RELAZIONE SISTEMI OPERATIVI

2019/2020

Docente: Prof. Francesco Quaglia

Macroarea Ingegneria

Anuar Elio Magliari, 0267412
Michele Tosi, 0253591

Sommario

Introduzione
Il modulo di Sistemi Operativi prevede lo svolgimento di un progetto da sviluppare utilizzando il linguaggio C in ambiente Unix. Questo documento descrive la struttura complessiva del progetto. Esso ha come scopo la realizzazione di un servizio “talk” via internet gestito tramite server.

Specifica
Realizzazione di un servizio "talk" via internet gestito tramite server.

Il servizio tiene traccia di un elenco di client pronti ad iniziare una "conversazione".

I client (residenti, in generale, su macchine diverse), dopo essersi connessi al servizio acquisiscono l'elenco e lo mostrano all'utente il quale potrà collegarsi a un altro utente disponibile iniziando così la conversazione.
Due client collegati tra loro permetteranno ai relativi utenti di scambiarsi messaggi finche' uno dei due non interrompe la conversazione, e saranno indisponibili ad altre conversazioni. Al termine di una conversazione i client ritorneranno disponibili fino a che non si scollegano dal server.

Si precisa che la specifica richiede la realizzazione del software sia per l'applicazione client che per l'applicazione server.

Per progetti misti Unix/Windows è a scelta quale delle due applicazioni sviluppare per uno dei due sistemi.

Descrizione generale
Il servizio tiene traccia di un elenco di client pronti ad iniziare una "conversazione". I client (residenti, in generale, su macchine diverse), dopo essersi connessi al servizio acquisiscono l'elenco e lo mostrano all'utente il quale potrà collegarsi a un altro utente disponibile iniziando così la conversazione. Due client collegati tra loro permetteranno ai relativi utenti di scambiarsi messaggi finche' uno dei due non interrompe la conversazione, e saranno non disponibili ad inziare altre conversazioni. Al termine di una conversazione i client ritorneranno disponibili fino a che non si scollegano dal server.

La comunicazione tra client e server avviene tramite socket di tipo AF_INET. Il server al suo interno implementa un sistema multi-thread in grado di gestire concorrentemente sia nuove connessioni da nuovi client che richieste su connessioni già stabilite.

Un client può collegarsi al server in ascolto identificandosi e rendendosi disponibile ad altri client. Su richiesta può intraprendere una conversazione con i client disponibili nel display della tabella, se e solo se viene accettata da quest'ultimo. La comunicazione viene interrotta a seguito del messaggio "quit" oppure con il segnale "SIGINT" rendendosi di nuovo disponibili. Possono scollegarsi dal server solo nel momento in cui non hanno nessuna comunicazione in corso.

Il server può gestire fino ad un massimo di connessioni concorrenti stabilito dalla "define" PENDING, se tale numero viene superato, il client rimarrà in attesa di un posto disponibile o altrimenti si scollegherà. Il server può essere terminato solo se viene chiuso il suo terminale oppure tramite un segnale di "SIGKILL".

Breve descrizione sulla logica dei programmi

Server

- client_online: fornisce una tabella dei client disponibili ad intraprendere una conversazione, in caso di nessuna disponibilità il thread assegnato rimane in attesa
- comunicazione: permette una conversazione tra i client tramite socket fino a che uno dei due client coinvolti non digita la stringa "quit"
- identificazione: chiede al client il nome per poi inserirlo all'interno della tabella e imposta lo stato del client come disponibile
- cambia_linea: serve a mettere in relazione un client con quello che l'ha contattato
- listen_client: tiene traccia dei client che si disconnettono dal server
- cerca_client: controlla se il client con cui si vuole comunicare è connesso al server ed è disponibile per iniziare una nuova conversazione, in caso invia a quest'ultimo una richiesta per iniziare una conversazione
- dealloca_memoria: si occupa in caso di disconnesione di un client dal server di deallocare la memoria allocata per esso

Client

- gestione_comunicazione: controlla se il canale di comunicazione con l'altro client è stato chiuso
- gestione_uscita: gestisce l'uscita dalla chat con conseguente chiusura della comunicazione

Gestione segnali

Il segnale **SIGINT** viene gestito all'interno del client per uscire da una conversazione se questa è in corso mentre per terminare l'esecuzione dell'applicativo nel caso in cui questo sia in attesa di iniziare una nuova conversazione.

Semafori

Nel server sono stati utilizzati semafori per evitare che i thread accedano in concorrenza ai dati condivisi che servono per istanziare la comunicazione tra client diversi evitando così un malfunzionamento del programma.

Socket

Per la comunicazione sono stati utilizzati socket AF_INET via IPV4 e TCP/IP tra il server e i client.

Manuale d'uso

Manuale di compilazione:

È stato preparato un makefile per la compilazione, è possibile eseguire il comando:

- make per generare gli eseguibili del client e del server
- make Server per generare l'eseguibile del server
- make Client per generare l'eseguibile del client
- make clean per eliminare gli eseguibili generati

Manuale del server:

> Una volta avviato il server questo va lasciato in attesa delle richieste dei vari client e funge come ponte tra essi

Manuale del client:

> Una volta avviato il client inserire il nome per poter essere identificati dagli altri client connessi al server.
>
> Una volta connessi si può digitare 'aggiorna' per aggiornare la lista delle persone connesse con cui poter iniziare una nuova conversazione oppure si può digitare il nome di un utente connesso al server per inviargli una richiesta che può essere accettata o meno da quest'ultimo.
>
> Per uscire la conversazione si può digitare quit oppure si può utilizzare il segnale "SIGINT" facendo attenzione al fatto che se non si è in una conversazione questo chiuderà l'applicativo.

server.c

#include "lib.h"

#define PENDING 100

```
int fd[PENDING];
        //mantiene tutti i socket client descriptor

int busy[PENDING];
        //tiene traccia dei client che stanno occupati a rispondere a una richiesta di chat

int connessione_richiesta[PENDING];
    //tiene traccia se un client propone di chattare

void *identificazione(void *argument);
    //identifica il client

void client_online(void *argument);
    //stampa i client disponibili online

int cerca_client(int, char *, int);
    //cerca il client selezionato

void comunicazione(int client1, long client2, long i);          //mette in
comunicazione i due client



struct tabella{
    char *nome_client;
    char *indirizzo_ip;
    int disponibile;        //-1 non attivo, 1 disponibile, 0 occupato
    pthread_t id_thread;
};

struct value{
    int dsc;
    int numero;
    pthread_t testa;
};



struct tabella **tabella_thread;

pthread_mutex_t *sem;
```

```
int passaggio[3];
int entry, exit_client;
char **risposta_connessione_richiesta;


void client_exit_attesa(){
        pthread_exit(NULL);
}

void dealloca_memoria(int i){

        pthread_mutex_lock(sem);
        free(tabella_thread[i]->nome_client);
        free(tabella_thread[i]->indirizzo_ip);
        tabella_thread[i]->disponibile = -1;
        fd[i] = -1;
        busy[i] = -1;
        connessione_richiesta[i] = 0;
        risposta_connessione_richiesta[i] = (char *)malloc(MAX_READER);
        sprintf(risposta_connessione_richiesta[i], "%s", "quit client");
        pthread_mutex_unlock(sem);

        pthread_exit(NULL);
}


void *listen_client(void *argument){
        struct value *head_listen = (struct value *) argument;
        char *ric_attesa;
        int cl, id_l = head_listen->numero;
```

```
        while(cl >= 0){
                ric_attesa = (char *)malloc(MAX_READER);
                cl = recv(head_listen->dsc, ric_attesa, MAX_READER, 0);
                ric_attesa[cl] = '\0';

                exit_client = 1;
                while(entry == 0);

                if(cl == 0){
                        pthread_kill(head_listen->testa, SIGUSR1);
                        printf("Client[%d] si è disconnesso dal server\n", id_l);

                        free(head_listen);
                        free(ric_attesa);
                        dealloca_memoria(id_l);

                        return((void *)0);
                }
                exit_client = 0;
        }

        pthread_exit(NULL);
}



int main(){
   struct sockaddr_in server, client[PENDING];
   int sd;
        long id;
   socklen_t addrlen;
        pthread_t thread[PENDING];
```

```
        printf("\n\t\t\e[91m\e[1m@@@@@@  @@@@@@  @@@@@@  @@   @@
@@@@@@  @@@@@@@\e[22m\e[39m\n");

        printf("\t\t\e[91m\e[1m@@     @@     @@ @@  @@   @@  @@     @@ @@\
e[22m\e[39m\n");

        printf("\t\t\e[91m\e[1m@@@@@@  @@@@@@  @@@@@@  @@   @@
@@@@@@  @@@@@@@\e[22m\e[39m\n");

        printf("\t\t\e[91m\e[1m   @@  @@     @@ @@   @@ @@   @@     @@ @@\
e[22m\e[39m\n");

        printf("\t\t\e[91m\e[1m@@@@@@  @@@@@@  @@  @@   @@    @@@@@@
@@   @@\e[22m\e[39m\n");


   if((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1 ){
      printf("Errore init socket\n");
      exit(EXIT_FAILURE);
   }


   server.sin_family = AF_INET;
   server.sin_port = htons(PORT);
        server.sin_addr.s_addr = htonl(INADDR_ANY);



   if(setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &(int){1}, sizeof(int))<0){
        printf("Error setsockopt(SO_REUSEADDR)");
        printf("Error code: %d\n", errno);
        exit(EXIT_FAILURE);
   }



   if( (bind(sd, (struct sockaddr *)&server, sizeof(server)))== -1){
      printf("Error binding\n");
      printf("Error code: %d\n", errno);
      exit(EXIT_FAILURE);
   }
```

```
if( listen(sd, PENDING) == -1){
        printf("error listen\n");
        printf("error code %d\n", errno);
        exit(EXIT_FAILURE);
    }

    printf("\nServer in ascolto sulla porta %d\n", PORT);


    addrlen = sizeof(client[0]);
    sem = malloc(sizeof(pthread_mutex_t *));
    tabella_thread = malloc(PENDING*sizeof(struct tabella *));
    risposta_connessione_richiesta = (char **)malloc(PENDING*sizeof(char*));

    for(int x=0; x<PENDING; x++){
        tabella_thread[x] = malloc(sizeof(struct tabella *));
        fd[x] = -1;
        busy[x] = -1;
        connessione_richiesta[x] = 0;
    }

    id = 0;
    int ct;
    while(1){
        while(1){
            pthread_mutex_lock(sem);
            ct = fd[id];
            pthread_mutex_unlock(sem);

            if(ct == -1){                    //trovato un posto per un client
```

```
                    break;
                }
                id = (id+1)%PENDING;
            }

            while((ct = accept(sd, (struct sockaddr *)&client[id], &addrlen)) == -1 );

            pthread_mutex_lock(sem);
            fd[id] = ct;
            tabella_thread[id]->indirizzo_ip = (char
*)malloc(strlen(inet_ntoa(client[id].sin_addr)));
            strcpy(tabella_thread[id]->indirizzo_ip, inet_ntoa(client[id].sin_addr));
            printf("\nConnesione aperta sul server\n");
            printf("Ip Client[%ld]: %s\n\n", id, tabella_thread[id]->indirizzo_ip);
            pthread_mutex_unlock(sem);

            pthread_create(&thread[id], NULL, (void *)identificazione, (void *)id);
            id = (id+1)%PENDING;
        }

        printf("Chiusura della connessione\n");

    close(sd);
    return 0;
}

void *identificazione(void *argument){
        long i = (long) argument;                    //indice del thread
        int c;

        char *nome;
```

```
        nome = (char *)malloc(MAX_READER);
        c = recv(fd[i], nome, MAX_READER, 0);
        if(c == 0){
                free(nome);
                fd[i] = -1;
                pthread_exit(NULL);
        }
        nome[c] = '\0';

        pthread_mutex_lock(sem);
        tabella_thread[i]->nome_client = (char *)malloc(MAX_READER);
        strcpy(tabella_thread[i]->nome_client, nome);
        tabella_thread[i]->nome_client[strlen(tabella_thread[i]->nome_client)] = '\0';
        tabella_thread[i]->disponibile = 1;
        tabella_thread[i]->id_thread = pthread_self();
        pthread_mutex_unlock(sem);

        free(nome);


        client_online((void *)i);
        return((void *)0);
}

void client_online(void *argument){
        long i = (long)argument;
        int c, dsc, count;

        pthread_mutex_lock(sem);
        dsc = fd[i];
        pthread_mutex_unlock(sem);
```

```
char *stampa;
restart_list:
        nanosleep((const struct timespec[]){{0, 5000000L}}, NULL);

send(dsc, "\n   --- Elenco delle persone disponibili ---", strlen("\n   --- Elenco delle persone disponibili ---"), 0);

count = 0;
for(int j=0; j<PENDING; j++){
        count++;
        pthread_mutex_lock(sem);
        c = tabella_thread[j]->disponibile;
        pthread_mutex_unlock(sem);

        if(c == 1){
                count--;
                stampa = (char *)malloc(MAX_READER);

                pthread_mutex_lock(sem);
                        sprintf(stampa, " - \e[1m%-15s\e[22mIndirizzo IP:  \e[1m%s\e[22m", tabella_thread[j]->nome_client, tabella_thread[j]->indirizzo_ip);
                pthread_mutex_unlock(sem);

                nanosleep((const struct timespec[]){{0, 50000000L}}, NULL);
                send(dsc, stampa, strlen(stampa), 0);

                free(stampa);
        }
}

pthread_mutex_lock(sem);
c = tabella_thread[1]->disponibile;
```

```
pthread_mutex_unlock(sem);

if(count == PENDING-1 && c == -1){
        signal(SIGUSR1, client_exit_attesa);
        struct value *head;
        head = malloc(sizeof(struct value *));
        pthread_t thread_ric_attesa;

        head->dsc = dsc;
        head->numero = (int) i;
        head->testa = pthread_self();

        pthread_create(&thread_ric_attesa, NULL, (void *)listen_client, (void *)head);

        int x = 0;
        exit_client = 0;
        while(1){
                if(i!=x){

                        entry = 0;

                        pthread_mutex_lock(sem);
                        c = tabella_thread[x]->disponibile;
                        pthread_mutex_unlock(sem);

                        if(c == 1){
                                pthread_kill(thread_ric_attesa, SIGUSR1);
                                free(head);
                                goto restart_list;
                        }

                        entry = 1;
```

```
                        while(exit_client == 1);
                }
                x = (x+1)%PENDING;
        }
}

char *seleziona;
while(1){
        nanosleep((const struct timespec[]){{0, 5000000L}}, NULL);
        send(dsc, "\n   --- Digita il \e[1mnome\e[22m della persona da contattare ---",
strlen("\n   --- Digita il \e[1mnome\e[22m della persona da contattare ---"), 0);
        nanosleep((const struct timespec[]){{0, 5000000L}}, NULL);
        send(dsc, "   *** Digita '\e[1maggiorna\e[22m' per aggiornare la lista  ***", strlen("
*** Digita '\e[1maggiorna\e[22m' per aggiornare la lista  ***"), 0);

        start_from_here:
                seleziona = (char *)malloc(MAX_READER);

        c = recv(dsc, seleziona, MAX_READER, 0);
        seleziona[c] = '\0';

        if(c == 0 || strcmp(seleziona, "quit") == 0){
                printf("Client[%ld] si è disconnesso dal server\n", i);
                free(seleziona);
                nanosleep((const struct timespec[]){{0, 50000000L}}, NULL);

                dealloca_memoria(i);

                pthread_exit(NULL);
        }

        pthread_mutex_lock(sem);
```

```
            if(connessione_richiesta[i] == 1){

                risposta_connessione_richiesta[i] = (char *)malloc(MAX_READER);
                sprintf(risposta_connessione_richiesta[i], "%s", seleziona);
                risposta_connessione_richiesta[i][strlen(risposta_connessione_richiesta[i])] =
'\0';
                connessione_richiesta[i] = 0;

                pthread_mutex_unlock(sem);

                while(1){

                    if(strcmp(seleziona, "Si") == 0 || strcmp(seleziona, "si") == 0 ||
strcmp(seleziona, "SI") == 0 || strcmp(seleziona, "sI") == 0 || strcmp(seleziona, "yes") == 0){
                        free(seleziona);
                        while(1){
                            pthread_mutex_lock(sem);
                            c = connessione_richiesta[i];
                            pthread_mutex_unlock(sem);

                            nanosleep((const struct timespec[]){{0,
50000000L}}, NULL);

                            if(c == 2) comunicazione(-1, -1, -1);
                        }
                        goto start_from_here;
                    }

                    if(strcmp(seleziona, "No") == 0 || strcmp(seleziona, "no") == 0 ||
strcmp(seleziona, "NO") == 0){
                        free(seleziona);
                        goto restart_list;
                    }
```

```
                send(dsc, "   --- Devi rispondere '\e[1mSi\e[22m' o '\e[1mNo\e[22m'!
---", strlen("   --- Devi rispondere '\e[1mSi\e[22m' o '\e[1mNo\e[22m'! ---"), 0);
                free(seleziona);

                seleziona = (char *)malloc(MAX_READER);
                c = recv(dsc, seleziona, MAX_READER, 0);
                seleziona[c] = '\0';

                pthread_mutex_lock(sem);
                risposta_connessione_richiesta[i] = (char
*)malloc(MAX_READER);
                sprintf(risposta_connessione_richiesta[i], "%s", seleziona);
                risposta_connessione_richiesta[i]
[strlen(risposta_connessione_richiesta[i])] = '\0';

        risposta_connessione_richiesta[strlen(risposta_connessione_richiesta[i])] = '\0';
                connessione_richiesta[i] = 0;
                pthread_mutex_unlock(sem);
            }

            goto start_from_here;
        }

        c = busy[i];
        pthread_mutex_unlock(sem);


        if(strcmp(seleziona, "aggiorna") == 0){
            free(seleziona);
            goto restart_list;
        }

        if((count = cerca_client(i, seleziona, dsc)) && c == 0){
```

```
                        free(seleziona);
                        if(count == 0) goto restart_list;
                    }
                }
            }
```

```
int cerca_client(int i, char *seleziona, int descpritor){
        char *risposta, **cerca;
        int dsc[2], c;

        dsc[0] = descpritor;

        cerca = (char **)malloc(3*sizeof(char *));

        for(long j=0; j<PENDING; j++){

                if(i == j){
                        pthread_mutex_lock(sem);
                        if(tabella_thread[j]->nome_client == NULL){
                                pthread_mutex_unlock(sem);
                                continue;
                        }

                        if(strcmp(tabella_thread[j]->nome_client, seleziona) == 0){
                                if(busy[j] == 1 || tabella_thread[j]->disponibile == 0){
                                        send(dsc[0], "\n   --- \e[1mUtente occupato in un'altra
richiesta!\e[22m ---\a", strlen("\n   --- \e[1mUtente occupato in un'altra richiesta!\e[22m ---\a"), 0);
                                        pthread_mutex_unlock(sem);
                                        free(cerca);
                                        return 1;
                                }
```

```
                        send(dsc[0], "\n   --- Non è possibile parlare con \e[1mse stessi :)\
e[22m ---", strlen("\n   --- Non è possibile parlare con \e[1mse stessi :)\e[22m ---"), 0);
                        free(cerca);
                        pthread_mutex_unlock(sem);

                        return 1;
                }

                pthread_mutex_unlock(sem);
        }

        if(i!=j){

                pthread_mutex_lock(sem);
                if(tabella_thread[j]->nome_client == NULL){
                        pthread_mutex_unlock(sem);
                        continue;
                }

                if(strcmp(tabella_thread[j]->nome_client, seleziona) == 0){
                        if(busy[j] == 1 || tabella_thread[j]->disponibile == 0){
                                send(dsc[0], "\n   --- \e[1mUtente occupato in un'altra
richiesta!\e[22m ---\a", strlen("\n   --- \e[1mUtente occupato in un'altra richiesta!\e[22m ---\a"), 0);
                                pthread_mutex_unlock(sem);
                                free(cerca);
                                return 1;
                        }
                }
                cerca[0] = (char *)malloc(MAX_READER);
                strcpy(cerca[0], tabella_thread[j]->nome_client);
                pthread_mutex_unlock(sem);

                cerca[0][strlen(cerca[0])] = '\0';
```

```
                    if(strcmp(seleziona, cerca[0]) == 0){
                        free(seleziona);
                        free(cerca[0]);

                        send(dsc[0], "   --- \e[1m\e[5mInvio della richiesta, attendere...\
e[25m\e[22m ---\a", strlen("\t--- \e[1m\e[5mInvio della richiesta, attendere...\e[25m\e[22m ---\a"),
0);

                        pthread_mutex_lock(sem);
                        dsc[1] = fd[j];
                        busy[j] = 1;
                        pthread_mutex_unlock(sem);

                        connessione_richiesta[j] = 0;
                        cerca[1] = (char*)malloc(MAX_READER);

                        pthread_mutex_lock(sem);
                        sprintf(cerca[1], "\n\t--- \e[1m\e[5mHai una richiesta di chat da %s\
e[25m\e[22m ---\a", tabella_thread[i]->nome_client);
                        pthread_mutex_unlock(sem);

                        send(dsc[1], cerca[1], strlen(cerca[1]), 0);
                        nanosleep((const struct timespec[]){{0, 500000L}}, NULL);
                        free(cerca[1]);
send(dsc[1], "  --- Digitare '\e[1mSi\e[22m' per accettare altrimenti digitare '\e[1mNo\e[22m' ---",
strlen("  --- Digitare '\e[1mSi\e[22m' per accettare altrimenti digitare'\e[1mNo\e[22m'---"), 0);

                        while(1){

                            pthread_mutex_lock(sem);
                            connessione_richiesta[j] = 1;
```

```
                pthread_mutex_unlock(sem);

                while(1){
                        pthread_mutex_lock(sem);
                        c = connessione_richiesta[j];
                        pthread_mutex_unlock(sem);

                        nanosleep((const struct timespec[]){{0, 5000000L}},
NULL);

                        if(c == 0) break;
                }

                risposta = (char *)malloc(MAX_READER);

                pthread_mutex_lock(sem);
                sprintf(risposta, "%s", risposta_connessione_richiesta[j]);
                free(risposta_connessione_richiesta[j]);
                pthread_mutex_unlock(sem);

                risposta[strlen(risposta)] = '\0';

                if(strcmp(risposta, "quit client") == 0){
                        send(dsc[0], "\n   --- L'utente si è appena \
e[1mscollegato\e[22m dal server :( ---", strlen("   --- L'utente si è appena \e[1mscollegato\e[22m dal
server :( ---"), 0);
                        free(risposta);
                        free(cerca);
                        return 0;
                }


                if(strcmp(risposta, "Si") == 0 || strcmp(risposta, "si") == 0 ||
strcmp(risposta, "SI") == 0 || strcmp(risposta, "sI") == 0 || strcmp(risposta, "yes") == 0){
```

```
                                        free(risposta);

                                        passaggio[0] = dsc[1];
                                        passaggio[1] = dsc[0];
                                        passaggio[2] = j;

                                        pthread_mutex_lock(sem);
                                        connessione_richiesta[j] = 2;                   //via
libera per entrare in funzione
                                        pthread_mutex_unlock(sem);



                                        cerca[1] = (char *)malloc(MAX_READER);
                                        send(dsc[0], "     ------------------------------------",
strlen("     ------------------------------------"), 0);
                                        nanosleep((const struct timespec[]){{0, 500000L}},
NULL);

                                        pthread_mutex_lock(sem);
                                        sprintf(cerca[1], "   --- \e[1m%s ha accettato la
richiesta!\e[22m ---", tabella_thread[j]->nome_client);
                                        tabella_thread[i]->disponibile = 0;
                                        tabella_thread[j]->disponibile = 0;
                                        pthread_mutex_unlock(sem);

                                        send(dsc[1], "     ------------------------------------",
strlen("     ------------------------------------"), 0);
                                        send(dsc[0], cerca[1], strlen(cerca[1]), 0);
                                        free(cerca[1]);
                                        nanosleep((const struct timespec[]){{0, 500000L}},
NULL);

                                        send(dsc[1], "\n\t*** \e[91m\e[1mDigita per parlare!\
e[22m\e[39m ***\n", strlen("\n\t*** \e[91m\e[1mDigita per parlare!\e[22m\e[39m ***\n"), 0);
```

```
                send(dsc[0], "\n\t*** \e[91m\e[1mDigita per parlare!\
e[22m\e[39m ***\n", strlen("\n\t*** \e[91m\e[1mDigita per parlare!\e[22m\e[39m ***\n"), 0);

                pthread_mutex_lock(sem);
                busy[j] = 0;
                connessione_richiesta[j] = 2;                    //via
libera di sicurezza
                pthread_mutex_unlock(sem);

                free(cerca);

                comunicazione(dsc[0], dsc[1], i);
                pthread_exit(NULL);
            }

            if(strcmp(risposta, "No") == 0 || strcmp(risposta, "no") == 0 ||
strcmp(risposta, "NO") == 0){
                free(risposta);
                free(cerca);
                risposta = (char *) malloc(MAX_READER);

                pthread_mutex_lock(sem);
                busy[j] = 0;
                sprintf(risposta, "\n   --- \e[1m%s ha rifiutato la
richiesta!\e[22m ---\a", tabella_thread[j]->nome_client);
                pthread_mutex_unlock(sem);

                send(dsc[0], risposta, strlen(risposta), 0);
                //free(risposta);

                return 0;
            }
            free(risposta);
```

```
                        }
                    }

                    free(cerca[0]);
                }
            }
            send(dsc[0], "\n\t--- \e[1mNome non trovato!\e[22m ---\a", strlen("\n\t--- \e[1mNome non trovato!\e[22m ---\a\n"), 0);
            free(cerca);
            return 1;
}



void comunicazione(int client1, long client2, long i){
            int c;


            if(client1 == -1 && client2 == -1 && i == -1){
                        client1 = passaggio[0];
                        client2 = passaggio[1];
                        i = passaggio[2];
            }

            char *messaggio, *uscita;
            uscita = (char *)malloc(MAX_READER);

            pthread_mutex_lock(sem);
            sprintf(uscita, "%s: quit", tabella_thread[i]->nome_client);
            pthread_mutex_unlock(sem);

            while(1){
```

```
messaggio = (char *)malloc(MAX_READER);
c = recv(client1, messaggio, MAX_READER, 0);
messaggio[c] = '\0';

if(strcmp(messaggio, "***exItnOw***") == 0){
	free(messaggio);
	free(uscita);

	pthread_mutex_lock(sem);
	tabella_thread[i]->disponibile = 1;
	pthread_mutex_unlock(sem);

	client_online((void *)i);
	pthread_exit(NULL);
}

if(strcmp(messaggio, uscita) == 0 || c == 0){
	send(client1, "\n     ------------------------------------", strlen("\n
------------------------------------"), 0);
	send(client2, "\n     ------------------------------------", strlen("\n
------------------------------------"), 0);
	free(messaggio);
	messaggio = (char *)malloc(MAX_READER);
	nanosleep((const struct timespec[]){{0, 500000L}}, NULL);

	pthread_mutex_lock(sem);
	sprintf(messaggio, "\n\t\e[1m%s ha abbandonato la conversazione\e[22m\n\
a", tabella_thread[i]->nome_client);
	pthread_mutex_unlock(sem);

	send(client2, messaggio, strlen(messaggio), 0);
	send(client1, messaggio, strlen(messaggio), 0);
	nanosleep((const struct timespec[]){{0, 500000L}}, NULL);
```

```
                send(client2, "***exItnOw***", strlen("***exItnOw***"), 0);
                free(messaggio);
                free(uscita);

                pthread_mutex_lock(sem);
                tabella_thread[i]->disponibile = 1;
                pthread_mutex_unlock(sem);

                client_online((void *)i);
                pthread_exit(NULL);
            }

            send(client2, messaggio, strlen(messaggio), 0);
            free(messaggio);
        }

        pthread_exit(NULL);

}
```

client.c

```
#include "lib.h"



char*nome_client;
int sent;
long sd;
pthread_t thread_padre;
pthread_t thread;
void *gestione_comunicazione(void *argument);
```

```
void exit_thread(){
	pthread_exit(NULL);
}


void gestione_uscita(){

	if(sent == 0){
		printf("\t*** \e[1mChiusura del canale di comunicazione\e[22m ***\n\n\n");
		send(sd, "quit", strlen("quit"), 0);
		close(sd);
		if(thread == pthread_self()){
			pthread_kill(thread_padre, SIGUSR1);
		}
		else{
			pthread_kill(thread, SIGUSR1);
		}
		pthread_exit(NULL);
	}

	sent = 0;
	char *close_chat;
	close_chat = (char *)malloc(MAX_READER);
	sprintf(close_chat, "%s: %s", nome_client, "quit");
	send(sd, close_chat, strlen(close_chat), 0);
	printf("\033[2J\033[H");
}


int main(){
	struct sockaddr_in server, client;
```

```
	signal(SIGINT, gestione_uscita);
	signal(SIGUSR1, exit_thread);

	printf("\n\t\t\e[34m\e[1m@@@@@@  @@    @@  @@@@@@  @@@@ @@ @@@@@@\e[22m\e[39m\n");
	printf("\t\t\e[34m\e[1m@@    @@    @@  @@    @@ @@ @@   @@\e[22m\e[39m\n");
	printf("\t\t\e[34m\e[1m@@    @@    @@  @@@@@@  @@  @@@   @@\e[22m\e[39m\n");
	printf("\t\t\e[34m\e[1m@@    @@    @@  @@    @@  @@  @@\e[22m\e[39m\n");
	printf("\t\t\e[34m\e[1m@@@@@@  @@@@@@  @@  @@@@@@  @@  @@ @@\e[22m\e[39m\n");

	printf("\n\033[40m\033[1;33mInserisci il tuo nome: \033[0m");
	if(scanf("%ms", &nome_client) == EOF){
		free(nome_client);
		exit(-1);
	}
	nome_client[strlen(nome_client)] = '\0';

	if((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1){
		printf("error init socket\n");
		exit(EXIT_FAILURE);
	}

	client.sin_family = AF_INET;
	client.sin_addr.s_addr = htonl(INADDR_ANY);
	client.sin_port = htons(0);

	if(bind(sd, (struct sockaddr *)&client, sizeof(client)) == -1){
		printf("error bind\n");
		exit(EXIT_FAILURE);
	}
```

```
char *IP;
IP = (char *)malloc(14);
strcpy(IP, "127.0.0.1");
IP[strlen(IP)] = '\0';
server.sin_family = AF_INET;
server.sin_addr.s_addr = inet_addr(IP);
server.sin_port = htons(PORT);

if(connect(sd, (struct sockaddr *)&server, sizeof(server)) == -1){
        printf("error connect\n");
        printf("error code %d\n", errno);
        exit(EXIT_FAILURE);
}


printf("Connesso al server - Digita '\e[1mquit\e[22m' per interrompere la connessione\n\n");
thread_padre = pthread_self();

send(sd, nome_client, strlen(nome_client), 0);


pthread_create(&thread, NULL, (void *)gestione_comunicazione, (void *)sd);

sent = 0;
char *message_input, *message_input_send;
while(1){
        reset:
                message_input = (char *)malloc(MAX_READER);
                if(fgets(message_input, MAX_READER, stdin) == NULL){
                        free(message_input);
                        goto reset;
```

```
                }

                message_input[strlen(message_input)-1] = '\0';
                message_input_send = (char *)malloc(MAX_READER);
                sprintf(message_input_send, "%s: %s", nome_client, message_input);

                if(strcmp(message_input, "quit") == 0 && sent == 0){
                        printf("\t*** \e[1mChiusura del canale di comunicazione\e[22m ***\
n");
                        close(sd);
                        free(message_input);
                        free(message_input_send);
                        pthread_kill(thread, SIGINT);
                        printf("\033[2J\033[H");
                        return 0;
                }

                if(sent == 1){
                        send(sd, message_input_send, strlen(message_input_send), 0);
                }
                else{
                        send(sd, message_input, strlen(message_input), 0);
                }

                if(strcmp(message_input, "quit") == 0){
                        sent = 0;
                }

                free(message_input);
                free(message_input_send);
        }
        return 0;
```

```
}

void *gestione_comunicazione(void *argument){
	long sd, c;
	sd = (long) argument;
	char *message_server;

	while(1){
		message_server = (char *)malloc(MAX_READER);
		c = recv(sd, message_server, MAX_READER, 0);

		if(c == 0){
			free(message_server);
			pthread_kill(thread_padre, SIGUSR1);
			pthread_exit(NULL);
		}
		message_server[c] = '\0';

		if(strcmp(message_server, "\n\t*** \e[91m\e[1mDigita per parlare!\e[22m\e[39m
***\n") == 0){
			printf("%s\n", message_server);
			sent = 1;
			free(message_server);
			continue;
		}

		if(strcmp(message_server, "***exItnOw***") == 0){
			sent = 0;
			free(message_server);
			send(sd, "***exItnOw***", strlen("***exItnOw***"), 0);
			continue;
		}
```

```
            printf("%s\n", message_server);
            free(message_server);
        }
        exit(0);
}
```

lib.h

```
#include <stdio.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <stdlib.h>
#include <time.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <pthread.h>
#include <signal.h>
#include <errno.h>
#include <string.h>

#define PORT 5000
#define MAX_READER 100
```

Makefile

```
CC = gcc
OTP = -pthread
all:
	$(CC) -Wall -Wextra server.c -o server $(OTP)
	$(CC) -Wall -Wextra client.c -o client $(OTP)
server:
	$(CC) -Wall -Wextra server.c -o server $(OTP)
client:
	$(CC) -Wall -Wextra client.c -o client $(OTP)
clean:
	rm client server
```