Sistemi Operativi e Reti - Esercitazione su socket e protocolli di livello applicazione

Durante l'esecuzione dei seguenti esercizi, monitorare l'andamento delle connessioni in corso usando il comando `netstat` e catturando il traffico usando `wireshark`. Si ricordi che è possibile sapere quali sono gli indirizzi IP associati alle interfacce di rete del proprio host usando il comando ifconfig (Linux) o ipconfig (Windows).

Esercizio 1

E' dato il codice che implementa il server "Maiuscolizzatore" visto a lezione (Sorgenti disponibili su http://bit.ly/Im7zwB).

Si modifichi il protocollo di comunicazione tra programma client e programma server in maniera tale che il processo client possa inviare una sequenza di stringhe (anziché una sola come nel sorgente fornito) separate dal carattere di fine linea. Ogni volta che il client invia una stringa deve attendere che questa venga posta in maiuscolo dal processo server e ritornata al processo client. Il client deve poter terminare la conversazione inviando una linea costituita dai soli caratteri 'END' che devono essere riconosciuti dal server come segnale di fine conversazione. Si sperimenti il proprio sorgente facendo girare il programma client e il programma server su host rispettivamente differenti.

Esempio di conversazione (Linee emesse dal server inizianti con S, linee emesse dal client inizianti con C):

```
C: Ciao
S: CIAO
C: Come stai
S: COME STAI
C: Scusa se te lo chiedo
S: SCUSA SE TE LO CHIEDO
C: E non gridare
S: E NON GRIDARE
C: END
```

(Il server chiude il socket assumendo che il client faccia altrettanto).

Domanda: come deve essere modificato il programma Client se il Server memorizza tre linee alla volta prima di rispondere con tutte e tre le linee poste in maiuscolo?

Esercizio 2

E' data la seguente conversazione SMTP di esempio:

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Dove 'C' (comandi in rosso) indica i comandi che devono essere impartiti per depositare una e-mail su un server SMTP specificato. Le parti in blu indicano valori parametrici che possono essere cambiati (mittente, destinatario, testo della e-mail).

Si scriva un programma C++ (o Java) dal nome `sendMail` che possa essere invocato da linea di comando con la seguente sintassi:

```
sendMail nomeServer mitt dest  < testoemail.txt
```

Il comando deve contattare `nomeServer` sulla porta 25 ed effettuare una opportuna conversazione SMTP, il cui effetto finale deve essere la consegna di una e-mail il cui testo è contenuto nel file `testoemail.txt`, con indirizzo mittente specificato da `mitt` e indirizzo destinatario specificato da `dest`. Durante la conversazione SMTP si deve verificare che il server ritorni i corretti codici di risposta a seconda dei casi (220 = saluto iniziale, 250=OK, 354=pronto a ricevere testo, 221=saluto finale), e stampare gli opportuni messaggi relativi a eventuali anomalie. Si può usare `ml.mat.unical.it` come SMTP server di prova e il povero <`ianni@mat.unical.it`> come destinatario.

Esercizio 3

Avendo a disposizione il sorgente che implementa il conteggio dei numeri primi parallelizzato su più thread (Qui: http://bit.ly/18dcaaG), e visto nelle precedenti lezioni, si progetti una applicazione client/server che implementi la funzione

`contaprimi(int min, int max)` distribuendo il carico di lavoro su almeno 3 host differenti.

Suggerimento: si può pensare di individuare un host *master* e un insieme di host *slave* e di disegnare un protocollo per lo scambio di messaggi opportuno. L'host *master* comunica a tutti gli slave l'intervallo di numeri naturali rispettivo, attende la rispettiva risposta e mette insieme il risultato finale.

Ad esempio, un messaggio da master M verso uno slave S_i può essere "MIN=400 MAX=500" e la rispettiva risposta può essere "PRIMI=17".

Si assuma che le coppie <IP-Slave,Porta-Di-Ascolto-Slave> siano note a priori.