Elenco telefonico - Relazione

Studente: Dario Sella
Data: Settembre 2025

https://github.com/dariosella/elenco-telefonico

Introduzione

Per l'A.A. 2024/2025 ho deciso di sviluppare il progetto per l'esame di Sistemi Operativi tenuto dal Prof. Francesco Quaglia.

Il progetto è un servizio di elenco telefonico per sistemi operativi Unix, interamente scritto in linguaggio C. Il client connesso al server può loggarsi (o registrarsi) e decidere quale operazione effettuare sulla rubrica (se ha i permessi necessari).

Specifica del progetto

Realizzazione di un servizio "elenco telefonico" supportato da un server che gestisce sequenzialmente o in concorrenza (a scelta) le richieste dei client (residenti in generale su macchine diverse).

L'applicazione client deve fornire le seguenti funzioni:

1. Aggiunta di un record all'elenco telefonico (operazione accessibile solo ad utenti autorizzati).
2. Ricerca di un record all'interno dell'elenco telefonico (anche in questo caso l'operazione deve essere accessibile solo ad utenti autorizzati).

Scelte progettuali, tecniche e metodologie

Sistema Operativo

Il progetto è pensato per l'utilizzo sui sistemi operativi della famiglia **Unix**.

File sorgenti

Il progetto si compone di 5 parti:

- `server.c` - codice del server
- `client.c` - codice del client
- `user.h/user.c` - funzioni per la gestione degli utenti
- `contact.h/contact.c` - funzioni per la gestione dei contatti
- `helper.h/helper.c` - macro e funzioni di utilità

File di configurazione

Il server utilizza 3 file di testo:

- **rubrica**: l'elenco telefonico.
  Ogni riga rappresenta un contatto tramite la sintassi **"Nome [Nomi secondari] Cognome Numero"**.
- **utenti**: l'elenco di ogni utente registrato al servizio.
  Ogni riga rappresenta un utente tramite la sintassi **"Username Password"**.
- **permessi**: l'elenco dei permessi.
  Ogni riga rappresenta il permesso associato ad un'utente specifico tramite la sintassi **"Username Permesso"**.
  I permessi consentiti sono: **r** (lettura), **w** (scrittura), **rw** (lettura e scrittura).

Questi file, se non sono presenti all'interno della cartella del server, verranno creati dal server stesso al momento dell'accesso tramite la funzione `open()` con flag `O_CREAT` della libreria `fcntl.h`.

Per la lettura/scrittura di questi file vengono utilizzate le funzioni `read()/write()` della libreria `unistd.h`.

Comunicazione Client-Server

Per la comunicazione tra il client e il server ho deciso di utilizzare i socket del dominio `AF_INET` (IPv4) di tipo `SOCK_STREAM` con protocollo `TCP` orientato alla connessione.

Il server, quando viene avviato, si mette in ascolto sulla porta passata come parametro da linea di comando.

Il client, quando viene avviato, si connette all'indirizzo IP e numero di porta passati come parametro da linea di comando. Se la connessione tramite IP non

va a buon fine, il server tenterà di connettersi al server tramite DNS nel caso in cui sia stato inserito il nome del server anziché l'indirizzo.

Durante la comunicazione client-server, la formattazione corretta dei dati in **Big Endian/Little Endian** viene garantita dall'utilizzo delle funzioni `ntohl()` e `htonl()` durante l'invio e la ricezione dei dati. Ciò rende questo progetto indipendente dall'architettura hardware su cui viene eseguito.

La funzione `handle()`, definita in `helper.h`, assicura la chiusura del socket di connessione dell'applicazione ancora attiva (server o client) nel caso in cui l'altra termini durante la comunicazione.

Server

Il server è **multithreaded** e per ogni client con cui stabilisce una connessione creerà un thread dedicato.

Il massimo numero di client che il server può gestire contemporaneamente è 8, come definito nell' header `helper.h` dalla macro `LISTENQ`.

Il server utilizza le funzioni definite in `user.h` per fornire al client un servizio di registrazione, login e controllo dei permessi.

Nel caso del login, il sever utilizza la funzione `usrLogin()` che:

- Consulta il file "utenti" in modo sequenziale alla ricerca della coppia "Username Password" inviata dal client.
- Invia al client un codice di successo o fallimento.

Nel caso della registrazione, il server utilizza la funzione `usrRegister()` che:

- Controlla che l'username inviato dal client non sia già utilizzato consultando in modo sequenziale il file "utenti"
- Controlla che il permesso inviato dal client sia ammissibile
- Aggiunge il nuovo utente (username e password) alla fine del file "utenti" e il suo permesso alla fine del file "permessi"
- Invia al client un codice di successo o fallimento

Nel caso dell'aggiunta/ricerca di un contatto da parte di un client, il server prima utilizza la funzione `checkPermission()` che:

- Consulta in modo sequenziale il file "permessi" alla ricerca del permesso associato allo username del client che vuole fare l'operazione.

- Se il permesso combacia con quello necessario per l'operazione allora il server la permette, altrimenti no.

Il server utilizza le funzioni definite in `contact.h` per fornire al client un servizio di aggiunta contatto e ricerca contatto.

Nel caso dell'aggiunta di un contatto, se il client ha i permessi necessari, il server riceve il nome del contatto dal client ed utilizza la funzione `addContact()` che:

- Controlla che sia stato inviato il contatto completo
- Controlla che il contatto non sia un duplicato consultando in modo sequenziale il file "rubrica"
- Aggiunge il contatto alla fine del file "rubrica"

Nel caso della ricerca di un contatto, se il client ha i permessi necessari, il server riceve il contatto dal client ed utilizza la funzione `searchContact()` che:

- Cerca il contatto per nome in modo sequenziale sul file "rubrica"
- Se lo trova invia il contatto completo (nome + numero) al client, altrimenti gli invia un messaggio di fallimento.

Il server utilizza 2 **mutex** per evitare le race condition sui file rubrica e utenti, che potrebbero venire aperti e modificati da più client (thread) contemporaneamente:

- `r_mutex` è utilizzato per la mutua esclusione sul file "rubrica" nel caso di aggiunta o ricerca di un contatto da parte di più client.
- `up_mutex` è utilizzato per la mutua esclusione sul file "utenti" nel caso di registrazione o login da parte di più client.

Il server rimane in ascolto fino a quando non viene interrotto, ad esempio tramite `Ctrl+C`.

Il server gestisce il segnale di interruzione `SIGINT` in modo da garantire una chiusura pulita del socket di ascolto.

Client

Il client una volta connesso al server mostrerà all'utente il menù per il login e registrazione. L'utente dovrà digitare il numero corrispondente all'operazione che vuole fare.

Dopo che l'utente si sarà loggato (o registrato), l'applicazione mostrerà all'utente il menù per l'aggiunta del contatto, ricerca del contatto o l'uscita. L'utente dovrà digitare il numero corrispondente all'operazione che vuole fare.

Prima di eseguire ciascuna di queste operazioni (ad eccezione dell'uscita), il client richiederà al server l'autorizzazione. Se dispone dei permessi necessari, l'operazione verrà eseguita; in caso contrario, verrà mostrato un messaggio all'utente che segnala l'assenza dei permessi.

L'applicazione client in caso di ricerca del contatto eseguirà la funzione `search()`, che chiederà all'utente di inserire il nome del contatto che vuole cercare e lo invierà al server. Poi attenderà la risposta che, una volta ricevuta, la stamperà all'utente.

L'applicazione client in caso di aggiunta del contatto eseguirà la funzione `add()`, che chiederà all'utente di inserire il nome e il numero del contatto che vuole aggiungere all'elenco telefonico e lo invierà la server. Poi attenderà la risposta che, una volta ricevuta, la stamperà all'utente

Per evitare delle connessioni bloccate (deadend), ho deciso di implementare nel client un timer durante il quale l'utente deve inserire l'input.

Per fare ciò ho utilizzato il segnale `SIGALRM` settato al tempo definito dalla macro `TIMEOUT` in `helper.h`. Se l'utente non digita input in tempo, il client riceverà il segnale `SIGALRM` e chiuderà il socket di connessione tramite il gestore `timeout()`. Se invece l'utente digita input, il timer verrà disattivato e poi riattivato alla prossima chiamata di input, come definito dalle funzioni `safeFgets()` e `safeScanf()` in `helper.h`.

Il client gestisce il segnale di interruzione `SIGINT` tramite il gestore `interruptHandler()` garantendo una chiusura pulita del socket di connessione in caso di interruzione.

Contact

I file sorgente `contact.h/contact.c` inglobano le strutture e le funzioni utilizzate dal server per la gestione dei contatti.

Il contatto viene rappresentato nel server tramite una `struct Contact` con attributi:

- Nome completo
- Numero telefonico

User

I file sorgente `user.h/user.c` inglobano le strutture e le funzioni utilizzate dal server per la gestione degli utenti.

L'utente viene rappresentato nel server tramite una `struct User` con attributi:

- Username
- Password

Helper

I file sorgente `helper.h/helper.c` inglobano le macro e le funzioni utilità, utilizzate sia dal client che dal server, che garantiscono il corretto funzionamento del software e una corretta gestione degli errori.

- `handle()` viene utilizzata per controllare il valore di ritorno di `recv()`
- `flushInput()` viene utilizzato per svuotare il buffer `stdin`
- `safeFgets()` viene utilizzato per attivare/disattivare il timer nell'applicazione client e per gestire il valore di ritorno di fgets
- `safeScanf()` viene utilizzato per attivare/disattivare il timer nell'applicazione client e per gestire il valore di ritorno di scanf

---

Manuale d'uso

Per compilare ed eseguire correttamente il programma è necessario avere una macchina con un sistema operativo Unix.

È necessario avere il compilatore **gcc** ed eseguire su shell il comando:

```
make all
```

Dopodiché sulla macchina che dovrà essere il server, eseguire:

```
./server -p <porta>
```

Sulla macchina che dovrà essere il client, eseguire:

```
./client -a <indirizzo IP del server> -p <porta del server>
```

Il server chiederà al client di autenticarsi, dopodiché verrà mostrato il menù dove scegliere l'operazione da fare sull'elenco telefonico. Selezionare l'operazione inserendo i numeri corrispondenti.

Tutti i sorgenti del progetto

Makefile

```
all: server client

server: server.o helper.o user.o contact.o
    gcc server.o helper.o user.o contact.o -o server

client: client.o helper.o
    gcc client.o helper.o -o client

server.o: server.c
    gcc -c server.c

client.o: client.c
    gcc -c client.c

helper.o: helper.c
    gcc -c helper.c

user.o: user.c
    gcc -c user.c

contact.o: contact.c
    gcc -c contact.c

clean:
    rm -f *.o
    rm -f server
    rm -f client
```

server.c

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

#include "helper.h"
#include "user.h"
#include "contact.h"

pthread_mutex_t r_mutex;
pthread_mutex_t up_mutex;
int l_sock; // listen socket

void *clientThread(void *arg); // avvio thread
void interruptHandler(int sig);

int parseCmdLine(int argc, char *argv[], char **sPort);
bool checkPermission(char *username, char *perm);

int main(int argc, char *argv[]){
    struct sockaddr_in server;
    struct sockaddr_in client;
    int c_sock; // client socket

    pthread_mutex_init(&r_mutex, NULL); // inizializza mutex per lettura/scritt
ura su rubrica
    pthread_mutex_init(&up_mutex, NULL); // mutex per login/registra utenti
    signal(SIGINT, interruptHandler);
```

```
    char *sPort; // porta su cui il server ascolta
    parseCmdLine(argc, argv, &sPort); // acquisizione porta passata come ar
gomento da linea di comando

    char *end;
    int port = strtol(sPort, &end, 0);
    if (*end){
        // entra solo se *end non è \0
        puts("porta non riconosciuta");
        exit(EXIT_FAILURE);
    }

    printf("server in ascolto sulla porta %d\n", port);
    l_sock = socket(AF_INET, SOCK_STREAM, 0); // creazione listen socket
    if (l_sock < 0){
        perror("socket");
        exit(EXIT_FAILURE);
    }

    memset(&server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_port = htons(port);
    server.sin_addr.s_addr = htonl(INADDR_ANY);

    // associazione listen socket con indirizzo
    if (bind(l_sock, (struct sockaddr *)&server, sizeof(server)) < 0){
        perror("bind");
        exit(EXIT_FAILURE);
    }

    // server in ascolto
    if (listen(l_sock, LISTENQ) < 0){
        perror("listen");
        exit(EXIT_FAILURE);
    }

    pthread_t tid;
    socklen_t c_size = sizeof(client);
```

```
    while (1){
        // accetta la connessione del client
        if ((c_sock = accept(l_sock, (struct sockaddr *)&client, &c_size)) < 0){
            perror("accept");
            exit(EXIT_FAILURE);
        }

        printf("connessione al server da %s\n", inet_ntoa(client.sin_addr));

        int *c_sockptr = malloc(sizeof(int));
        if (c_sockptr == NULL){
            perror("malloc");
            exit(EXIT_FAILURE);
        }

        *c_sockptr = c_sock; // copia separata da passare al thread

        if (pthread_create(&tid, NULL, clientThread, (void *)c_sockptr) < 0){
            perror("pthread_create");
            exit(EXIT_FAILURE);
        }

        pthread_detach(tid); // se il thread termina non devo joinarlo per rilasci
are le risorse
    }

    return 0;
}

void *clientThread(void *arg){
    int c_sock = *((int *)arg);
    free(arg); // evita memory leak
    int res = -1, net_res = -1;
    int choice, net_choice;
    bool answer;

    // creazione utente del client
    char perm[PERM_SIZE];
```

```
    User *c_user = malloc(sizeof(User));
    if (c_user == NULL){
        perror("malloc");
        pthread_exit(NULL);
    }

    c_user->usr[0] = '\0';
    c_user->pwd[0] = '\0';

    // il client sceglie se registrarsi o loggarsi
    while (res != 0){
        handle(recv(c_sock, &net_choice, sizeof(net_choice), 0), c_sock, SERV
ER); // ricevo scelta del client
        choice = ntohl(net_choice);
        switch (choice){
            case 1:
                // REGISTRAZIONE
                handle(recv(c_sock, c_user->usr, USR_SIZE, 0), c_sock, SERVER);
// client manda username
                handle(recv(c_sock, c_user->pwd, PWD_SIZE, 0), c_sock, SERVE
R); // client manda password
                handle(recv(c_sock, perm, PERM_SIZE, 0), c_sock, SERVER); // cli
ent manda permesso

                pthread_mutex_lock(&up_mutex);
                res = usrRegister(c_user, perm);
                pthread_mutex_unlock(&up_mutex);

                net_res = htonl(res);
                send(c_sock, &net_res, sizeof(net_res), 0);
                break;
            case 2:
                // LOGIN
                handle(recv(c_sock, c_user->usr, USR_SIZE, 0), c_sock, SERVER);
// client manda username
                handle(recv(c_sock, c_user->pwd, PWD_SIZE, 0), c_sock, SERVE
R); // client manda password
```

```
            pthread_mutex_lock(&up_mutex);
            res = usrLogin(c_user);
            pthread_mutex_unlock(&up_mutex);

            net_res = htonl(res);
            send(c_sock, &net_res, sizeof(net_res), 0);
            break;
        }
    }

    // il client sceglie se aggiungere un contatto, cercare un contatto o uscire
    handle(recv(c_sock, &net_choice, sizeof(net_choice), 0), c_sock, SERVE
R); // client manda scelta
    choice = ntohl(net_choice); // conversione in host byte order
    while (choice != 3){
        switch (choice){
            case 1:
                // AGGIUNGI CONTATTO
                answer = checkPermission(c_user->usr, "w");
                // answer è bool (1 byte) quindi non serve convertirlo in network b
yte order
                send(c_sock, &answer, sizeof(answer), 0); // invia risultato al clien
t
                if (answer){
                    // accesso consentito
                    char buffer[BUF_SIZE];
                    handle(recv(c_sock, buffer, BUF_SIZE, 0), c_sock, SERVER); // il
client invia contatto
                    Contact *contatto = createContact(buffer);
                    if (contatto == NULL){
                        perror("malloc");
                        pthread_exit(NULL);
                    }
                    memset(buffer, 0, BUF_SIZE); // azzero per poi memorizzare la
risposta del server

                    pthread_mutex_lock(&r_mutex);
                    addContact(contatto, buffer); // aggiungo il contatto in rubrica
```

```
                pthread_mutex_unlock(&r_mutex);

                send(c_sock, buffer, BUF_SIZE, 0); // invio del risultato al client
                free(contatto); // evita memory leak
            }
            break;
        case 2:
            // CERCA CONTATTO
            answer = checkPermission(c_user->usr, "r");
            send(c_sock, &answer, sizeof(answer), 0); // invia risultato al clien
t
            if (answer){
                // accesso consentito
                char buffer[BUF_SIZE];
                handle(recv(c_sock, buffer, BUF_SIZE, 0), c_sock, SERVER); // il
client invia il contatto
                Contact *contatto = createContact(buffer);
                if (contatto == NULL){
                    perror("malloc");
                    pthread_exit(NULL);
                }
                memset(buffer, 0, BUF_SIZE);

                pthread_mutex_lock(&r_mutex);
                searchContact(contatto, buffer); // cerca il contatto in rubrica
                pthread_mutex_unlock(&r_mutex);

                send(c_sock, buffer, BUF_SIZE, 0); // invio del risultato al client
                free(contatto); // evita memory leak
            }
            break;
    }
    handle(recv(c_sock, &net_choice, sizeof(net_choice), 0), c_sock, SERV
ER);
    choice = ntohl(net_choice);
  }

  puts("client in uscita");
```

```
    close(c_sock);
    free(c_user);
    pthread_exit(NULL);
}

int parseCmdLine(int argc, char *argv[], char **sPort) {
    if (argc < 3){
        printf("Usage: %s -p (port) [-h]\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    for (int i = 1; i < argc; i++){
        if (!strcmp(argv[i], "-p") || !strcmp(argv[i], "-P")){
            *sPort = argv[i + 1];
        } else if (!strcmp(argv[i], "-h") || !strcmp(argv[i], "-H")){
            printf("Usage: %s -p (port) [-h]\n", argv[0]);
            exit(EXIT_FAILURE);
        }
    }

    return 0;
}

void interruptHandler(int sig){
    puts("Server interrotto, chiusura connessione");
    close(l_sock);
    exit(0);
}
```

client.c

```
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
```

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdbool.h>
#include <string.h>
#include <signal.h>

#include "helper.h"

int parseCmdLine(int argc, char *argv[], char **sAddr, char **sPort);
void timeout(int sig);
void interruptHandler(int sig);

void search();
void add();

int s_sock; // socket di connessione al server

int main(int argc, char *argv[]){
    struct sockaddr_in server;
    struct hostent *hp;

    signal(SIGALRM, timeout); // timer di inattività
    signal(SIGINT, interruptHandler);

    char *sAddr, *sPort, *end;
    parseCmdLine(argc, argv, &sAddr, &sPort);

    int port = strtol(sPort, &end, 0);
    if (*end){
        puts("porta non riconosciuta");
        exit(EXIT_FAILURE);
    }

    if ( (s_sock = socket(AF_INET, SOCK_STREAM, 0) ) < 0){
        perror("socket");
        exit(EXIT_FAILURE);
    }
```

```
memset(&server, 0, sizeof(server));
server.sin_family = AF_INET;
server.sin_port = htons(port);
if (inet_aton(sAddr, &server.sin_addr) <= 0){
    perror("inet_aton");
    if ( (hp = gethostbyname(sAddr)) == NULL){
        perror("gethostbyname");
        exit(EXIT_FAILURE);
    }
    server.sin_addr = *((struct in_addr *)hp->h_addr);
}

if ( connect(s_sock, (struct sockaddr *)&server, sizeof(server)) < 0){
    perror("connect");
    exit(EXIT_FAILURE);
}

// client connesso
char username[USR_SIZE];
char password[PWD_SIZE];
char perm[PERM_SIZE];
int choice, net_choice;
int res = -1, net_res = -1;
bool answer;

// scegliere se loggare o registrarsi
while (res != 0){
    printf("%s", "Scegliere se:\n1. Registrarti\n2. Loggarti\n");
    safeScanf(&choice);
    net_choice = htonl(choice);
    send(s_sock, &net_choice, sizeof(net_choice), 0);
    switch (choice){
        case 1:
            // REGISTRAZIONE
            memset(username, 0, USR_SIZE);
            memset(password, 0, PWD_SIZE);
            memset(perm, 0, PERM_SIZE);
```

```
            printf("%s", "Inserisci username: ");
            safeFgets(username, USR_SIZE);

            printf("%s", "Inserisci password: ");
            safeFgets(password, PWD_SIZE);

            printf("%s", "Inserisci un permesso tra:\nlettura - 'r'\nscrittura -
'w'\nlettura e scrittura - 'rw'\n");
            safeFgets(perm, PERM_SIZE);

            while (strcmp(perm, "r") != 0 && strcmp(perm, "w") != 0 && strcm
p(perm, "rw") != 0){
                // controllo se non scrive r, w, o rw
                memset(perm, 0, PERM_SIZE);
                printf("%s", "Inserisci un permesso tra:\nlettura - 'r'\nscrittura -
'w'\nlettura e scrittura - 'rw'\n");
                safeFgets(perm, PERM_SIZE);
            }

            send(s_sock, username, USR_SIZE, 0); // invia username al server
            send(s_sock, password, PWD_SIZE, 0); // invia password al server
            send(s_sock, perm, PERM_SIZE, 0); // invia permesso al server

            handle(recv(s_sock, &net_res, sizeof(net_res), 0), s_sock, CLIEN
T); // riceve il risultato
            res = ntohl(net_res);
            switch (res){
                case 1:
                    printf("%s già utilizzato\n", username);
                    break;
                case -1:
                    puts("Errore");
                    break;
            }
            break;
        case 2:
            // LOGIN
            memset(username, 0, USR_SIZE);
```

```
            memset(password, 0, PWD_SIZE);

            printf("%s", "Inserisci username: ");
            safeFgets(username, USR_SIZE);

            printf("%s", "Inserisci password: ");
            safeFgets(password, PWD_SIZE);

            send(s_sock, username, USR_SIZE, 0); // invia username al server
            send(s_sock, password, PWD_SIZE, 0); // invia password al server

            handle(recv(s_sock, &net_res, sizeof(net_res), 0), s_sock, CLIEN
T); // riceve il risultato
            res = ntohl(net_res);
            switch (res){
               case 1:
                  puts("Password sbagliata");
                  break;
               case 2:
                  printf("%s Non esiste\n", username);
                  break;
               case -1:
                  puts("Errore");
                  break;
            }
            break;
      }
   }

   printf("Benvenuto %s!\n", username);
   printf("%s", "Scegliere:\n1. Aggiungere contatto\n2. Cercare contatto\n3.
Uscire\n");
   safeScanf(&choice);

   while (choice != 3){
      switch (choice){
         case 1:
            net_choice = htonl(choice);
```

```
            send(s_sock, &net_choice, sizeof(net_choice), 0);
            // il server mi dice se ho i permessi necessari
            handle(recv(s_sock, &answer, sizeof(answer), 0), s_sock, CLIEN
T);
            if (answer){
               search();
            } else {
               puts("Non hai i permessi necessari per aggiungere contatti");
            }
            break;
         case 2:
            net_choice = htonl(choice);
            send(s_sock, &net_choice, sizeof(net_choice), 0);
            // il server mi dice se ho i permessi necessari
            handle(recv(s_sock, &answer, sizeof(answer), 0), s_sock, CLIEN
T);
            if (answer){
               add();
            } else {
               puts("Non hai i permessi necessari per cercare contatti");
            }
            break;
         default:
            puts("Scelta non valida");
      }
      printf("%s", "Scegliere:\n1. Aggiungere contatto\n2. Cercare contatto\n
3. Uscire\n");
      safeScanf(&choice);
   }

   net_choice = htonl(choice);
   send(s_sock, &net_choice, sizeof(net_choice), 0); // se choice è 3
   close(s_sock);
   exit(EXIT_SUCCESS);
}

int parseCmdLine(int argc, char *argv[], char **sAddr, char **sPort){
   if (argc < 5){
```

```
        printf("Usage: %s -a (indirizzo remoto) -p (porta remota) [-h]\n", argv
[0]);
        exit(EXIT_FAILURE);
    }

    for (int i = 0; i < argc; i++){
        if (!strcmp(argv[i], "-a") || !strcmp(argv[i], "-A")){
            *sAddr = argv[i + 1];
        } else if (!strcmp(argv[i], "-p") || !strcmp(argv[i], "-P")){
            *sPort = argv[i + 1];
        } else if (!strcmp(argv[i], "-h") || !strcmp(argv[i], "-H")){
            printf("Usage: %s -a (indirizzo remoto) -p (porta remota) [-h]\n", ar
gv[0]);
            exit(EXIT_FAILURE);
        }
    }

    return 0;
}

void timeout(int sig){
    puts("Sei stato inattivo troppo tempo, chiusura connessione");
    close(s_sock);
    exit(0);
}

void interruptHandler(int sig){
    puts("Client interrotto, chiusura connessione");
    close(s_sock);
    exit(0);
}

void search(){
    char buffer[BUF_SIZE];
    printf("%s", "Inserisci 'Nome [Nomi secondari] Cognome Numero'\n");
    safeFgets(buffer, BUF_SIZE);

    send(s_sock, buffer, BUF_SIZE, 0);
```

```
    memset(buffer, 0, BUF_SIZE);
    // risposta dal sever di successo o fallimento
    handle(recv(s_sock, buffer, BUF_SIZE, 0), s_sock, CLIENT);
    printf("%s", buffer);
}

void add(){
    char buffer[BUF_SIZE];
    printf("%s", "Inserisci 'Nome [Nomi secondari] Cognome'\n");
    safeFgets(buffer, BUF_SIZE);

    send(s_sock, buffer, BUF_SIZE, 0);
    memset(buffer, 0, BUF_SIZE);
    // risposta dal server con contatto e numero in caso di successo
    handle(recv(s_sock, buffer, BUF_SIZE, 0), s_sock, CLIENT);
    printf("%s", buffer);
}
```

user.h/user.c

```
#include "helper.h"

#include <stdbool.h>

typedef struct {
    char usr[USR_SIZE];
    char pwd[PWD_SIZE];
} User;

int usrLogin(User *utente);
int usrRegister(User *utente, char *perm);
bool checkPermission(char *username, char *perm);
```

```
#include "user.h"

#include <stdio.h>
#include <stdlib.h>
```

```
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <stdbool.h>

int usrLogin(User *utente){
    // ogni riga di users è "username password\n"
    int fd = open("utenti", O_RDONLY);
    if (fd == -1){
        perror("open");
        return -1;
    }

    char buffer[BUF_SIZE];

    int i = 0;
    char c;
    while (read(fd, &c, 1) == 1){
        // leggo un carattere alla volta
        buffer[i++] = c;
        if (c == '\n'){
            // ho letto una riga: "username password\n"
            buffer[i] = '\0';
            char *username = strtok(buffer, " \n");
            char *password = strtok(NULL, " \n");

            if (username != NULL && password != NULL){
                if (strcmp(username, utente->usr) == 0 && strcmp(password, ute
nte->pwd) == 0){
                    close(fd);
                    return 0; // utente riconosciuto
                } else if (strcmp(username, utente->usr) == 0 && strcmp(passwor
d, utente->pwd) != 0){
                    close(fd);
                    return 1; // password sbagliata
                }
            }
```

```
            memset(buffer, 0, BUF_SIZE);
            i = 0;
        }
    }

    close(fd);
    return 2; // utente non esiste
}

int usrRegister(User *utente, char *perm){
    int fd = open("utenti", O_CREAT | O_RDWR | O_APPEND, 0600);
    int fd2 = open("permessi", O_CREAT | O_WRONLY | O_APPEND, 0600);
    if (fd == -1 || fd2 == -1){
        perror("open");
        return -1;
    }

    char buffer[BUF_SIZE];
    int i = 0;
    char c;
    // controllo se lo username è già utilizzato
    while (read(fd, &c, 1) == 1){
        // leggo un carattere alla volta
        buffer[i++] = c;
        if (c == '\n'){
            // ho letto una riga: "username password\n"
            buffer[i] = '\0';
            char *username = strtok(buffer, " \n");
            char *password = strtok(NULL, " \n");

            if (username != NULL){
                if (strcmp(username, utente->usr) == 0){
                    close(fd);
                    close(fd2);
                    return 1; // username già utilizzato
                }
            }
```

```
            memset(buffer, 0, BUF_SIZE);
            i = 0;
        }
    }

    // scrittura "username password\n" sul file utenti
    memset(buffer, 0, BUF_SIZE);
    snprintf(buffer, BUF_SIZE, "%s %s\n", utente->usr, utente->pwd);
    write(fd, buffer, strlen(buffer));

    // scrittura "username permission\n" sul file permessi
    memset(buffer, 0, BUF_SIZE);
    snprintf(buffer, BUF_SIZE, "%s %s\n", utente->usr, perm);
    write(fd2, buffer, strlen(buffer));

    close(fd);
    close(fd2);
    return 0; // utente registrato
}

bool checkPermission(char *username, char *perm){
    // ogni riga di permission è "username permission\n"
    int fd = open("permessi", O_RDONLY);
    if (fd == -1){
        perror("open");
        return false;
    }

    char buffer[BUF_SIZE];
    int i = 0;
    char c;
    while (read(fd, &c, 1) == 1){
        buffer[i++] = c;
        if (c == '\n'){
            // ho letto una riga "username permission\n"
            buffer[i] = '\0';
            char *buffer_username = strtok(buffer, " \n");
            char *buffer_perm = strtok(NULL, " \n");
```

```
            if (buffer_username != NULL && buffer_perm != NULL){
                if (strcmp(username, buffer_username) == 0) {
                    if (strstr(buffer_perm, perm) != NULL) {
                        // l'utente ha il permesso
                        close(fd);
                        return true;
                    } else {
                        close(fd);
                        return false;
                    }
                }
            }

            memset(buffer, 0, BUF_SIZE);
            i = 0;
        }
    }

    close(fd);
    return false;
}
```

contact.h/contact.c

```
#include "helper.h"

typedef struct {
    char name[NAME_SIZE];
    char number[NUMBER_SIZE];
} Contact;

Contact *createContact(char *buffer);
void addContact(Contact *contatto, char *answer);
void searchContact(Contact *contatto, char *answer);
```

```
#include "contact.h"
```

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>

Contact *createContact(char *buffer){
    // funzione per la creazione di un nuovo contatto da inserire o cercare
    Contact *contatto = malloc(sizeof(Contact));
    if (contatto == NULL){
        perror("malloc");
        return NULL;
    }

    contatto->name[0] = '\0';
    contatto->number[0] = '\0';

    char *token = strtok(buffer, " \n");
    while (token != NULL){
        char *end;
        long num = strtol(token, &end, 10);

        if (*end == '\0'){
            // il token è il numero di telefono
            strcpy(contatto->number, token);
        } else {
            // mantiene la sintassi "nome [nomi secondari] cognome"
            strcat(contatto->name, token);
            strcat(contatto->name, " ");
        }

        token = strtok(NULL, " \n");
    }

    int namelen = strlen(contatto->name);
    contatto->name[namelen - 1] = '\0'; // toglie lo spazio finale

    return contatto;
```

```
}

void addContact(Contact *contatto, char *answer){
   // funzione per l'aggiunta di un nuovo contatto alla fine della rubrica
   int fd = open("rubrica", O_CREAT | O_WRONLY | O_APPEND, 0600);
   if (fd == -1){
      perror("open");
      strcpy(answer, "Errore nell'apertura della rubrica\n");
      return; // errore
   }

   // controllo che l'utente abbia inserito sia il nome che il numero
   if (contatto->name[0] == '\0' || contatto->number[0] == '\0'){
      strcpy(answer, "Nome o Numero non inseriti\n");
      close(fd);
      return;
   }

   char buffer[BUF_SIZE];
   snprintf(buffer, BUF_SIZE, "%s %s\n", contatto->name, contatto->numbe
r);

   // controllo se contatto già esiste
   char temp[BUF_SIZE];
   searchContact(contatto, temp);
   if (strcmp(buffer, temp) == 0){
      // se il contatto già esiste
      strcpy(answer, "Il contatto già esiste\n");
      close(fd);
      return;
   }
   write(fd, buffer, strlen(buffer));

   strcpy(answer, "Il contatto è stato aggiunto con successo\n");
   close(fd);
   return; // contatto aggiunto
}
```

```
void searchContact(Contact *contatto, char *answer){
   int fd = open("rubrica", O_RDONLY);
   if (fd == -1){
      perror("open");
      strcpy(answer, "Errore nell'apertura della rubrica\n");
      return;
   }

   char buffer[BUF_SIZE];
   int i = 0;
   char c;

   while (read(fd, &c, 1) == 1){
      buffer[i++] = c;\
      if (c == '\n'){
         buffer[i] = '\0';

         // temp contiene il contatto preso dalla rubrica
         Contact *temp = createContact(buffer);
         if (temp == NULL){
            perror("malloc");
            strcpy(answer, "Errore di memoria insufficente\n");
            close(fd);
            return; // errore
         }

         if (strcmp(temp->name, contatto->name) == 0){
            strcpy(contatto->number, temp->number);

            snprintf(answer, BUF_SIZE, "%s %s\n", contatto->name, contatto-
>number); // contatto trovato

            free(temp);
            close(fd);
            return;
         }

         free(temp);
```

```
            memset(buffer, 0, BUF_SIZE);
            i = 0;
        }
    }

    strcpy(answer, "Il contatto non esiste\n"); // contatto non trovato
    close(fd);
    return;
}
```

helper.h/helper.c

```
#define TIMER 30
#define LISTENQ (8)
#define BUF_SIZE (256)
#define NAME_SIZE (64)
#define NUMBER_SIZE (32)
#define USR_SIZE (64)
#define PWD_SIZE (64)
#define PERM_SIZE (3)
#define SERVER (0)
#define CLIENT (1)

#include <stddef.h>

void handle(int res, int sock, int who);
void flushInput();
void safeFgets(char *buffer, size_t size);
void safeScanf(int *val);
```

```
#include "helper.h"

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <string.h>
```

```
void handle(int res, int sock, int who){
    if (res == 0){
        // la connessione è stata chiusa
        close(sock);
        if (who == SERVER){
            puts("Connessione chiusa");
            pthread_exit(NULL);
        } else if (who == CLIENT){
            puts("Connessione chiusa");
            exit(EXIT_SUCCESS);
        }
    } else if (res == -1){
        // errore
        perror("recv");
        if (who == SERVER){
            pthread_exit(NULL);
        } else if (who == CLIENT){
            exit(EXIT_FAILURE);
        }
    }
}

void flushInput(){
    char c;
    while ( (c = getchar()) != '\n' && c != EOF);
}

void safeFgets(char *buffer, size_t size){
    alarm(TIMER);
    if (fgets(buffer, size, stdin) == NULL) {
          alarm(0);
          puts("Errore nella scrittura o fine del file");
          exit(0);
    }

    alarm(0);
    if (strchr(buffer, '\n') != NULL){
```

```
        buffer[strcspn(buffer, "\n")] = '\0';
    } else {
        flushInput();
    }

    return;
}

void safeScanf(int *val){
    alarm(TIMER);
    if (scanf("%d", val) == 1){
        alarm(0);
        flushInput();
        return;
    } else {
        alarm(0);
        puts("Input non valido");
        flushInput();
        exit(0);
    }
}
```