Sistemi Operativi

SERVIZIO DI MESSAGGISTICA

Antonio D'Orazio | Mat. 0242178
Prof. Francesco Quaglia

Specifica dell’applicazione

Realizzazione di un servizio di scambio messaggi supportato tramite un server sequenziale o concorrente (a scelta). Il servizio deve accettare messaggi provenienti da client (ospitati in generale su macchine distinte da quella dove risiede il server) ed archiviarli.

L'applicazione client deve fornire ad un utente le seguenti funzioni:
1. Lettura tutti i messaggi spediti all'utente.
2. Spedizione di un nuovo messaggio a uno qualunque degli utenti del sistema.
3. Cancellare dei messaggi ricevuti dall'utente.

Un messaggio deve contenere almeno i campi Destinatario, Oggetto e Testo.

Si precisa che la specifica prevede la realizzazione sia dell'applicazione client che di quella server. Inoltre, il servizio potrà essere utilizzato solo da utenti autorizzati (deve essere quindi previsto un meccanismo di autenticazione).

Metodologie utilizzate

Si è scelto di organizzare i dati nella seguente maniera:

Il client ha il solo scopo di presentazione. Si occupa di raccogliere le richieste dell’utente, inviarle al server e di restituire le informazioni precedentemente elaborate in maniera grafica. Ciò favorisce l’aggiornamento delle funzionalità mantenendo intatte le chiamate.

Il server si occupa di gestire ed elaborare le informazioni dell’applicazione e le richieste del client, a cui restituirà i risultati dopo aver elaborato le informazioni rendendole utilizzabili.

Per garantire coerenza e sincronizzazione tra le due applicazioni, viene utilizzata una variabile di esito il cui contenuto è inviato e letto fra i due eseguibili in modo da indirizzare l’esecuzione verso il giusto flusso a seconda che un’operazione sia riuscita oppure no.

Tra server e client avvengono scambi di sole stringhe, in modo da evitare problemi circa la diversa codifica dei numeri interi fra diverse architetture.

Gli applicativi sono robusti in caso di interruzioni improvvise: ciò avviene gestendo il segnale SIGPIPE e verificando la consistenza dei dati ricevuti in lettura tra client e server. In caso invece di attesa indefinita da parte del server di un dato dal client che non avverrà mai (esempio: l'utente apre il programma e poi si allontana dal computer) scatta un meccanismo di timeout che consente di terminare la connessione in maniera sicura.

Inoltre, è possibile accedere con più utenti in concorrenza, in quanto una volta accettata una connessione con un client tutte le operazioni relative sono gestite da un nuovo processo creato mediante fork. L'accesso ai file avviene in modo sicuro e sincronizzato fra i diversi processi attivi grazie all'utilizzo di un semaforo.

L'applicazione è stata sviluppata utilizzando le librerie Posix per sistemi UNIX.

Descrizione delle funzionalità

Server

Il server comprende la seguente lista di funzioni:

```
int registerUser(char *username, char *password);
int requestLogin(char *username, char *password);
void sendMessage(int socket_to);
void delMessage(int socket_to);
void getMessagesByUser(int socket_to);
int query_id();
```

REGISTER USER

La funzione riceve in input username e password inseriti dall'utente, e li concatena in un'unica stringa assieme ad un separatore. Dopodiché apre in lettura ed in append il file contenente la lista degli utenti registrati, e per ogni coppia username~password ne estrae l'username che viene così confrontato con il nome dell'utente da creare. Se l'utente è presente nel file la funzione restituisce 0, altrimenti inserisce la nuova stringa concatenata e restituisce 1.

REQUEST LOGIN

La funzione riceve in input username e password inseriti dall'utente, e li concatena in un'unica stringa assieme ad un separatore. Dopodiché apre in sola lettura il file contenente la lista degli utenti registrati, ed effettua un controllo coppia per coppia tra il contenuto e la stringa concatenata. Se l'utente è presente nel file la funzione restituisce 1, altrimenti 0.

SEND MESSAGE

La funzione riceve in input il socket attraverso cui comunicare con il client. A questo punto apre in append il file dei messaggi, e crea una struct contenente le informazioni sul messaggio da inviare, a cui aggiunge l'ultimo id libero nel file generato tramite la funzione query_id. Infine, esegue una fwrite sul file con il nuovo messaggio.

DEL MESSAGE

La funzione riceve in input il socket attraverso cui comunicare con il client.

Dopo aver richiamato getMessagesByUser per far visualizzare all'utente i messaggi ricevuti, attende in input dal client l'ID del messaggio da rimuovere. A questo punto esegue due operazioni sui file:

- Apre il file dei messaggi esistente in modalità lettura
- Crea un nuovo file temporaneo in scrittura

Dopo aver fatto questo, copia tutto il contenuto del file originale in quello temporaneo, escludendo tramite un controllo la riga da eliminare. Il controllo verifica anche che il destinatario del messaggio corrisponda con l'utente connesso, in modo da evitare cancellazioni indesiderate.

Dopo aver chiuso entrambi i file, elimina messages.dat originario e rinomina temp.dat in messages.dat

GET MESSAGES BY USER

La funzione riceve in input il socket a cui inviare i dati.

Dopo aver aperto il file dei messaggi in modalità di sola lettura, effettua una scansione sui messaggi presenti filtrandoli per nome utente. Per ogni messaggio trovato, dopo averlo memorizzato in una struct, prima invia il valore "1" tramite la variabile next_message, per avvisare di un nuovo messaggio in arrivo, e poi invia il messaggio stesso. Al termine, invia il valore "0" per avvisare il client che non ci sono più messaggi in arrivo.

QUERY ID

Legge tutti gli id presenti all'interno del file e restituisce il più grande incrementato di uno. Se non è presente alcun record, restituisce 0, che sarà quindi utilizzato come id iniziale.

MAIN

Dopo il controllo sulla quantità di parametri immessi tramite argv, vengono eseguite tutte le operazioni per aprire un socket di ascolto. A questo punto si esegue un ciclo

infinito in cui il socket rimane in ascolto e, per ogni client che richiede la connessione, viene stabilita una connessione tramite accept.

Ora, per ogni client connesso viene eseguita una fork() per istanziare un nuovo processo, che eseguirà le seguenti operazioni.

- Chiusura del socket di ascolto
- Richiesta al client della scelta tra login e registrazione, insieme a password e nome utente, e conseguente reindirizzamento alle funzioni requestLogin o registerUser, le quali gestiranno in maniera opportuna i dati immessi
- Autenticazione con le credenziali immesse (eventualmente appena create)
- Passaggio ad un loop infinito contenente le scelte disponibili nel menu principale dell'applicazione, e reindirizzamento alle rispettive funzioni dell'operazione scelta.
- Quando richiesto dall'utente, chiusura del socket di connessione con conseguente terminazione dell'istanza.

Client

Il client comprende la seguente lista di funzioni:

```
int registrazione();
int eseguiLogin();
void messaggiRicevuti();
void inviaMessaggio();
void eliminaMessaggio();
void sigpipeHandler();
void checkTimeout(int res);
```

REGISTRAZIONE

La funzione richiede in input username e password per la registrazione. Dopo aver ricevuto i dati, li invia al server e richiede l'esito. Se l'esito è positivo (valore 1) copia il nuovo username nella variabile active_user per convalidare l'autenticazione con l'utente appena creato. In entrambi i casi, restituisce l'esito.

ESEGUI LOGIN

La funzione richiede in input username e password per il login. Dopo aver ricevuto i dati, li invia al server e richiede l'esito. Se l'esito è positivo (valore 1) copia l'username nella variabile active_user per convalidare l'autenticazione. In entrambi i casi, restituisce l'esito.

MESSAGGI RICEVUTI

Dopo aver verificato che esistano dei messaggi in arrivo, richiedendo il valore della variabile next_message, richiede dal server il messaggio successivo, lo memorizza in una struct e ne stampa a schermo il contenuto.

INVIA MESSAGGIO

Richiede in input all'utente tutte le informazioni necessarie per creare un messaggio, e le memorizza in una struct. Dopodiché invia sequenzialmente ogni informazione e richiede al server l'esito della scrittura, in base al quale elabora il messaggio da mostrare all'utente.

ELIMINA MESSAGGIO

Effettua una chiamata a messaggiRicevuti() e prende in input dall'utente l'id del messaggio da eliminare. A questo punto invia al server l'id ricevuto, e richiede l'esito dell'operazione, in base al quale elabora il messaggio da mostrare all'utente.

SIGPIPE HANDLER

È la funzione chiamata in caso di terminazione inaspettata della connessione con il server. Stampa all'utente un segnale di errore e termina il programma.

CHECK TIMEOUT

È chiamata al termine di ogni read() e riceve in ingresso il valore restituito dalla read stessa. In base a questo controlla se vi è stato un timeout nella connessione ed in caso termina il programma.

MAIN

Dopo il controllo sulla quantità di parametri immessi tramite argv, il client un socket ed impostata ip e porta desiderate, successivamente si connette al server.

A connessione riuscita viene creato un primo ciclo, che terminerà solamente ad autenticazione eseguita. In questo ciclo viene presa in input la scelta dell'utente ("1" per effettuare il login, "2" per creare un nuovo utente), che viene inviata anche al server, e viene chiamata la funzione corrispondente alla scelta desiderata, ossia eseguiLogin o registrazione.

Una volta convalidata l'autenticazione, viene creato un loop infinito contente il menu principale dell'applicazione. L'utente dovrà inserire il numero corrispondente alla funzionalità desiderata (oppure "4" per chiudere il loop e terminare così l'applicazione), che sarà inviato anche al server per coordinare client e server con la funzionalità richiesta. Successivamente il client chiama la funzione necessaria dove verrà proseguito lo svolgimento della funzionalità, per poi ritornare al menu principale.

Installazione ed esecuzione

Compilare i sorgenti con

```
gcc -o server.o server.c
```

```
gcc -o client.o client.c
```

Per eseguire, avviare prima il server e successivamente il client. Nel server sarà necessario specificare la porta a cui connettersi, nel client la porta e l'indirizzo del server. Supponendo di utilizzare la porta 5302 e di effettuare una connessione in locale, i comandi da dare in due diverse istanze di terminale Linux saranno:

```
./server.o 5302          ./client.o 127.0.0.1 5302
```

Sorgenti del programma

Server.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <sys/sem.h>

typedef struct messaggio messaggio;
struct messaggio {
    char id_messaggio[4];
    char mittente[32];
    char destinatario[32];
    char oggetto[64];
    char testo[2048];
};

int registerUser(char *username, char *password);
int requestLogin(char *username, char *password);
void sendMessage(int socket_to);
void delMessage(int socket_to);
void getMessagesByUser(int socket_to);
int query_id();

char current_user[32];

int id_sem;
int chiave_sem = 1234;
struct sembuf sem_lock;
struct sembuf sem_unlock;

int main(int argc, char *argv[])
```

```
{
    // Controllo sugli argomenti
    if (argc < 2) {
        printf("Usage: ./server.o port\n");
        exit(EXIT_FAILURE);
    }

    int port = atoi(argv[1]);

    int listening_socket;
    int connection_socket;
    struct sockaddr_in server_address;
    struct sockaddr_in socket_address;
    int sin_size;

    printf("Server in ascolto sulla porta %d \n", port);

    // Creazione listening socket
    if ((listening_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("Errore nella creazione del listening socket \n");
        exit(EXIT_FAILURE);
    }

    //Imposta i byte nel socket a zero e riempi i dati significativi
    memset(&server_address, 0, sizeof(server_address));
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = htonl(INADDR_ANY);
    server_address.sin_port = htons(port);

    // Bind il socket address sul socket di ascolto
    if (bind(listening_socket, (struct sockaddr *) &server_address,
sizeof(server_address)) < 0)
    {
        printf("Errore durante bind \n");
        exit(EXIT_FAILURE);
    }

    // Chiamo la listen()
    if (listen(listening_socket, 5) < 0)
    {
        printf("Errore durante listen \n");
        exit(EXIT_FAILURE);
    }

    // Creo il semaforo per gestire gli accessi ai file
    id_sem = semget(chiave_sem, 1, IPC_CREAT | 0666);
    if (id_sem == -1) {
```

```
        printf("Errore nella creazione del semaforo \n");
        exit(EXIT_FAILURE);
    }
    semctl(id_sem, 0, SETVAL, 1); // Inizializzo il semaforo a valore 1

    sem_lock.sem_num = 0;
    sem_lock.sem_op = -1;
    sem_lock.sem_flg = SEM_UNDO;

    sem_unlock.sem_num = 0;
    sem_unlock.sem_op = 1;
    sem_unlock.sem_flg = SEM_UNDO;

    // Ciclo per accettare piu collegamenti con i client
    while (1)
    {
        sin_size = sizeof(struct sockaddr_in);

        if ((connection_socket = accept(listening_socket, (struct sockaddr *)
&socket_address, &sin_size)) < 0)
        {
            printf("Connessione non accettata \n");
            exit(EXIT_FAILURE);
        }

        printf("Connessione accettata da %s \n",
inet_ntoa(socket_address.sin_addr));

        // Creo nuovo processo per sessione
        if (!fork())
        {
            close(listening_socket);

            int res;

            // Imposto il timeout per una lettura
            struct timeval tv;
            tv.tv_sec = 300; // 5 Minuti
            tv.tv_usec = 0;
            setsockopt(connection_socket, SOL_SOCKET, SO_RCVTIMEO, (const
char*)&tv, sizeof tv);

            char username[32];
            char password[32];

            char scelta[2];
            strcpy(scelta, "0");
            char esito[2];
```

```
strcpy(esito, "0");

// Ciclo per autenticazione/registrazione
printf("In attesa di autenticazione \n");
while (1)
{
    res = read(connection_socket, scelta, 1);
    if (res == -1 || res == 0)
    {
        printf("Disconnected from client \n");
        exit(EXIT_FAILURE);
    }

    printf("Attendo username\n");
    res = read(connection_socket, username, 32);
    if (res == -1 || res == 0)
    {
        printf("Disconnected from client \n");
        exit(EXIT_FAILURE);
    }
    printf("Username: %s\n", username);

    printf("Attendo password\n");
    res = read(connection_socket, password, 32);
    if (res == -1 || res == 0)
    {
        printf("Disconnected from client \n");
        exit(EXIT_FAILURE);
    }
    printf("Password: %s\n", password);

    if (strcmp(scelta, "1") == 0) {
        if (!requestLogin(username, password)) {

            strcpy(esito, "0");
            write(connection_socket, esito, 1);
        }
        else { break; }
    }
    else if (strcmp(scelta, "2") == 0) {
        if (!registerUser(username, password)) {
            strcpy(esito, "0");
            write(connection_socket, esito, 1);
        }
        else { break; }
    }
    strcpy(scelta, "0");
```

```
        }
        strcpy(esito, "1");
        write(connection_socket, esito, 1);

        // Copio l'username nella variabile dedicata all'utente connesso
        memcpy(current_user, username, sizeof(current_user));
        printf("Utente %s autenticato \n", current_user);

        // Ciclo per menu principale
        strcpy(scelta, "0");
        while (1)
        {
            printf("Pronto\n");
            res = read(connection_socket, scelta, 1);
            if (res == -1 || res == 0)
            {
                printf("Disconnected from client \n");
                exit(EXIT_FAILURE);
            }
            printf("Scelta effettuata: %s \n", scelta);

            if (strcmp(scelta, "1") == 0)
            {
                printf("Messaggi ricevuti\n");
                getMessagesByUser(connection_socket);
            }
            else if (strcmp(scelta, "2") == 0)
            {
                printf("Invia messaggio\n");
                sendMessage(connection_socket);
            }
            else if (strcmp(scelta, "3") == 0) {
                printf("Elimina messaggi\n");
                delMessage(connection_socket);
            }
            else if (strcmp(scelta, "4") == 0) {
                printf("Chiusura connessione \n");
                break;
            }
            strcpy(scelta, "0");
        }
        close(connection_socket);
        exit(EXIT_SUCCESS);
    }
}

// Termino la conessione
if (close(listening_socket) < 0)
```

```
    {
        printf("Errore durante close \n");
        exit(EXIT_FAILURE);
    }

    exit(EXIT_SUCCESS);

}

// Registrazione utente
int registerUser(char *username, char *password) {
    char toWrite[1024];
    char line[1024];

    // Concateno username e password nel formato utilizzato per il file
    strcpy(toWrite, username);
    strcat(toWrite, "~");
    strcat(toWrite, password);
    strcat(toWrite, " ");


    printf("registerUser -> Attendo il semaforo \n");
    semop(id_sem, &sem_lock, 1);

    // Apro il file in lettura ed append
    FILE *list_ptr;
    list_ptr = fopen("user_list.dat", "a+");
    if (list_ptr == NULL) {
        printf("Impossibile aprire il file di utenti\n");
        printf("registerUser -> Rilascio il semaforo\n");
        semop(id_sem, &sem_unlock, 1);
        return 0;
    }

    // Tramite il separatore ricavo l'username per ogni registrazione
    // e verifico se il nuovo nome
    // utente e' gia esistente
    while (fscanf(list_ptr, "%s", line) != EOF) {
        char *tokens = NULL;
        // Tokenizzo ogni coppia nome~password dal separatore '~' per estrarne
        // l'username e confrontarlo
        tokens = strtok(line, "~");

        if (strcmp(tokens, username) == 0) {
            printf("Utente gia presente\n");
            fclose(list_ptr);
            printf("registerUser -> Rilascio il semaforo\n");
            semop(id_sem, &sem_unlock, 1);
            return 0;
```

```
        }
    }
    printf("#### sto registrando %s ####\n", toWrite);

    fprintf(list_ptr, "%s", toWrite);

    fclose(list_ptr);

    printf("registerUser -> Rilascio il semaforo\n");
    semop(id_sem, &sem_unlock, 1);

    printf("Registrazione completata\n");
    return 1;
}

// Login utente
int requestLogin(char *username, char *password) {
    char compare[1024];
    char line[1024];

    // Concateno username e password nel formato utilizzato per il file
    strcpy(compare, username);
    strcat(compare, "~");
    strcat(compare, password);

    printf("requestLogin -> Attendo il semaforo\n");
    semop(id_sem, &sem_lock, 1);

    // Apro il file in sola lettura e ne verifico l'esistenza
    FILE *list_ptr;
    list_ptr = fopen("user_list.dat", "r");
    if (list_ptr == NULL) {
        printf("Nessun utente registrato\n");
        printf("requestLogin -> Rilascio il semaforo\n");
        semop(id_sem, &sem_unlock, 1);
        return 0;
    }

    // Verifico se le credenziali sono valide ed eventualmente
    // convalido l'autenticazione
    while (fscanf(list_ptr, "%s", line) != EOF) {

        if (strcmp(line, compare) == 0) {
            printf("Utente trovato\n");
            fclose(list_ptr);
            printf("requestLogin -> Rilascio il semaforo\n");
            semop(id_sem, &sem_unlock, 1);
            return 1;
```

```
        }

    }
    fclose(list_ptr);

    printf("requestLogin -> Rilascio il semaforo\n");
    semop(id_sem, &sem_unlock, 1);

    printf("Utente non trovato \n");
    return 0;
}

// Invio messaggio
void sendMessage(int socket_to) {
    int res;
    int next_id = -1;
    FILE *msg_ptr;
    messaggio msg;
    char esito[2];
    strcpy(esito, "1");

    printf("sendMessage -> Attendo il semaforo\n");
    semop(id_sem, &sem_lock, 1);

    // Apro il file che contiene i messaggi
    msg_ptr = fopen("messages.dat", "a");
    if (!msg_ptr) {
        printf("Impossibile aprire messages.dat \n");
        strcpy(esito, "0");
        printf("sendMessage -> Rilascio il semaforo \n");
        semop(id_sem, &sem_unlock, 1);
        write(socket_to, esito, 1);
        return;
    }

    // Ottengo il messaggio dal client ed ottengo il primo ID libero
    res = read(socket_to, &msg, sizeof(msg));
    if (res == -1 || res == 0)
    {
        printf("Disconnected from client \n");
        semop(id_sem, &sem_unlock, 1);
        exit(EXIT_FAILURE);
    }
    if ((next_id = query_id()) == -1) {
        printf("Impossibile ottenere l'ID da inserire. Messaggio annullato.\n");
        strcpy(esito, "0");
        printf("sendMessage -> Rilascio il semaforo \n");
        semop(id_sem, &sem_unlock, 1);
```

```
        write(socket_to, esito, 1);

        return;
    }

    // Converto l'id ottenuto in stringa e lo inserisco nella struct
    snprintf(msg.id_messaggio, 4, "%d", next_id);

    // Scrivo il messaggio nel file
    printf("Id impostato: %s \n", msg.id_messaggio);
    if (!fwrite(&msg, sizeof(messaggio), 1, msg_ptr)) {
        strcpy(esito, "0");
    }

    printf("sendMessage -> Rilascio il semaforo \n");
    semop(id_sem, &sem_unlock, 1);

    fclose(msg_ptr);
    write(socket_to, esito, 1);

    return;
}

// Elimino messaggio
void delMessage(int socket_to) {
    int res;
    char esito[2];
    strcpy(esito, "1");

    char remove_id[4];
    // Ottengo tutti i messaggi ricevuti per far scegliere all'utente cosa
eliminare
    getMessagesByUser(socket_to);

    res = read(socket_to, remove_id, 4);
    if (res == -1 || res == 0)
    {
        printf("Disconnected from client \n");
        exit(EXIT_FAILURE);
    }
    FILE *actual_file_ptr;
    FILE *new_file_ptr;
    messaggio msg;

    printf("delMessage -> Attendo il semaforo\n");
    semop(id_sem, &sem_lock, 1);

    // Apro in lettura il file dei messaggi
```

```
    actual_file_ptr = fopen("messages.dat", "r");
    if (!actual_file_ptr) {
        printf("Impossibile aprire il file di messaggi \n");
        strcpy(esito, "0");
        printf("delMessage -> Rilascio il semaforo \n");
        semop(id_sem, &sem_unlock, 1);
        write(socket_to, esito, 1);
        return;
    }

    // Creo un file temporaneo in scrittura
    new_file_ptr = fopen("temp.dat", "w");
    if (!new_file_ptr) {
        printf("Impossibile creare il file temporaneo\n");
        strcpy(esito, "0");
        printf("delMessage -> Rilascio il semaforo \n");
        semop(id_sem, &sem_unlock, 1);
        write(socket_to, esito, 1);
        return;
    }

    // Ricopio tutti i messaggi da messages.dat a temp.dat ad eccezione di quello
da eliminare
    while (fread(&msg, sizeof(messaggio), 1, actual_file_ptr)) {
        if ((strcmp(msg.destinatario, current_user) == 0) &&
(strcmp(msg.id_messaggio, remove_id) == 0)) {
            printf("Record marcato per eliminazione \n");
        }
        else {
            fwrite(&msg, sizeof(messaggio), 1, new_file_ptr);
        }
    }

    fclose(actual_file_ptr);
    fclose(new_file_ptr);

    // Elimino il vecchio file e rinomino temp.dat in messages.dat
    remove("messages.dat");
    rename("temp.dat", "messages.dat");

    printf("delMessage -> Rilascio il semaforo \n");
    semop(id_sem, &sem_unlock, 1);

    write(socket_to, esito, 1);

    return;
}
```

```c
// Ottengo tutti i messaggi ricevuti da un utente
void getMessagesByUser(int socket_to) {
    messaggio msg;
    FILE *msg_list_ptr;

    char next_message[2];
    strcpy(next_message, "0");

    printf("getMessagesByUser -> Attendo il semaforo \n");
    semop(id_sem, &sem_lock, 1);

    // Apro in lettura il file dei messaggi
    msg_list_ptr = fopen("messages.dat", "r");
    if (!msg_list_ptr)
    {
        printf("Impossibile aprire il file di messaggi \n");
        strcpy(next_message, "0");
        printf("getMessagesByUser -> Rilascio il semaforo \n");
        semop(id_sem, &sem_unlock, 1);
        write(socket_to, next_message, 1);
        return;
    }

    // Leggo il file dei messaggi
    while (fread(&msg, sizeof(messaggio), 1, msg_list_ptr))
    {
        // Confronto destinatario del messaggio con l'utente della sessione
        if (strcmp(msg.destinatario, current_user) == 0)
        {
            // Avviso il client di un nuovo messaggio in arrivo e lo invio
            strcpy(next_message, "1");
            write(socket_to, next_message, 1);
            write(socket_to, &msg, sizeof(msg));
        }
    }

    strcpy(next_message, "0");
    write(socket_to, next_message, 1);
    fclose(msg_list_ptr);

    printf("getMessagesByUser -> Rilascio il semaforo \n");
    semop(id_sem, &sem_unlock, 1);

    return;
}

// Ottengo il primo ID disponibile per l'utilizzo
int query_id()
```

```
{
    messaggio msg;
    FILE *msg_list_ptr;
    int id;

    // Apro in lettura il file dei messaggi
    msg_list_ptr = fopen("messages.dat", "r");
    if (!msg_list_ptr) {
        printf("Impossibile aprire il file di messaggi \n");
        return -1;
    }

    // Verifico che il file sia vuoto. Per farlo vado all'End of File e controllo
la posizione
    fseek(msg_list_ptr, 0, SEEK_END);
    if (ftell(msg_list_ptr) == 0)
    {
        return 0;
    }

    // Se non e' vuoto ritorno l'ultimo id utilizzato incrementato di uno
    fseek(msg_list_ptr, 0, SEEK_SET);
    while (fread(&msg, sizeof(messaggio), 1, msg_list_ptr))
    {
        id = atoi(msg.id_messaggio) + 1;
    }

    fclose(msg_list_ptr);

    return id;
}
```

Client.c

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

typedef struct messaggio messaggio;
struct messaggio {
    char id_messaggio[4];
    char mittente[32];
    char destinatario[32];
    char oggetto[64];
    char testo[2048];
};

int registrazione();
int eseguiLogin();
void messaggiRicevuti();
void inviaMessaggio();
void eliminaMessaggio();
void sigpipeHandler();
void checkTimeout(int res);

int conn_socket;
char active_user[32];

int main(int argc, char *argv[])
{
    if (argc < 3) {
        printf("Usage: ./client.o ipaddress port\n");
        exit(-1);
    }

    int port_number = atoi(argv[2]);
    char ip_address[16];
    memcpy(ip_address, argv[1], sizeof(ip_address));

    struct sockaddr_in server_address; // Socket address string

    conn_socket = socket(AF_INET, SOCK_STREAM, 0);
```

```
    if (conn_socket < 0) {
        printf("Socket non aperto\n");
        exit(EXIT_FAILURE);
    }


    // Imposto il socket address
    memset(&server_address, 0, sizeof(server_address));
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(port_number);

    // Imposto indirizzo ip
    if (inet_aton(ip_address, &server_address.sin_addr) <= 0 )
    {
        printf("Ip non valido\n");
        exit(EXIT_FAILURE);
    }

    // Connect al server remoto
    if (connect(conn_socket, (struct sockaddr *) &server_address,
sizeof(server_address) ) < 0 )
    {
        printf ("Errore durante connect \n");
        exit (EXIT_FAILURE);
    }

    // Gestisco disconnessione del server
    struct sigaction action;
    sigset_t set;
    sigfillset(&set);
    action.sa_sigaction = sigpipeHandler;     // handler  la funzione a cui va il
controllo
    action.sa_mask = set;
    action.sa_flags = 0;

    sigaction(SIGPIPE, &action, NULL);

    // Imposto il timeout per una richiesta al server
    struct timeval tv;
    tv.tv_sec = 60; // 60 secondi
    tv.tv_usec = 0;
    setsockopt(conn_socket, SOL_SOCKET, SO_RCVTIMEO, (const char*)&tv, sizeof tv);


    printf("############ SCAMBIO DI MESSAGGI - Client ############\n\n");
    int auth = 0;
    char scelta[2];
    strcpy(scelta, "0");
```

```
    // Ciclo per autenticazione/registrazione
    while (!auth) {
        printf("Accedi: 1\nRegistrati: 2\n>> ");

        scanf("%s", scelta);

        printf("La scelta e' %s", scelta);

        if ((strcmp(scelta, "1") != 0) && (strcmp(scelta, "2") != 0))
            continue;

        write(conn_socket, scelta, 1);
        printf("ho scritto - %s -", scelta);


        if ( strcmp(scelta, "1") == 0 ) {
            auth = eseguiLogin();
        } else if ( strcmp(scelta, "2") == 0 ) {
            auth = registrazione();
        }
    }

    // Menu principale
    strcpy(scelta, "0");
    while ( strcmp(scelta, "4") != 0 )
    {
        strcpy(scelta, "0");
        printf("\nPrego selezionare un'operazione\n");
        printf("1 - Messaggi ricevuti\n");
        printf("2 - Invia un messaggio\n");
        printf("3 - Elimina dei messaggi ricevuti\n");
        printf("4 - Esci dall'applicazione\n");
        printf("\n>> ");
        scanf("%s", scelta);

        if ((strcmp(scelta, "1") != 0) && (strcmp(scelta, "2") != 0) &&
(strcmp(scelta, "3") != 0) && (strcmp(scelta, "4") != 0))
            continue;

        write(conn_socket, scelta, 1);


        if (strcmp(scelta, "1") == 0)
        {
            messaggiRicevuti();
        }
        else if (strcmp(scelta, "2") == 0)
        {
            inviaMessaggio();
```

```
        }
        else if (strcmp(scelta, "3") == 0)
        {
            eliminaMessaggio();
        }

    }

    return 0;
}

// Registrazione per nuovo utente
int registrazione()
{
    int res;
    char username[32];
    char password[32];
    char esito[2];
    strcpy(esito, "0");

    // Invio al server username e password
    printf("Inserire username: ");
    scanf("%s", username);
    write(conn_socket, username, 32);
    printf("Inserire password: ");
    scanf("%s", password);
    write(conn_socket, password, 32);

    res = read(conn_socket, esito, 1);
    checkTimeout(res);

    if (strcmp(esito, "1") == 0) {
        printf("Registrazione completata\n ");
        printf("\nBenvenuto %s\n", username);
        memcpy(active_user, username, 32);
        return 1;
    } else {
        printf("Impossibile completare la registrazione\n");
        return 0;
    }

}

// Login utente esistente
int eseguiLogin()
{
    int res;
    char username[32];
```

```
    char password[32];
    char esito[2];
    strcpy(esito, "0");

    // Invio al server username e password
    printf("Inserire username: ");
    scanf ("%s", username);
    write(conn_socket, username, 32);
    printf("Inserire password: ");
    scanf ("%s", password);
    write(conn_socket, password, 32);

    // Richiedo l'esito
    res = read(conn_socket, esito, 1);
    checkTimeout(res);

    if (strcmp(esito, "1") == 0){
        printf("\nBenvenuto %s\n", username);
        // Autenticazione effettuata
        memcpy(active_user, username, 32);
        return 1;
    }
    else {
        printf("Impossiile eseguire il login \n");
        return 0;
    }
}

// Visualizzo tutti i messaggi ricevuti da un utente
void messaggiRicevuti()
{
    int res;
    char next_message[2];
    strcpy(next_message, "0");
    messaggio msg;
    res = read(conn_socket, next_message, 1);

    checkTimeout(res);

    while( strcmp(next_message, "0") != 0 ) {
        res = read(conn_socket, &msg, sizeof(msg));
        checkTimeout(res);
        printf("\n## MESSAGGIO ##\n");
        printf("ID\n   %s\n", msg.id_messaggio);
        printf("Destinatario:\n   %s\n",msg.destinatario);
        printf("Oggetto:\n   %s\n", msg.oggetto);
        printf("Testo:\n   %s\n", msg.testo);
        res = read(conn_socket, next_message, 1);
```

```
        checkTimeout(res);
    }

    return;
}

// Invio un nuovo messaggio al destinatario scelto
void inviaMessaggio()
{
    int res;
    char esito[2];
    strcpy(esito, "0");

    messaggio msg;

    strcpy(msg.id_messaggio, "0");
    printf("Inserisci il destinatario \n");
    scanf(" %49[^\n]", msg.destinatario);
    printf("Inserisci oggetto \n");
    scanf(" %49[^\n]", msg.oggetto);
    printf("Inserisci testo \n");
    scanf(" %49[^\n]", msg.testo);

    memcpy(msg.mittente, active_user, 32);

    write(conn_socket, &msg, sizeof(msg));

    res = read(conn_socket, esito, 1);
    checkTimeout(res);

    if ( strcmp(esito, "1") == 0 ){
        printf("Messaggio inviato correttamente\n");
    }
    else {
        printf("Impossibile inviare il messaggio\n");
    }
    return;
}

// Elimino un messaggio ricevuto dato l'ID inserito
void eliminaMessaggio()
{
    int res;

    char id_msg[4];
    char esito[2];

    strcpy(id_msg, "-1");
```

```
    strcpy(esito, "0");

    messaggiRicevuti();

    printf("Quale messaggio vuoi eliminare? Inserisci l'ID \n>> ");

    scanf("%s", id_msg);
    write(conn_socket, id_msg, 4);

    res = read(conn_socket, esito, 1);
    checkTimeout(res);

    if ( strcmp(esito, "1") == 0 ) {
        printf ("Messaggio eliminato \n");
    } else {
        printf ("Impossibile eliminare il messaggio\n");
    }

    return;
}

void sigpipeHandler() {
    printf("\nErrore: server disconnesso. Il programma verra chiuso\n");
    exit(EXIT_FAILURE);
}

void checkTimeout(int res){
    if(res == -1) {
        printf("Disconnesso per inattivita. Il programma verra chiuso\n");
        exit(EXIT_FAILURE);
    }

    return;
}
```