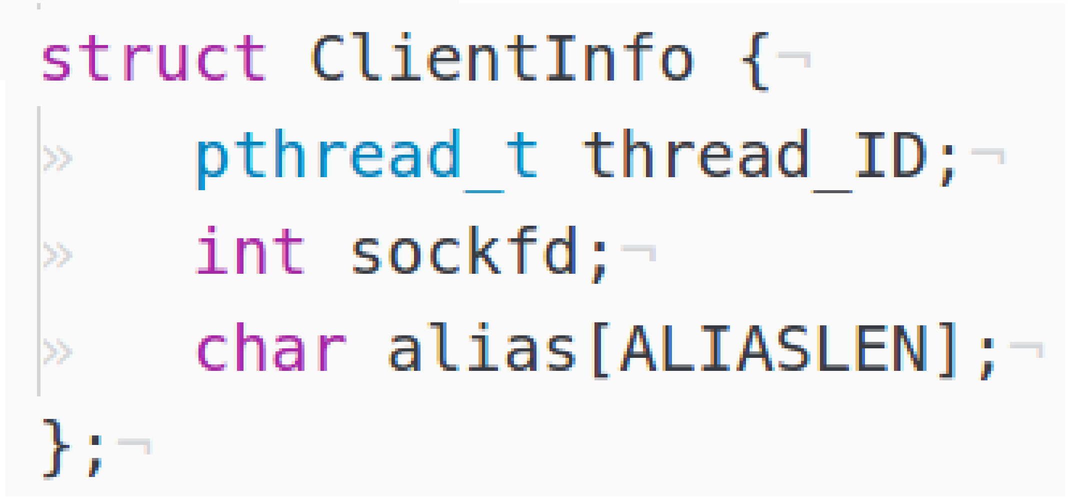
Architettura
Lista di client

Linked List

- Implementazione in clientlist.c
- Semplici operazioni di aggiunta e rimozione



```c
struct LLNode {
	struct ClientInfo client_info;
	struct LLNode *next;
};
```



```c
struct ClientInfo {
	pthread_t thread_ID;
	int sockfd;
	char alias[ALIASLEN];
};
```

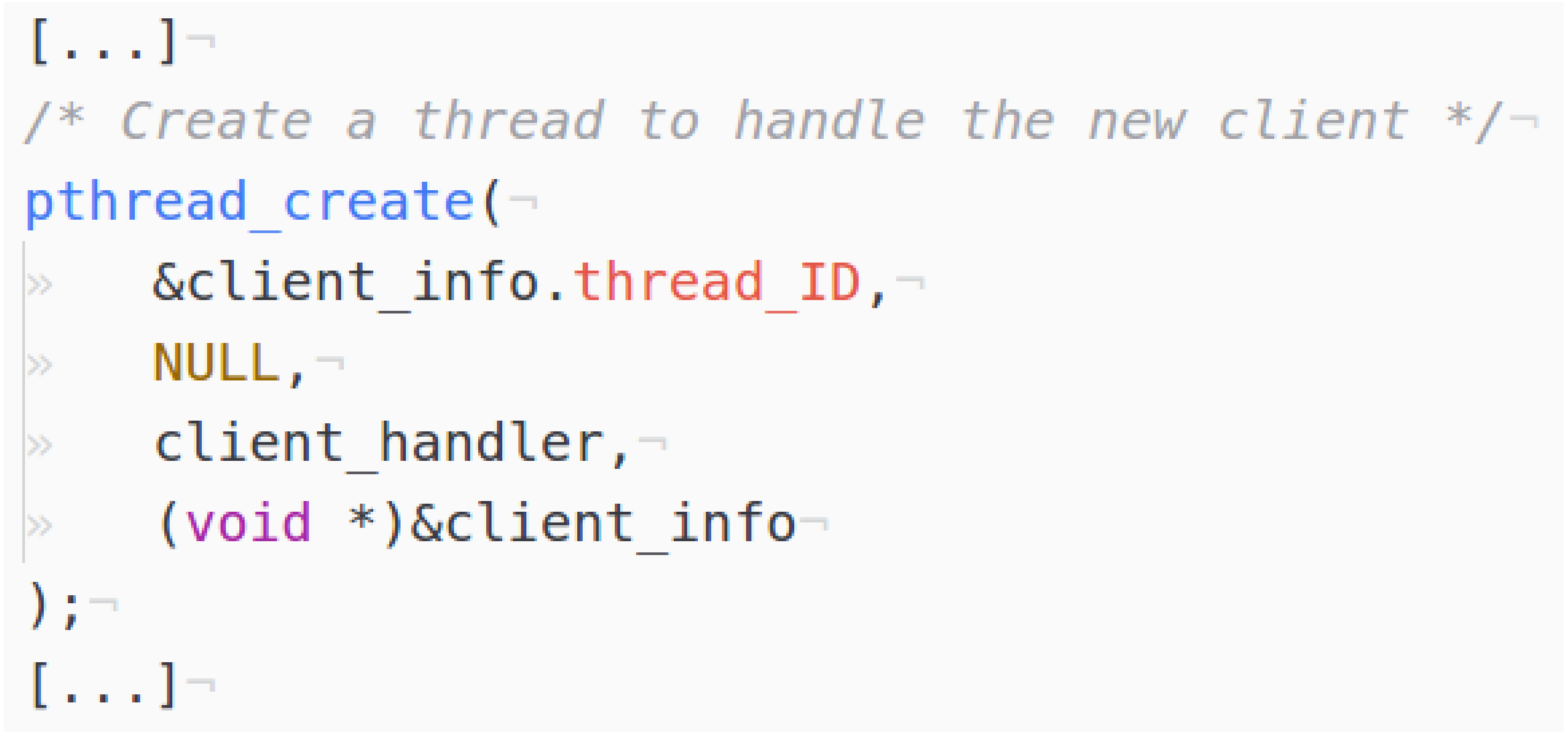
Architettura
Programmazione concorrente

- **Utilizzo di pthread.h**

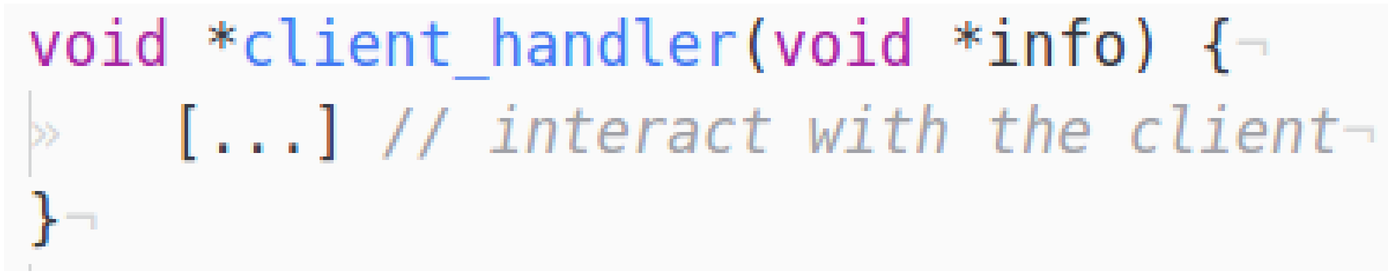
Threads utilizzati per:

- Socket listening (client e server)
- Gestione comandi utente (server)
- Gestione dei client (server)

Architettura
Programmazione concorrente



```c
[...]
/* Create a thread to handle the new client */
pthread_create(
	&client_info.thread_ID,
	NULL,
	client_handler,
	(void *)&client_info
);
[...]
```



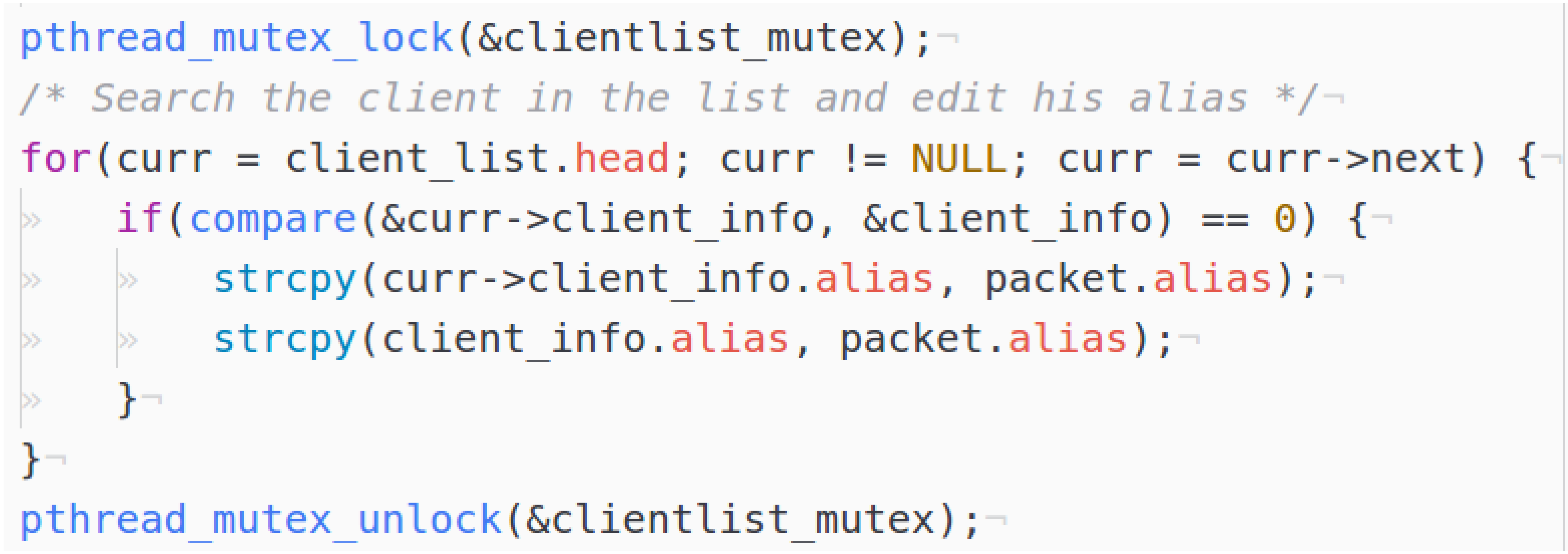
```c
void *client_handler(void *info) {
	[...] // interact with the client
}
```

Architettura

Programmazione concorrente

- **Problema:** Scrittura concorrente nella lista di client

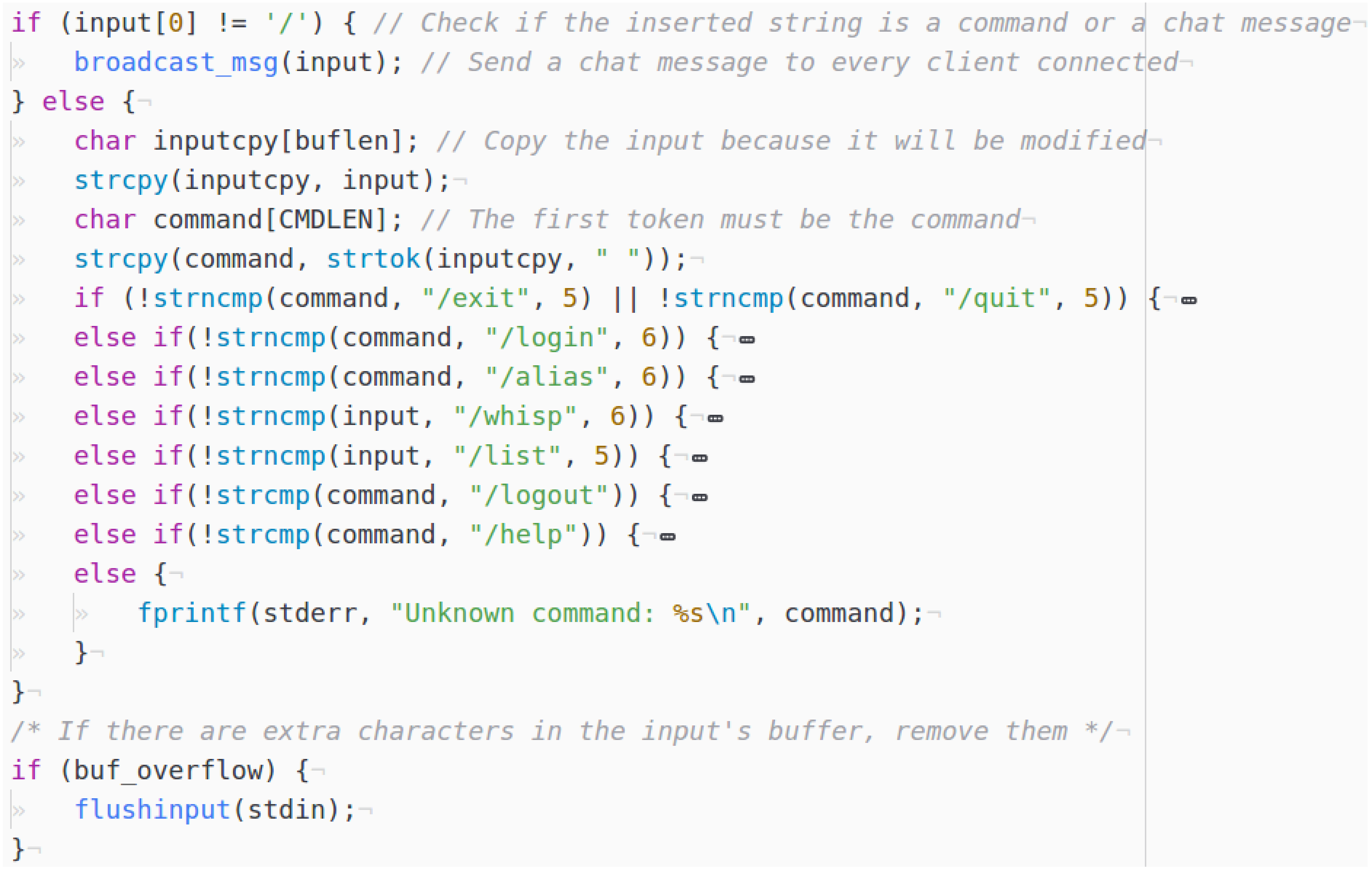
- **Soluzione:** Utilizzo di una variabile di lock



```c
pthread_mutex_lock(&clientlist_mutex);
/* Search the client in the list and edit his alias */
for(curr = client_list.head; curr != NULL; curr = curr->next) {
	if(compare(&curr->client_info, &client_info) == 0) {
		strcpy(curr->client_info.alias, packet.alias);
		strcpy(client_info.alias, packet.alias);
	}
}
pthread_mutex_unlock(&clientlist_mutex);
```

Architettura

Gestione dei comandi



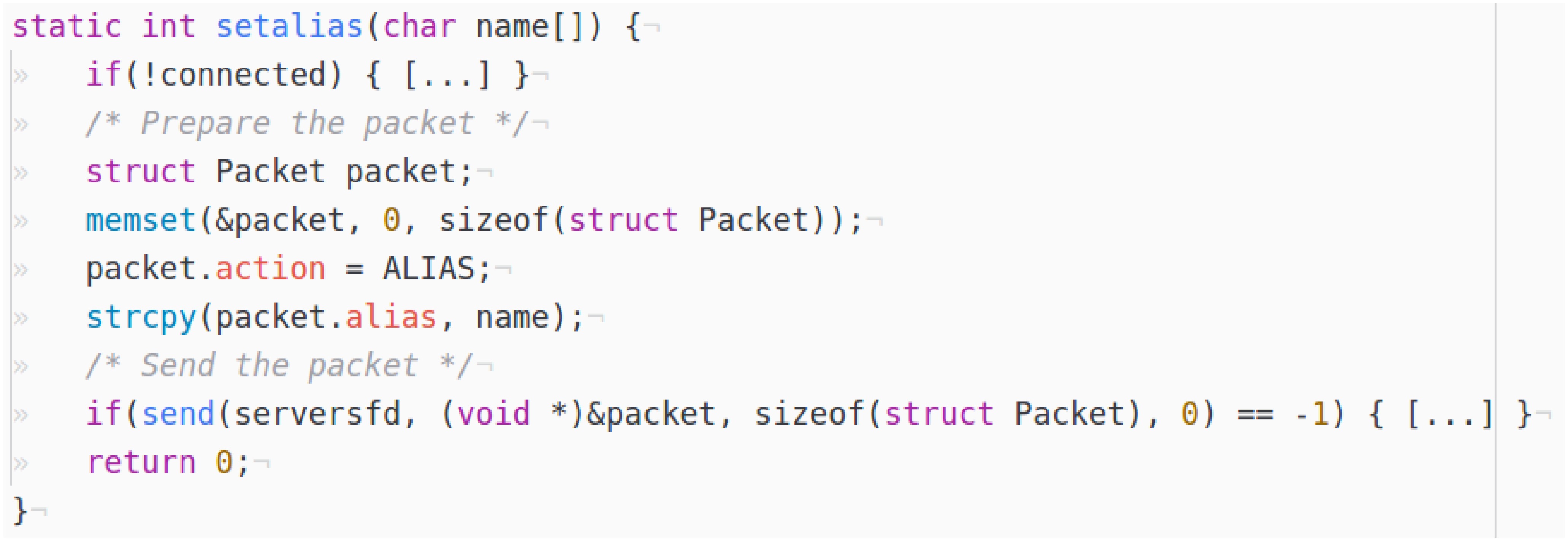
```c
if (input[0] != '/') { // Check if the inserted string is a command or a chat message
    broadcast_msg(input); // Send a chat message to every client connected
} else {
    char inputcpy[buflen]; // Copy the input because it will be modified
    strcpy(inputcpy, input);
    char command[CMDLEN]; // The first token must be the command
    strcpy(command, strtok(inputcpy, " "));
    if (!strncmp(command, "/exit", 5) || !strncmp(command, "/quit", 5)) {
    else if(!strncmp(command, "/login", 6)) {
    else if(!strncmp(command, "/alias", 6)) {
    else if(!strncmp(input, "/whisp", 6)) {
    else if(!strncmp(input, "/list", 5)) {
    else if(!strcmp(command, "/logout")) {
    else if(!strcmp(command, "/help")) {
    else {
        fprintf(stderr, "Unknown command: %s\n", command);
    }
}
/* If there are extra characters in the input's buffer, remove them */
if (buf_overflow) {
    flushinput(stdin);
}
```

Architettura

Invio di pacchetti

- Per ogni comando, metodo che gestisce lo scambio di pacchetti

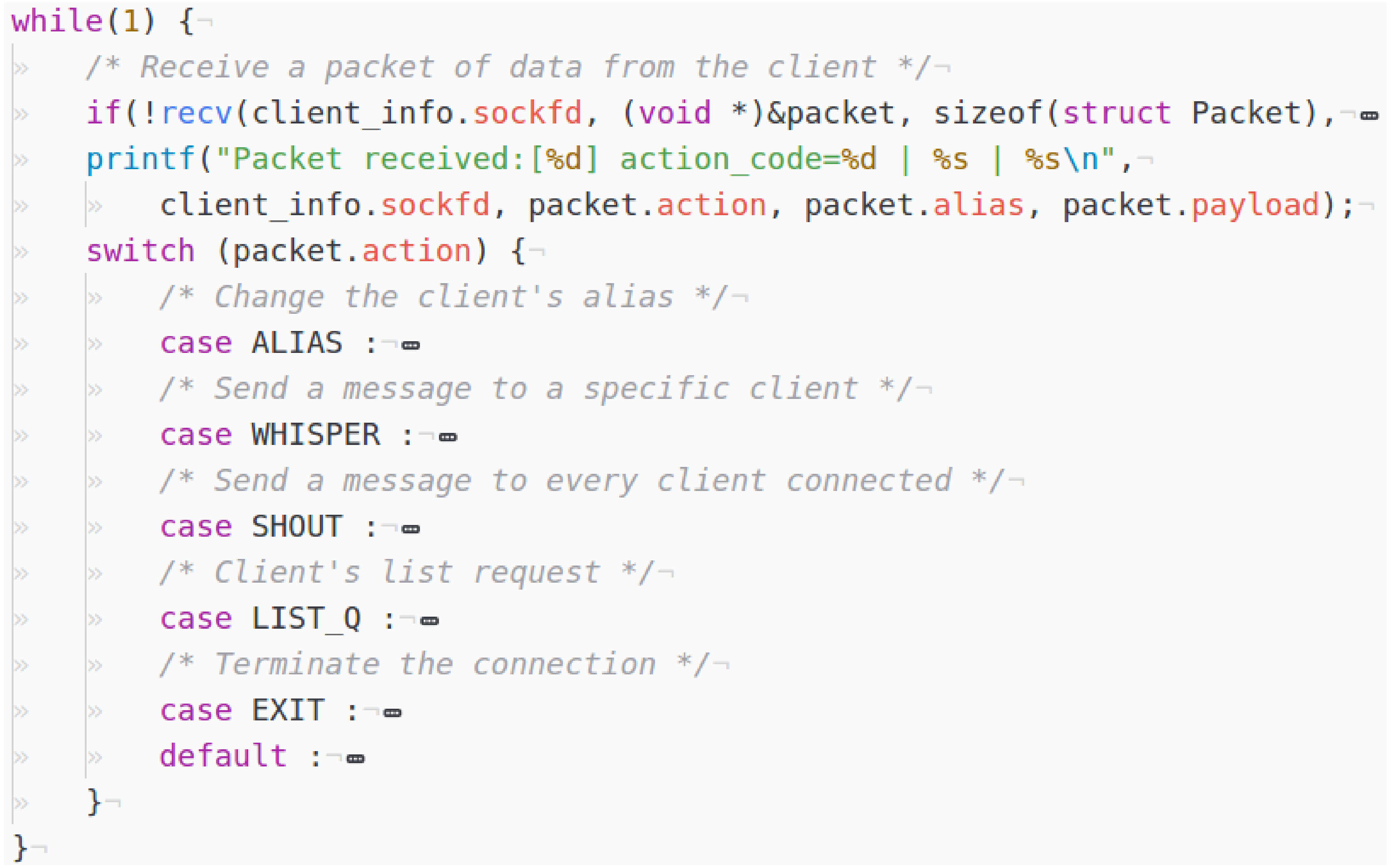
Esempio: metodo per impostare alias



```c
static int setalias(char name[]) {
    if(!connected) { [...] }
    /* Prepare the packet */
    struct Packet packet;
    memset(&packet, 0, sizeof(struct Packet));
    packet.action = ALIAS;
    strcpy(packet.alias, name);
    /* Send the packet */
    if(send(serversfd, (void *)&packet, sizeof(struct Packet), 0) == -1) { [...] }
    return 0;
}
```

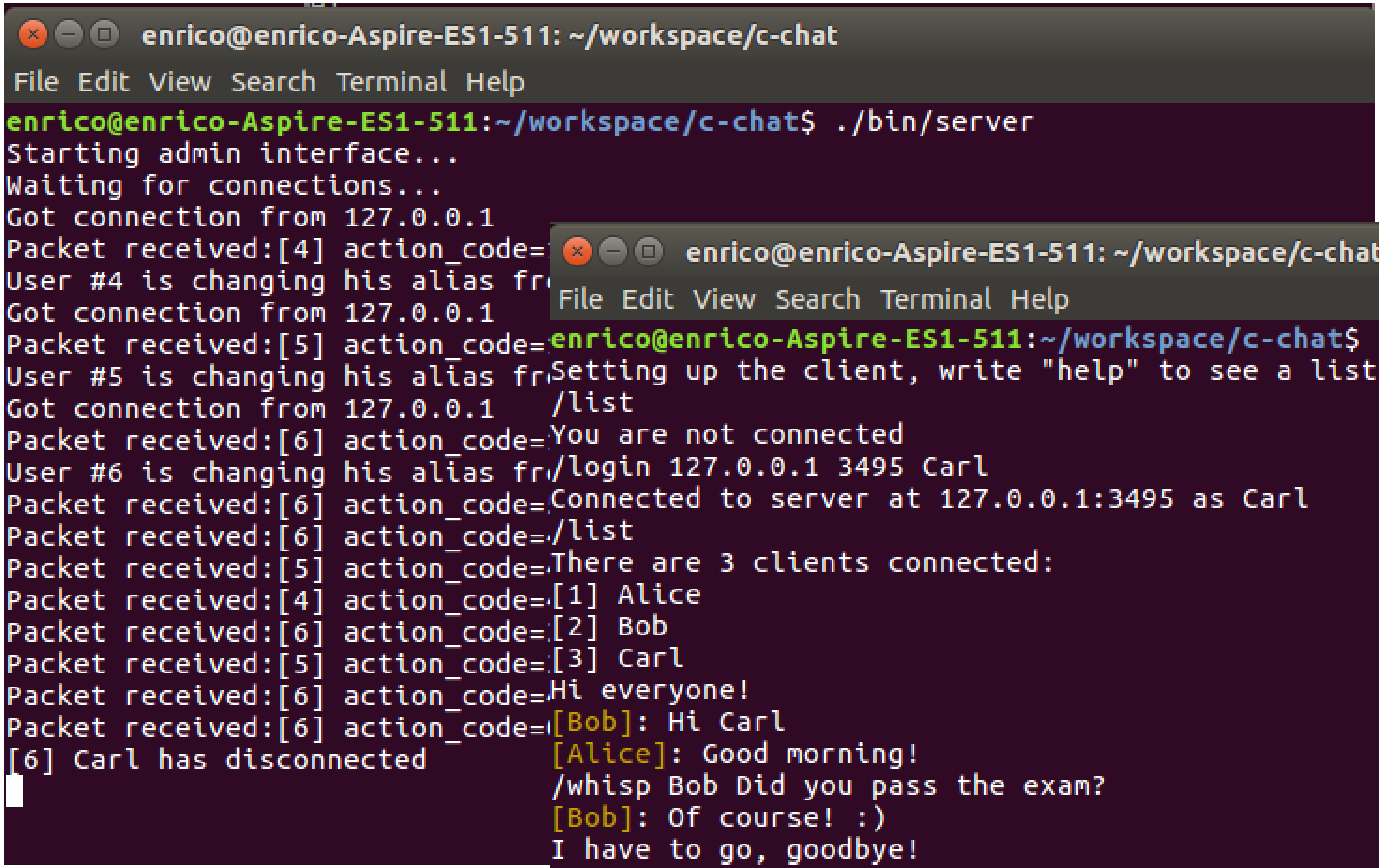
Architettura

Ricezione da parte del Server

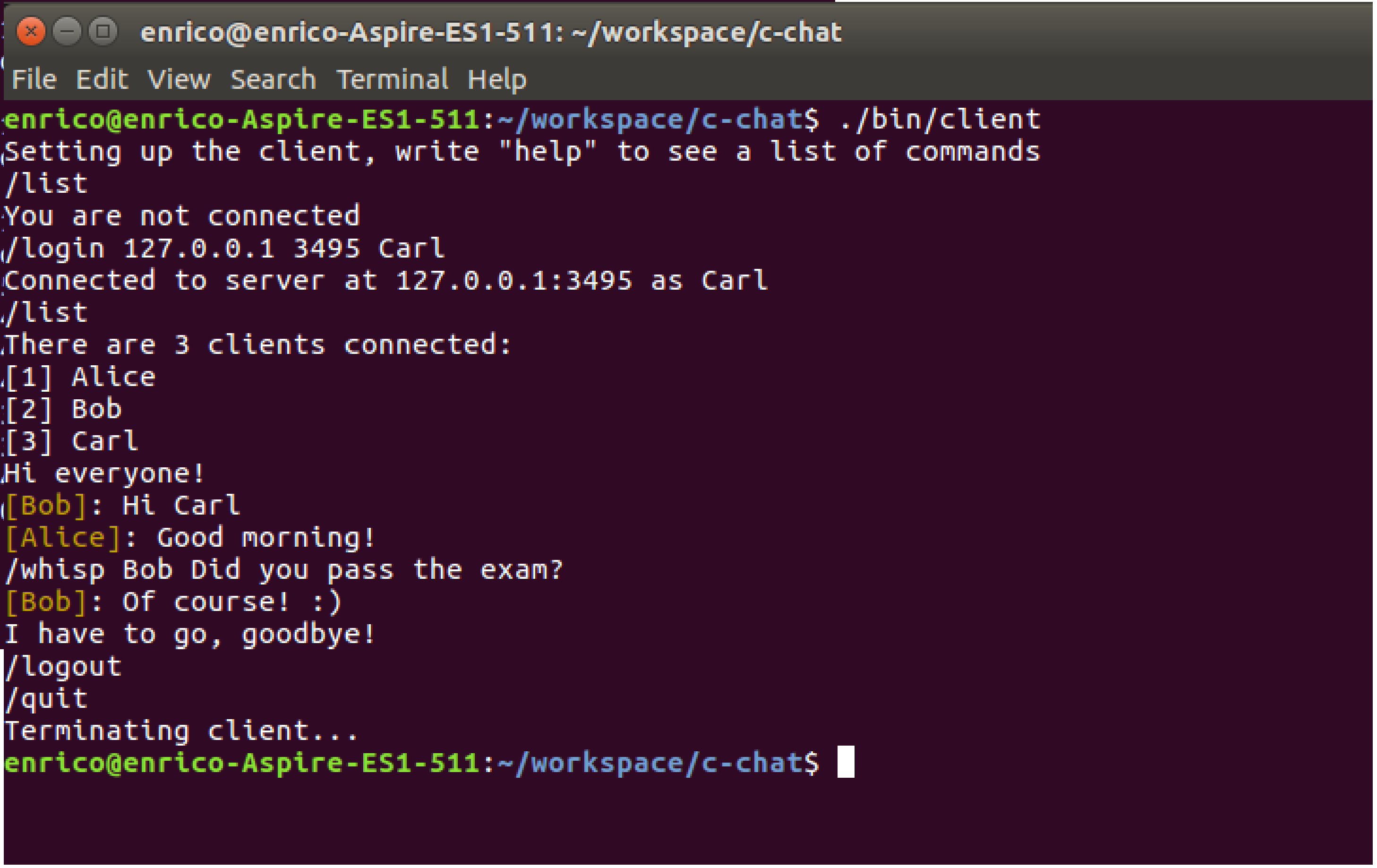


```c
while(1) {
    /* Receive a packet of data from the client */
    if(!recv(client_info.sockfd, (void *)&packet, sizeof(struct Packet),
    printf("Packet received:[%d] action_code=%d | %s | %s\n",
        client_info.sockfd, packet.action, packet.alias, packet.payload);
    switch (packet.action) {
        /* Change the client's alias */
        case ALIAS :
        /* Send a message to a specific client */
        case WHISPER :
        /* Send a message to every client connected */
        case SHOUT :
        /* Client's list request */
        case LIST_Q :
        /* Terminate the connection */
        case EXIT :
        default :
    }
}
```

Conclusioni


enrico@enrico-Aspire-ES1-511: ~/workspace/c-chat
File Edit View Search Terminal Help

```
enrico@enrico-Aspire-ES1-511:~/workspace/c-chat$ ./bin/server
Starting admin interface...
Waiting for connections...
Got connection from 127.0.0.1
Packet received:[4] action_code=
User #4 is changing his alias fr
Got connection from 127.0.0.1
Packet received:[5] action_code=
User #5 is changing his alias fr
Got connection from 127.0.0.1
Packet received:[6] action_code=
User #6 is changing his alias fr
Packet received:[6] action_code=
Packet received:[6] action_code=
Packet received:[5] action_code=
Packet received:[4] action_code=
Packet received:[6] action_code=
Packet received:[5] action_code=
Packet received:[6] action_code=
Packet received:[6] action_code=
[6] Carl has disconnected
```


enrico@enrico-Aspire-ES1-511: ~/workspace/c-chat
File Edit View Search Terminal Help

```
enrico@enrico-Aspire-ES1-511:~/workspace/c-chat$ ./bin/client
Setting up the client, write "help" to see a list of commands
/list
You are not connected
/login 127.0.0.1 3495 Carl
Connected to server at 127.0.0.1:3495 as Carl
/list
There are 3 clients connected:
[1] Alice
[2] Bob
[3] Carl
Hi everyone!
[Bob]: Hi Carl
[Alice]: Good morning!
/whisp Bob Did you pass the exam?
[Bob]: Of course! :)
I have to go, goodbye!
/logout
/quit
Terminating client...
enrico@enrico-Aspire-ES1-511:~/workspace/c-chat$
```

Possibili sviluppi

Sviluppo di funzionalità esistenti

- ID per ogni client
- Moderazione assegnazione alias
- Supporto per messaggi di qualsiasi lunghezza
- Comandi lato server per moderazione

Possibili sviluppi

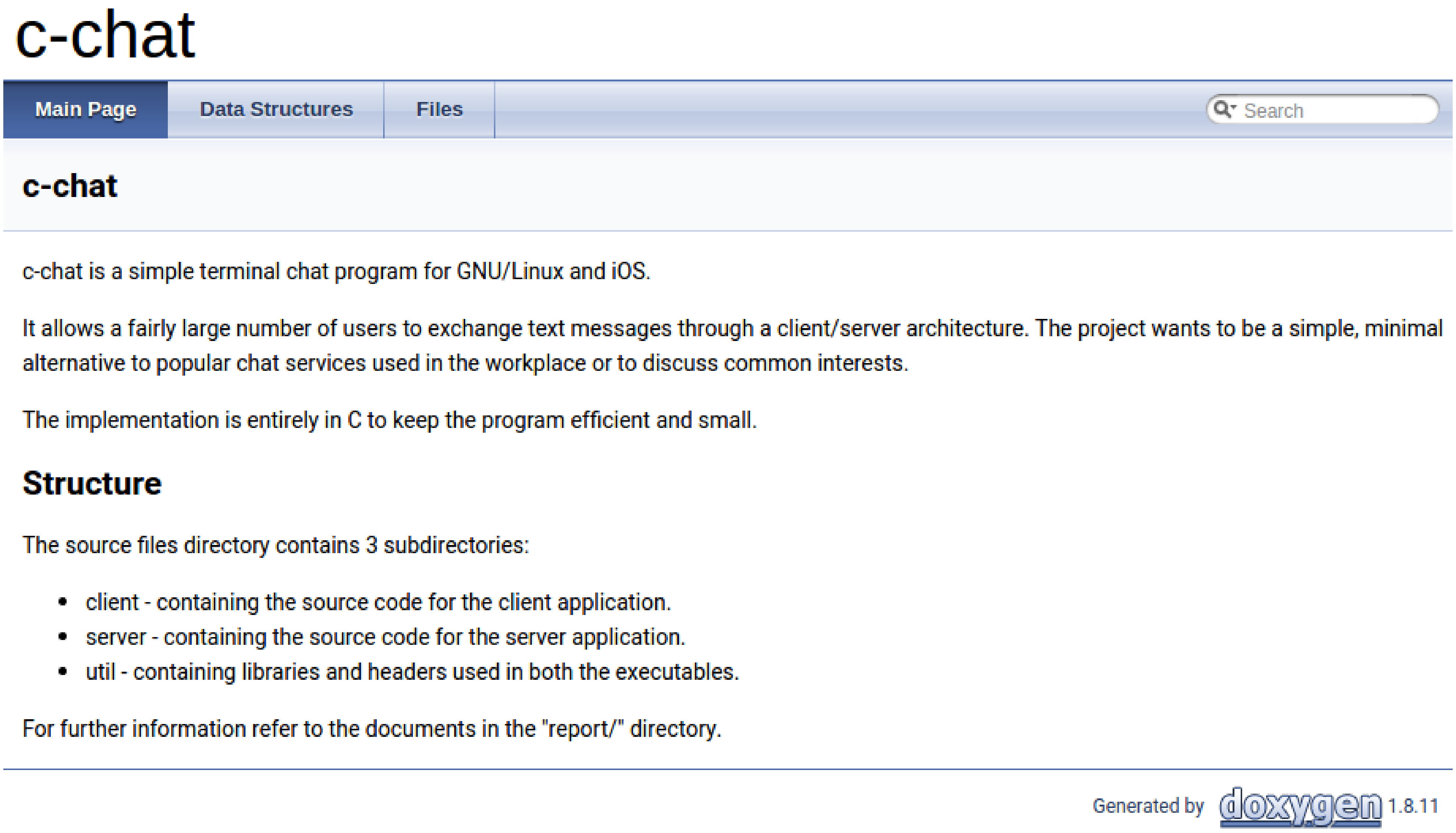
Introduzione di ulteriori features

- Semplice interfaccia grafica
- Porting su windows
- Chat room multiple
- Registrazione di alias con password

Strumenti
Cmake

```
# Specify the minimum version for CMake
cmake_minimum_required(VERSION 3.4)

# Project's name, version, and languages
project(c-chat VERSION 1.0 LANGUAGES C)

# Set the output folder where the executables will be created
set(CMAKE_BINARY_DIR ${CMAKE_SOURCE_DIR}/../bin)
set(EXECUTABLE_OUTPUT_PATH ${CMAKE_BINARY_DIR})
set(LIBRARY_OUTPUT_PATH ${CMAKE_BINARY_DIR})

# Directory containing the header included
include_directories(${CMAKE_SOURCE_DIR}/util)

# Add the subdirectories where the source files are present
add_subdirectory(util)
add_subdirectory(client)
add_subdirectory(server)
```

Strumenti
doxygen



c-chat

c-chat is a simple terminal chat program for GNU/Linux and iOS.

It allows a fairly large number of users to exchange text messages through a client/server architecture. The project wants to be a simple, minimal alternative to popular chat services used in the workplace or to discuss common interests.

The implementation is entirely in C to keep the program efficient and small.

Structure

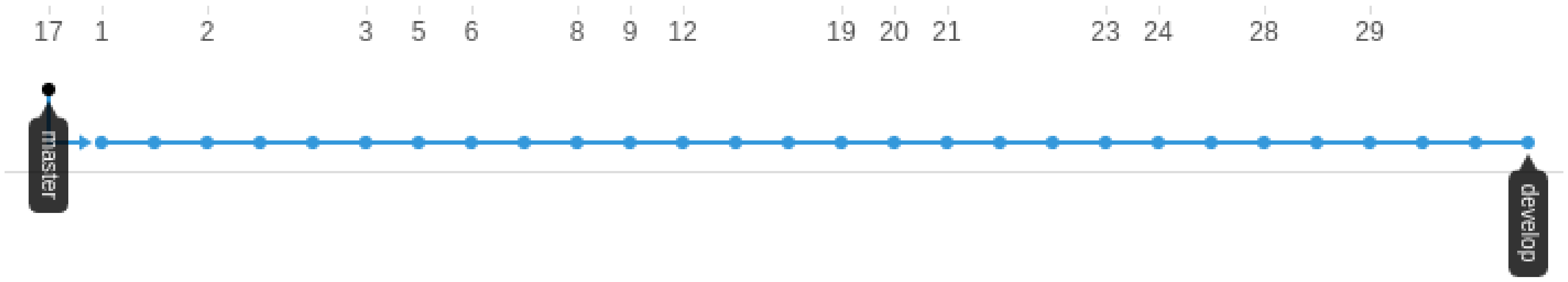
The source files directory contains 3 subdirectories:

- client - containing the source code for the client application.
- server - containing the source code for the server application.
- util - containing libraries and headers used in both the executables.

For further information refer to the documents in the "report/" directory.

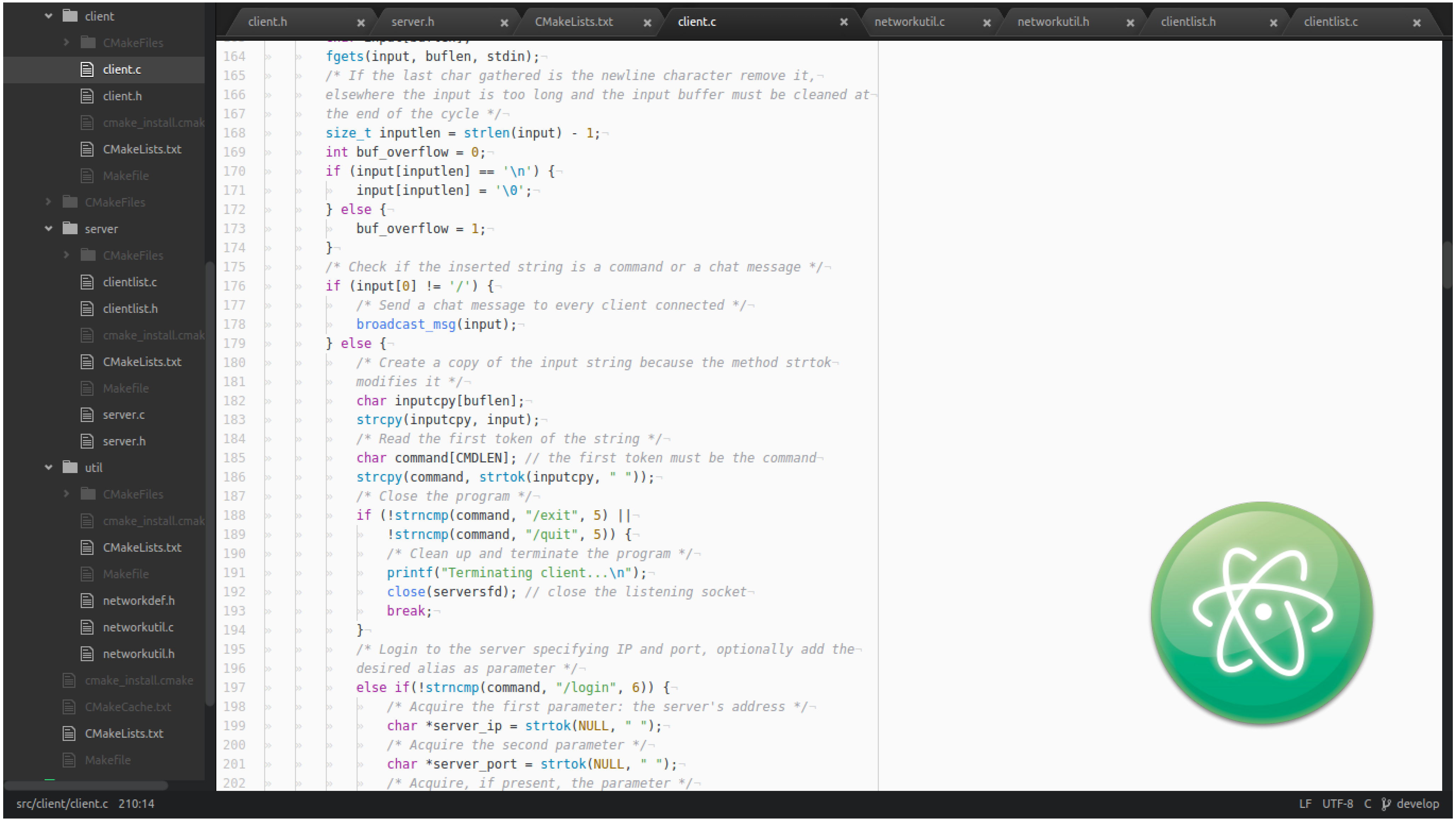
Generated by doxygen 1.8.11

Strumenti
git & Github



Strumenti

atom



```
client
  CMakeFiles
  client.c
  client.h
  cmake_install.cmak
  CMakeLists.txt
  Makefile
CMakeFiles
server
  CMakeFiles
  clientlist.c
  clientlist.h
  cmake_install.cmak
  CMakeLists.txt
  Makefile
  server.c
  server.h
util
  CMakeFiles
  cmake_install.cmak
  CMakeLists.txt
  Makefile
  networkdef.h
  networkutil.c
  networkutil.h
cmake_install.cmake
CMakeCache.txt
CMakeLists.txt
Makefile
```

```
client.h | server.h | CMakeLists.txt | client.c | networkutil.c | networkutil.h | clientlist.h | clientlist.c
```

```c
        fgets(input, buflen, stdin);
        /* If the last char gathered is the newline character remove it,
        elsewhere the input is too long and the input buffer must be cleaned at
        the end of the cycle */
        size_t inputlen = strlen(input) - 1;
        int buf_overflow = 0;
        if (input[inputlen] == '\n') {
            input[inputlen] = '\0';
        } else {
            buf_overflow = 1;
        }
        /* Check if the inserted string is a command or a chat message */
        if (input[0] != '/') {
            /* Send a chat message to every client connected */
            broadcast_msg(input);
        } else {
            /* Create a copy of the input string because the method strtok
            modifies it */
            char inputcpy[buflen];
            strcpy(inputcpy, input);
            /* Read the first token of the string */
            char command[CMDLEN]; // the first token must be the command
            strcpy(command, strtok(inputcpy, " "));
            /* Close the program */
            if (!strncmp(command, "/exit", 5) ||
                !strncmp(command, "/quit", 5)) {
                /* Clean up and terminate the program */
                printf("Terminating client...\n");
                close(serversfd); // close the listening socket
                break;
            }
            /* Login to the server specifying IP and port, optionally add the
            desired alias as parameter */
            else if(!strncmp(command, "/login", 6)) {
                /* Acquire the first parameter: the server's address */
                char *server_ip = strtok(NULL, " ");
                /* Acquire the second parameter */
                char *server_port = strtok(NULL, " ");
                /* Acquire, if present, the parameter */
```

```
src/client/client.c  210:14                                LF  UTF-8  C  develop
```