Programmazione distribuita I

(01NVWOV)

AA 2018-2019, 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)

PER L’ESONERO, questo esercizio deve essere sottomesso entro il 20 maggio 2019, ore 11:59am (del mattino).

Sviluppare un server TCP concorrente (in ascolto sulla porta specificata come primo parametro sulla riga di comando come numero intero decimale) che dopo aver stabilito una connessione TCP con un client, accetti richieste di trasferimento file dal client ed invii i files richiesti usando lo stesso protocollo dell’esercizio di laboratorio 2.3. Il server deve creare processi a richiesta (un nuovo processo per ogni nuova connessione TCP).

Quando si sviluppa questo nuovo server, si usi la stessa struttura di directory già usata per il client e il server dell’esercizio di laboratorio 2.3 (cartella `lab2.3`). Copiare i files di nome `test.sh` e `makezip.sh` (forniti nella cartella `lab3.1` dell’archivio zip dove si trova il materiale del laboratorio 3) nella cartella `lab2.3`, e nella cartella `lab2.3/source`, creare una nuova cartella chiamata `server2` (che sarà allo stesso livello di `server1` e `client1`). Notate che il vecchio test.sh deve essere sovrascritto. Scrivere il codice del nuovo server dentro questa cartella (si possono usare le librerie che sono state già inserite nella cartella `source`).

Il codice del server2 sarà considerato valido solo se potrà essere compilato con il seguente comando, lanciato dalla cartella `source`:

```
gcc -std=gnu99 -o server server2/*.c *.c -Iserver2 -lpthread -lm
```

Usando il client sviluppato nell’esercizio di laboratorio 2.3 (client1), provare a lanciare più client contemporaneamente e verificare che ognuno può stabilire una connessione e trasferire files.

Ripetere gli stessi tests fatti per il server dell’esercizio di laboratorio 2.3: in particolare, cercare di collegare il proprio client e server, e il client di riferimento con il proprio server per verificare la corretta interoperabilità.

Provare a terminare forzatamente un client (premendo CTRL+C nel suo terminale) e verificando che il server padre sia ancora attivo e che risponda correttamente ad altri client.

Quando i tests sul nuovo server sono terminati, è possibile far girare i tests di accettazione lanciando il seguente comando dalla cartella `lab2.3`:

```
./test.sh
```

Lo script testerà sia server1 (sviluppato nel laboratorio 2.3) sia server2, e dirà se i tests obbligatori sono stati passati oppure no.

Per sottomettere la soluzione, lanciare il comando:

`./makezip.sh`

dalla stessa cartella. Lo script creerà un file zip chiamato `lab2.3_3.1.zip` con la propria soluzione. Infine, andare sul portale https://pad.polito.it:8080/enginframe/index.html e sottomettere questo file zip (si riceveranno le credenziali per il login nei prossimi giorni).

Esercizio 3.2 (server TCP con pre-forking)

Creare un'altra versione della soluzione dell'esercizio 3.1 in cui 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 un parametro della 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.

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

Esercizio 3.3 (dati in standard XDR)

Modificare il client TCP sviluppato nel primo laboratorio (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 è necessario gestire errori: il server restituisce sempre un unico valore di tipo intero. Utilizzare il server di prova reso disponibile nell'esercizio 1.1, lanciato usando l'opzione -x.

Esercizio 3.4 (XDR-based file transfer)

Creare un'altra versione del client e del 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 nel materiale del laboratorio.

Esercizio 3.5 (client TCP con multiplexing) - Facoltativo

Modificare il client dell'esercizio di laboratorio 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.