**Reti di Calcolatori ed Ingegneria del Web**

**A.A. 2019/20**

**Progetto B2:**

**Trasferimento file su UDP**

**Angeloni Ilaria 0220229**

**Lattanzi Michela 0227803**

**Indice**

|  |  |
| --- | --- |
| 1. ***Introduzione***    1. ***Panoramica UDP***    2. ***Socket*** 2. ***Architettura del sistema e scelte progettuali***   ***2.1 Istruzioni***   * 1. ***Strutture***   ***2.3 Funzioni***  ***3. Implementazione***  ***3.1 Client***  ***3.2 Server***  ***3.3 Trasmissione affidabile***  ***3.4 Protocollo Selective Repeat***  ***3.5 Parametri di configurazione***  ***3.6 Thread e trasferimento dati***   * 1. ***File log***   ***4. Limitazioni riscontrate***  ***5. Piattaforma software (sviluppo e testing)***  ***6. Esempi di funzionamento***  ***7. Analisi delle prestazioni***  ***8. Manuale di installazione***  ***8.1 Configurazione del sistema***  ***8.2 Comandi client*** | p.3  p.3  p.3  p.3  p.4  p.5  p.  p.  p.  p.  p.  p.  p.  p.  p.  p.  p.  p. |

***1. Introduzione***

Il progetto ha lo scopo di realizzare ed implementare nel linguaggio C, utilizzando l’API del socket di Berkeley, un’applicazione basata sull’architettura client-server che permette la comunicazione tra host servendosi di un servizio di rete non orientato alla connessione.

***1.1 Panoramica UDP***

UDP è un protocollo di trasporto non orientato alla connessione e non affidabile: non garantisce né l’arrivo dei pacchetti a destinazione, né la ritrasmissione in caso di perdita d’informazione, né la corretta sequenza. Il protocollo UDP fornisce soltanto i servizi basilari del livello di trasporto come la consegna di dati da processo a processo e il controllo degli errori. Un’applicazione che usa UDP deve risolvere problemi di affidabilità, perdita di pacchetti, duplicazione, controllo di sequenza, controllo di flusso, controllo di congestione.

***1.2 Socket***

Il socket è un oggetto di I/O associato ad un canale di I/O. Il suo compito è quello di fare da intermediario tra il livello di trasporto e il livello di applicazione.

Un socket UDP viene identificato univocamente dall’indirizzo IP e dal numero di porta di destinazione.

***2. Architettura del sistema e scelte progettuali***

L’applicazione, basata su un’architettura client-server, utilizza UDP come protocollo di trasporto e permette il trasferimento di file in modo affidabile.

L’applicazione si costituisce di tre moduli:

* ***program.h*** nel quale sono definite le principali strutture dati, le variabili globali e le procedure che verranno poi condivise dagli altri due moduli;
* ***client.c*** costituito dal main che implementa procedure e strutture dati utilizzate solamente dal client;
* ***server.c*** costituito dal main che implementa procedure e strutture dati utilizzate solamente dal server.

Per il funzionamento dell’applicazione sono richiesti un ***file di configurazione***, con i relativi valori dei parametri di configurazione, e un ***file log***, per registrare le operazioni svolte e analizzare la sequenza delle azioni compiute in trasmissione e ricezione.

Nell’applicazione, client e server sono di tipo concorrente. La concorrenza è stata implementata tramite la creazione di processi figli che gestiscono la connessione client/ server in base al relativo comando.

***2.1 Istruzioni***

Sono stati sviluppati tre comandi principali, con le rispettive funzionalità per client e server:

• ***list*** per la visualizzazione sul client dei file disponibili sul server;

• ***get*** per il download di un file dal server al client;

• ***put*** per l’upload di un file sul server.

Il processo figlio, creato appositamente dal client per la gestione delle richieste, si comporta in modo diverso in base al comando ricevuto.

In caso di comando ***list***, alloca un’area di memoria per i nomi degli oggetti delle directory che riceverà dal server; nel caso di comando ***get***, apre il file destinazione in scrittura affinchè possa esserci copiato il file sorgente ed infine per quanto concerne il ***put***, apre il file origine in lettura.

Il server riceve il messaggio CMD dal client e crea un processo figlio che rispettivamente: alloca un’area di memoria per i nomi degli oggetti contenuti nella directory ( list); apre il file origine in lettura (get); apre il file destinazione in scrittura (put). Il server invia quindi un messaggio del tipo ACK al client.

Per le istruzione ***list*** e ***get***: il client risponde all’ACK con un messaggio START e si predispone alla ricezione dei dati; il server inizia il trasferimento dei dati e quando il client riceve l’ultimo pacchetto, invia il messaggio END al server.

Per l’istruzione ***put***: il server è disponibile alla ricezione; quando il client riceve il messaggio ACK, inizia il trasferimento dei dati; una volta ricevuto l’ultimo pacchetto, il server invia il messaggio END al client.

Sono presenti inoltre altri due comandi che consentono la chiusura della comunicazione. Con “***stop***” il server viene arrestato e con “***quit***” termina anche il client.

***2.2 Strutture***

Le strutture principali vengono definite in *program.h* e sono:

* ***struct frame***: struttura dedicata alla definizione del frame; vengono infatti dichiarate le variabili tipo di frame, istruzione, nome del file o della directory (a seconda dell’istruzione) da trasferire, esito della richiesta, numero di sequenza del frame, posizione nel buffer di trasmissione dell'ultimo di eof, buffer contenente i dati da trasmettere, stato del frame (ovvero se è stato trasmesso o meno) e porta del server concorrente associata al processo.
* ***struct ack***: definisce le variabili necessarie per la ricezione dei messaggi di ack ovvero tipo di frame, numero di sequenza del frame a cui si riferisce l’ack, tempo di trasmissione del frame a cui si riferisce l’ack, variabile che indica se si tratta di una prima trasmissione o di una ritrasmissione dell’ack.
* ***struct timeout\_args***: struttura utilizzata nel caso in cui sia necessaria una ritrasmissione del frame; vengono definite le variabili id del socket, intervallo di ritrasmissione, frame da ritrasmettere e indirizzo del computer.
* ***struct send\_slot***: vengono dichiarate le variabili per la definizione di uno slot del buffer di trasmissione, vengono richiamate le strutture timeout\_args e frame rispettivamente per i dati e per il frame da trasmettere, viene definito il numero del thread che esegue la procedura di timeout e una variabile che indica se il frame ha o meno ricevuto l’ack.
* ***struct send\_file\_args***: struttura che verrà poi utilizzata per trasferire i parametri tra le diverse funzioni; le variabili inizializzate sono quelle per gli indirizzi remoto e locale del computer, id del socket, nome del file di origine e destinazione, puntatori ai file di origine e destinazione, path da visualizzare in remoto (tramite lo specifico comando list), istruzione in esecuzione, numero di sequenza dell’ultimo ack ricevuto, numero di sequenza dell’ultimo frame trasmesso, ampiezza della finestra di trasmissione, massimo numero di sequenza, e la struttura send\_slot per la dichiarazione del frame buffer. La definizione di alcune di queste variabili deriva dalla scelta di utilizzare il protocollo Selective Repeat.
* ***struct config***: struttura che permette di definire i parametri di configurazione che sono: ip del server, porta del server, tipo di intervallo di ritrasmissione (che può essere adattativo o fisso), intervallo di ritrasmissione iniziale, massimo numero di sequenza, ampiezza della finestra, intervallo di trasmissione tra due pacchetti, ritardo minimo per la ricezione di un ack per un pacchetto, numero massimo di ritrasmissioni, probabilità di perdita di un frame, tempo di attesa massimo per un frame, variabile booleana che indicherà se si vuole o meno raddoppiare il tempo di ritrasmissione e coefficiente moltiplicativo del ritardo sulla rete.
* ***struct rtt\_info***: struttura necessaria per la gestione del tempo di ritrasmissione in cui vengono definite le variabili ultima rilevazione del Round Trip Time (RTT), media pesata del tempo di andata e ritorno, valore corrente del tempo di ritrasmissione, numero massimo di ritrasmissioni per un frame e tempo inizio trasmissione.

***2.3 Funzioni***

In ***program.h*** vengono definite le principali funzioni:

* ***int read\_config():*** legge i parametri di configurazione dal file fornito come primo parametro di input al main e aggiorna la struct dei parametri di configurazione; apre in scrittura il file di log il cui nome è fornito come secondo parametro di input; restituisce 0 in caso di successo (vedi paragrafo *3.5 Parametri di configurazione)*.
* ***void rtt\_init()***: imposta i valori iniziali della struct per la gestione del timeout adattativo e il tempo di inizio sul campo rtt\_base.
* ***uint32\_t rtt\_ts()***: calcola e restituisce il tempo trascorso dal tempo di inizio
* ***float rtt\_calc()*** : verifica se è stato configurato un retrasmission timeout fisso o adattivo; nel secondo caso calcola e restituisce il valore di RTO
* ***void rtt\_print()***: scrive (opzionalmente) sul file di log i valori dei campi della struct.
* ***int read\_mem()***: legge DATA\_SIZE bytes dall’area di memoria in cui è stato scritto il contenuto della directory da trasferire e li scrive sull’array; assegna il valore ad alcuni campi del frame come il numero di sequenza, il tipo, il cursore end-of-file; restituisce 1 se incontra la fine dell’area di memoria.
* ***int write\_mem()***: scrive in un’area di memoria i dati ricevuti tramite un pacchetto; restituisce 1 quando riconosce l’ultimo pacchetto.
* ***int read\_frame()***: legge dal file origine DATA\_SIZE bytes e li scrive sull’ array data; restituisce 1 se incontra la fine del file.
* ***int write\_frame()***: scrive sul file destinazione i dati ricevuti tramite un pacchetto; restituisce 1 quando riconosce l’ultimo pacchetto.
* ***int seq\_check():*** controlla che il numero di sequenza del pacchetto rientri all'interno della finestra e ritorna 1 in caso affermativo; serve per l’implementazione del protocollo selective repeat.
* ***void\* timeout():*** ripete periodicamente la trasmissione di uno specifico frame fino a quando il thread viene cancellato (perché è arrivato il relativo riscontro ) o viene ecceduto il numero massimo di tentativi (il processo termina).
* ***void\* send\_data():*** vedi paragrafo *3.6 Thread e trasferimento dati*
* ***void\* wait\_ack():*** vedi paragrafo *3.6 Thread e trasferimento dati*
* ***void transmit():*** vedi paragrafo *3.6 Thread e trasferimento dati*
* ***void receive():*** vedi paragrafo *3.6 Thread e trasferimento dati*

Nel client:

* ***int check\_ex():*** verifica che i file sorgente e destinazione abbiano la stessa estensione.
* ***int read\_cmd():*** legge l’istruzione da riga di comando e ne controlla la correttezza sintattica.

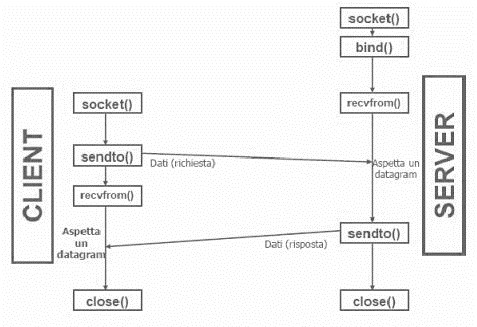
Nel server:

* ***char\* get\_directory\_contents():*** restituisce la lista dei file contenuti nella directory specificata.
* ***void srv\_ric():*** apre il file specificato nell’istruzione in scrittura, invia al client la risposta con l’esito dell’operazione ed avvia la procedura di ricezione; se si verifica un errore nell'apertura del file, arresta il processo figlio.
* ***void srv\_trasm():*** apre il file specificato nell’istruzione in lettura (o copia il contenuto della directory richiesta in un’area di memoria), invia al client la risposta con l’esito dell’operazione e si pone in attesa di un comando START e avvia la trasmissione; se la risposta non arriva entro un numero massimo di secondi, si interrompe.

***3. Implementazione***

***3.1 Client***

Il ***main client*** legge il file di configurazione e imposta i parametri. Nel ciclo while vengono lette le istruzioni da riga di comando e viene eseguita una fork. Il genitore torna in attesa di un comando digitato da stdin e termina in caso di istruzione “quit”. Il processo figlio gestisce la connessione con il server, crea il socket ed estrae l'indirizzo (la porta viene assegnata automaticamente dal kernel del sistema), invia l'istruzione al server e attende una risposta. In caso di esito positivo, se l'istruzione è *get* o *list* invia il comando START al server e avvia la procedura di ricezione; se è *put*, avvia la procedura di trasmissione; se entro un tempo limite (variabile TMaxS) non arriva la risposta, si arresta. Il client può arestare il server da remoto con l’istruzione “stop”.



***3.2 Server***

Il ***main server*** legge il file di configurazione e apre il relativo file log. Viene creato un socket per gestire i comandi ricevuti dal client; per ogni comando viene eseguita una fork() . Il processo figlio crea un socket, il cui indirizzo verrà reso noto nella risposta inviata al client, e avvia la procedura relativa al comando ricevuto. *srv\_ric()* apre il file in scrittura, invia al client l’esito dell’operazione e avvia la ricezione con *receive()*; *srv\_trasm()* apre il file in lettura, invia al client l’esito dell’operazione e quando riceve dal client il comando START avvia la trasmisssione con *transmit()*. Il processo figlio gestisce quindi la connessione con il client, mentre il padre resta in attesa di istruzioni dal client; termina se riceve il comando “stop”.

***3.3 Trasmissione affidabile***

Trasmissione e ricezione vengono eseguite applicando un protocollo ***Selective Repeat*** che permette una comunicazione affidabile tra client e server. SR è un protocollo a finestra scorrevole nel quale è necessaria la gestione di timer multipli, inoltre mittente e destinatario devono gestire un buffer per mantenere i frame. Vengono quindi definiti determinati parametri, settati nel file di configurazione.

***3.4 Protocollo Selective Repeat***

Si tratta di un protocollo con pipeline a livello di trasporto che serve per rendere il trasferimento dei dati affidabile. È basato sul concetto fondamentale di ritrasmissione dei soli pacchetti per i quali c'è un sospetto di errore, che può essere dovuto dallo smarrimento o dall’alterazione. È presente un’ampiezza di finestra pari ad N allo scopo di limitare il numero di pacchetti privi di ack nella pipeline. Gli acknowledgment vengono inviati per ogni pacchetto ricevuto in modo corretto, ma non è necessario che venga ricevuto in ordine. I pacchetti ricevuti correttamente verranno infatti memorizzati in un buffer fin quando non verranno ricevuti tutti i pacchetti mancanti. Nel momento in cui gli N pacchetti vengono ricevuti il blocco viene trasportato in ordine al livello superiore. Il pacchetto smarrito può anche essere quello inviato dal ricevente al mittente, ossia il pacchetto contenente l’ack, perciò può accadere che le finestre di mittente e destinatario non coincidano. La mancanza di sincronizzazione tra le finestre può avere importanti conseguenze per cui si richiede che la finestra abbia un’ampiezza inferiore o al più uguale alla metà dello spazio dei numeri di sequenza dei protocolli SR.

***3.5 Parametri di configurazione***

I ***parametri di configurazione*** sono definiti in ***struct config*** e sono i seguenti:

* ***char srv\_ip[50]****: Server IP*, ha formato xxx.x.x.x
* ***int srv\_port****: Server port*
* ***int prob****: Loss packet probability,* per settare la probabilità di perdita dei pacchetti in rete.
* ***int tip\_RTO****: RTO type* (0=adattivo, 1=fisso)
* ***float RTO\_in****: Starting RTO*, valore iniziale del retransmission timeout.
* ***seqnum\_t seq\_max****: Max sequence number,* massimo numero di sequenza.
* ***seqnum\_t sws****: Window size,* dimensione della finestra scorrevole (deve essere <= alla metà del massimo numero di sequenza, per garantire che i dati vengano trasferiti correttamente)
* ***int rit\_send****: Transmission interval*, intervallo di tramissione fra due pacchetti, in millisecondi. 0 è il valore di default.
* ***int rit\_ack****: Waiting time for ack reception*, tempo minimo che deve intercorrere tra la trasmissione del pacchetto e la ricezione del riscontro.
* ***int max\_tent****: Max number of retransmission,* numero massimo di tentativi di ritrasmissione.
* ***int t\_max****: Maximum seconds of waiting for a package,* tempo di attesa massimo di un frame; se il pacchetto non viene ricevuto entro TMax secondi, il processo in ricezione si arresta; a sua volta il trasmittente, non ricevendo più ACK, esaurisce il numero massimo di trasmissioni e si arresta.
* ***int coef\_RTO****: Multiplicative coefficient RTO,* permette discegliere se raddoppiare il tempo di trasmissione del pacchetto ad ogni ritrasmissione dello stesso (1= si, 0=no) fino ad un massimo di 64 volte il valore della prima trasmissione
* ***int coef\_RTD****: Multiplicative coefficient of delay,* coefficiente moltiplicativo ritardo sulla rete; consete di simulare un incremento del tempo di transito sulla rete secondo la formula: *RTD(k)=RTD(k-1)(1+)*

con *RTD(0)=rit\_ack.*

Nel caso RTO fisso la base di calcolo è il valore di RTO impostato nel parametro di configurazione, nel caso di timeout di ritrasmissione adattivo, il calcolo viene eseguito attraverso un apposito algoritmo:

* calcolo di RTO: ***RTO(k)=SRTT(k)+4⋅MSDEV(k)***

• calcolo del valore della media pesata (il valore raccomandato di g è 0.125):

***S R T T ( k ) = S R T T ( k − 1) + g ⋅ E r r ( k )***

• calcolo dell’errore: ***E r r ( k ) = R T T ( k ) − S R T T ( k − 1)***

• calcolo della deviazione media pesata (il valore raccomandato di h è 0.25):

***MSDEV(k)=MSDEV(k−1)+h⋅[Err(k) −MSDEV(k−1)]***

I parametri di configurazione vengono letti tramite la funzione ***read\_config()*** che li scrive sul file di log e li memorizza sulla variabile globale di tipo ***struct config***. Viene controllata l’esistenza del file di configurazione ma non la correttezza formale dei parametri.

***3.6 Thread e trasferimento dati***

Vengono utilizzati due thread, definiti all’interno della procedura ***transmit()*** in *program.h* : uno per l’invio dei dati, procedura *send\_data()*, l’altro per la gestione dei riscontri, procedura *wait\_ack()*.

In ***send\_data()***viene calcolato il numero di sequenza, controllato tramite ***seq\_check()***: se il numero non rientra nella finestra scorrevole, si attende la ricezione del riscontro con numero di sequenza pari alla base della finestra; se il numero è valido, la procedura legge dal file (comandi put/get) o dalla memoria (comando list), copia DATASIZE bytes da inserire nel paccchetto da trasmettere.

Per ogni pacchetto viene creato un ***timeout thread*** che rispedisce periodicamente il pacchetto fino a quando non arriva il riscontro associato oppure viene superato il massimo numero di tentativi. Queste attività vengono eseguite in modo ciclico fino a quando non viene letto tutto il file (terminazione).

In ***wait\_ack()*** si attende la ricezione del riscontro; se arriva l’ack viene eliminato il relativo thread di timeout e viene aggiornata la finestra scorrevole fino al primo numero di sequenza relativo ad un pacchetto che non ha ancora ricevuto riscontro. In caso di RTO adattivo viene ricalcolato il valore del timeout di ritrasmissione. Il ciclo termina quando viene ricevuto un riscontro relativo all’ultimo pacchetto oppure se il file giunge correttamente al processo ricevente.

La procedura ***receive()*** inizializza un buffer per la memorizzazione dei pacchetti e attende l’arrivo di un pacchetto. Viene controllato il numero di sequenza e, se appartiene alla finestra, il pacchetto viene scritto sul buffer. Se il numero di sequenza del frame corrisponde alla base della finestra scorrevole, i pacchetti vengono scritti sul file o in memoria, a seconda del tipo di comando, e si scorre la finestra.

Questo ciclo si ripete fino a quando non viene letto l’ultimo frame dal buffer; viene quindi inviato al mittente il comando END che indica la fine del trasferimento.

***3.7 File log***

Vengono generati due files log distinti, in trasmissione e ricezione, che hanno la stessa struttura: valori dei parametri di configurazione, sequenza delle operazioni effettuate, dati statistici di riepilogo delle elaborazioni (inseriti nel file log, rispettivamente tramite ***transmit()*** e ***receive()***).

I dati statistici riepilogativi nel log di trasmissione hanno la seguente struttura:

==================================================================

TRANSMISSION STATS

total frame :

of which lost at first transmission :

repeated frame :

of which lost :

ack product (remotely) :

of which lost :

frame out of sequence :

duration of the process :

==================================================================

* numero di prime trasmissioni, comprese quelle eventualmente perse ( pari al numero totale dei pacchetti necessari a trasferire il file/contenuto della directory) 🡪 ***a***;
* numero dei pacchetti persi alla prima trasmissione 🡪***b***;
* numero di ritrasmissioni 🡪 ***c***;
* numero di ritrasmissioni perse 🡪 ***d***;
* numero di Ack di ritorno (compresi quelli che si simulano persi) 🡪 ***e***;
* numero di Ack persi in rete ***f***;
* numero di frame fuori sequenza (numero di volte in cui il posizionamento della finestra non consente la trasmissione del successivo pacchetto)
* durata del processo di trasmissione , dato utilizzato nelle analisi prestazionali dell’applicazione (*vedi paragrafo 7. Analisi delle prestazioni*).
* numero dei pacchetti immessi in rete 🡪 ***a+c***;
* numero totale dei pacchetti persi in rete 🡪 ***b+d***;
* numero totale dei pacchetti che arrivano al ricevente 🡪***e= (a+c) ‐ (b+d)***

I dati statistici riepilogativi nel log di ricezione hanno la seguente struttura:

==================================================================

RECEPTION STATS

total received frames/ transmitted :

frames out of sequence :

total number of written frames :

==================================================================

* numero di pacchetti pervenuti al ricevente 🡪 ***g***;
* numero di frame fuori sequenza (numero di volte in cui il posizionamento della finestra non consente l’acquisizione dei dati sul pacchetto) 🡪 ***h***;
* numero totale dei pacchetti scritti sul file 🡪 ***i***;

In caso di corretto trasferimento si ha che ***a=i***, ovvero il numero di pacchetti scritti sul file è pari al numero dei frame trasmessi.

**4. Limitazioni riscontrate**

L’applicazione ritrasmette i pacchetti scartati, tuttavia non verifica che i dati vengano integralmente ricevuti.

I file log permettono di analizzare il flusso delle azioni compiute in trasmissione e ricezione; l’applicazione non gestisce però l’utilizzo dello stesso file log da parte di più processi.

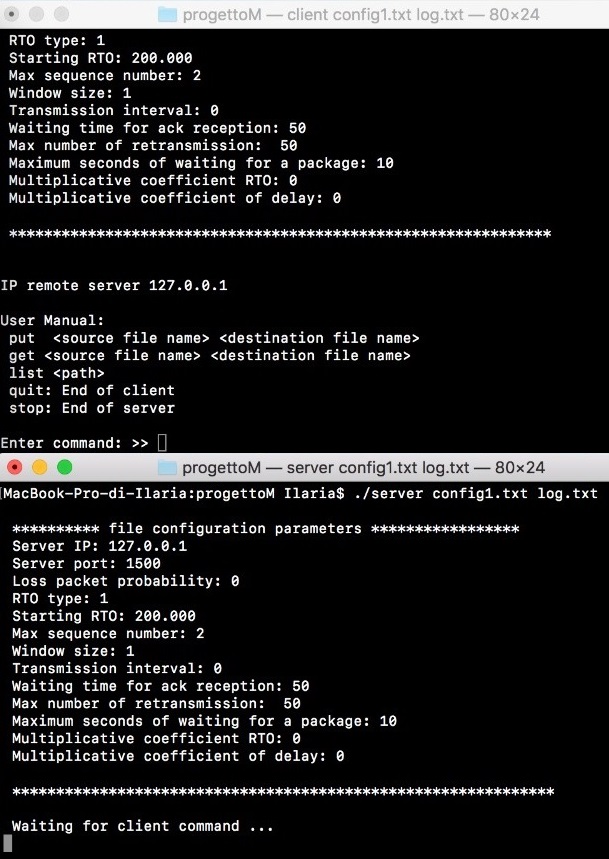
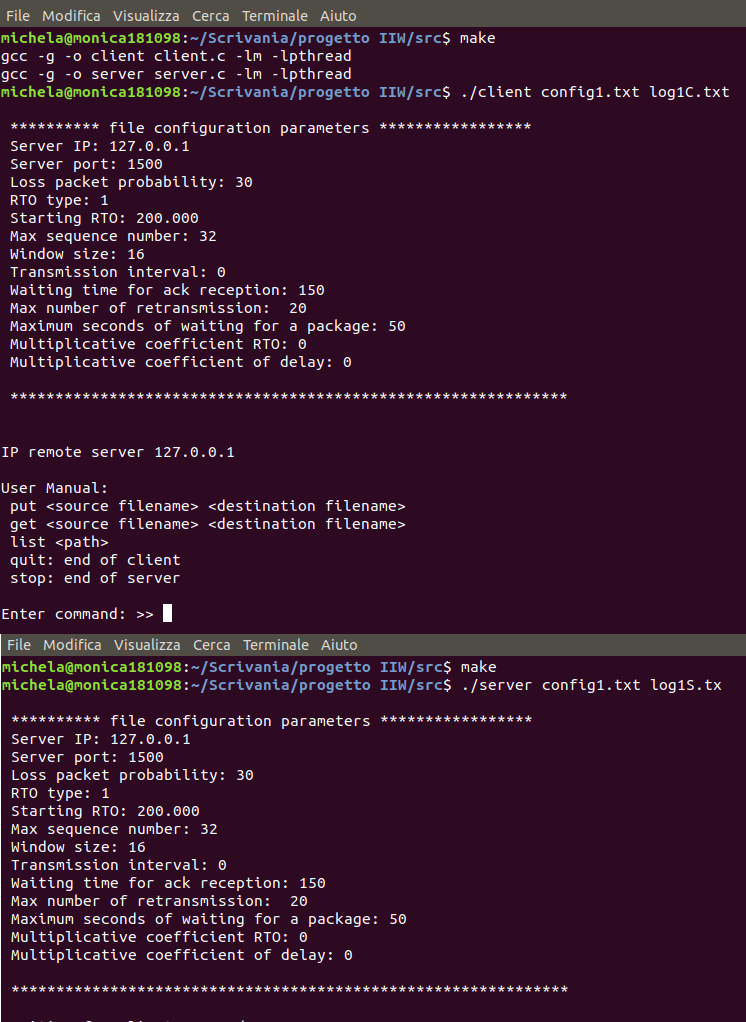
La procedura read\_config verifica se il file di configurazione esiste, ma non controlla che i dati del file siano conformi allo standard.

***5.*** ***Piattaforma software (sviluppo e testing)***

Il funzionamento dell’applicazione è stato testato su:

* Ubuntu 17.10 - SO 64 bit
* Mac-OS

***6. Esempi di funzionamento***



***7. Analisi delle prestazioni***

Per testare l’applicazione, si è scelto di procedere in modo iterativo partendo da un file di configurazione base “config1.txt” e variando la dimensione della finestra di spedizione, la probabilità di perdita dei messaggi e la durata del timeout, nel caso di RTO fisso e adattivo.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | config1 | config2 | config3 | config4 | config5 | config6 | config7 |
| srv\_ip  srv\_port  prob  tip\_RTO  RTO  seq\_max  sws  rit\_send  rit\_ack  max\_tent  TMax  coef\_RTO coef\_RTD | 127.0.0.1  1500  0  1  200  2  1  0  50  50  10  0  0 | 127.0.0.1  1500  0  0  200  8  4  0  50  50  10  0  0 | 127.0.0.1  1500  0  1  200  32  16  0  300  50  10  0  0 | 127.0.0.1  1500  10  0  200  8  4  0  300  50  10  0  0 | 127.0.0.1  1500  10  0  50  32  16  0  300  50  10  0  0 | 127.0.0.1  1500  20  1  50  32  16  0  300  50  10  0  0 | 127.0.0.1  1500  20  1  50  8  4  0  300  50  10  1  0 |
|  | config8 | config9 | config10 | config11 | config12 | config13 | config14 |
| srv\_ip  srv\_port  prob  tip\_RTO  RTO  seq\_max  sws  rit\_send  rit\_ack  max\_tent  TMax  coef\_RTO coef\_RTD | 127.0.0.1  1500  20  0  50  8  4  0  200  50  10  0  0 | 127.0.0.1  1500  0  0  10  32  16  0  200  10  10  0  0 | 127.0.0.1  1500  0  1  10  32  16  0  200  10  10  0  0 | 127.0.0.1  1500  30  1  10  64  32  0  200  10  10  1  0 | 127.0.0.1  1500  30  0  200  8  4  0  200  30  10  0  0 | 127.0.0.1  1500  20  0  200  4  2  0  50  50  10  0  1 | 127.0.0.1  1500  20  1  200  8  4  0  50  50  10  0  4 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | config15 | config16 | config17 | config18 | config19 | config20 | config21 |
| srv\_ip  srv\_port  prob  tip\_RTO  RTO  seq\_max  sws  rit\_send  rit\_ack  max\_tent  TMax  coef\_RTO coef\_RTD | 127.0.0.1  1500  40  1  200  32  16  0  50  10  100  0  1 | 127.0.0.1  1500  40  0  200  32  16  0  50  10  50  0  0 | 127.0.0.1  1500  10  1  50  8  4  0  50  10  30  0  2 | 127.0.0.1  1500  10  0  50  8  4  0  50  10  30  0  2 | 127.0.0.1  1500  0  1  200  32  8  0  50  10  50  1  2 | 127.0.0.1  1500  0  0  200  64  32  0  150  20  50  0  2 | 127.0.0.1  1500  0  1  200  64  32  0  150  20  50  1  2 |

***8. Manuale di installazione***

Prima di eseguire l’applicazione, è necessario generare i file di configurazione secondo il seguente formato (vedi paragrafo *3.5 Parametri di configurazione*):

127.0.0.1 srv\_ip Server IP

1500 srv\_port Server port

30 prob Loss packet probability

1 tip\_RTO RTO type (0=adaptive, 1=fixed)

200 RTO Starting RTO value

32 seq\_max Max sequence number

16 sws Window size

0 rit\_send Transmission interval (default=0)

150 rit\_ack Waiting time for ack reception

20 max\_tent Max number of retransmission

50 TMax Maximum seconds of waiting for a package

0 coef\_RTO Multiplicative coefficient RTO (0=NO, 1=SI)

0 coef\_RTD Multiplicative coefficient of delay

-------------------------------------------------------------------------------

In alternativa è possibile scegliere tra i file di configurazione utilizzati in fase di test, i quali hanno una denominazione standard del tipo <config**N**.txt> con ***N*** numero identificativo. Tali file si trovano nella directory “***configuration file***”.

La cartella “***src***” contiene i codici sorgente: ***program.h***, ***client.c***, ***server.c***, moduli descritti nel paragrafo *2. Architettura del sistema e scelte progettuali*.

Sono inoltre presenti una directory “***clt***” e “***srv***” che contengono rispettivamente gli eseguibili ***client*** e ***server*** ed i file sorgente/destinazione del client e del server.

***8.1 Configurazione del sistema***

Per compilare da linea di comando è possibile procedere aprendo il terminale, posizionandosi nella directory dove è presente il progetto ed eseguendo il comando ***make.*** All’interno del progetto è infatti presente un **Makefile** che compila sia il codice del client che quello del server.

Lanciare gli eseguibili con:

./ client <nome file configurazione> <nome file log>

./ server <nome file configurazione> <nome file log>

È importante utilizzare lo stesso file di configurazione lato client e lato server.

Per il nome dei file log è consigliabile utilizzare la convenzione utilizzata anche in fase di test, ovvero <log**N**.txt> con **N** numero identificativo del file.

L’applicazione è in grado di controllare che il file di origine e destinazione abbiano la stessa estensione.

***8.2 Comandi client***

Dopo aver lanciato l’eseguibile, viene richiesto all’utente di inserire il comando da eseguire (vedi paragrafo *2.1 Istruzioni )* Enter a command: >>

Utilizzare la seguente sintassi, separando i parametri con uno spazio:

*put* <nome file sorgente> <nome file destinazione>

*get* <nome file sorgente> < nome file destinazione>

*list* <path>

*quit*

*stop*

Per le istruzioni ***put*** e ***get***, i file sorgente devono essere già presenti nelle directory relative al client e al server, rispettivamente, nel momento in cui viene lanciato il comando. Per l’istruzione ***list*** “path” è il percorso della directory server (questo comando permette infatti la visualizzazione sul client dei file disponibili sul server).