Programmazione distribuita I

(01NVWOV)

AA 2017-2018, 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 per trasferimento file TCP iterativo)

PER L’ESONERO, questo esercizio deve essere sottomesso entro il 21 maggio 2018, ore 11:59 (del mattino).

Sviluppare un server TCP sequenziale (in ascolto sulla porta specificata come primo parametro sulla riga di comando come numero decimale) che, dopo aver stabilito una connessione con un client, accetti richieste di trasferimento di file dal client e spedisca i files richiesti indietro al client, seguendo il protocollo specificato nel seguito. I files disponibili per essere inviati dal server sono quelli accessibili dal server nel suo file system nella sua directory di lavoro.

Sviluppare un client che possa collegarsi ad un server TCP (all’indirizzo e porta specificati come primo e secondo parametro sulla riga di comando). Dopo aver stabilito la connessione, il client richiede il trasferimento dei files il cui nome è specificato sulla linea di comando dal terzo parametro in poi, e li salva localmente nella propria directory di lavoro.

Dopo aver trasferito e salvato localmente un file, il client deve stampare su standard output un messaggio circa l'avvenuto trasferimento, includendo il nome del file, seguito dalla dimensione del file (in bytes, come numero decimale), e dal timestamp di ultima modifica (come numero decimale).

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 dai caratteri ASCII carriage return (CR) e line feed (LF):

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 usando la stessa connessione TCP inviando più comandi GET, uno dopo l'altro. 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 la connessione con il client.

Si usi la struttura di directories inclusa nel file zip fornito con questo testo. Dopo aver un-zippato l'archivio, si troverà una directory chiamata `lab2.3`, che include una cartella `source` con una sottocartella `server1` dove si deve scrivere il client. Sono già presenti degli scheletri vuoti di files sorgenti (uno per il client ed uno per il server). Si riempiano semplicemente questi files con i propri programmi senza spostarli. Possono essere usate librerie di funzioni (es. quelle dello Stevens). I files sorgente C di tali librerie devono essere copiati nella cartella `source` (non metterli nelle sottocartelle `client1` o `server1`, perché tali sottocartelle devono contenere solamente il proprio codice!). Inoltre, si ricordi che se si vogliono includere alcuni di tali files nei propri sorgenti è necessario specificare il percorso "`..`" nella direttiva include).

La soluzione sarà considerata valida se e solo se può essere compilata tramite i seguenti comandi lanciati dalla cartella `source` (gli scheletri forniti compilano già tramite questi comandi, non spostarli, riempirli solo):

```
gcc –std=gnu99 -o server server1/*.c *.c –Iserver1 -lpthread -lm

gcc –std=gnu99 -o client client1/*.c *.c –Iclient1 -lpthread -lm
```

La directory `lab2.3` contiene anche una sottocartella chiamata `tools` che include alcuni strumenti di test, tra i quali è possibile trovare i files eseguibili di un client e un server di riferimento che si comportano secondo il protocollo stabilito e che possono essere usati per effettuare tests di interoperabilità.

Provare a collegare il proprio client con il server di riferimento, ed il client di riferimento con il proprio server per testare l'interoperabilità (notare che i files eseguibili sono forniti per architetture sia a 32 bit sia a 64 bit. I files con suffisso _32 sono compilati 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 tra loro al termine delle modifiche. Al termine dell'esercizio, si dovrà avere un client e un server che possono comunicare tra loro e che possono operare correttamente con il client ed il server di riferimento.

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 non raggiungibile.
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.

Quando il test del proprio client e server è terminato, si possono usare gli script di test forniti nella directory `lab2.3` per effettuare un test finale che verifichi che il proprio client e server siano conformi ai requisiti essenziali dell'esercizio, e che passino i tests obbligatori per la sottomissione. Per lanciare lo script di test, semplicemente dare il seguente comando dalla directory `lab2.3`:

```
./test.sh
```

Lo script dirà se la soluzione è accettabile. Se non lo è, correggere gli errori e riprovare fino a quando si passano i tests. Gli stessi tests verranno eseguiti sul nostro server quando la soluzione viene sottomessa. Altre caratteristiche della soluzione sottomessa verranno testate dopo la chiusura delle sottomissioni, al fine di decidere l'esito dell'esonero ed assegnare il corrispondente voto.

Se si desidera far girare lo script di test su un sistema a 32 bit è necessario per prima cosa sovrascrivere i files eseguibili sotto la directory `tools` con la loro versione corrispondente a 32 bit. Per esempio:

```
mv server_tcp_2.3_32 server_tcp_2.3
```

Per sottomettere la propria soluzione, per prima cosa lanciare il seguente comando dalla directory `lab2.3`:

```
./makezip.sh
```

Si avrà la possibilità di sottomettere la soluzione da considerare per l'esonero **insieme alla soluzione di un altro esercizio che verrà assegnato nel lab3**. Le istruzioni di sottomissione saranno fornite con il lab3.

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 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 usando l'opzione -x.