**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
3. Implementazione
4. Limitazioni riscontrate
5. Piattaforma software (sviluppo e testing)
6. Esempi di funzionamento
7. 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
2. **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.

Un socket UDP viene identificato univocamente dalla coppia: indirizzo IP, porta di destinazione.

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 di una specifica directory;

• ***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; ***\**** apre il file origine in lettura; ***\**** apre il file destinazione in scrittura. 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.

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 ***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.

Procedura trasmetti()

Procedura ricevi()

**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; è necesssario definire determinati parametri che vengono settati nel file di configurazione.

**Parametri principali**

I ***parametri di configurazione*** sono i seguenti:

* *Server IP*
* *Server port*
* *Loss packet probability*: probabilità di perdita dei pacchetti
* *RTO type*
* *Starting RTO* : valore iniziale del retransmission timeout
* *Max sequence number*: massimo numero di sequenza
* *Window size*: dimensione della finestra
* *Transmission interval*: intervallo di tramissione
* *Waiting time for ack reception*: tempo minimo che deve intercorrere tra la trasmissione e la ricezione del riscontro
* *Max number of retransmission*: numero massimo di ritrasmissioni
* *Maximum seconds of waiting for a package*
* *Multiplicative coefficient RTO*
* *Multiplicative coefficient of delay*

Vengono utilizzati due thread: uno per l’invio dei dati, l’altro per la gestione dei riscontri.