UNIVERSITÀ
DEGLI STUDI
FIRENZE

Sistemi Operativi – modulo Laboratorio
Anno Accademico 2023-2024

Virginia Guiggiani, 7034306, virginia.guiggiani@edu.unifi.it
Sultan Zhunushov, 7073339, sultan.zhunushov@edu.unifi.it
Carmelo Casula, 7085019, carmelo.casula@edu.unifi.it

June 2, 2024

Istruzioni per compilazione

Dopo avere scaricato i file .zip, avviare il terminale e spostarsi nella directory `server-SO`, per la compilazione del progetto eseguire il comando:

```
make
```

In alternativa è possibile compilare il progetto eseguendo il comando:

```
gcc -o server src/main.c src/server.c src/commands.c src/user.c
src/contact.c -I./include -pthread
```

Similmente alla compilazione per il server, eseguire nella directory client-SO il comando:

```
make
```

Altrimenti digitare:

```
gcc -o client src/client.c src/commands.c -I./include
```

Progettazione e implementazione

Per realizzare un'infrastruttura per la realizzazione, gestione e consultazione di una rubrica telefonica abbiamo creato un file `main.c`, che si occupa di accettare connessioni da parte di client e la gestione delle relative richieste, e un file `client.c`, che dopo essersi connesso al server può svolgere le varie operazioni sulla rubrica.

Il server resta in attesa dei client per svolgere le possibili operazioni, finché resta attivo salva i contatti in una linked-list definita opportunamente.

Per poter eseguire le operazioni sulla rubrica ogni client deve prima registrarsi ed effettuare il login, qualora non venga eseguito l'accesso, o questo fallisca, si potrà solo visualizzare la rubrica salvata dal server fino a quel momento. Per la registrazione è necessario che l'utente fornisca username e password, il server salverà i dati per il login in una linked-list. Nel caso in cui l'utente inserisca uno username già presente, verrà stampato un messaggio di errore *"Username already exists"*, per evitare che nel momento del login si possano verificare casi ambigui dovuti a più user con diverse password, procedendo poi con una nuova operazione.

Dopo che l'utente effettua l'accesso può svolgere le operazioni di inserimento, cancellazione, modifica e stampa.

- **Inserimento**: l'utente dovrà fornire il nome con cui verrà salvato in rubrica il contatto e il numero ad esso associato, qualora fosse già presente un contatto con lo stesso nominativo verrà stampato un messaggio di errore *"Contact*

already exists", questo per evitare che nel momento in cui si esegue la cancellazione o la modifica di un contatto, vi sia la possibilità che l'operazione venga eseguita sul contatto sbagliato; quindi, si dovrà procedere con una nuova operazione.

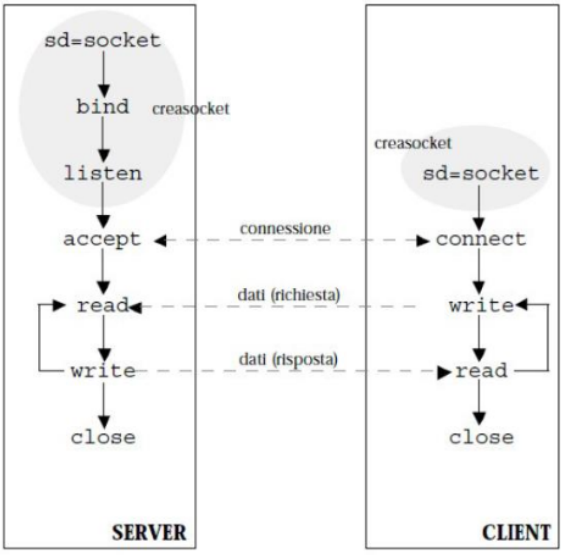
- **Cancellazione**: consente all'utente di eliminare dalla rubrica il contatto desiderato andando a indicare il nome con il quale era stato inserito; nel caso in cui non fosse presente verrà stampato un messaggio di errore *"Contact not found"*, si dovrà quindi procedere con una nuova operazione.

- **Modifica**: inserendo il nome del contatto da modificare, l'utente può procedere alla modifica del numero ad esso associato; verrà stampato un messaggio di errore *"Contact not found"* nel caso in cui il nome specificato non sia presente nella rubrica.

- **Stampa**: stampa tutti i contatti presenti nella rubrica.

Abbiamo scelto di utilizzare una **"lista concatenata"** per memorizzare gli utenti registrati e una per i contatti salvati in rubrica, perché tale struttura dati ci consente di allocare memoria nel momento in cui ne abbiamo la necessità, permettendo di espandere la lista quando avviene un inserimento in rubrica, o la registrazione di un nuovo utente, evitando spreco di memoria o una limitazione del numero di inserimenti. Per fare ciò abbiamo implementato due funzioni [**readString()** e **writeString()**] per allocare la memoria in modo dinamico. Inoltre, garantisce una maggiore velocità nelle operazioni di cancellazione e inserimento in una posizione arbitraria della lista concatenata.

Abbiamo creato una connessione tra client e server utilizzando **TCP** (Transmission Control Protocol), in quanto garantisce una buona affidabilità e il tempo di trasmissione risulta meno critico. TCP assicura che i dati che vengono trasferiti rimangano intatti e vengono consegnati nello stesso ordine in cui sono stati inviati. Questo protocollo esegue il controllo del flusso richiedendo tre pacchetti per impostare una connessione socket, prima che i dati del client possano essere inviati. I socket sono un meccanismo di comunicazione tra processi che consento la comunicazione bidirezionale anche tra processi residenti su macchine diverse.



La precedente tabella riassume i passaggi del processo: il server crea un socket anonimo **[socket()]** e gli assegna un indirizzo [**bind()**], successivamente dichiara quante connessioni è disposto ad accettare **listen()**, infine, dopo aver accettato una connessione [**accept()**], opera sul socket [**read(), write()**]. Il client crea un socket anonimo [**socket()**] e lo collega all'indirizzo del socket del server [**connect()**], quindi opera sul socket [**read(), write()**]. Una volta terminate le operazioni col socket sia il server che il client terminano [**close()**].

Una volta creata la connessione col client, il server crea un processo figlio che si occupi della gestione della connessione, mentre il processo originario continua ad accettare altre connessioni da client. Per garantire che ciò avvenga abbiamo deciso di implementare un **server TCP multi-thread**, in quanto consente a più client di condividere la stessa risorsa allo stesso tempo.

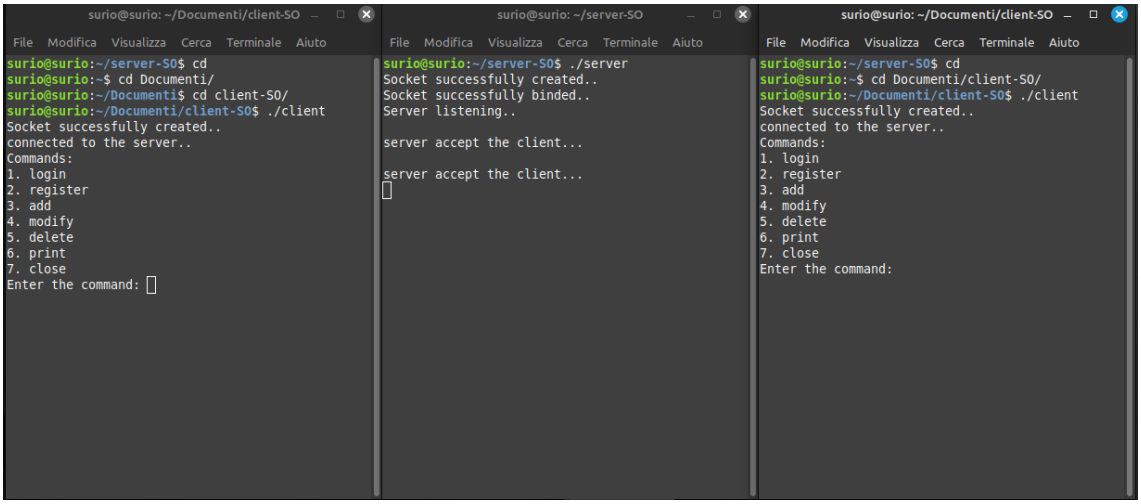
Esecuzione

Per eseguire il programma, dopo aver effettuato la compilazione con il comando **make**, dalla directory **server-SO** avviare il server eseguendo il comando:

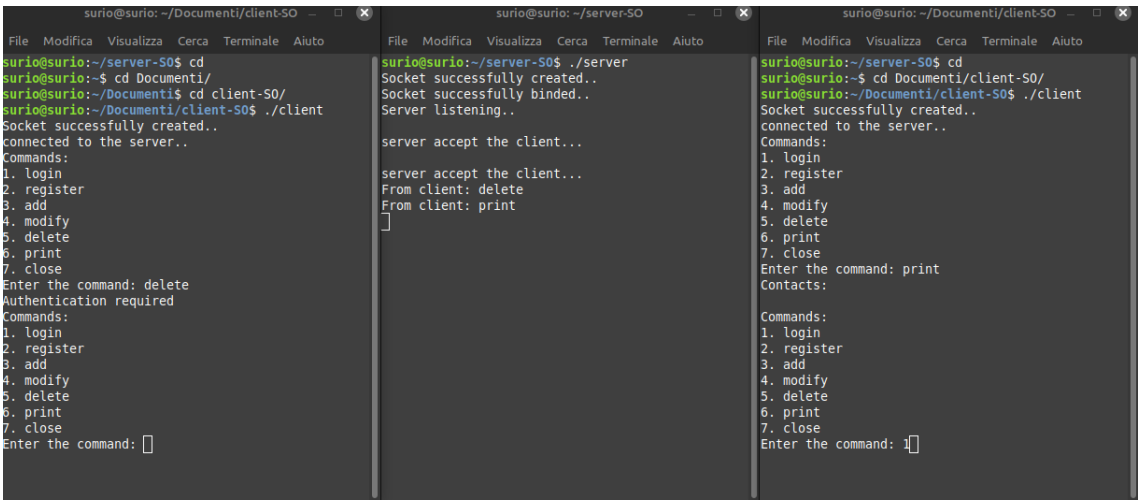
```
./server
```

Successivamente, da altre istanze del terminale, dopo aver eseguito il comando per la compilazione **make**, è possibile avviare uno o più client dalla directory **client-SO**, eseguendo il comando:

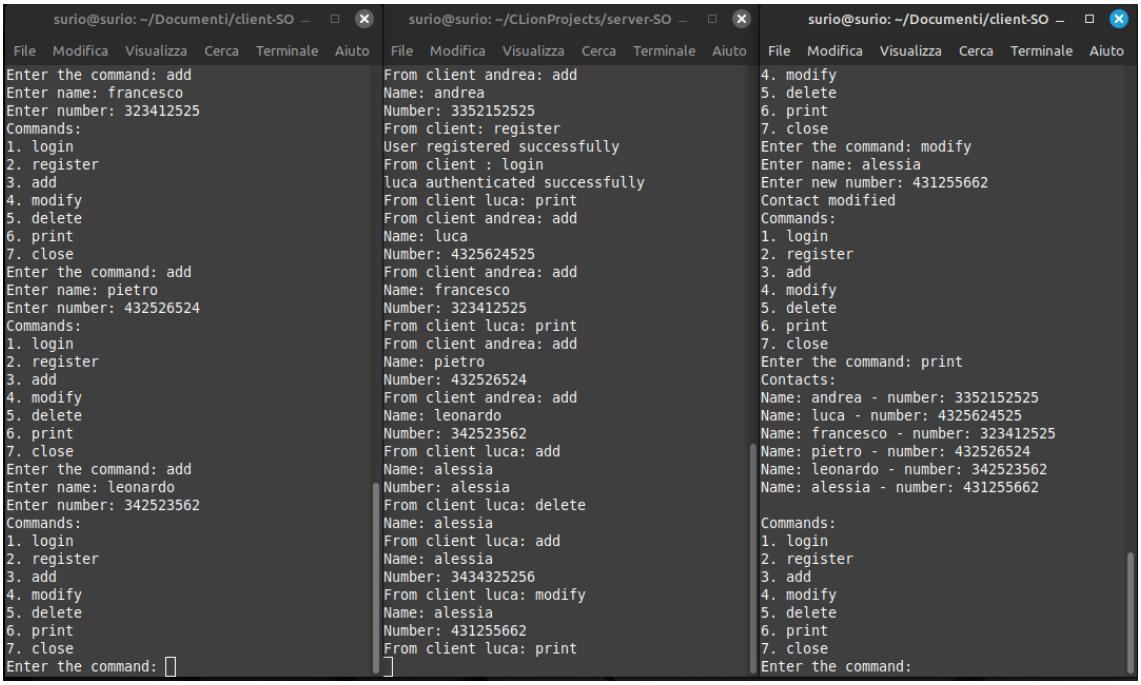
```
./client
```



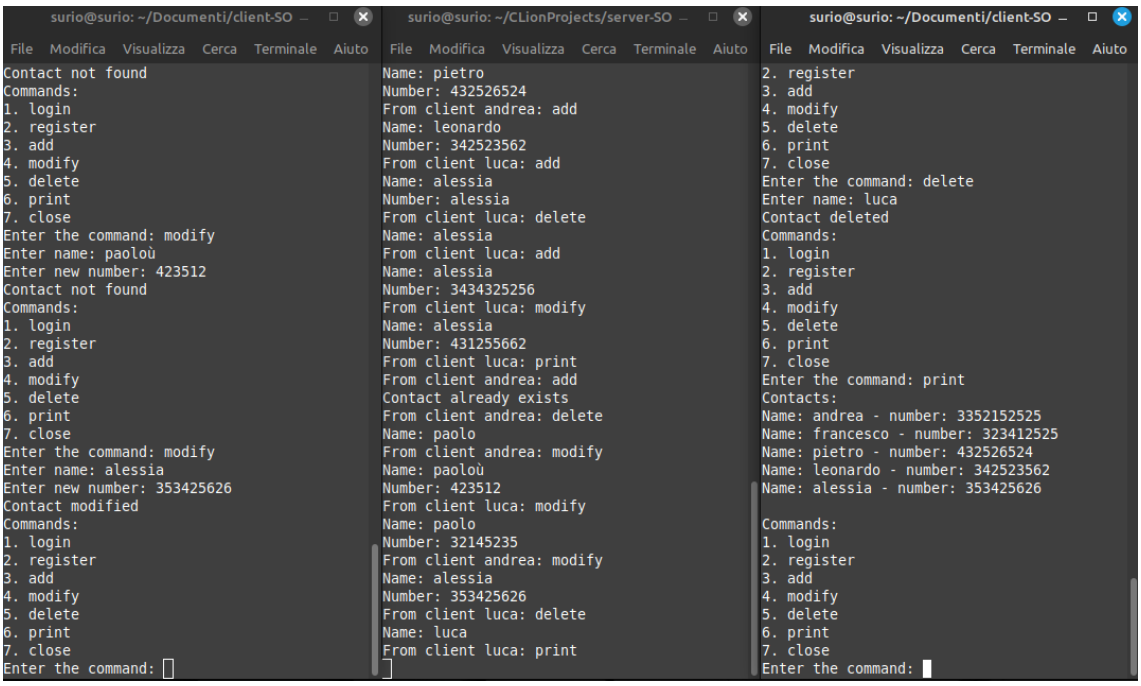
Adesso se proviamo ad eseguire due comandi [**print**, **delete**], senza essersi prima registrati, possiamo vedere come i due processi falliscono. Quindi siamo andati ad eseguire la regisrazione per entrambi i client, effettuando successivamente il login.



Abbiamo eseguito alcune operazioni come **add** e **modify**, in entrambi i client, stampando poi i contatti [**print**].

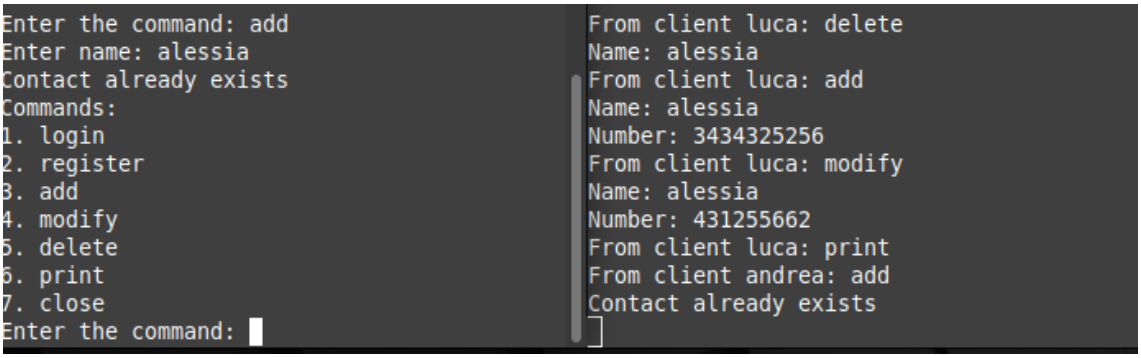


Successivamente abbiamo eseguito **delete** e **modify** andando nuovamente a stampare il risultato per vedere i cambiamenti.



Se proviamo ad aggiungere un contatto già presente nella rubrica otterremo un

messaggio di errore, come possiamo vedere dall'immagine anche nei log del server avremo lo stesso messaggio. Similmente accade se proviamo a modificare o ad eliminare un contatto non presente in rubrica.



Infine, dopo aver disconnesso entrambi i client, abbiamo eseguito il comando

`make clean`

per eliminare i file .o e l'eseguibile, facendo lo stesso per il server.