

# Programmazione distribuita I

(01NVWOV)

## AA 2016-2017, Esercitazione di laboratorio n. 3

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 3.1 (server TCP concorrente)

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.

Il server deve implementare lo stesso protocollo usato nell'esercizio 2.3 ed essere di tipo concorrente, attivando processi figli a richiesta (un nuovo processo per ogni richiesta).

Usando il client sviluppato nell'esercizio 2.3 provare ad attivare più client simultaneamente e verificare che ognuno riesca a stabilire la connessione e trasferire files.

#### Parte facoltativa

Modificare il server in modo che non più di **tre** client vengano serviti contemporaneamente. Quindi, provare ad attivare quattro client allo stesso tempo e verificare che solo 3 sono in grado di ottenere il servizio, cioè, a ricevere il file. Si potrebbe notare che i client riescono comunque a collegarsi al server svolgendo il 3-way handshake del TCP anche se il server non ha ancora chiamato la funzione accept(). Questo è un comportamento standard del kernel Linux per migliorare il tempo di risposta dei server. Comunque, fino a quando il server non ha chiamato la accept(), non può usare la connessione, cioè, i comandi non verranno ricevuti. Nel caso in cui il server non chiamasse la accept() per un certo tempo, la connessione aperta dal kernel verrà automaticamente chiusa.

Provare a terminare un client brutalmente (battendo ^C nella sua finestra) e verificare che il server (padre) sia ancora attivo ed in grado di rispondere ad altri client.

**Suggerimento:** per vedere i processi attivi sul sistema e la loro relazione padre-figlio, utilizzare il comando ps -uf

### Esercizio 3.2 (client TCP con multiplexing)

Modificare il client dell'esercizio 2.3 per gestire un'interfaccia utente interattiva che preveda i seguenti comandi:

GET file (richiede di fare GET del file indicato)

Q (richiede di chiudere il collegamento col server col comando QUIT dopo che un eventuale trasferimento in corso è terminato)

A (richiede di terminare immediatamente il collegamento col server, anche interrompendo un eventuale collegamento in corso)

L'interfaccia utente deve essere sempre attiva e permettere quindi il "type-ahead", ossia fornire input anche se c'è un trasferimento in corso dal server.

### Esercizio 3.3 (server TCP con pre-forking)

Modificare l'esercizio 3.1 per fare in modo che i processi che accettano richieste di trasferimento files dai client siano creati non appena il server viene lanciato (pre-forking), e rimangano in attesa finché un client si connette. Se un client chiude la connessione, il processo che serviva il client deve procedere a servire un nuovo client in attesa, se presente, o rimanere in attesa di nuovi client.

Rendere configurabile il numero di processi figli che vengono lanciati all'avvio del server tramite linea di comando, imponendo un massimo di 10 figli.

Verificare il corretto funzionamento lanciando il server con 2 figli e collegandosi con 3 client, da ciascuno dei quali è stata effettuata una richiesta per un file. Chiudere uno dei tre client e verificare che viene immediatamente servito il client in attesa.

Verificare che il server sia in grado di gestire anche il caso in cui un client collegato vada in crash (per esempio se chiuso tramite il comando kill), ossia che il processo che serviva il client si accorga di questa condizione e si renda disponibile per un eventuale nuovo client in attesa.

Infine, modificare il server in modo che se la connessione con il client rimane inattiva per 2 minuti essa viene terminata dal server, in modo da consentire al figlio che gestiva il client considerato inattivo di essere pronto a gestire un eventuale altro client in attesa di connessione.

**Suggerimento:** per vedere i processi attivi sul sistema e la loro relazione padre-figlio, utilizzare il comando `ps -uf`

### Esercizio 3.4 (XDR-based file transfer)

Modificare il client ed il server sviluppati fino a questo punto per il trasferimento di file in modo che accettino un parametro opzionale `-x` prima degli altri argomenti (cioè come primo argomento). Se questo argomento è presente sulla linea di comando, un protocollo simile al precedente, ma basato su XDR, deve essere usato. Secondo questo nuovo protocollo, i messaggi dal client al server e viceversa sono tutti rappresentati tramite il tipo XDR "message" definito come segue:

```
enum tagtype {
    GET = 0,
    OK   = 1,
    QUIT = 2,
    ERR  = 3
};

struct file {
    opaque contents<>;
    unsigned int last_mod_time;
};

union message switch (tagtype tag) {
    case GET:
        string filename<256>;
    case OK:
        struct file fdata;
    case QUIT:
        void;
    case ERR:
        void;
};
```

Verificare che il client sviluppato funzioni correttamente con il server fornito (per linux), e analogamente che il server sviluppato funzioni con il client fornito.