

Relazione primo assegnamento

Scucchia Matteo, Ziani Andrea

Agosto 2018

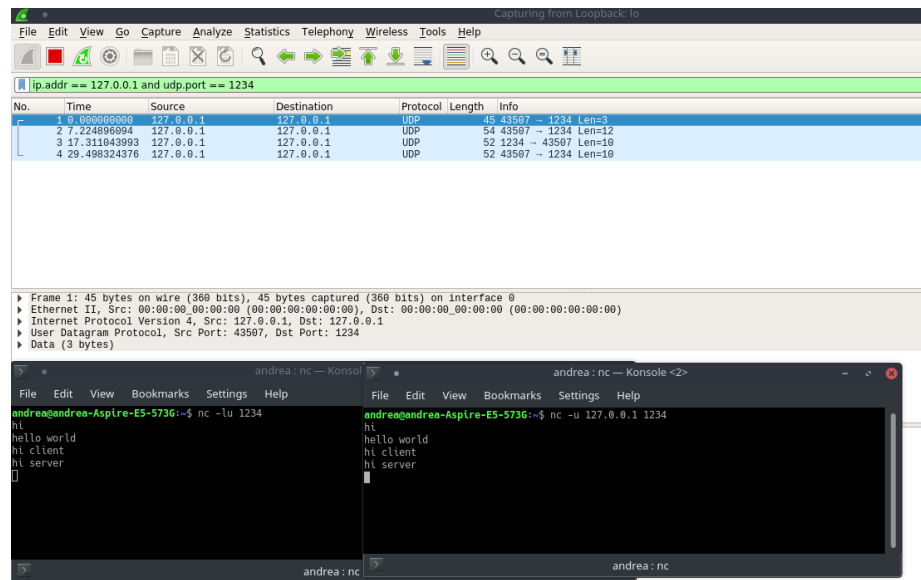
1 Prima richiesta

Per compiere la prima richiesta è stato sufficiente consultare il manuale della bash per comprendere il funzionamento di Netcat. Per svolgere l'esercizio è bastato utilizzare il flag "-u" per utilizzare il protocollo UDP e seguire il manuale alla sezione "Client/Server Model". In questa parte viene spiegato che, per costruire un semplice modello client/server, basta eseguire i comandi "nc -l 1234" in una prima console e "nc 127.0.0.1 1234" in una seconda console.

Il primo comando citato mette in ascolto la console sulla socket port 1234, il secondo comando specifica l'ip e la socket port ai quali la console deve inviare i pacchetti. Eseguiti i due comandi aggiungendo il flag "-u" ad entrambi la connessione viene stabilita la connessione con tra client e server con protocollo UDP.

Una volta stabilita la connessione Netcat non si preoccupa più di quale sia il lato client e quale sia il lato server, infatti tutto ciò che viene scritto sulla prima console viene concatenato a ciò che viene scritto sulla seconda console.

Per catturare il traffico generato dal modello client/server di Netcat è bastato eseguire la cattura Wireshark in "Loopback interface" e filtrando la cattura con "ip.addr == 127.0.0.1 and udp.port == 1234".



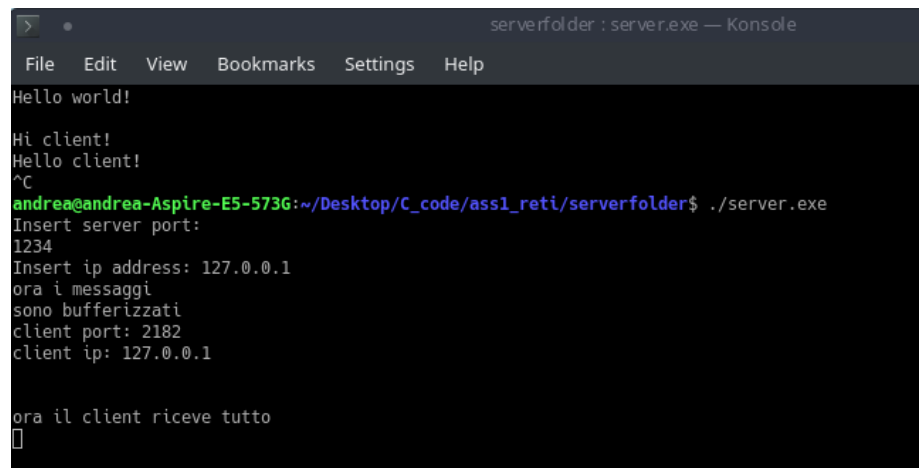
Screen della cattura Wireshark sul funzionamento di Netcat

2 Seconda Richiesta

Per compiere la seconda richiesta per prima cosa abbiamo analizzato l'esercizio visto in laboratorio e, una volta comprese le funzioni utilizzate, abbiamo riadattato il codice per svolgere la richiesta di emulare il comportamento di Netcat.

Per farlo al meglio abbiamo utilizzato la programmazione concorrente facendo uso dei pthread, con annesse funzioni di mutua esclusione. In una struttura di utilità sono state racchiuse tutte le variabili condivise dai thread, mentre abbiamo lasciato variabili globali solamente le mutex e le condition variable.

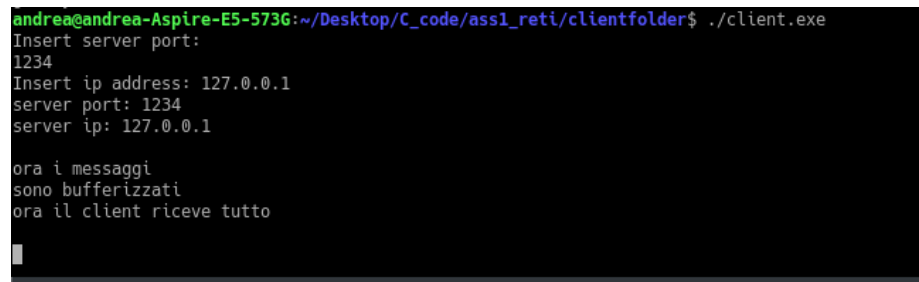
Abbiamo inoltre visto che Netcat bufferizza i messaggi che vengono inviati dal server prima che esso sappia l'indirizzo IP del client. Abbiamo quindi anche noi creato una coda nella quale memorizziamo temporaneamente i messaggi per poi inviarli tutti al primo messaggio scritto dopo che si è reso noto l'IP del client.



```
serverfolder : server.exe — Konsole
File Edit View Bookmarks Settings Help
Hello world!
Hi client!
Hello client!
^C
andrea@andrea-Aspire-E5-5736:~/Desktop/C_code/ass1_reti/serverfolder$ ./server.exe
Insert server port:
1234
Insert ip address: 127.0.0.1
ora i messaggi
sono bufferizzati
client port: 2182
client ip: 127.0.0.1

ora il client riceve tutto
█
```

Screen del funzionamento del buffer del server



```
andrea@andrea-Aspire-E5-5736:~/Desktop/C_code/ass1_reti/clientfolder$ ./client.exe
Insert server port:
1234
Insert ip address: 127.0.0.1
server port: 1234
server ip: 127.0.0.1

ora i messaggi
sono bufferizzati
ora il client riceve tutto
█
```

Screen del funzionamento della ricezione dei messaggi dal buffer.

Per prima cosa il server chiede all'utente su quale port si deve mettere in ascolto e quale sia il suo IP; quindi imposta la famiglia degli indirizzi, il protocollo utilizzato (in questo caso UDP) e la porta sulla quale mettersi in ascolto. Fatto questo il server divide il suo comportamento in tre thread.

Un thread si mette in attesa di ricevere pacchetti sulla port specificata e ogni volta che riceve un messaggio lo stampa su console insieme all'IP e alla socket port del client dal quale il messaggio è stato inviato.

Alla ricezione del messaggio "exit" il server non stampa nulla a console, ma risveglia un thread che era in attesa che manda al client un messaggio "goodbye". Il thread della ricezione dei messaggi continua a svolgere le sue funzioni, mentre il thread che manda il messaggio di goodbye una volta spedito il messaggio si mette in attesa.

Un terzo thread attende che l'utente inserisca i caratteri a console e manda

il messaggio al client. Si considera un singolo messaggio composto da tutti i caratteri fino all'andata a capo.

In Netcat se un secondo client si connette al server, il server non considera i suoi messaggi. Inoltre il secondo client non riceve nemmeno nessun messaggio da parte del server.

Per emulare questo comportamento, salviamo il primo indirizzo con il quale il server comunica e se gli indirizzi successivi non coincidono con il primo indirizzo, lasciamo perdere i messaggi.

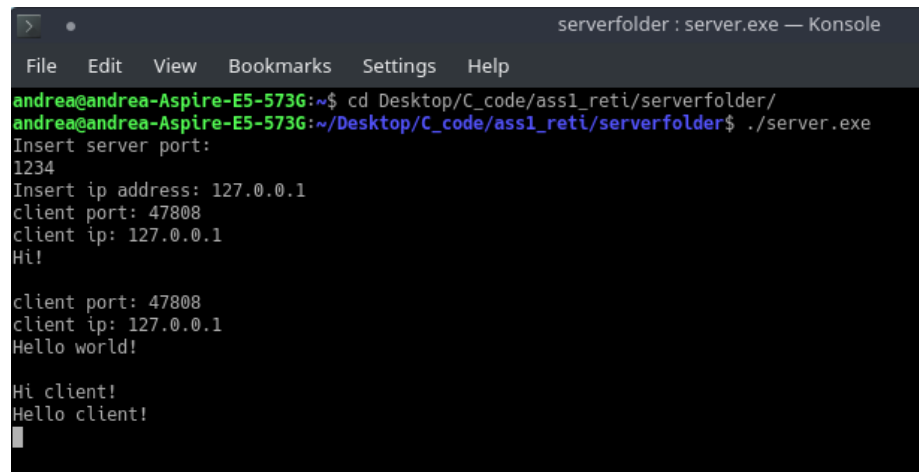
Dall'altro lato il client chiede all'utente l'IP address e la port verso il quale deve inviare i pacchetti; quindi imposta la famiglia degli indirizzi, il protocollo utilizzato (in questo caso UDP) e la porta sulla quale inviare i pacchetti.

Fatto questo il client divide il suo comportamento in due thread.

Il primo attende che l'utente inserisca dei caratteri da console per mandarli al server.

Il secondo thread invece si occupa di stampare i messaggi ricevuti dal server. Per interrompere la connessione tra client e server basterà digitare la stringa "exit" premendo invio e attendere il messaggio "goodbye" dal server.

Il client, ricevuto il messaggio di terminazione dal server, visualizza il contenuto del pacchetto ricevuto su console e termina la connessione.

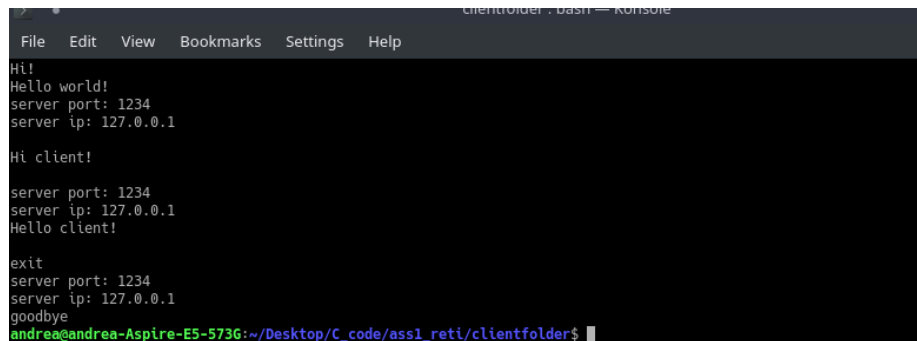


```
serverfolder : server.exe — Konsole
File Edit View Bookmarks Settings Help
andrea@andrea-Aspire-E5-573G:~$ cd Desktop/C_code/ass1_reti/serverfolder/
andrea@andrea-Aspire-E5-573G:~/Desktop/C_code/ass1_reti/serverfolder$ ./server.exe
Insert server port:
1234
Insert ip address: 127.0.0.1
client port: 47808
client ip: 127.0.0.1
Hi!

client port: 47808
client ip: 127.0.0.1
Hello world!

Hi client!
Hello client!
█
```

Screen del funzionamento del server



```
clientfolder - Bash -- Konsole
File Edit View Bookmarks Settings Help
Hi!
Hello world!
server port: 1234
server ip: 127.0.0.1

Hi client!

server port: 1234
server ip: 127.0.0.1
Hello client!

exit
server port: 1234
server ip: 127.0.0.1
goodbye
andrea@andrea-Aspire-E5-5736:~/Desktop/C_code/ass1_reti/clientfolder$
```

Screen del funzionamento del client

3 Terza Richiesta

Per rispondere alla terza richiesta abbiamo preso in esame il codice visto in classe, dato che il nostro modello client server emulando Netcat non accetta i messaggi provenienti da un secondo client.

In un primo luogo il comportamento del server del codice visto in classe potrebbe sembrare concorrente, questo perché le operazioni che deve svolgere non richiedono molte computazioni. Il comportamento reale è però sequenziale, infatti se si prova a inserire la funzione "sleep()" nella porzione di codice che processa i messaggi in entrata, il comportamento del server e del client è chiaramente sequenziale in quanto è possibile notare le stampe su console dei messaggi in entrata uno alla volta intervallate dal periodo di tempo nel quale il server è in pausa.