RELAZIONE DEL PROGETTO DI RETI
"TRASPORTO AFFIDABILE MULTIPERCORSO"

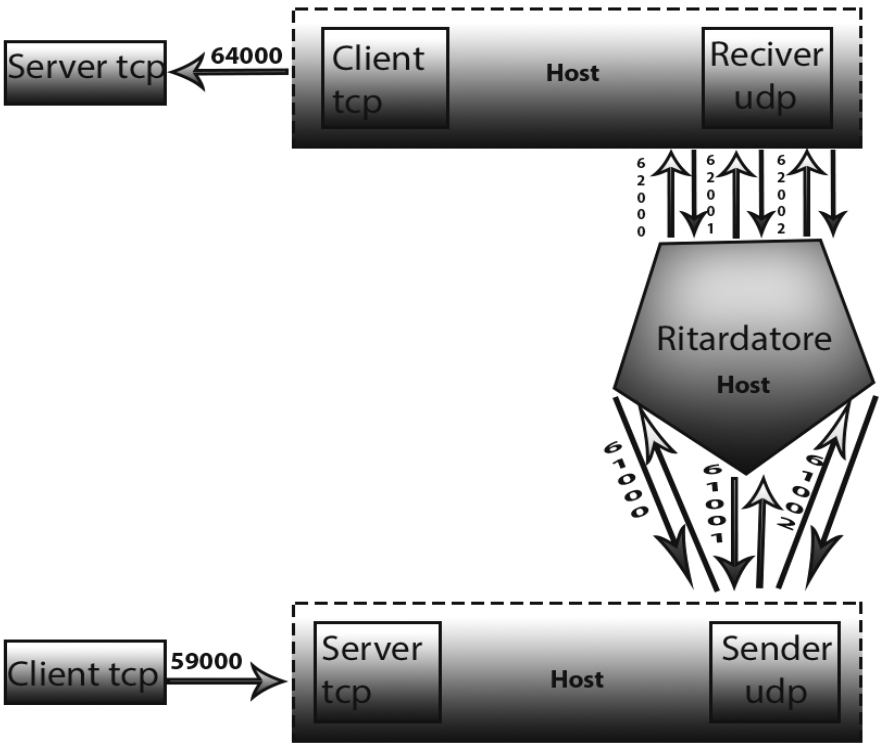
Partecipanti:

Gianmarco Ascenzo 0000604925
Giovanni Ricucci 0000415377

Introduzione Al Progetto:

L'obbiettivo è la realizzazione di un sistema affidabile che mantenga integri i dati spediti tra due processi (Sender e Reciver) che lavorano su due host differenti, nonostante la perdita di pacchetti da parte del Ritardatore eseguito su un altro host. Il Sender, che lavora come client TCP, si avvia sulla porta 59000. Dall'altra parte il Reciver, che lavora come Server TCP, è in ascolto sulla porta 64000 dal ProxyReciver.
Abbiamo deciso di implementare le funzioni sul file Includi.h che contiene sia le funzioni per il ProxySender sia quelle del ProxyReciver.

Dimostrazione grafica del flusso di dati



INFORMAZIONI SUL CONTENUTO

Il progetto è costituito dai seguenti file:

1. **Proxysender.c**
2. **Proxyreciver.c**
3. **Includi.h**
4. **Makefile**

L'esecuzione avviene su due host distinti, i comandi sono i seguenti:
./Proxysender (IP Ritardatore)
./Proxyreciver (IP Ritardatore)

DESCRIZIONE DELLE FUNZIONALITA'

Struttura di memorizzazione dei pacchetti:

- **Campo id** → l'indice del pacchetto;
- **Campo b** → Contenente la lettera B per poter ricevere e spedire dati attraverso il Ritardatore;
- **Campo body** → Contenuto del pacchetto;
- **Campo dim** → Contenente la dimensione del pacchetto.

MODULO ProxySender.c:

Contiene il main() del programma. Si occupa dell'inizializzazione e della configurazione del socket per l'ascolto di eventuali richieste da parte del Client TCP sulla porta nota 59000 e della spedizione dei dati in UDP al Ritardatore. L'implementazione delll' I/0 MULTIPLEXING e della funzione select garantisce il controllo sul socket attraverso uno dei due file descriptor utilizzati per i due tipi di socket (TCP e UDP); attraverso le macro cancello e reimposto i settaggi dei file descriptor nel caso in cui per esempio il Client TCP si disconnettesse, di modo da essere sempre in ascolto sul lato "Server" e poter ricevere altri dati. Nel momento in cui si è instaurata la connessione il ProxySender si occupa di immagazzinare all'interno della struttura precedentemente illustrata il contenuto spedito dal Client TCP e di spedire subito dopo tramite udp al Ritardatore sulle tre porte note.Per assicurare l'integrità del flusso spedito dal Client TCP, il ProxySender si occuperà di ricevere indietro (dal ProxyReciver) i pacchetti spediti, rimuovendoli dalla lista e rispedendo i restanti alla scadenza di un TimeOut, anche se il Client TCP si dovesse disconnettere e se avvenisse un'altra connessione da parte di un altro Client, fin quando la lista non risulta vuota e i dati spediti correttamente al ProxyReciver.

MODULO ProxyReciver.c:

Anch'esso contiene il main() del programma, ma si occupa di ricevere dal Ritardatore i dati UDP spediti da ProxySender. Al momento della ricezione si occupa di spedire l'id del pacchetto indietro al Ritardatore su una delle tre porte note e di contrassegnarlo (modificando il tipo 'B') come già spedito (dato che eliminarli mi comporterebbe la perdita dei dati da spedire al Server TCP in ascolto). Successivamente vi è l' utilizzo di un contatore che effettua una ricerca ordinata dei pacchetti ricevuti, passando il puntatore risultante alla funzione "inviatcp", che si occupa di spedire i dati al Server. Inoltre per evitare doppie scritture al Server anche questi pacchetti vengono contrassegnati come spediti. Viene gestita anche la dimensione del pacchetto spedito al Server attraverso il campo dim della struttura cosicchè la dimensione del pacchetto spedito sia uguale a quello ricevuto. Anche questo modulo implementa l'I/0 MULTIPLEXING attraverso la funzione select per potersi mettere in ascolto quando il ritardatore ha dati da spedire.

MODULO includi.h:

Contiene le funzioni usate:

- **la funzione di inizializzazione:** attraverso un intero passato, inizializza o un socket tcp o uno udp facendo la bind su una porta determinata;
- **la funzione di ascolto** sul socket tcp con successiva accept;
- **la funzione di ricezione** di dati dal client tcp,che dopo aver ricevuto i dati,li inserisce all'interno del body del pacchetto incrementando l'id per ognuno di questi e spedendo i pacchetti appena creati in udp al ritardatore.
- **la funzione di eliminazione** del pacchetto che prendendo in input il buffer dei dati udp ricevuti e la lista effettua una ricerca dei pacchetti, se trova una corrispondenza fa l'eliminazione del pacchetto e restituisce la lista modificata.
- **la funzione di ordinamento** è da parte del proxyreciver. Predendo in input l'elemento sentinella della lista fa una ricerca ordinata di pacchetti passando il puntatore alla spedizione dei dati al server tcp.
- **la funzione della scrittura:** Spedisce i pacchetti in modo ordinato al server tcp contrassegnando quelli già spediti.

Tecniche di Progettazione:

L'elaborato è stato compilato con questi parametri:
-ansi -Wall -Wunused -pedantic

Il progetto è bastato sull I/O MULTIPLEXING, utilizzando tecniche di progettazione basate su I/O sincrono . In particolare:

1. Sono stati usati i socket unix anonimi , cioè creati con il comando socket (AF_INET, ,);
2. Non sono state utilizzate tecniche di compressione dei dati.
3. Non sono stati utilizzati pthread per evitare sovraccarico della CPU.
4. Non sono state utilizzate tecniche di progettazione asincrone,ma implementata la funzione select per l'attesa di I/O

Demo:

I due ProxySender e ProxyReciver sono stati provati tra due macchine (o anche in locale) collegate alla rete. E' stato testato il metodo di spedizione (file/testo) con netcat. I due moduli si metteranno rispettivamente in ascolto da lato server/client tcp/udp con le rispettive chiamate:

1. ./Proxysender 127.0.0.1 → Indirizzo IP Ritardatore;

2. netcat 127.0.0.1 59000 → Client tcp indirizzo locale e porta nota;

3. netcat -l 127.0.0.1 64000 → Server tcp indirizzo locale e porta nota;

4. ./Proxyreciver 127.0.0.1 → Indirizzo IP Ritardatore.

Test Effettuati:

Sono state testate le performance del programma,attraverso l'applicazione iperf,che si mette in ascolto come server TCP,qui di seguito i dati restituiti:

Interval	Transfer	Bandwidth
0.0 – 9.7sec	4.31MBytes	3.73Mbits/sec
0.0 – 13.2sec	9.10MBytes	5.77Mbits/sec
0.0 – 15.1sec	14.0Mbytes	7.52Mbits/sec