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 (Difficoltà: FACILE)

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 a un certo server SMTP remoto per depositare una e-mail. Le parti in blu indicano valori parametrici che possono essere cambiati (mittente, destinatario, testo della e-mail).

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

```
python3 sendMail.py nomeServer mitt dest < testoemail.txt
```

Il comando deve contattare `nomeServer` sulla porta `PORT` 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. Come SMTP server di prova è possibile utilizzare le coordinate IP:Porta che vi verranno fornite direttamente durante l'esercitazione. Le email di prova devono essere destinate all'utente <ianni@maia>.

N.B. Il l'indirizzo ip del server `nomeServer` e la porta `PORT` su cui effettuare la connessione vi saranno comunicati direttamente durante la lezione in laboratorio.

Esercizio 2 (Difficoltà: FACILE)

È dato il codice che implementa il server "Maiuscolizzatore" visto a lezione (Sorgenti disponibili qui).

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 per volta prima di rispondere con tutte e tre le linee poste in maiuscolo?

Esercizio 3 (Difficoltà: MEDIA)

Scrivere un programma che si comporti come un semplice file server in grado di rendere disponibile per i client ad esso collegati una serie di file pronti per il download.
Dopo che il server è stato avviato, ogni client potrà connettersi ad esso utilizzando la specifica porta di ascolto. Il server è preimpostato per lavorare sui contenuti di una certa *cartella corrente*.

Un client può inviare al server **3 diversi comandi**:

1. **ls --->** Il server risponde con la lista dei file contenuti nella directory corrente;
2. **cat filename --->** Il client richiede il download del file filename.
   ATTENZIONE: non inserire il nome di una cartella. Verificare prima che il file richiesto sia un file di testo. Per verificare la tipologia di un file si può usare il comando shell Linux ***`file`***;
3. **cd new/directory/path --->** Ci si sposta tra le cartelle del Server (ammesso che il path specificato esista);

Il server risponde a un client con **"OK"** o con **"ERROR"** al comando **"cat",** rispettivamente se il file richiesto esiste o meno nel server. In caso positivo dopo l'OK, seguito da fine linea, il server stamperà in STDOUT il contenuto del file.
Quando il server avrà completato la stampa del file invierà al client la stringa **<EOF>.**
Per semplicità si assuma che nei file non sia mai presente la stringa <EOF>.

Il server può anche rispondere al client con la stringa **"comando sconosciuto"** se il comando ricevuto non è esistente.

NOTA BENE:

- La modalità di gestione degli errori più comuni (file inesistente, directory non esistente, ecc.) è delegata allo studente.
- I comandi (ls, cat e cd) **NON** sono case sensitive (le stringhe, come i nomi dei file, invece **SI**)
- Per semplicità si può supporre di specificare una cartella di default appena avviato il server e che al suo interno vi siano situati **SOLO file di testo**.
- Non consentire all'utente di effettuare il download ricorsivo di intere cartelle.

Esempio di comunicazione client-server: (C: → Client ; S: → Server)

C: ls
S: file2.txt
file1.txt

C: CAT file2.txt
S: riga 1
riga 2
…

C: cd /home/user/Scrivania
S: New working directory is: /home/user/Scrivania

Esercizio 4 (Difficoltà: ALTA)

Gioco del Quiz in Rete

Creare un gioco del quiz in cui un server pone domande a una serie di client e aspetta le risposte. I client devono connettersi al server e rispondere alle domande. Il server deve valutare le risposte e per poi comunicare l'esito al client connesso

Istruzioni:

1. **Server**
   - Il server genera domande casuali su argomenti come storia, scienza, sport, ecc.
   - Invia le domande ai client con le opzioni di risposta (A, B, C, D).
   - Riceve le risposte dai client e verifica se sono corrette.
   - *Tiene traccia dei punteggi dei giocatori. (punto extra)*

2. **Client**
   - I client si connettono al server.
   - Ricevono le domande e le opzioni di risposta dal server.
   - Consentono all'utente di inserire la risposta (1, 2, 3, 4).
   - Inviano la risposta al server.

3. **Requisiti**
   - Utilizzare il modulo `socket` di Python per gestire la comunicazione tra il server e i client.
   - Creare una lista di domande e risposte corrette sul lato del server.
   - *Gestire in modo appropriato le risposte dei client e aggiornare i punteggi. (punto extra)*
   - *Aggiungere un timer per limitare il tempo di risposta dei giocatori. (punto extra)*
   - *Visualizzare il punteggio dei giocatori in tempo reale durante il gioco. (punto extra)*