RETI INFORMATICHE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

SPECIFICHE DI PROGETTO A.A. 2018/2019

Il progetto consiste nello sviluppo di un'applicazione client/server scritta in C. Nello specifico, l'applicazione da sviluppare è un sistema di ***trasferimento file*** basato sul protocollo Trivial File Transfer Protocol **TFTP**, che definisce un protocollo molto semplice per lo scambio di files. Il protocollo di comunicazione tra il client e il server viene descritto all'interno del seguente documento:

https://tools.ietf.org/html/rfc1350

Per semplicità riportiamo di seguito una breve descrizione del protocollo. Si richiede comunque la lettura del documento di specifiche.

PROTOCOLLO TFTP

Il protocollo è composto dai messaggi illustrati in Figura 1, mentre in Figura 2 si illustra un esempio di sequenza dei messaggi. I messaggi sono trasmessi utilizzando il protocollo UDP. Lo scambio dei messaggi è originato dal client che invia un messaggio di tipo RRQ o WRQ. Il messaggio RRQ richiede il trasferimento di un file (download) dal server al client, mentre il messaggio RRQ richiede il trasferimento di un file dal client al server (upload). Per semplicità si illustrerà in seguito solo la modalità di trasferimento dal server al client (download).

Il file da essere trasferito è individuato attraverso il nome. Considerando che un file può essere letto/scritto in formato binario o testuale (si veda la pagina https://www.programiz.com/c-programming/c-file-input-output per chiarezza), il messaggio di richiesta (RRQ o WRQ) include un campo 'mode' che indica il formato del file da trasferire.

Il trasferimento comincia con un messaggio RRQ inviato dal client al server. A seguito di un messaggio RRQ il server risponde con una serie di messaggi di tipo DATA, ciascuno contenenti una porzione del file (al massimo 512 bytes). Il trasferimento termina implicitamente quando si invia (o viene ricevuto) un frammento di dimensioni inferiori a 512 bytes. Ogni blocco del file trasferito è contraddistinto da un numero di sequenza (block number) a 16 bit, inserito nel pacchetto DATA. La ricezione di ogni messaggio dati è confermato dall'invio di un messaggio di tipo ACK, il quale contiene il numero del blocco ricevuto come riferimento.

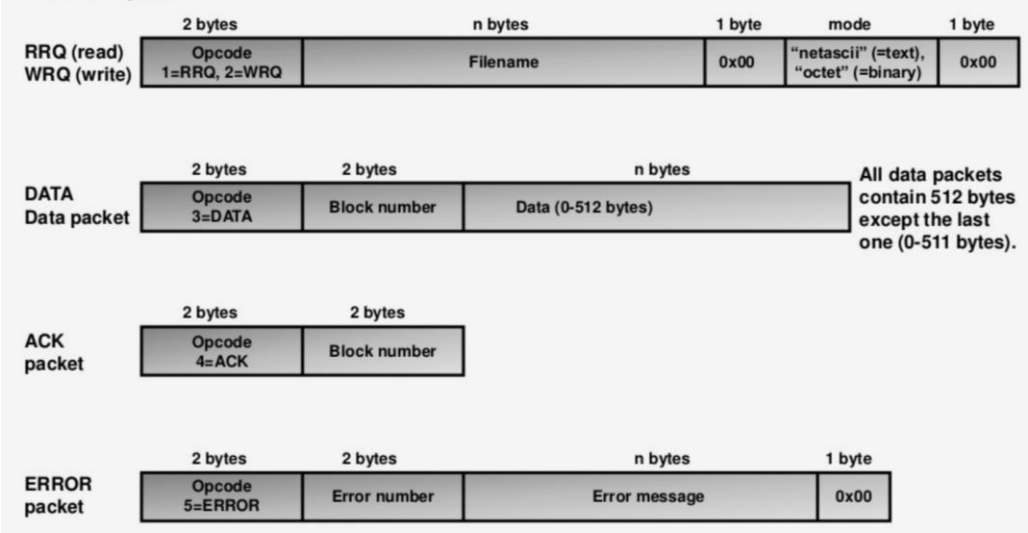

Figura 1. Messaggi del protocollo TFTP.

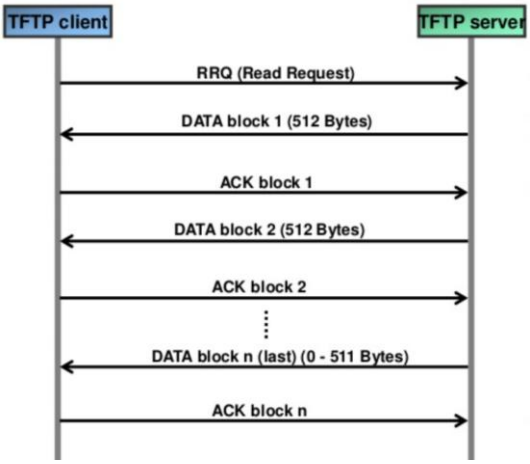

Figura 2. Sequenza dello scambio messaggi.

Nel caso in cui un messaggio DATA o ACK venga perso, dopo un certo periodo il mittente del messaggio dovrà prendersi cura di inviare nuovamente il messaggio stesso.

A seguito di messaggi di errore, il client o il server possono inviare un messaggio ERROR, il quale termina il trasferimento.

IMPLEMENTAZIONE

Si realizzi un'applicazione client e un'applicazione server che implementino il protocollo TFTP per lo scambio di files. Nello specifico, il server dovrà dare la possibilità di scaricare uno dei file presenti in una data directory locale, mentre il client dovrà mostrare all'utente una schermata attraverso la quale potrà specificare il nome del file da scaricare e il nome con cui questo dovrà essere salvato localmente.

Rispetto al protocollo TFTP completo, al fine di semplificare l'implementazione, si facciano le seguenti assunzioni:

1. Nonostante sia definito il trasferimento dal Client al Server (operazione Write) e dal Server al Client (operazione Read), **il server e il client dovranno implementare solamente il trasferimento dal Server al Client** (operazione Read o download).
2. **Si assuma che i messaggi non possano perdersi, in altre parole non si devono implementare le ritrasmissioni**.
3. Vengano gestiti **solamente i seguenti errori**:
    a. File not found.
    b. Illegal TFTP operation, da generare in risposta ad un messaggio di richiesta con un'operazione diversa da RRQ.

Il server non dovrà mantenere nessuno stato sulle richieste effettuate dai clients, se non i dati relativi al trasferimento corrente. Il server dovrà essere in grado di gestire richieste concorrenti.

LATO SERVER

Il server **tftp_server** si occupa di ricevere le richieste dai clients e di inviare a ciascuno il file richiesto.

```
./tftp_server <porta> <directory files>
```

<porta> è la porta su cui il server è in ascolto (la 69 di default nel caso del protocollo TFTP, richiede che il server sia lanciato con privilegi di super utente). Il server dovrà essere raggiungibile da tutti gli indirizzi IP della macchina su cui è in esecuzione.
<directory files> è la directory dalla quale leggere i files da inviare ai clients.

Una volta eseguito, **tftp_server** deve stampare a video delle informazioni descrittive sullo stato del server (creazione del socket di ascolto, connessioni ricevute, operazioni richieste dai client ecc.).

PROGRAMMA CLIENT

Il client deve essere avviato con la seguente sintassi:

```
./tftp_client <IP server> <porta server>
```

dove:

- **<IP server>** è l'indirizzo dell'host su cui è in esecuzione il server;
- **<porta server>** è la porta su cui il server è in ascolto.

All'avvio, i comandi disponibili per l'utente devono essere:

- `!help`
- `!mode`
- `!get`
- `!quit`

Il client deve stampare tutti gli eventuali errori che si possono verificare durante l'esecuzione.
Un *esempio* di esecuzione è il seguente:

```
$ ./tftp_client 127.0.0.1 69


Sono disponibili i seguenti comandi:
!help --> mostra l'elenco dei comandi disponibili
!mode {txt|bin} --> imposta il modo di trasferimento dei files (testo o
binario)
!get filename nome_locale --> richiede al server il nome del file
<filename> e lo salva localmente con il nome <nome_locale>
!quit --> termina il client
```

Nel caso in cui il nome del file non sia presente sul server, il client deve continuare l'esecuzione stampando un messaggio di errore.

Comandi base

`!help`: mostra l'elenco dei comandi disponibili.
Esempio di esecuzione:

```
Sono disponibili i seguenti comandi:
!help --> mostra l'elenco dei comandi disponibili
!mode {txt|bin} --> imposta il modo di trasferimento dei files (testo o
binario)
!get filename nome_locale --> richiede al server il nome del file
<filename> e lo salva localmente con il nome <nome_locale>
!quit --> termina il client
```

`!mode {txt|bin}:` attraverso questo comando l'utente specifica il modo di trasferimento del file, testuale o binario.

Esempio di esecuzione:

```
> !mode bin
Modo di trasferimento binario configurato
>
```

`!get filename nome_locale`: il client richiede al server il trasferimento del file <filename>, il quale viene salvato localmente nel path (assoluto o relativo) <nome_locale>. Il trasferimento deve avvenire attraverso l'ultimo modo di trasferimento specificato dall'utente, se l'utente non ha ancora specificato un modo *deve essere usato il modo di trasferimento binario*.
Esempio di esecuzione:

```
> !get testo.txt ./testo_download.txt
Richiesta file testo.txt al server in corso.
Trasferimento file in corso.
Trasferimento completato (5/5 blocchi)
Salvataggio ./testo_download.txt completato
>
```

Nel caso in cui il file non sia presente sul server, il client deve mostrare un messaggio di errore, lasciando all'utente la possibilità di continuare (il programma non deve terminare).
Esempio di esecuzione:

```
> !get pippo.txt ./testo_download.txt
Richiesta file pippo.txt al server in corso.
File non trovato.
>
```

REQUISITI

- Si utilizzino gli autotools (comando make) per la compilazione del progetto.

CONSEGNA DEL PROGETTO

Il progetto dovrà essere consegnato almeno 24h prima dell'esame attraverso il sistema elearn.ing.unipi.it sulla pagina apposita del corso (http://elearn.ing.unipi.it/course/view.php?id=1383) usando le proprie credenziali di ateneo. Per ogni appello verrà creato una nuova sezione apposita per le consegne.

VALUTAZIONE DEL PROGETTO

Il progetto viene valutato prima dello svolgimento dell'esame. Il tutto verrà fatto girare su una macchina con sistema operativo **Debian 8**. Si consiglia caldamente di testare il sorgente su una macchina Debian 8 prima della consegna. Durante lo svolgimento dell'esame potrà essere chiesta l'esecuzione del programma. La valutazione prevede le seguenti fasi:

1. **Compilazione dei sorgenti**. Il client e il server vengono compilati attivando l'opzione **`-Wall`** che abilita la segnalazione di tutti i warning, e **non dovranno presentare warning o errori**. Si consiglia vivamente di usare tale opzione anche durante lo sviluppo del progetto, **interpretando i messaggi forniti dal compilatore.**
2. **Esecuzione dell'applicazione**. Il client e il server vengono eseguiti simulando una tipica sessione di utilizzo. In questa fase si verifica **il corretto funzionamento dell'applicazione e il rispetto delle specifiche fornite e della corretta implementazione del protocollo.**
3. **Esame del codice sorgente**. Il codice sorgente di client e server viene esaminato per controllarne l'implementazione. **Può essere richiesto di apportare modifiche al codice sul momento. Il codice verrà analizzato anche da software per l'individuazione del plagio sulle consegne dei colleghi e altre implementazioni presenti in rete.**

FUNZIONI DI UTILITA

Una documentazione di alto livello per le funzioni necessarie per leggere e scrivere files a blocchi possono essere visualizzate al seguente indirizzo:

https://www.programiz.com/c-programming/c-file-input-output