Immagine che contiene computer

Descrizione generata automaticamente

Relazione progetto B2: Trasferimento file su UDP

Ingegneria di Internet e del Web | Settembre 2020

Minut Robert Adrian

Caliandro Pierciro

Falcone Gian Marco

**Indice**

• Traccia del progetto \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3

• Descrizione dell'architettura del sistema e delle scelte progettuali effettuate \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3

• Descrizione dell'implementazione \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3

• Eventuali limitazioni riscontrate \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3

• l'indicazione della piattaforma software usata per lo sviluppo ed il testing del sistema \_\_\_\_\_\_\_\_ 3

• Esempi di funzionamento \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3

• Valutazione delle prestazioni del protocollo \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3

1. al variare della dimensione della finestra di spedizione N \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3
2. al variare della probabilità di perdita dei messaggi p \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3
3. al variare della durata del timeout T \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3

• Manuale per l'installazione, la configurazione e l'esecuzione del sistema \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3

**Traccia del progetto**

Lo scopo de progetto è quello di progettare ed implementare in linguaggio C usando l’API del socket di Berkeley un’applicazione client-server per il trasferimento di file che impieghi il servizio di rete senza connessione (socket tipo SOCK\_DGRAM, ovvero UDP come protocollo di strato di trasporto).

Il software deve permettere:

* Connessione client-server senza autenticazione;
* La visualizzazione sul client dei file disponibili sul server (comando list);
* Il download di un file dal server (comando get);
* L’upload di un file sul server (comando put);
* Il trasferimento file in modo affidabile.

La comunicazione tra client e server deve avvenire tramite un opportuno protocollo. Il protocollo di comunicazione deve prevedere lo scambio di due tipi di messaggi:

* messaggi di comando: vengono inviati dal client al server per richiedere l’esecuzione delle diverse operazioni;
* messaggi di risposta: vengono inviati dal server al client in risposta ad un comando con l’esito dell’operazione.

**Funzionalità del server**

Il server, di tipo concorrente, deve fornire le seguenti funzionalità:

* L’invio del messaggio di risposta al comando list al client richiedente; il messaggio di risposta contiene la filelist, ovvero la lista dei nomi dei file disponibili per la condivisione;
* L’invio del messaggio di risposta al comando get contenente il file richiesto, se presente, od un opportuno messaggio di errore;
* La ricezione di un messaggio put contenente il file da caricare sul server e l’invio di un messaggio di risposta con l’esito dell’operazione.

**Funzionalità del client**

I client, di tipo concorrente, devono fornire le seguenti funzionalità:

* L’invio del messaggio list per richiedere la lista dei nomi dei file disponibili;
* L’invio del messaggio get per ottenere un file
* La ricezione di un file richiesta tramite il messaggio di get o la gestione dell’eventuale errore
* L’invio del messaggio put per effettuare l’upload di un file sul server e la ricezione del messaggio di risposta con l’esito dell’operazione.

**Trasmissione affidabile**

Lo scambio di messaggi avviene usando un servizio di comunicazione non affidabile. Al fine di garantire la corretta spedizione/ricezione dei messaggi e dei file sia i client che il server implementano a livello applicativo il protocollo di comunicazione affidabile di TCP con dimensione della finestra di spedizione fissa N (cfr. Kurose & Ross “Reti di Calcolatori e Internet”, 7° Edizione).

Per simulare la perdita dei messaggi in rete (evento alquanto improbabile in una rete locale per non parlare di quando client e server sono eseguiti sullo stesso host), si assume che ogni messaggio sia scartato dal mittente con probabilità p.

La dimensione della finestra di spedizione N, la probabilità di perdita dei messaggi p sono configurabili ed uguali per tutti i processi. I client ed il server devono essere eseguiti nello spazio utente senza richiedere privilegi di root. Il server deve essere in ascolto su una porta di default (configurabile).

Opzionale: Implementare anche il controllo di flusso e il controllo di congestione.

Riguardo l'implementazione del server abbiamo deciso di attuare una soluzione ibrida a processi e thread, così da poter sfruttare i vantaggi di ognuna delle due tecnologie.

Per rendere il nostro server più robusto abbiamo utilizzato i processi, così che se un determinato processo fosse andato giù gli altri avrebbero potuto continuare a funzionare senza problemi. In particolare, il nostro server utilizza il pre-forking, istanziando un numero fisso di processi all'avvio, in modo da non doversene occupare nel momento in cui arriva una nuova richiesta ci connessione, cosa che avrebbe altrimenti rallentato il three-way handshake iniziale.

Per indirizzare ogni richiesta verso un diverso processo è stata creata nel processo padre una socket di ascolto principale il quale, grazie alla funzione select(), si rende conto della nuova connessione e avvisa con un segnale un ben determinato processo figlio incaricato di gestirla.

A questo punto il processo figlio istanzia un nuovo thread che si occupa di mantenere attiva la connessione con il client e di rispondere ai comandi che quest'ultimo invia. La scelta di far gestire le connessioni a thread piuttosto che a nuovi processi è giustificata dal fatto che creare un nuovo thread è un’operazione meno dispendiosa.