Informazioni

Data consegna progetto: 03/07/2024

Autori: Miranda Bellezza 7109803
Daniele Fallaci 7112588
Email di ateneo: miranda.bellezza@edu.unifi.it
daniele.fallaci@edu.unifi.it

Sistemi Operativi utilizzati: Ubuntu 24.04 e EndeavourOS (arch-based)
Shell usate: bash e zsh

Indice

Descrizione Progetto

Il progetto consiste nella realizzazione di una rubrica telefonica in C, tramite socket del dominio AF INET, che opera secondo un'architettura client-server. La rubrica permette agli utenti di eseguire operazioni di gestione dei contatti, come cercare contatti, aggiungere, modificare e eliminare. Il server prevede delle funzionalità per ogni utente (consultazione della rubrica e ricerca contatti) e delle operazioni che possono essere eseguite solo da un admin. Le operazioni di modifica (modifica, aggiunta, rimozione) possono essere svolte solo da un admin per volta. Se qualcuno sta già modificando i contatti, i client successivi verranno messi in attesa (la sezione critica viene gestita tramite un semaforo).
Per evitare che il file del database diventi troppo pesante è stata data la possibilità di limitare il numero dei contatti, al momento il limite impostato è 100 (`RECORDS_MAX` in “server.h”).
Data la complessità dei metodi main, questi sono descritti in questa relazione con l’uso di pseudo-diagrammi di flusso, che non sono completi dal punto di vista algoritmico ma che permettono di capire il funzionamento logico.

Struttura file ed esecuzione

Entrambe le cartelle hanno un Makefile per compilare automaticamente i file.
Utilizzando il comando `make` vengono create le cartelle ../build, che contiene i file *.o, e ../bin che contiene i file compilati. I Makefile possono eseguire il comando `make clean` per eliminare i file *.o e `make run` per far partire direttamente i programmi. In più il Makefile del server può eseguire il comando `make deps` per installare le librerie mancanti, individuando il package manager di riferimento analizzando il file “/etc/os-release”.
All’interno della cartella vendor è stata inserita la libreria cJSON utilizzata dal server, in modo che sia disponibile e che non sia necessario installarla.

Client

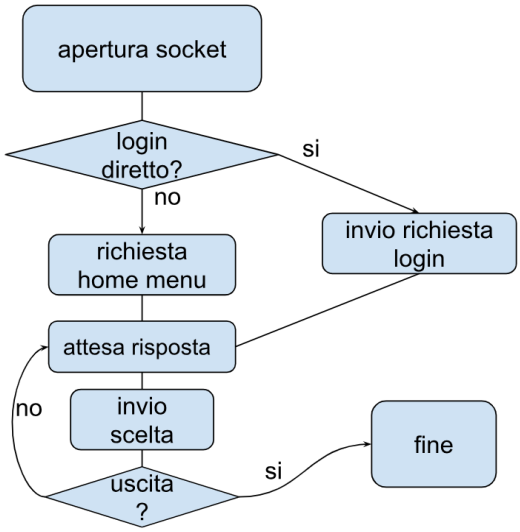
Il client fornisce un'interfaccia utente per interagire con la rubrica. Invia le richieste al server e visualizza i risultati. Aggiungendo lo username e la password come parametri è possibile fare il login direttamente sul server `./client -a <username> <password>` oppure mentre senza parametri o con username e password errati si entrerà come visitatori.

Header

Sono stati creati due nuovi tipi di dato utilizzando `typedef` in combinazione con `struct` per rappresentare il messaggio `MSG` che viene scambiato tra client e server e le `credenziali` per il login.

Main

Diagramma del funzionamento del client:



Apertura socket e Connessione al server

Viene aperto un socket di tipo `AF_INET` che utilizza un protocollo per lo scambio di messaggi di tipo `SOCK_STREAM`.

Controllo argomenti

Il client controlla se è stato avviato con degli argomenti:

- se non ci sono argomenti si passa alla richiesta di homepage (`requestHome()`) come visitatore;
- se ci sono almeno 3 argomenti (“`-a <username> <password>`”) `argc >= 4` (perché il primo argomento è il nome del file) questi vengono passati ad un parser (`parser()`) che li controlla e se il ritorno è affermativo viene avviato il login (`login()`);
- altrimenti vuol dire che ci sono argomenti ma non sono formattati correttamente per l’avvio del login diretto, viene quindi stampato un messaggio che spiega come avviare il login diretto ma si procede comunque alla richiesta di homepage come visitatore (`requestHome()`).

Lettura della risposta del Server

Il client entra in un ciclo while(1) e prova a leggere la risposta del server attraverso una `recv()`.
Se c’è una risposta:

- se la risposta è “x”: il server sta comunicando al client di disconnettersi, viene chiuso il socket, stampato un messaggio per indicare la disconnessione e il programma termina,
- altrimenti viene stampato il messaggio ricevuto.

Gestione Input dell'utente

L'utente ha la possibilità di scegliere l'operazione da effettuare attraverso un input da tastiera. `stdin`, viene ripulito (`clean_stdin()`) in modo da assicurarci che l'input sia sempre inserito dopo il messaggio del server. Viene quindi richiamato `fgets()` che legge sul buffer una nuova riga.
Il contenuto del buffer appena inserito viene controllato e tagliato inserendo `'\0'` se ha una dimensione maggiore di `INPUT_MAX` (definito nell'header). Infine, la stringa viene inviata al server attraverso una `send()`. Ricomincia quindi il ciclo attendendo una nuova risposta dal server.

Funzioni

requestHome

La funzione `requestHome()` prende come argomenti il socket sul quale inviare il messaggio e un messaggio da inviare. Inserisce nel messaggio la richiesta dell'homepage (nel nostro caso `"h"`) e la invia al server attraverso una `send()` sul socket.

parser

La funzione `parser()` prende come argomento i parametri passati da terminale. Attraverso un ciclo for scorre un puntatore `ptr` ad ogni stringa finché il puntatore non punta alla terzultima stringa presente. Se il puntatore punta a `loginArg` (per noi `-a`) il puntatore alle due stringhe successive viene copiato nelle credenziali (`cred.user` e `cred.password`).

login

La funzione `login()` segue la procedura manuale di login del server. Copia nel buffer e invia la richiesta di login al server `loginChar` (nel nostro caso `"l"`), attende una risposta, copia nel buffer e invia il nome utente prendendolo dalle credenziali `cred.user`, attende la risposta del server e infine copia nel buffer e invia la password prendendola dalle credenziali `cred.password`.

clean_stdin

Le funzioni `check_stdin()` e `clear_stdin()` servono a controllare la presenza di dati all'interno dello standard input e ad eliminarli se presenti. Sono utili per regolare la quantità di input da tastiera che l'utente può inviare. La funzione `check_stdin()` richiama la funzione `select()` che ci permette di monitorare descrittori di file e qui in particolare viene usata per monitorare 0 (`stdin`).
La funzione `clean_stdin()` richiama `fgets()` che legge una riga per volta (quindi fino a che non trova `"\n"`) finchè è presente qualcosa nello stream dello standard input controllando con `check_stdin()`.

Server

Il server ha il compito di gestire la memorizzazione e la manipolazione dei dati dei contatti. Riceve le richieste dai client, esegue le operazioni richieste e invia i risultati. Per la gestione dei dati utilizza due file:

- database.json: un file json che contiene i contatti salvati;
- setting.txt: che contiene i dati dell'amministratore.

Libreria cJSON

La libreria cJSON consente un utilizzo di una notazione degli oggetti più sofisticata ed usata in numerose applicazioni moderne. L'utilizzo della Javascript Object Notation ci permette di passare facilmente anche ad altri tipi di tecnologie come web-app, siti internet e database NoSQL(come mongoDB), garantendo una scalabilità del progetto per futuri ampliamenti.
È possibile visualizzare la documentazione qui: https://github.com/DaveGamble/cJSON

Settings

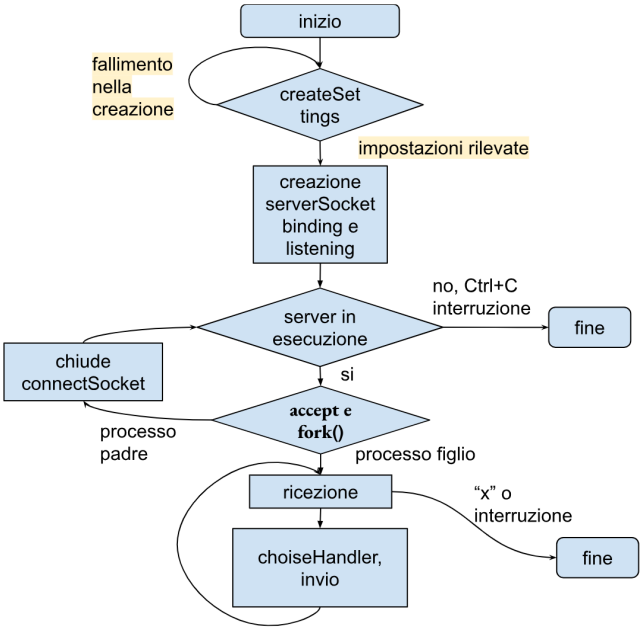
Le impostazioni del server sono composte semplicemente da un file settings.txt contenente le credenziali dell'amministratore. Viene applicata una funzione hash alla password per proteggerla.
Se all'avvio non viene trovato il file delle impostazioni, si provvede alla creazione di questo e al salvataggio delle credenziali. E' possibile tramite il parametro `-r` resettare le credenziali. `./server -r`

Header

Sono stati definiti tre nuovi tipi `MSG`, parallelo a quello in Client per consentire la comunicazione, `t_credenziali` che è usato per salvare le credenziali e `operationOnList` usato per definire il tipo di operazione applicabile a seguito di una ricerca. Inoltre sono definite alcune macro usate all'interno del codice.

Main

Diagramma del funzionamento del server:



Settings

Viene richiamata la funzione `createSetting()` che controlla se esiste un file setting.txt e lo ripristina in caso non esista o in caso si apra il server utilizzando il flag di reset `-r`.

Socket

Viene creato un socket di tipo `AF_INET` che utilizza un protocollo per lo scambio di messaggi di tipo `SOCK_STREAM`. Viene assegnato un indirizzo al socket attraverso la funzione `bind()` e il socket si mette in ascolto con la funzione `listen()`.

Accept e Fork

Appena un client si connette al socket si ha una `fork()` del processo. Il processo si divide in:

- processo padre: che chiude la connessione appena creata e attende una nuova connessione;
- processo figlio: che gestisce le richieste del client connesso.

Richiesta del client

Il processo figlio attende con una `recv()` il messaggio del client. Se c'è un errore chiude la connessione, altrimenti lo stampa. Se il messaggio è uguale a `"x"` il client ha deciso di chiudere la connessione, il server risponde rimandando `"x"` e chiude la connessione e il processo.
Se invece si ha una risposta diversa questa viene passata alla funzione `choiseHandler()` che la gestisce.

Funzioni

createSettings

Si occupa di creare le impostazioni del server. E' invocata al primo avvio e in modalità di reset col parametro `-r` da terminale. Si occupa della scelta di user e password dell'amministratore e le immagazzina nel file di riferimento. Utilizza il metodo `intoSha256()` per la protezione del server e un metodo `parser` per controllare la presenza del parametro di reset. Fa due importanti accessi:

- Apertura in lettura del file
- apertura in scrittura e creazione in caso di assenza del file o richiesta di reset

Le impostazioni sono salvate in un file .txt per semplicità e necessità di salvare solo delle credenziali.

choiseHandler

La funzione `choiseHandler()` legge il messaggio inviato dal client e attraverso uno switch richiama le funzioni richieste.
`"h"` -> `sendMenu()`, `"v"` -> `readContent()`, `"s"` -> `search()`, `"l"` -> `login()`, `"a"` -> procedura per aggiunta, `"m"` -> procedura per modifica, `"r"` -> procedura per rimozione, `default`: invia "Comando non riconosciuto."
Le procedure per aggiunta, modifica e rimozione sono strutturate in modo analogo.
Ad esempio nella procedura per l'aggiunta, si controlla se il messaggio arriva da un admin. Si guarda il semaforo e se è occupato si chiede al client se vuole rimanere in attesa o tornare al menu. Allo sblocco del semaforo, si richiama la funzione `aggiungiPersona()` e si procede a liberare il semafono.
La modifica e la rimozione procedono in modo analogo, richiamando però la funzione `search()` con le rispettive operazioni `editFromList()` e `removeFromList()`.
Dopo la conclusione della procedura principale si torna al menù con `sendMenu()` oppure dopo l'esecuzione di visita o ricerca si richiede al client se vuole continuare con il menù o uscire.

sendMenu

Inserisce nel buffer il banner grafico della rubrica e il menù con le scelte, diverso se si è admin o visitatore.

loadDatabase

La funzione restituisce un array cJSON con il contenuto del file `database.json` (NULL se il file non viene trovato). Viene aperto il file `database.json`.
Se non nullo, la dimensione del file viene calcolata utilizzando `fseek()` e `ftell()`. Con `fseek()` si punta alla fine del file, si legge con `ftell()` la posizione del puntatore e si torna all'inizio del file sempre con `fseek()`.
Viene allocata una quantità di spazio necessaria per la lettura del file utilizzando `calloc()` e il file viene letto e chiuso. Il contenuto del file viene quindi sottoposto ad un parse `cJSON_Parse()` (libreria cJSON) che crea l'array JSON.

saveDatabase

La funzione salva sul file `database.json` il contenuto di un array di cJSON.
Gli elementi presenti nell'array vengono ordinati per chiave (`"surname"`) e con `cJSON_Print()` (libreria cJSON) viene creata la stringa corrispondente all'array.
Viene quindi aperto il file `database.json` e sovrascritto con la nuova stringa.

readContent

I contatti vengono caricati su un array cJSON attraverso `loadDatabase()`. Viene poi invocata `printContent()`.

printContent

La funzione `printContent()` prende un array cJSON, crea una stringa per ogni elemento nell'array e li invia al client una "pagina" per volta. La variabile `i` indica l'indice del contatto +1, quindi il numero del contatto (partendo da 1).
Per l'inizio della pagina viene inserita una stringa con il numero di contatti trovati e il numero di pagina, poi vengono inserite le stringhe dei contatti fino ad avere un numero di contatti per pagina prestabilito (`RECORDS_INPAGE = 4`). Infine, se ci sono ancora contatti da stampare viene inserita nel buffer una stringa per chiedere al client se vuole visualizzare la pagina successiva.
Il tutto viene inviato al client sul socket.

search

La funzione è in grado di individuare i contatti che hanno un determinato nome, cognome, nome cognome, cognome nome o una qualsiasi sottostringa di questi. I contatti vengono caricati sulla variabile `foundArray` con `loadDatabase()`. Poi viene richiesto al client un nome da ricercare e viene salvato nella variabile `searchName`. Viene fatto un ciclo sugli elementi dell'array cJSON (`jsonArray`) e ricercato il nome inserito come sottostringa `strstr()` della stringa "nome cognome" o "cognome nome" dell'elemento dell'array. Se viene trovata una sottostringa l'elemento viene aggiunto ad un nuovo array `foundArr`. Se quest'ultimo array non è vuoto viene invocata `printContent()` per inviare il risultato della ricerca al client.
La funzione search supporta due operazioni i cui puntatori possono essere passati come argomento della funzione: `removeFromList` e `editFromList`. Passando invece NULL la funzione termina qui (questo avviene nel caso della ricerca da visitatore).

removeFromList

La funzione `removeFromList()` continua la `search()`. Riprende da dopo il `printContent()` e al client viene richiesto di inserire l'indice del contatto da eliminare tra quelli che gli vengono proposti sullo schermo (oppure tornare al menù).
Il numero inviato dal client viene controllato (deve rientrare tra quelli proposti, altrimenti viene chiesto di nuovo) e successivamente il contatto corrispondente viene ricercato nella lista contatti.
Viene richiesto al client di confermare l'eliminazione e se la conferma è positiva il contatto viene eliminato con `cJSON_DeleteItemFromArray()` (libreria cJSON) e la funzione restituirà 1. Altrimenti se non c'è nessun contatto eliminato la funzione restituisce 0.

editFromList

Ha una struttura simile alla `removeFromList()`. Dopo aver richiesto un indice valido e dopo la conferma di voler modificare viene avviata la funzione `creaPersona()`. Se viene creato correttamente un nuovo contatto il precedente viene sostituito tramite `cJSON_ReplaceItemInArray()` (libreria cJSON).

aggiungiPersona

La funzione `aggiungiPersona()` ritorna -1 se si è raggiunto il massimo dei contatti permessi, 1 se il nuovo contatto è stato aggiunto correttamente e 0 se il contatto non è stato aggiunto.
La funzione carica i contatti attraverso `loadDatabase()`. Se il file non esiste crea un nuovo array e se i contatti sono maggiori di `RECORD_MAX` la funzione ritorna -1. Altrimenti invoca `creaPersona()` e se questa non è NULL la aggiunge all'array e salva il database con `saveDatabase()`.

creaPersona

La funzione `creaPersona()` costruisce l'elemento cJSON contenente i dati di un contatto e lo restituisce. Se ci sono errori restituisce il valore NULL. Per ogni campo del contatto (nome, cognome, età, email e telefono) viene richiesto al client di inviare il dato.
Al messaggio mandato dal client vengono applicati i seguenti controlli: il nome deve essere non nullo e massimo 12 caratteri, il cognome deve avere massimo 12 caratteri, l'età deve essere un numero compreso tra 1 e 100, l'email può essere lasciata vuota e se inserita deve avere massimo 30 caratteri ed essere un'email valida `utils_isValidEmail()` e il numero di telefono può essere lasciato vuoto e se inserito deve avere massimo 16 caratteri ed essere formato solo dai caratteri "+ 1234567890" funzione `utils_strIncludeOnly()`. Dopo la creazione del contatto viene richiesta al client la conferma.

login

La funzione `login()` richiede le credenziali al client e le salva su una variabile.
Il login utilizza una funzione di `verifica()` che apre il file settings.txt definito nell'header, effettua l'Hash della password e lo confronta con le credenziali fornite. Utilizza `inToSha256()` per la conversione.

Gestione Hashing

La tecnica hashing è usata spesso nel contesto client-server o più in generale nei database.
Si basa su una funzione irreversibile che permette di non salvare in chiaro le credenziali di utenti o informazioni sensibili generiche. Esistono molti algoritmi di hash come MD5, SHA o SHA256, nel nostro contesto usiamo l'ultimo per salvare la password dell'admin.

inToSha256

La funzione utilizza la libreria openssl, la quale gestisce molti tipi di algoritmi hash.
Viene creato un contesto di uso della conversione con `EVP_MD_CTX_NEW`.
Si inizializza con `EVP_DigestInit_ex`, si aggiorna il contesto aggiungendo la stringa da convertire in `EVP_DigestUpdate` e la conversione vera e propria avviene tramite `EVP_DigestFinal_ex`. In caso di errori o comportamenti anomali si ha una funzione di gestione degli errori (`handleErrors`).
Le funzioni di openssl non usano char, ma unsigned char come destinazione della conversione, quindi dobbiamo convertirla in un formato comprensibile tramite l'uso di `hashToHexString`.
Questa funzione tronca il byte in cui è stato tradotto ogni carattere hash in uno esadecimale;
Ogni elemento è 1 byte quindi 8 bit, proprio per questo la destinazione della conversione è di lunghezza definita da: `#define CONVERTION_SHA256_MAX (EVP_MAX_MD_SIZE *2 + 1)`
quindi ad ogni elemento della bytestring ne corrisponde 2 esadecimali (+ 1 per il carattere di fine stringa).

Altre funzioni in server_utils

clean_stdin

Ripulisce lo Standard Input con `getchar()` leggendo un char per volta.

utils_strIncludeOnly

Controlla se una stringa `str` è composta solo da caratteri della seconda stringa `restriction`. Scorre un puntatore attraverso `str` e vede se il primo carattere è una sottostringa di `restriction`. Se non lo è vuol dire che il carattere non è presente in `restriction`.

utils_isValidEmail

Prende come argomento una stringa `email` e controlla se ha le caratteristiche di un'email valida.
Separa la stringa in `afterAt` dopo la "@" e `beforeAt` prima della chiocciola e controlla che questi abbiano solo caratteri validi (`emailDigits`) con `utils_strIncludeOnly()`.

utils_sortByKey

È stata effettuata una piccola modifica alla funzione `sort_object()` della libreria cJSON_Utils. È stata cambiata la stringa su cui veniva fatto il sort. Nella libreria il sort veniva fatto su tutta la stringa generata dal `cJSON_Print()` dell'oggetto cJSON, nella nostra funzione è possibile specificare una chiave e fare il sort solo sui valori di quella chiave.

Signal Handler

Sul server, il processo padre e i processi figlio condividono la tabella dei segnali facilitando molto la gestione dei segnali.

SIGINT

Il segnale di interruzione è generato quando si immette la combinazione di tasti CTRL+C.
La gestione delle interruzioni è delegata al metodo `customSigHandler`, presente sia nel server che nei client. L'handler si occupa di chiudere il socket prima di terminare il processo.

SIGPIPE

Il segnale `SIGPIPE` viene generato quando un processo tenta di scrivere su una connessione senza lettore. Utilizziamo un puntatore globale al semaforo di riferimento per sbloccare tutte le sezioni critiche dei client, questo viene fatto perché nel caso di disconnessione di un client il server deve sbloccarsi. Se un processo si disconnette in sezione critica si crea uno stallo se non si rilascia il semaforo. In maniera parallela nel client si effettua la disconnessione se il server non è più in ascolto.

Gestione comportamenti anomali

Server

Arresto improvviso

Se il server si arresta all'improvviso la connessione si chiude e viene segnalato al client tramite il segnale di `SIGPIPE`. Il client si interrompe e viene restituito un messaggio che indica la chiusura della connessione da parte del server. L'arresto improvviso del server non comporta la perdita di dati ma è possibile che l'ultima modifica non venga salvata se l'arresto avviene prima del salvataggio sul file `database.json`.

Client

Inserimento troppi caratteri nell'input da tastiera

L'utente può inserire solo fino a 35 caratteri in input, tutto quello che viene inserito successivamente viene eliminato e non inviato al server.

Inserimento linee di troppo durante l'attesa al semaforo

Può capitare che durante l'attesa l'utente provi ad inviare numerose righe di testo digitando caratteri e premendo invio quando il server non è pronto per ricevere messaggi. Queste righe vengono cancellate prima di prendere un nuovo messaggio da inviare al server.

Arresto del client in sezione critica

Attraverso il segnale di `SIGPIPE` al server viene segnalato che il client si è disconnesso. Se il pid del processo che si disconnette è uguale al processo che era in sezione critica il semaforo viene sbloccato.