RELAZIONE LABORATORIO RETI 1 – MATTEO SCHIRINZI MAT.: 20035542

SERVER

Il server è stato implementato seguendo la traccia fornita in modo rigoroso, all'interno del codice sono presenti numerosi commenti con riferimenti alla traccia e ai suoi punti specifici.

Da riga 7 a riga 16 vengono importate le librerie standard di sistema, non vi è nessuna dipendenza con librerie esterne; inoltre viene definita una costante **DIM** di valore 256 per la massima dimensione dei messaggi.

Da riga 21 a 27 vengono definite quattro funzioni utili al corretto svolgimento del server:

- **int isAlpha(char *parola)**: funzione che determina se una parola è formata solo da lettere e non contine altri simboli
- **void estraiParola(char *parolaServer)**: funzione che estrae casualmente dal file elenco-parole.txt le parole che devono essere indovinate (punto 3)
- **void toUpper(char* s)**: funzione che trasforma la parola passata come parametro in maiuscolo
- **char *trimwhitespace(char *str)**: funzione che rimuove eventuali spazi bianchi rimasti in coda ad una stringa

Successivamente, fino a riga 85 inizia il main del programma dove vengono definite le variabili per la connessione e il numero massimo di tentativi di default, vengono presi da riga di comando la porta di ascolto e l'eventuale terzo parametro opzionale (max_tentativi); vengono effettuati i controlli sull'input da riga di comando e sulla creazione della socket.

Viene poi effettuato il bind e verificata la connessione tramite l'apposito controllo, il server si mette poi in ascolto (listen).

Da riga 118 si entra nel loop del server, vengono definite le variabili utilizzate per la comunicazione ed effettuata l'accept; successivamente viene creato il messaggio di benvenuto ed inviato al client.

Viene poi estratta la parola ed inizializzato il contatore dei tentativi, si entra così da riga 163 nel loop in cui avviene la comunicazione tra server e client, vengono puliti i buffer utilizzati e il server rimane in attesa di risposta dal client.

Da riga 178 il server controlla la risposta ricevuta dal client, in particolare:

- se il messaggio è vuoto
- se il comando differisce da **WORD** o **QUIT** (tutto maiuscolo)
- controlla la presenza di caratteri non alfabetici nella parola del client
- controlla che la parola inserita dal client sia esattamente lunga cinque caratteri

in tutti questi casi viene generato un messaggio di errore ed inviato al client.

Altrimenti da riga 258 il server distingue se ha ricevuto un comando di **WORD** e verifica che nel caso la parola corrisponda a quella estratta dal server invia la risposta **OK PERFECT** altrimenti, se rimangono ancora tentativi, crea la stringa di risposta e la invia al client; altrimenti invia il comando di end con la parola da indovinare.

Da riga 326 viene invece gestito il caso del comando **QUIT** in cui il server genera il messaggio di commiato seguito dalla parola che si doveva indovinare.

Si esce così dal loop più interno e viene chiusa la connessione con il client ma il server rimane attivo e in ascolto in attesa di un altro client.

CLIENT

Il client è stato implementato seguendo la traccia fornita in modo rigoroso, all'interno del codice sono presenti numerosi commenti con riferimenti alla traccia e ai suoi punti specifici.

Come per il server, all'inizio del programma client vengono incluse le librerie di sistema senza dipendenze da librerie esterne e definita una costante **DIM** per la massima dimensione dei messaggi.

Vengono poi dichiarate due funzioni utili al corretto funzionamento del client:

- **int isAlpha(char *s)**: funzione che determina se una parola è formata solo da lettere e non contiene altri simboli
- **void trimTrailing(char *str)**: rimuove eventuali spazi bianchi in coda alla parola passata come parametro

Si entra a riga 25 nel main del client dove vengono dichiarate le variabili per la comunicazione con il server (i buffer) e le variabili dove verranno salvati i tentativi e le parole che inserirà l'utente.

A riga 47 viene effettuato il controllo sui parametri e il client controlla che il formato dei parametri sia rigorosamente <eseguibile> <IPv4 Server> <Porta Server>, altrimenti si genera un errore.

Viene poi creata la socket ed effettuato il controllo sulla sua corretta creazione, successivamente a riga 77 viene creata la connessione con il server (anche qui viene effettuato un apposito controllo: riga 80)

Da riga 92 a 133 viene letto il messaggio di benvenuto del server che il client si aspetta di ricevere, viene poi mostrato all'utente senza delimitatore e senza il numero di tentativi; i tentativi vengono poi salvati in una variabile locale per poi essere mostrati all'utente come indicato nel punto 5 della traccia.

Da riga 136 a riga 314 viene eseguito il loop del client nel quale avviene la comunicazione con il server; dopo aver mostrato il messaggio di benvenuto viene pulito il buffer tramite **memset** per evitare "caratteri sporchi", successivamente viene proposta all'utente una scelta tra indovinare la parola oppure abbandonare l'esecuzione.

Viene controllato che l'utente inserisca solo valori di tipo numerico compresi tra **1** e **2**, dopodiché è presente un if di selezione in base alla scelta effettuata precedentemente dall'utente:

- **1**: vengono mostrati i tentativi rimasti e richiesta la parola che poi verrà inviata al server (viene effettuato anche nel client il controllo che la parola sia composta solo da caratteri numerici)
- **2**: è il caso in cui l'utente vuole abbandonare l'esecuzione e quindi viene inviato al server il comando di **QUIT**

Al termine di questo if di selezione il client si pone in attesa di una risposta del server, vengono puliti i buffer e controllata la risposta del server; nel caso in cui la risposta non genera errore il client:

- Controlla inizialmente se la risposta del server è di tipo **OK PERFECT**, in tal caso l'utente ha indovinato e il client gli offre il riscontro positivo per poi terminare la sua esecuzione
- Altrimenti viene scomposto il buffer ricevuto in chiave di protocollo, numero di tentativi e messaggio del server (da riga 243 a riga 268)

Dopo aver scomposto il comando vengono effettuati i seguenti controlli:

- La chiave di protocollo è **OK** e sono rimasti ancora tentativi a disposizione, il client mostra all'utente la risposta del server (con una legenda per rendere chiaro all'utente il significato dei simboli)
- La chiave di protocollo è **END** e quindi i tentativi sono esauriti, viene opportunamente mostrato all'utente il numero di tentativi effettuati e la parola che si sarebbe dovuto indovinare

- La chiave di protocollo è **QUIT** il server ha risposto in maniera positiva alla richiesta dell'utente di abbandonare l'esecuzione; viene mostrato all'utente il messaggio di commiato del server
- La chiave di protocollo è **ERR**, si ricade in questo ramo del codice quando viene mandato dal server un qualsiasi messaggio di errore e il client riporta all'utente il messaggio di errore senza i delimitatori e la chiave di protocollo.

Alla fine di questo ciclo il client chiude la connessione e ciò viene comunicato in maniera opportuna all'utente, successivamente il client termina la sua esecuzione.

L'applicazione è stata sviluppata in ambiente Linux (Kali Linux 2022.3) ed è stato utilizzato VisualStudio come editor per l'implementazione del codice; al termine dello sviluppo client e server sono stati testati attraverso le seguenti modalità:

TEST SERVER

È stato eseguito il server: "./server.c 1002" e attraverso l'uso del comando netcat: "nc 127.0.0.1 1002" sono stati eseguiti i seguenti test:

- **TEST COMANDO ERRATO:** dopo il messaggio di benvenuto del server viene inviato un comando diverso da WORD o QUIT (anche word e quit in minuscolo generano errore)
- **TEST PAROLA VUOTA:** è stato inviato un comando di WORD senza che fosse seguito dalla parola generando un messaggio di errore dovuto alla parola mancante
- **TEST TENTATIVI:** è stato eseguito il server passando come parametro da riga di comando (dopo il valore della porta) un numero > 10 o < 6 oppure un input non numerico, generando un errore
- **TEST PARAMETRI ECCESSIVI:** nel caso in cui il server venga avviato passando più di due argomenti (porta e tentativi) si genera un messaggio di errore.
- **TEST BUFFER VUOTO:** è stato effettuato il test di invio di un buffer vuoto al server o un buffer contente solo il carattere '\n', si genera un messaggio di errore.
- **TEST PAROLA ISALPHA:** viene controllata la parola inviata attraverso il comando WORD, deve essere composta solo da caratteri alfabetici altrimenti si genera un errore
- **TEST LUNGHEZZA PAROLA:** viene controllata la lunghezza della parola inviata attraverso il comando WORD e nel caso sia diversa da 5 si genera un errore.
- **TEST TERMINE TENTATIVI:** è stato verificato che il server permetta di effettuare esattamente i tentativi dichiarati (o in caso non fossero stati dichiarati -> 6) al client senza eccedere o far effettuare meno tentativi al client.
- **TEST QUIT:** è stato verificato che in qualsiasi momento dell'esecuzione se il client invia un comando di QUIT il server risponde correttamente con il comando QUIT + messaggio di commiato.

TEST CLIENT

Il client è stato testato con il server appena creato (dopo aver effettuato i test sul server), inoltre è stato testato con l'implementazione di riferimento disponibile all'indirizzo prezzemolo.polito.it 130.192.9.131, alla porta 10010.

Per ridondanza nel client vengono effettuati controlli che già vengono effettuati sul server come, ad esempio, che la parola inserita dall'utente sia isAlpha e che sia lunga 5; è un controllo maggiore ma se client e server dovessero essere utilizzati con altre implementazioni in modo separato riuscirebbero comunque a gestire i casi di errore in modo corretto.

Il codice compila regolarmente senza errori, è stato utilizzato il seguente comando per le compilazioni sia per il client che per il server: gcc -Wall -Werror -pedantic -o server/client server.c/client.c; per l'esecuzione invece: ./server <porta> (tentativi opzionale), ./client <Ipv4 server> <porta server>