UNIVERSITA' DEGLI STUDI DI MILANO

Dipartimento di Informatica – Sicurezza dei Sistemi e delle Reti Informatiche

SOCKET PROGRAMMING – MULTICHAT, FILE EXCHANGE AND MUSIC PLAYER

MATTEO DAL GRANDE

Anno Accademico 2019/2020

Questo progetto vuole realizzare un piccolo applicativo Client/Server utilizzando le socket in C, dove il Client si connette al server, il quale gli darà la possibilità di chattare con altri utenti se connessi nello stesso momento al server. Il server mette a disposizione agli utenti una metodologia per poter scaricare un file(in questa versione purtroppo non possiamo ancora far scegliere all'utente che tipo di file scaricare). Il file in questione in questa versione è un file di tipo .mp3 che se scaricato in modo corretto verrà automaticamente eseguito lato client.

Per avviare il server da terminale bisogna prima compilare il codice:

gcc server.c -o a.out

Successivamente avviamo il server con il comando:

./a.out

Se vi risponde con un messaggio così:

[Server-TCP] attende per un client alla porta 4950

Significa che è tutto apposto.

Per avviare il o i client basterà compilare una sola volta il codice con il comando:

gcc client.c -o b.out

Successivamente si ripete da vari terminali più volte questo comando:

./b.out

Tante volte quanti utenti vogliamo far connettere.

SERVER:

Funzioni:

void connect_request(int *sockfd, struct sockaddr_in *my_addr)

questa funzione crea il socket, setta la porta e l'indirizzo di se stesso, fa la bind ovvero lega al socket la porta, fa la listen() che definisce la coda di richieste massima di tcp.

- **sockfd** è il puntatore alla variabile dove viene messo il socket
- **my_addr** è la struttura che contiene l'indirizzo ip del server

void connection_accept(fd_set *master, int *fdmax, int sockfd, struct sockaddr_in *client_addr)

E' funzione che serve per accettare le connessioni dei nuovi utenti che si vogliono connettere

- **master** puntatore al file descriptor master
- **fdmax** puntatore al socket del file descriptor maggiore
- **sockfd** socket del master
- **client_addr** puntatore a struttura per l'indirizzo client

void send_recv(int i, fd_set *master, int sockfd, int fdmax)

send_recv è una funzione che serve per ricevere i dati da un dato client che vuole trasmetterceli e richiamerà la funzione send_to_all per il rinoltro dei messaggi ricevuti da un client. Inoltre chiama la funzione music() che serve per gestire l'invio della musica ad un client quando richiesto.

- **i** numero in array che dice quale file descriptor è pronto a leggere
- **master** puntatore del file descriptor master
- **socket** è il socket del master

- **fdmax** è il numero del socket massimo

void send_to_all(int j, int i, int sockfd, int nbytes_recvd, char *recv_buf, fd_set *master)

send_to_all è una funzione che serve per l'inoltro del messaggio inviato da un cliente a tutti gli altri client ed accetta:

- **j** il socket a chi deve inviare il messaggio
- **i** il socket del client che ha inviato il messaggio
- **sockfd** che indica il socket del master
- **nbytes_recvd** che indica il numero di byte che sono inviati
- **recv_buf** che è il buffer che contiene i dati da inviare
- **master** è il file descriptor del master

int strpos(char *str, char *search)

E' una funzione che prede due tipi char * come parametri e controlla se la parola search è contenuta nella frase str. Ritorna 1 se search è contenuta in str altrimenti 0

Socket:

Questa applicazione è realizzata usando una connessione persistente TCP/IP. In C una connessione TCP/IP viene creata per mezzo della funzione **socket(AF_INET, SOCK_STREAM, 0)**. Il primo parametro della funzione indica la famiglia di protocollo, il secondo indica il tipo di comunicazione che in questo caso è di tipo SOCK_STREAM che permette di implementare una comunicazione affidabile di tipo TCP, il terzo parametro non è sempre settato ma serve per specificare la versione del protocollo nel caso la famiglia e il tipo non bastino per specificarlo(in questo caso è 0, come nella maggior parte dell'utilizzo delle socket).

Necessario è assegnare una porta alla socket appena creata, in questo viene fatto sfruttando la funzione **bind(socket, localaddr, addrlen),** nel nostro caso specifico, il socket è quello ottenuto come valore di ritorno dalla funzione precedentemente utilizzata, il secondo parametro è localaddr che indica l'indirizzo IP locale e deve essere di tipo puntatore alla struttura sockaddr. Infatti nel nostro caso noi creiamo una struttura di tipo sockaddr_in che contiene sockaddr come primo parametro ma, siccome la nostra funzione bind richiede un tipo puntatore ad una struttura di tipo sockaddr allora casto sockaddr_in in sockaddr. Successivamente si trova il parametro addrlen che è di tipo intero e che serve per indicare la lunghezza dell'indirizzo passato al parametro due.

L'indirizzo locale che viene passato al parametro due, prima di essere castato a sockaddr è di tipo **sockaddr_in** che è una struttura per la gestione degli indirizzi internet così formata:

```
#include <netinet/in.h>

struct sockaddr_in {
    short          sin_family;   // e.g. AF_INET indica la famiglia
    unsigned short   sin_port;     // e.g. htons(3490) indica la porta
    struct in_addr   sin_addr;     // guarda struct in_addr, specificata sotto
    char           sin_zero[8];  // zero è quello che vogliamo
};


struct in_addr {
```

```
   unsigned long s_addr;  // load with inet_aton()
};
```

viene settato in questo modo lato server

```
my_addr->sin_family = AF_INET;
my_addr->sin_port = htons(4950);
my_addr->sin_addr.s_addr = INADDR_ANY;
memset(my_addr->sin_zero, '\0', sizeof(my_addr->sin_zero));//setto a zero il valore sin_zero
```

Un'altra cosa fondamentale per poter creare una connessione di tipo TCP è fare una **listen(socket, qlength)** questo tipo di funzione viene usata solo per le connessioni di tipo TCP e permette di definire la lunghezza della coda delle richieste in ingresso. La funzione viene usata lato server ed accetta come primo parametro il socket server al quale definire la lunghezza che viene passata come secondo parametro.

In seguito bisognerà far fronte alla gestione di canali di I/O multipli(ovvero supportare questo servizio offerto dal server gestendo le richieste multiple dei client). Le socket normalmente utilizzano un approccio bloccante ovvero che ad ogni richiesta si bloccano e non ricevono più richieste da altri client. E' necessario quindi adottare una metodologia che ci permetta di rendere le socket non bloccanti, il nome della funzione da noi adottata è la funzione **select()** che specifica una lista di descrittori che controlla facendoli rimanere in attesa nel caso il server non sia disponibile subito. La funzione select si blocca fino a quando uno dei descrittori non è pronto per scrivere, leggere o per qualche eccezione. Il valore di ritorno della select è un descrittore pronto a svolgere qualche operazione nel canale TCP di quelle elencate prima. La funzione select() è così composta:

int select(int maxDescPlus1, fd_set * readDescs, fd_set * writeDescs, fd * exceptionDescs, struct timeval timeout);

-**maxDescsPlus1** è di tipo intero, si riferisce al valore massimo di descrittori

-**readDescs** controllo per chi è disponibile per la lettura

-**writeDescs** controllo per chi è disponibile per la scrittura

-**exceptionDescs** controllo chi è disponibile per segnalare un eccezione

-**timeout** per quanto tempo si blocca sulla funzione(NULL→per sempre)

-**returns** il numero totale di descrittori pronti, -1 in caso di errore

-**changes** cambia le liste di descrittori solo nelle posizioni in cui sono settati.

int **FD_ZERO** (fd_set *descriptorVector); /* rimuove tutti i descrittori dal vettore */

int **FD_CLR** (int descriptor, fd_set *descriptorVector); /* rimuove un descrittore da un vettore*/

int **FD_SET** (int descriptor, fd_set *descriptorVector); /* aggiunge un descrittore in un vettore*/

int **FD_ISSET** (int descriptor, fd_set *descriptorVector); /* controlla se un bit in un vettore è settato */

La struttura timeval è così formata:

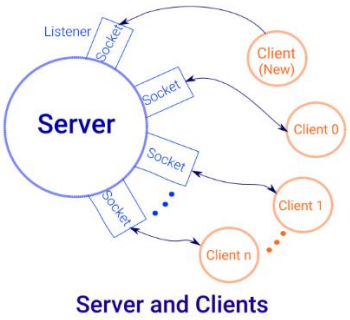
struct timeval {

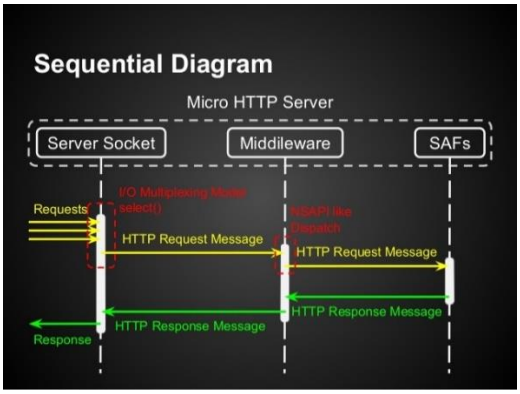
time_t tv_sec; /* secondi */

time_t tv_usec; /* microsecondi */

};

Il funzionamento della funzione select() avviene in questo modo:





tanti client si connettono al server come se sembra che ci siano molti socket attivi sul server, in realtà c'è un singolo socket che riceve i dati ma che lavora in multiplexing grazie ad un software middlewere che si trova tra la il livello socket e il livello applicativo.

Invio di dati:

- ### Messaggi:

  I messaggi spediti da un client vengono letti dal server con la funzione:

  recv(i, recv_buf, BUFSIZE, 0)

  - **i** rappresenta il socket dal quale leggere I dati
  - **recv_buf** indica il buffer dal quale leggere i dati
  - **BUFSIZE** indica la grandezza del buffer
  - **0** indica un flag

  Dopo essere stati letti i messaggi vengono inviati per mezzo della funzione send_to_all() a tutti i client. Questa funzione spedisce i messaggi con:

  send(j, recv_buf, nbytes_recvd, 0)

  - **j** rappresenta il socket al quale inviare i dati
  - **recv_buf** indica il buffer dal quale inviare i dati
  - **nbytes_recvd** indica quanti dati inviare
  - **0** rappresenta un flag

- ### File:

  Apro inizialmente il file, da inviare, in lettura con la funzione fopen(fs_name, "r").
  Invio il file spezzettandolo in pacchetti di grandezza 512byte. I pacchetti da inviare vengono creati con la funzione
  fread(sdbuf, sizeof(char), LENGTH, fs) dove:
  - **sdbuf** rappresenta il buffer di dati che si riempie(è il buffer che si invierà)
  - **sizeof(char)** invia la grandezza di ogni singolo dato inviato
  - **LENGTH** indica la grandezza del buffer
  - **fs** indica il file da dove leggere i dati

send(i, sdbuf, fs_block_sz, 0) serve per inviare i pacchetti creati al client

- **i** rappresenta il socket dove inviare i dati
- **sdbuf** indica il buffer da svuotare
- **fs_block_sz** indica la grandezza del buffer
- **0** è un flag

CLIENT:

Inizialmente al client viene fatto inserire un nome utente dopo di che viene creata una socket e stabilita la connessione TCP tra il client e il server e successivamente, l'invio e il ricevimento di messaggi tra i client passerà anche per questa socket.

Funzioni:

void send_recv(int i, int sockfd)

- **i** è l'indice del socket che indica chi è che vuole scrivere o leggere(server)
- **sockfd** è il puntatore al socket(file descriptor)

void connect_request(int *sockfd, struct sockaddr_in *server_addr)

- **sockfd** è il puntatore al socket(file descriptor)
- **server_addr** è la struttura che indica l'indirizzo IP del server

Socket:

Viene inizialmente creata una socket di tipo TCP con l'istruzione **socket(AF_INET, SOCK_STREAM, 0)** che è uguale a quella del server. A differenza del server a questa socket dobbiamo fare una connect con il server, quindi imposto l'indirizzo del server al quale connettermi analogamente a quanto fatto nella parte server, solo che al posto che INADDR_ANY inserisco 127.0.0.1 che è l'IP del server.

```
server_addr->sin_family = AF_INET;//uso l'operatore -> per settare i parametri nella struttura dell'indirizzo del server

server_addr->sin_port = htons(4950);

server_addr->sin_addr.s_addr = inet_addr("127.0.0.1");

memset(server_addr->sin_zero, '\0', sizeof(server_addr->sin_zero));//imposto la struttura come vuota.
```

Infine faccio la **connect(*sockfd, (struct sockaddr *)server_addr, sizeof(struct sockaddr))**

- **sockfd** è un puntatore alla socket del client la quale sarà in comunicazione con il server
- **server_addr** è la struttura che contiene l'indirizzo e la porta dove contattare il server
- **sizeof(struct sockaddr)** serve per passare la grandezza della struttura dove è contenuto l'ip del server.

con la connect parte anche il tree way handshake, si fa con la connect() si fa:

Client **Server**

Connection required

----------SYN j ---------->

Connection acknowledged

<----------SYN k ACK J+1----------

Application data

----------ACK k+1---------->

Ricezione di dati:

- **Messaggi:**
  I messaggi spediti dal server al client vengono letti con la funzione
  - **char *fgets(char *s, int size, FILE *stream)** legge da uno stream, stdin, una sequenaz di caratteri che l'utente scrive e li immagazzina nel buffer puntato da s. Il numero di dati che prende è definito da size.

  I messaggi che vengono scritti dal client vengono spediti al server con la funzione

- **File:**
  Apro il file, che devo ricevere, in scrittura. Quindi se non è creato lo creo, altrimenti lo sovrascrivo.

  char* fr_name = "/home/matteo/Desktop/nuvolebianche.mp3";
  FILE *fr = fopen(fr_name, "w");

  Ricevo i dati con questa funzione:

  recv(sockfd, revbuf, LENGTH, 0)
  - **sockfd** è il file descriptor dove si riceveranno i dati
  - **recvbuf** è il buffer dove inserire i dati ricevuti
  - **LENGTH** indica la lunghezza del buffer
  - **0** è il flag

Avvio della funzionalità musica:

L'invio della parola "musica" scatenerà un meccanismo per ricevere un file dal server che verrà successivamente avviato dal client con la funzionalità di vlc in linea di comando.

- L'utente inserisce muscia nel terminale, il client riconosce questo e con la funzione
  - **send(sockfd, send_buf, strlen(send_buf), 0);**

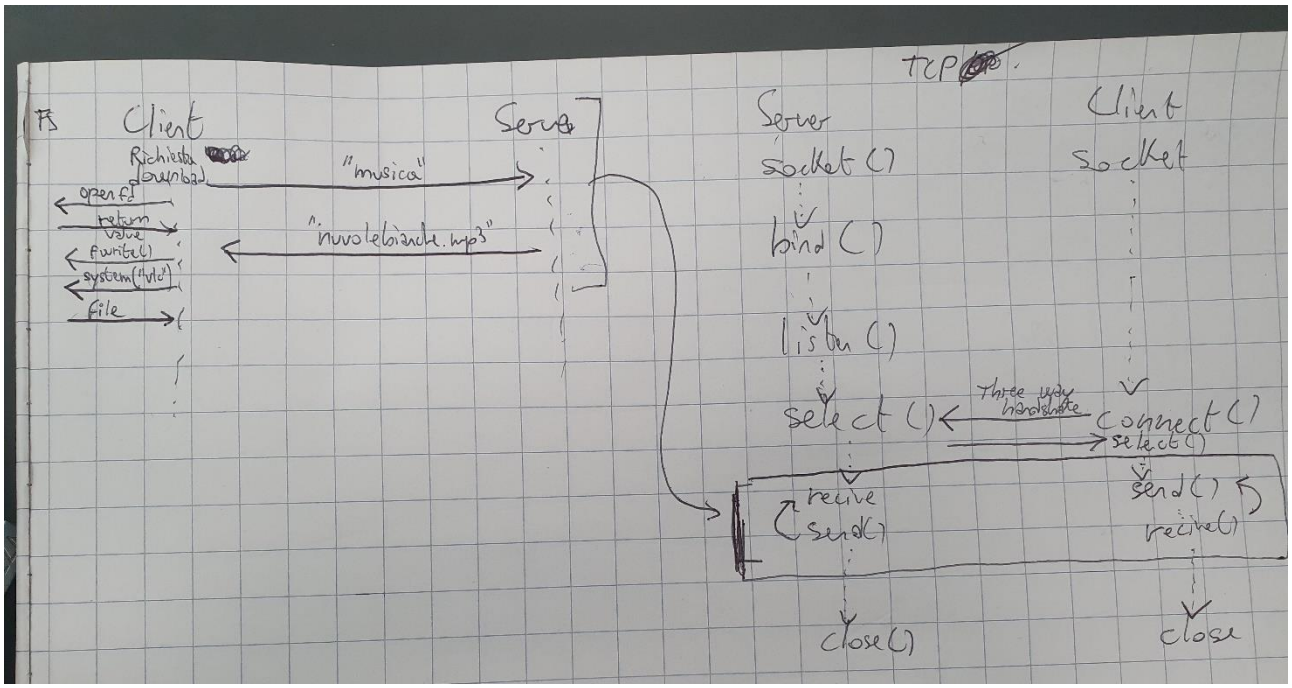
  indica al server che vuole scaricare la musica disponibile.

- Il client apre in scrittura il file:
  - **char* fr_name = "/home/matteo/Desktop/nuvolebianche.mp3";**
  - **FILE *fr = fopen(fr_name, "w");**
- Il client si mette in ascolto sulla scoket con la funzione
  - **recv(sockfd, revbuf, LENGTH, 0)**
- Il client salva il file nel File System
  - **fwrite(revbuf, sizeof(char), fr_block_sz, fr)**
- Una volta ricevuto il file il client lo avvia con la chiamata di sistema:
  - **system("cvlc /home/matteo/Desktop/nuvolebianche.mp3");**

system() è la chiamata di sistema che permette di eseguire i comandi che vengono passati da linea di comando come se ci trovassimo in un normale terminale.

Grazie alla funzione select() implementata lato client, permettiamo al server di ricevere tutti i messaggi che vengono inviati dagli altri utenti mentre è in esecuzione vlc e conseguentemente di stamparli a video facendo in modo che l'utente non si perda la conversazione della chat.

Questo è il Diagram Flow dell'applicazione:



FS→File System