# Programmazione distribuita I

(01NVWOV)

**AA 2016-2017, Esercitazione di laboratorio n. 2**

NB: In ambiente linux, per questo corso il programma “wireshark” e “tshark” sono configurati per poter catturare il traffico sulle varie interfacce di rete anche se il programma e' lanciato da utente normale (non super-user root). E' possibile utilizzarlo sull'interfaccia di loopback (“lo”) per verificare i dati contenuti nei pacchetti inviati dalle proprie applicazioni di test. Gli studenti del corso sono invitati a testare il funzionamento delle loro applicazioni anche utilizzando questo strumento.

L'utilizzo limitato all'interfaccia “lo” e' completamente sicuro. Si ricorda pero' che catturare il traffico su **altre** interfacce su cui transita **traffico non proprio**, in particolare credenziali di autenticazione (es. passwords o hash di tali dati) al fine di effettuare accessi impropri o non autorizzati e' un reato con conseguenze di natura civile e PENALE. E' quindi assolutamente vietato tale uso. Chi fosse sorpreso a catturare o tentare di catturare passwords o simili verra' immediatamente allontanato dal laboratorio e deferito alle apposite commissioni disciplinari del Politecnico, oltre a poter subire eventuali sanzioni di natura amministrativa e penale a norma di legge.

## Esercizio 2.1 (client UDP perseverante)

Modificare il client UDP dell’esercizio 1.4 in modo che – se non riceve qualsiasi risposta dal server entro 3 secondi – ritrasmetta la richiesta (fino ad un massimo di 5 volte) e quindi termini indicando se ha ricevuto risposta o meno.

Effettuare le stesse prove dell’esercizio 1.4.

## Esercizio 2.2 (server UDP limitante)

Modificare il server UDP dell’esercizio 1.4 in modo che invii risposta ad un client solo se questi non ha effettuato più di tre richieste dallo stesso indirizzo IP (dal momento dell’attivazione del server). Il server deve essere in grado di riconoscere gli ultimi 10 client che hanno fatto richiesta.

Provare quindi ad attivare verso questo server quattro volte il client dell’esercizio 1.4.

Provare infine ad attivare verso questo server due client posti su due nodi di rete diversi, in alternanza tra loro, quattro volte per ciascun client.

## Esercizio 2.3 (server TCP iterativo)

Sviluppare un server TCP (in ascolto sulla porta specificata come primo parametro sulla riga di comando) che accetti richieste di trasferimento file da client ed invii il file richiesto.

Sviluppare un client che possa collegarsi ad un server TCP (all’indirizzo e porta specificati come primo e secondo parametro sulla riga di comando) per richiedere dei file e memorizzarli localmente. I nomi dei file da richiedere vengono forniti su standard input, uno per riga. Ogni file richiesto deve essere salvato localmente e deve essere stampato su standard output un messaggio circa l’avvenuto trasferimento, con nome, dimensione del file e timestamp di ultima modifica.

Il protocollo per il trasferimento del file funziona come segue: per richiedere un file il client invia al server i tre caratteri ASCII “GET” seguito dal carattere ASCII dello spazio e dai caratteri ASCII del nome del file, terminati da CR LF (*carriage return* e *line feed*, cioè due bytes corrispondenti ai valori 0x0d 0x0a in esadecimale, ossia `'\r'` e `'\n'` in notazione C, sempre senza spazi):

| G | E | T | | …filename… | CR | LF |
|---|---|---|---|---|---|---|

(Nota: il comando include un totale di 6 caratteri più quelli del nome del file)

Il server risponde inviando:

| + | O | K | CR | LF | B1 | B2 | B3 | B4 | T1 | T2 | T3 | T4 | File content…… |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Notare che il messaggio è composto da 5 caratteri, seguiti dal numero di byte del file richiesto (un intero senza segno su 32 bit in network byte order - bytes B1 B2 B3 B4 nella figura), e quindi dal timestamp dell'ultima modifica (Unix time, cioè numero di secondi dall'inizio dell' "epoca"), rappresentato come un intero senza segno su 32 bit in network byte order (bytes T1 T2 T3 T4 nella figura), e infine dai byte del file in oggetto.

Per ottenere il timestamp dell'ultima modifica al file, si faccia riferimento alle chiamate di sistema *stat* o *fstat*.

Il client può richiedere più file inviando più comandi GET. Quando intende terminare la comunicazione invia:

| Q | U | I | T | CR | LF |
|---|---|---|---|---|---|

(6 caratteri) e chiude il canale.

In caso di errore (es. comando illegale, file inesistente) il server risponde sempre con

| - | E | R | R | CR | LF |
|---|---|---|---|---|---|

(6 caratteri) e quindi chiude il canale col client.

Si faccia attenzione a porre l'eseguibile del client in una directory DIVERSA da quella dove gira il server. Si chiami questa cartella con lo stesso nome del programma client (questo per evitare che, con prove sullo stesso PC, salvando il file si sovrascriva lo stesso file che viene letto dal server).

Provare a collegare il proprio client con il server incluso nel materiale fornito per il laboratorio, ed il client incluso nel materiale fornito con il proprio server (notare che i files eseguibili sono forniti per architetture sia a 32 bit sia a 64 bit. I files con suffisso _32 sono compilato per girare su sistemi Linux a 32 bit, quelli senza suffisso per sistemi a 64 bit. I computer al LABINF sono sistemi a 64 bit). Se sono necessarie delle modifiche al proprio client o al server, controllare attentamente che il client ed il server modificati comunichino correttamente sia tra loro sia con i client e server forniti. Al termine dell'esercizio, si deve avere un client e un server che possono comunicare tra loro e possono operare correttamente con il client ed il server forniti nel materiale del laboratorio.

Provare a trasferire un file binario di dimensioni notevoli (circa 100 MB). Verificare che il file sia identico tramite il comando cmp o diff e che l'implementazione sviluppata sia efficiente nel trasferire il file in termini di tempo di scaricamento.

Mentre è in corso un collegamento provare ad attivare un secondo client verso il medesimo server.

Provare ad attivare sul medesimo nodo una seconda istanza del server sulla medesima porta.

Provare a collegare il client ad un indirizzo esistente ma ad una porta su cui il server non è in ascolto.

Provare a disattivare il server (battendo CTRL+C nella sua finestra) mentre un client è collegato.

## Esercizio 2.4 (dati in standard XDR)

Modificare il client TCP sviluppato nella prima esercitazione (esercizio 1.3) per inviare i due numeri interi letti da standard input e ricevere la risposta (somma) dal server utilizzando lo standard XDR per la rappresentazione dei dati. Non e' necessario gestire errori: il server restituisce sempre un unico valore di tipo intero. Utilizzare il server di prova reso disponibile nell'esercizio 1.1 usando l'opzione -x.