PROGRAMMAZIONE DI RETI
REPORT ASSIGNMENT 1

Lorenzo Casini Sophia Fantoni

5 ottobre 2018

Indice

1 Task 1 **3**
1.1 Descrizione degli step 3
1.2 Avvio dell'applicazione lato Client 3
1.3 Avvio dell'applicazione lato Server 4
1.4 Note sul funzionamento di netcat 5

2 Task 2 **5**
2.1 Strategia di risoluzione client/server 5
2.2 Comportamento dettagliato client/server 5
2.3 Test 6

3 Task 3 **7**
3.1 Risposte alle domande 7
3.2 Motivazioni 7

1 Task 1

1.1 Descrizione degli step

Il primo passo che abbiamo eseguito per realizzare il primo task è la comprensione totale delle funzionalità di netcat. Digitando su terminale man nc è possibile leggere e comprendere le funzionalità che il comando ci mette a disposizione, digitando successivamente netcat (o nc) -h analizziamo solo le opzioni che ci interessa utilizzare.
Abbiamo deciso di sperimentare le funzionalità di netcat sia nel caso in cui Client e Server stiano sullo stesso host, sia nel caso in cui appartengano a due host diversi.

1.2 Avvio dell'applicazione lato Client

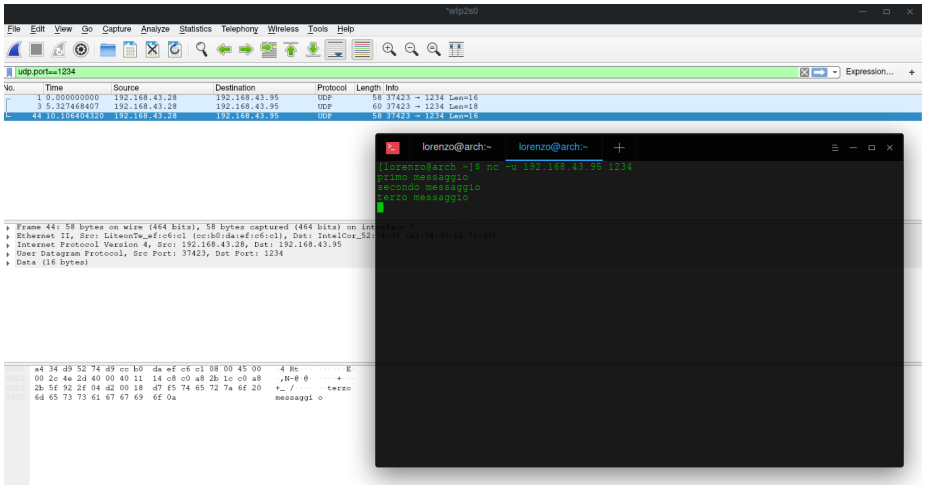
netcat -u localhost 1234
nc -u 192.168.43.95 1234

Il primo scopo è quello di creare un Client mediante la suite nc. Tra le opzioni scegliamo di specificare:

-u per dire esplicitamente di voler comunicare sfruttando il protocollo UDP,

hostname: nel protocollo UDP è la stringa a 4byte che identifica l'indirizzo IP (localhost-> variabile riconosciuta da tutti, in alternativa avremmo potuto inserire l'indirizzo esplicito 127.0.0.1 Loopback sulla macchina locale, se voglio collegarmi ad una macchina remota inserisco l'IP del calcolatore).

port 1234: il numero di tale porta è scelta in modo arbitrario, l'unica cosa di cui bisogna preoccuparsi è che quella porta sia libera, è necessario non utilizzare le porte già assegnate ad altri protocolli (es. 80, 21, 22).



1.3 Avvio dell'applicazione lato Server

netcat -u -l 1234

Successivamente istanziamo il Server, in questo caso non dobbiamo specificare l'hostname, per questo motivo non dovremo variare la riga di comando nel caso in cui gli host appartengano allo stesso o a due host differenti.

-u per far in modo che ascolti solo i pacchetti di tipo UDP.

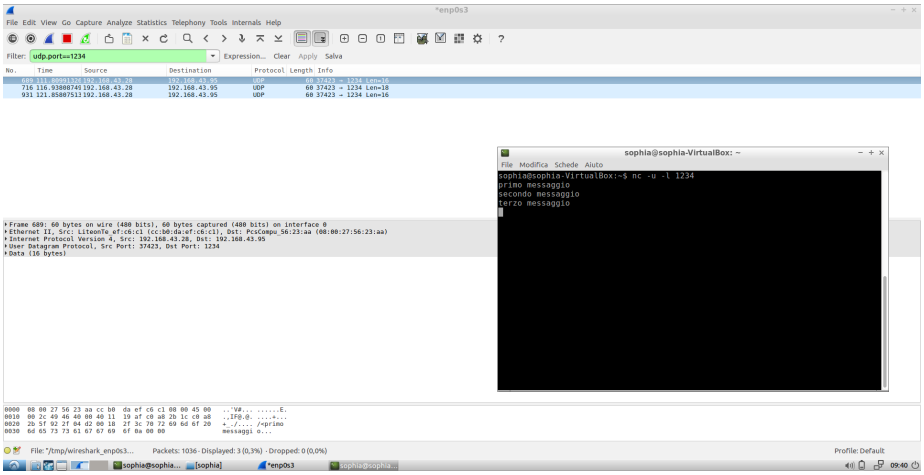
-l per dire che deve ascoltare (listen) qualsiasi connessione (inviato da un qualsiasi host) in entrata.

1234 è la porta su cui deve mettersi in ascolto.

Per analizzare il traffico che viene generato dalla loro connessione utilizziamo Wireshark.

Nel primo caso, quello in cui il Server e il Client appartengano allo stesso host, mi metto in ascolto sull'interfaccia di Loopback, e avvio la cattura dei pacchetti. Nel secondo filtro solo i dati che passano sull'interfaccia enp0s3.
Per prendere effettivamente il solo traffico che client e server si scambiano sulla porta 1234 con protocollo UDP devo inserire dei filtri. udp.port==1234 tale filtro mi permette di escludere tutte le connessioni che non siano locali di tipo UDP che girano sulla porta 1234.

Il client invia il pacchetto dalla porta 37423, tale porta viene scelta in modo arbitrario al momento dell'apertura del client; se eseguo uno stesso client più volte tale porta può cambiare.



1.4 Note sul funzionamento di netcat

Dai test effettuati (utilizzando i nostri comandi) con netcat abbiamo compreso il suo funzionamento. Per quanto riguarda la parte UDP abbiamo notato che una volta che un Client invia un pacchetto al Server, quest'ultimo rimane in ascolto solo dei pacchetti ricevuti da quel specifico client, perciò altri client vengono ignorati. Quando terminiamo l'esecuzione del client specifico è necessario terminare anche il server in quanto esso non è in grado di accettare pacchetti da altri client. Infine, una volta instaurata la connessione, appare chiaro che netcat non distingue lato client dal lato server, inquanto è possibile inviare messaggi anche dal Server verso il Client.

2 Task 2

2.1 Strategia di risoluzione client/server

Per realizzare il secondo task è stato necessario innanzitutto comprendere le funzionalità di nc, ciò è stato immediato dopo la realizzazione del task 1, e successivamente analizzare e modificare il codice consegnatoci durante le ore di laboratorio.

2.2 Comportamento dettagliato client/server

Il Client e il Server, per poter essere avviati, necessitano l'inserimento in input dell'indirizzo IP e del numero di porta. Il Server per poter inviare una risposta al Client, estrae da ogni datagramma ricevuto l'indirizzo IP e il numero di porta del mittente.

Il file udpclient.c crea la socket con i parametri presi al momento dell'avvio (IP_SERVER, PORTA).

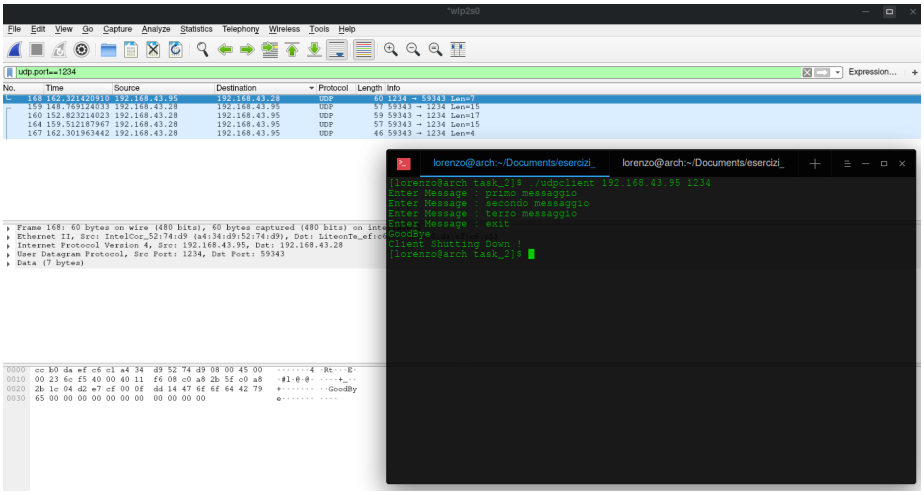
Il Client è in grado di inviare messaggi al Server, nel caso in cui la stringa inviata dal Client coincida con "exit", esso si metterà in attesa della risposta da parte del Server, se tale risposta risulta essere "Goodbye" la socket verrà chiusa.

Il Server crea la socket, legge i datagram ricevuti, controlla se contengono la stringa "exit", in tal caso invia come risposta al Client la stringa "Goodbye", altrimenti rimane in attesa di ulteriori messaggi. Il Server dopo la richiesta di chiusara della connnessione da parte del Client, rimane in esecuzione e non chiude la propria socket.

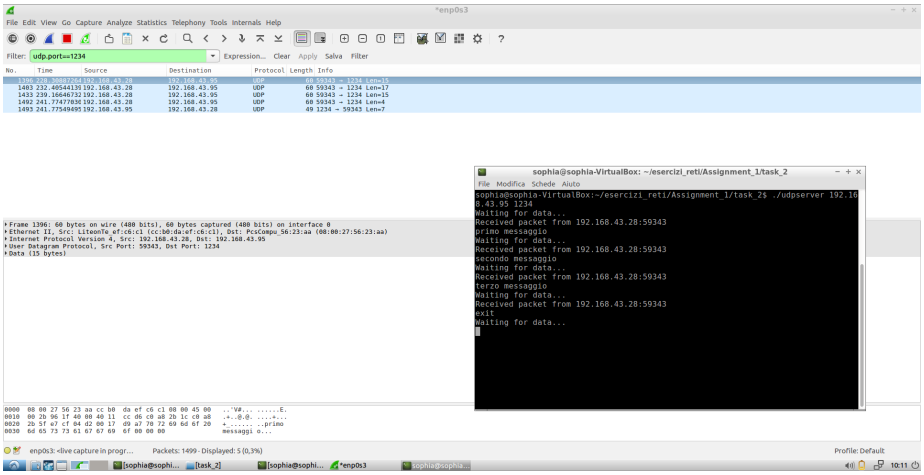
2.3 Test

Abbiamo sperimentato il codice avviando Client e Server su due host differenti.

client:



server:



3 Task 3

3.1 Risposte alle domande

Per effettuare il terzo task abbiamo utilizzato il codice scritto da noi e utilizzato nel task 2. Avviando due client che comunicano con lo stesso server, le operazioni svolte dal server sembrerebbero concorrenti perché le poche operazioni che esegue impiegano un lasso di tempo molto breve.
Successivamente, per simulare operazioni più complesse, decidiamo di inserire una sleep prima della funzione recivefrom. Questa operazione risulta più impegnativa dal punto di vista del tempo all'interno del codice. Eseguendo questa prova abbiamo osservato che il protocollo esegue in realtà le operazioni in modo sequenziale, in quanto queste vengono eseguite, quindi stampate a video, intervallate dal periodo di tempo della sleep inserita.

3.2 Motivazioni

Avendo implementato il server senza l'utilizzo di thread, esso è in grado di gestire un pacchetto alla volta completando le operazioni richieste.

Una volta completate tali operazioni il Server sarà in grado di ricevere un nuovo pacchetto e riavviare le operazioni su di esso.

Quindi, per ottenere un comportamento concorrente, è necessario creare lato server un thread in grado di gestire tutte le richieste in ingresso. Esso si preoccuperà di creare un nuovo thread per ogni nuova connessione richiesta. Il server effettua un controllo su ogni pacchetto, verificandone ip e porta e lo inoltrerà al thread che sta gestendo tale connessione, in caso il thread non esista verrà creato. Ovviamente questa tipo di gestione implica che il numero di connessioni accettate dal server venga controllato e/o limitato per evitare di incorrere in vari tipi di errori.