Università degli studi di Urbino “Carlo Bo”

Corso di Laurea in Informatica Applicata

Progetto di

Reti di Calcolatori

per la sessione annuale 2018/2019

File Transfer Protocol (FTP)
Studio e implementazione in C

Studente: Angelo Sasso

Matricola: 263143

a.sasso4@campus.uniurb.it

Prof.re: Antonio Della Selva

Introduzione

Specifiche del problema

Il progetto in questione è volto ad approfondire le conoscenze acquisite con il linguaggio C, unite al Livello di Applicazioni delle Reti di Calcolatori con particolare riferimento al File Transfer Protocol (FTP) basato sullo standard RFC959, volto alla trasmissione di dati sfruttando un'architettura client-server.

Il programma crea una connessione bidirezionale tra client e server. Il funzionamento di base consiste nell'avvio del server che rimane in attesa di un client e dopo aver eseguito i comandi o essersi disconnesso, rende nuovamente disponibile il server per un altro client.

Non si è riuscito purtroppo a creare una doppia connessione tra client e server per dividere il flusso di dati dal flusso di controllo, assente quindi la logica di autenticazione dei client.

Con l'utilizzo del FTP si prefiggono, principalmente, quattro obbiettivi:

1) promuovere la condivisione di file

2) promuovere indirettamente l' uso di computer remoti

3) proteggere le risorse di host dalle modifiche ad opera di utenti

4) trasferire dati in modo efficiente e corretto.

Implementazione

Si è scelto Linux come ambiente di sviluppo, in particolare la nuova release Ubuntu 18.04 e il Linguaggio C con compilatore gcc.

Il progetto, composto da file di codice e librerie per semplificarne le funzioni, sfrutta delle regole comuni per l'inclusione di librerie apposite quali *arpa/inet.h*, *netinet/in.h* e per la dichiarazione delle funzioni implementate; queste sono definite tra le funzioni del client. I metodi di trasferimento sono affidate a due principali prototipi: EOF (End Of Trasmission) e TERM.

Client

Il client_ftp viene inizializzato specificandone la famiglia, l'indirizzo IP del server e la porta del server.

Server

Il server_ftp contiene il codice per la sua inizializzazione e per i controlli sugli errori. Una volta avviato rimane in attesa di creare una connessione con un client.

Data Transfer Protocol

Riprodotto nel file_transfer_functions.

Lo scambio di comandi avviene tramite messaggi FTP, del quale ne vengono creati gli interpreti:

Server Protocol Interpreter

Implementato nel server_ftp_functions, che acquisisce i dati dal client .

Client Protocol Interpreter

Implementato nel client_ftp_functions, contiene le definizioni dei comandi .

Lo standard RFC959 (1985) prevede tutti gli strumenti necessari per la manipolazione dei file e la sincronizzazione degli host, al quale sono seguiti importanti update nel 2000 con RFC2773 (prevede la crittografia KEA e SKIPJACK), fino al 2010 con RFC5797.
Il progetto si propone di permetterne un numero esiguo, ma è possibile ampliarne le funzioni; queste sfruttano i puntatori *data* e *chp* alla struttura *packet* ed un puntatore al buffer *filename* impostando poi il comando.
Ogni qualvolta viene inoltrata una REQUEST al server, la funzione getcwd() effettua un controllo d'integrità, acquisisce il pathname assoluto della directory corrente di lavoro copiandone il nome nel buffer puntato da *buf* e di dimensione *size*.
Se il pathname assoluto richiede un buffer piu' grande di size, la funzione fallisce; viene inoltre sfruttata la size per controlli di integrità in caso di omissioni dell'utente.

Segue la lista dei comandi definiti e alcuni passi di codice:

GET - apre il file in scrittura

PUT - invia un file

```
void command_put(struct packet* chp, struct packet* data, int sfd_client, char* filename)
{
        FILE* f = fopen(filename, "rb");
        if(!f)
        {
                fprintf(stderr, "Il file non può essere aperto in lettura. Chiusura...\n");
                return;
        }
        int x;
        set0(chp);
        chp->type = REQU;
        chp->conid = -1;
        chp->comid = PUT;
        strcpy(chp->buffer, filename);
        data = htonp(chp);
        if((x = send(sfd_client, data, size_packet, 0)) != size_packet)
                er("send()", x);
        if((x = recv(sfd_client, data, size_packet, 0)) <= 0)
                er("recv()", x);
        chp = ntohp(data);

        if(chp->type == INFO && chp->comid == PUT && strlen(chp->buffer))
        {
                printf("\t%s\n", chp->buffer);
                chp->type = DATA;
                send_file(chp, data, sfd_client, f);
                fclose(f);
        }
        else
                fprintf(stderr, "Errore nell'invio del file.\n");
        send_EOT(chp, data, sfd_client);
}
```

MGET - ricezione multipla di file

MPUT - invio multiplo di file

MPUTWILD - invia tutti i file e le cartelle

MGETWILD -riceve tutti i file e le cartelle

DIR -lista del contenuto della directory remota

LDIR - lista del contenuto della directory locale

LS - lista del contenuto della directory remota

LLS - lista del contenuto della directory locale

RGET - riceve tutti i file

RPUT - invia tutti i file

CD - cambia la directory di lavoro remota

```
void command_cd(struct packet* chp, struct packet* data, int sfd_client, char* path)
{
        int x;
        set0(chp);
        chp->type = REQU;
        chp->conid = -1;
        chp->comid = CD;
        strcpy(chp->buffer, path);
        data = htonp(chp);
        if((x = send(sfd_client, data, size_packet, 0)) != size_packet)
                er("send()", x);
        if((x = recv(sfd_client, data, size_packet, 0)) <= 0)
                er("recv()", x);
        chp = ntohp(data);
        if(chp->type == INFO && chp->comid == CD && !strcmp(chp->buffer, "success"))
                ;
        else
                fprintf(stderr, "\tErrore di esecuzione del comando sul server.\n");
}
```

LCD - cambia la directory di lavoro locale

MKDIR - crea una cartella nella directory remota

```
void command_mkdir(struct packet* chp, struct packet* data, int sfd_client, char* dirname)
{
        int x;
        set0(chp);
        chp->type = REQU;
        chp->conid = -1;
        chp->comid = MKDIR;
        strcpy(chp->buffer, dirname);
        data = htonp(chp);
        if((x = send(sfd_client, data, size_packet, 0)) != size_packet)
                er("send()", x);
        if((x = recv(sfd_client, data, size_packet, 0)) <= 0)
                er("recv()", x);
        chp = ntohp(data);
        if(chp->type == INFO && chp->comid == MKDIR)
        {
                if(!strcmp(chp->buffer, "success"))
                        printf("\tDirectory creata sul server.\n");
                else if(!strcmp(chp->buffer, "gia' esistente"))
                        printf("\tDirectory gia' esistente sul server.\n");
        }
        else
                fprintf(stderr, "\tErrore durante l'esecuzione del comando sul server.\n");
}
```

LMKDIR - crea una cartella nella directory locale

PWD - stampa sul terminale della directory di lavoro

```
void command_pwd(struct packet* chp, struct packet* data, int sfd_client)
{
        int x;
        set0(chp);
        chp->type = REQU;
        chp->conid = -1;
        chp->comid = PWD;
        data = htonp(chp);
        if((x = send(sfd_client, data, size_packet, 0)) != size_packet)
                er("send()", x);
        if((x = recv(sfd_client, data, size_packet, 0)) <= 0)
                er("recv()", x);
        chp = ntohp(data);
        if(chp->type == DATA && chp->comid == PWD && strlen(chp->buffer) > 0)
                printf("\t%s\n", chp->buffer);
        else
                fprintf(stderr, "\tErrore nel recupero delle informazioni.\n");
}
```

LPWD - stampa sul terminale della directory locale

EXIT - termina il client

Conclusioni

Testing

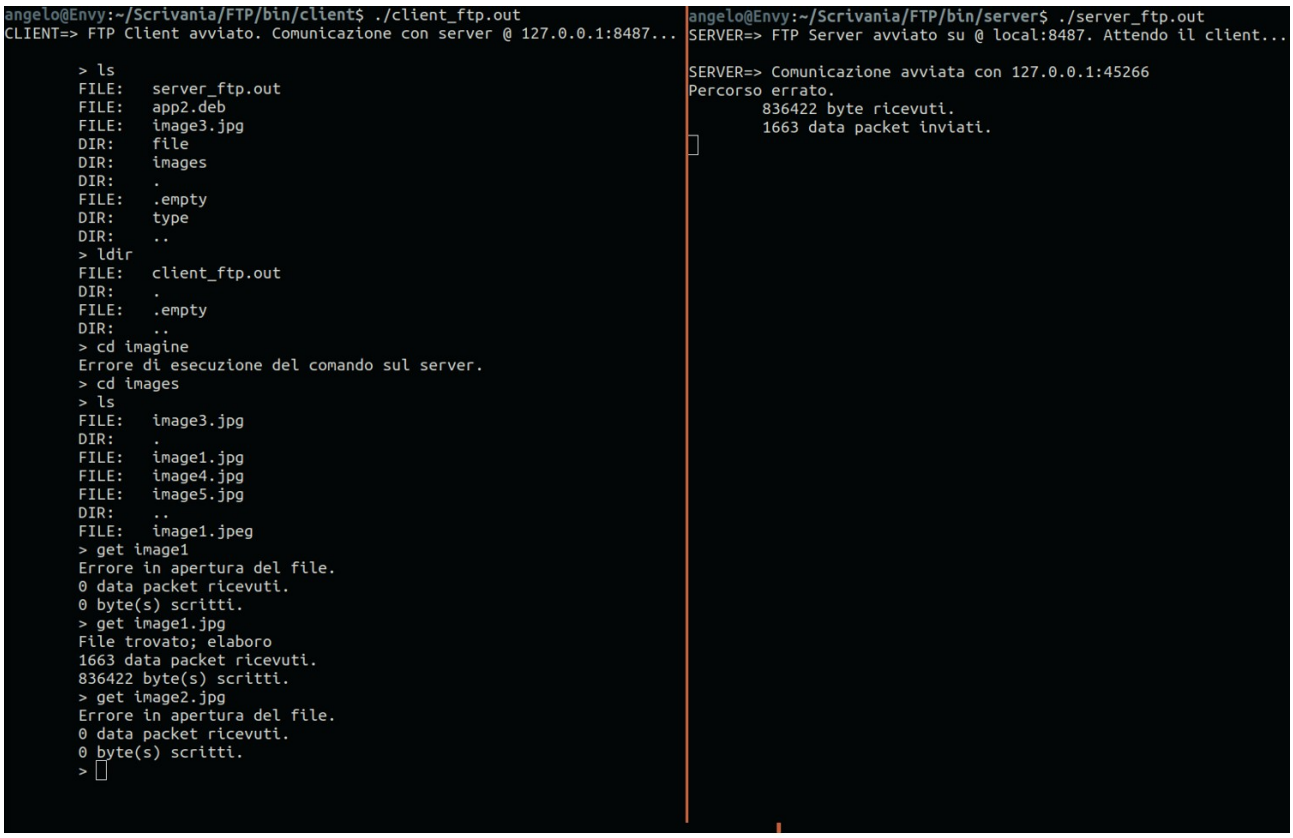
E’ possibile testare il programma spostandosi col terminale sulla directory FTP:

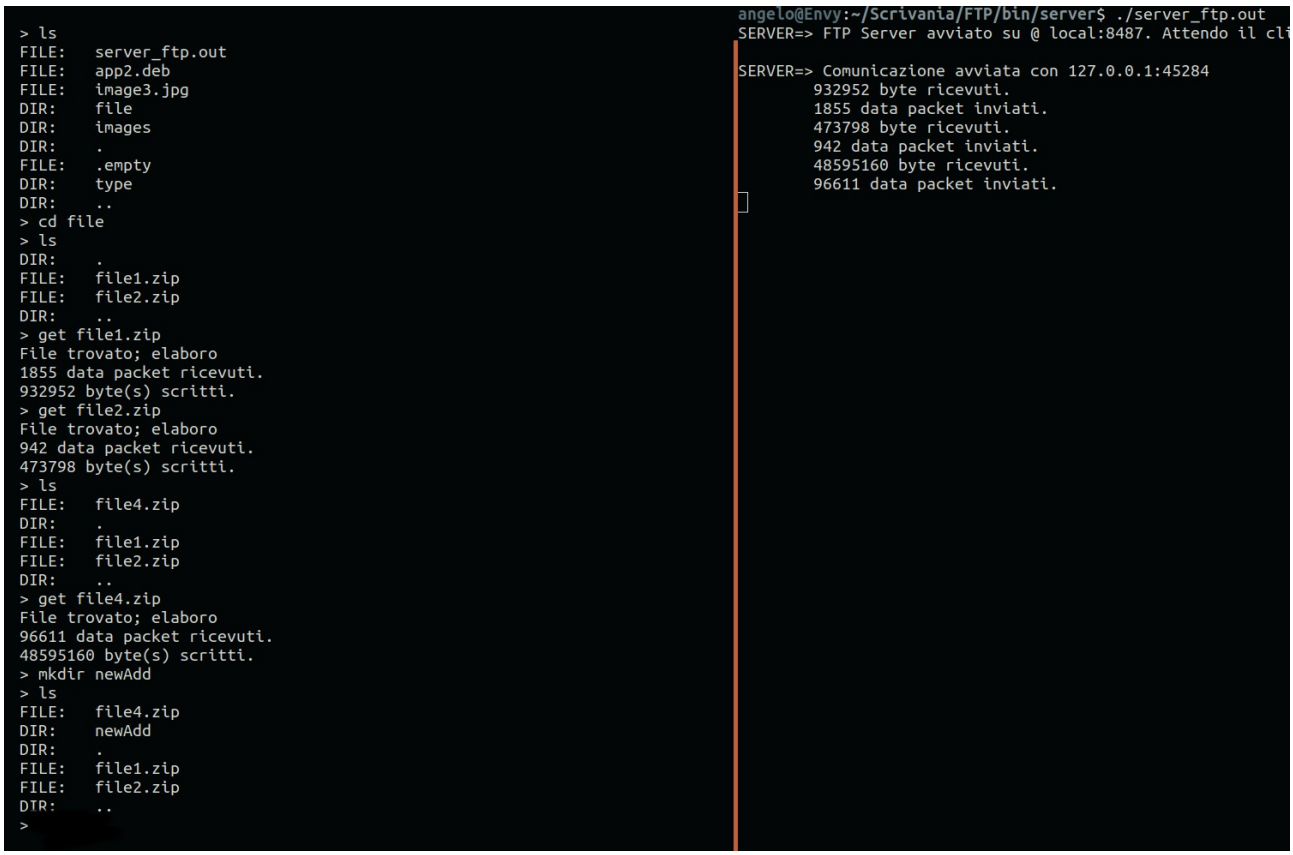
```
~<percorso>/FTP$    cd bin
                    cd server
                    ./server_ftp.out
```

Aprendo poi un altro terminale è possibile avviare il client:

```
~<percorso>/FTP$    cd bin
                    cd server
                    ./client_ftp.out
```

Test con gestione di alcuni errori, prelievo di file e creazione di directory:





Infine la chiusura del client e la disconnessione dal server: