**Reti di Calcolatori ed Ingegneria del Web**

**A.A. 2019/20**

**Progetto B2: Trasferimento file su UDP**

**Indice**

|  |  |
| --- | --- |
| 1. Panoramica UDP 2. Architettura del sistema e scelte progettuali   2.1 Strutture   1. Implementazione   3.1 Trasmissione affidabile  3.2 Parametri principali   1. Limitazioni riscontrate 2. Piattaforma software (sviluppo e testing) 3. Esempi di funzionamento 4. Analisi delle prestazioni   *al variare della dimensione della finestra di spedizione N,*  *della probabilità di perdita dei messaggi p, e della durata del timeout T (incluso il caso di T adattativo)*   1. Manuale di installazione e configurazione del sistema | p.3  p. 3 |

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. **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 è costituita da tre moduli: in ***program.h*** sono definite le principali strutture dati, le variabili globali e le procedure utilizzate in ***client.c*** e ***server.c*** che contengono i relativi main, procedure e tipi di dati.

Per il funzionamento dell’applicazione sono richiesti un ***file di configurazione***, con i 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.

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;

Generalmente, il client crea un processo figlio che:

***\**** alloca un’area di memoria per i nomi degli oggetti delle directory che riceverà dal server (comando list);

***\**** apre il file destinazione in scrittura (comando get);

***\**** apre il file origine in lettura (comando put).

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 (comando list);

***\**** apre il file origine in lettura (comando get);

***\**** apre il file destinazione in scrittura (comando 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.

**2.1 Strutture**

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

***struct frame***

***struct ack***

***struct timeout\_args***

***struct send\_slot***

***struct send\_file\_args***

***struct config***

***struct rtt\_info***

1. **Implementazione**

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”.

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”.

**3.1 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.

**3.2 Parametri principali**

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

* ***char srv\_ip[50]****: Server IP*
* ***int srv\_port****: Server port*
* ***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
* ***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 prob****: Loss packet probability,* per settare la probabilità di perdita dei pacchetti in rete
* ***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.
* ***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

*Sia RTD(0)=rit\_ack 🡪 RTD(k)=RTD(k-1)(1+)*

Nel caso di timetout 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)]***

**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\_ak()*** 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.

1. **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 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.

Togliere la possibilità di scrivere sul server

Nella fase di test è stato riscontrato che l’applicazione non controlla che il file di origine e destinazione abbiano la stessa estensione; il problema è stato risolto inserendo