Programmazione distribuita I

(01NVWOV)

AA 2018-2019, Esercitazione di laboratorio n. 1

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 delle proprie applicazioni**, 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.

NB2: Si suggerisce l’utilizzo, per quanto possibile, del programma `valgrind` per testare tutti gli accessi alla memoria fatti dal programma. Per usarlo, basta anteporlo, sulla linea di comando, al comando vero e proprio da lanciare. Esempio: `valgrind ./server_test -a 1500`. Se il programma è compilato con le opzioni di debug (`-g`), eventuali messaggi conterranno anche la linea di riferimento ed il file sorgente interessato.

Esercizio 1.1 (server di prova)

In ambiente linux, compilare il server di prova fornito, che si mette in ascolto sulla porta specificata, e somma i due numeri ricevuti.

Utilizzare il comando:

```
gcc -g -Wall -DTRACE -o server_test server_test.c errlib.c sockwrap.c
```

Per avviarlo:

```
./server_test formato porta
```

ove *formato* è una opzione che specifica il formato dei dati usato nel dialogo col client:

- **`-a`** per la rappresentazione dei dati in ASCII
- **`-x`** per la rappresentazione dei dati tramite XDR

e *porta* è il numero di porta sulla quale il server si mette in ascolto (per es. 1500).

Testare il funzionamento del server usando l'opzione -a e come client l'applicazione ***telnet***, che prende come parametro l'indirizzo del server cui collegarsi e la porta. Una volta connessi si devono scrivere due numeri decimali e premere INVIO.

Nota: Per interrompere il collegamento all'interno del programma telnet premere CTRL + ALT GR +] e poi Q seguito da INVIO.

Nota 2: Per comodità, è possibile utilizzare le funzioni presenti in **sockwrap.c** (Socket, Connect, etc...). Queste sono una selezione delle funzioni di libreria fornite con il libro di Stevens. Come spiegato a lezione, le funzioni con l’iniziale maiuscola sono la copia esatta delle API dei socket ma hanno gia' integrato il controllo dei valori di ritorno delle funzioni per segnalare e stampare eventuali errori, ed uscire dal programma. E' possibile modificare il codice contenuto in tali files per

adattarlo ai propri scopi, per esempio aggiungendo altre funzioni. Questo potrebbe essere utile soprattutto per l'esame finale.

Nota 3: IMPORTANTE: Chi volesse utilizzare un ambiente di sviluppo (Eclipse, XCode, ecc.) per compilare il codice C si ricordi di impostare l'opzione `-Wall` per la compilazione in modo che tutti i potenziali errori vengano segnalati dal compilatore. Si consiglia comunque di fare pratica con la compilazione, dal terminale, a linea di comando che è l'unica di cui si garantisce il funzionamento in sede di esame.

Suggerimento: per vedere tutti i processi in ascolto sulle varie porte sul sistema locale utilizzare il comando `netstat -atnp` (per tcp) o `-aunp` (per udp)

Esercizio 1.2 (test di connessione)

Scrivere un client che si colleghi ad un server TCP all'indirizzo e porta specificati come primo e secondo parametro sulla riga di comando. Il programma deve notificare all'utente se è riuscito a collegarsi correttamente oppure no. Il programma deve quindi chiudere correttamente il canale e terminare.

Esercizio 1.3 (client che interagisce con il server di test con dati ASCII)

Sviluppare un client che si colleghi ad un server TCP all'indirizzo e porta specificati come primo e secondo parametro sulla riga di comando. Il client poi legge da standard input due numeri interi senza segno e li invia al server (codificati come stringhe di caratteri ASCII, come spiegato oltre). Infine, il client riceve la risposta (somma, o errore) dal server, e la stampa a video. Per testare il client puo' essere utilizzato il programma compilato nell'esercizio 1.1

Gli interi sono rappresentati localmente internamente al client e al server come interi su 16 bit senza segno, mentre sono invece rappresentati usando caratteri ASCII quando trasmessi sulla connessione TCP (come spiegato oltre). Il server inoltre gestisce il caso di overflow segnalandolo con uno specifico errore (si veda l'esempio).

Il client invia al server i due numeri in notazione decimale espressi mediante caratteri numerici ASCII. I numeri devono essere separati da un solo spazio e la trasmissione deve essere terminata con CR LF (*carriage return* e *line feed*, cioè due bytes corrispondenti ai valori 0x0d 0x0a in esadecimale, ossia `'\r'` e `'\n'` in notazione C). Il server restituisce il risultato (un solo numero) espresso mediante caratteri ASCII, privo di spazi e terminato con CR-LF. In caso di errore, il server restituisce un singolo messaggio di errore (sequenza di caratteri ASCII) terminato anch'esso da CR-LF. Il messaggio di errore può essere differenziato dal risultato ottenuto con successo perché non inizia con una cifra (si veda l'esempio). Nel codice C si può usare il tipo char per la codifica ASCII.

Esempi (ogni elemento rappresenta un singolo byte trasmesso in codice ASCII):

(client → server) `1 2 3 4 5  3 CR LF`

(server → client) `1 2 3 4 8 CR LF`

o in caso di errore:

(server → client) `o v e r f l o w CR LF`

(server → client) `i n c o r r e c t  o p e r a n d s CR LF`

Esercizio 1.4 (client-server UDP base)

Scrivere un client che invii ad un server UDP (all'indirizzo e porta specificati come primo e secondo parametro sulla riga di comando) un datagramma contenente il nome (massimo 31 caratteri) passato come terzo parametro sulla riga di comando, attenda quindi un qualunque datagramma di risposta dal server. Il client termina mostrando il contenuto (testo ASCII) del datagramma ricevuto oppure segnalando che non ha ricevuto risposta dal server entro un certo tempo.

Sviluppare un server UDP (in ascolto sulla porta specificata come primo parametro sulla riga di comando) che risponda ad ogni datagramma ricevuto inviando un datagramma di risposta contenente lo stesso nome presente nel pacchetto ricevuto.

Provare ad effettuare degli scambi di datagrammi tra client e server con le seguenti configurazioni:

- client invia datagramma ad una porta su cui il server è in ascolto
- client invia datagramma ad una porta su cui il server NON è in ascolto
- client invia datagramma ad un indirizzo non raggiungibile (es. 10.0.0.1)

Nel materiale del laboratorio sono disponibili le versioni eseguibili del client e del server che si comportano secondo le specifiche fornite (si noti che i files eseguibili sono forniti per architetture sia a 32 bit sia a 64 bit. Files con suffisso _32 sono compilati per essere eseguiti su sistemi Linux a 32 bit, gli altri per sistemi a 64 bit. I computer del laboratorio sono sistemi a 64 bit).

Provare a collegare il proprio client col server fornito nel materiale del laboratorio, e il client fornito nel materiale con il proprio server. Se sono necessarie correzioni fare in modo che il proprio server e il proprio client continuino a comunicare correttamente con il client e il server fornito. Per terminare correttamente l'esercizio è necessario arrivare ad avere un client ed un server che comunicano correttamente tra loro e che possono comunicare correttamente con il client ed il server forniti nel materiale del laboratorio.