Esercitazione 2020-10-08

Esercizio 1.1

Per quanto riguarda questo primo esercizio, dobbiamo eseguire client e server scritti in precedenza ed osservare il traffico con wireshark.

Avviamo quindi il server e iniziamo a monitorare il traffico con wireshark; non appena lanciamo il client otteniamo:

Esercizio 1.2

Questo esercizio è uguale al primo, ma il protocollo usato è TCP.

Anche in questo caso ascoltiamo con wireshark:

Come possiamo osservare, il traffico è significativamente diverso rispetto al caso del protocollo UDP. I primi tre segmenti sono **segmenti di controllo**, usati per attivare la connessione; questa operazione viene chiamata **handshake**.

Esercizio 1.3

L'esercizio ci dice di scrivere un'applicazione client-server per realizzare la moltiplicazione di due valori interi. Deve essere utilizzato il protocollo TCP, quindi orientato ai flussi.

Client

Per quanto riguarda il client, il codice è uguale fino alla chiamata della **connect()**.

```
int values[2];
printf("Inserisci i due numeri da moltiplicare.\n Inserisci il primo valore: "); scanf("%d",
&values[0]);
printf("Inserisci il secondo valore: "); scanf("%d", &values[1]);

write(sd, &values, sizeof(values));

// leggo la risposta
long prod;
read(sd, &prod, sizeof(long)); //salvo il risultato nella variabile prod

printf("Il risultato e': %ld\n", prod);
close(sd); // chiudo la socket
return 0;
```

Per la soluzione che ho adottato, ho preferito spedire gli operandi tramite un array, a mio parere una soluzione più elegante.

Ci basterà quindi costruire un array, inserire i valori al suo interno ed effettuare la **write()** sul canale sd, che spedisce al server l'array.

Successivamente, il server ci darà una risposta che leggeremo con una **read()**; a questo punto non ci basta che stampare la soluzione.

Server

```
while (1) {
  sd2 = accept(sd, (struct sockaddr*) &cad, &alen);

  int values[2];
  read(sd2, &values, sizeof(values));
  long prod = values[0] * (long) values[1];

  write(sd2, &prod, sizeof(long));

  close (sd2);
}
```

Anche in questo caso, fino alla listen(), il codice è uguale.

Per leggere i dati, non ci basta che invocare la **accept()** e successivamente una **read()**. Successivamente calcoliamo il valore ed inviamo i dati al client.

Cosa vede wireshark?

Inizialmente, c'è la fase di handshaking; non sono ancora stati inviati messaggi perchè il client sta aspettando un input da utente.

Una volta inseriti i due operandi e premuto invio, viene inviato un messaggio al server contenente l'array, successivamente si ottiene una risposta:

Su wireshark non riusciamo a vedere i nostri operandi ne tantomeno il risultato; questo perchè non stiamo scambiando caratteri, ma interi, che sono quindi codificati in binario.

Anche se non riusciamo a vedere i dati scambiati, possiamo individuare i pacchetti che codificano i dati inviati: se notiamo è presente la voce **data**, ed in questo caso sono stati inviati 8 bytes; questo messaggio è stato inviato dal client al server, e lo notiamo dalla direzione delle porte.

🏁 1:20

Esercizio 1.4

La traccia di questo esercizio è uguale alla precedente, con l'unica differenza che dobbiamo usare un protocollo UDP, e quindi orientato ai datagram.

Siccome questa volta non abbiamo a disposizione un canale di comunicazione orientato ai flussi, abbiamo bisogno di incapsulare i dati all'interno di un datagram. Ogni datagram che spediamo deve essere caratterizzato dall'indirizzo di trasporto che usiamo.

Ad un certo punto il client deve invocare la **sendto()** per inviare il messaggio a destinazione; dobbiamo quindi copiare un contenuto di memoria di spazio utente, nel buffer di spedizione della socket.

A differenza dell'esercizio precedente, non possiamo invocare ripetutamente la funzione **write()**, ma dobbiamo costruire l'intero messaggio e poi invocare, in un unico messaggio, la funzione **sendto()**.

Inserire i valori

```
printf("Inserisci i due numeri da moltiplicare.\n Inserisci il primo valore: "); scanf("%d", (int*)buf); printf("Inserisci il secondo valore: "); scanf("%d", (int*) (buf + sizeof(int)));
```

Per inviare i due valori, dobbiamo usare un'unica send(), e per questo motivo dobbiamo incapsulare i due valori a e b all'interno del nostro buffer.

Il primo valore viene inserito dalla prima posizione del buf, mentre per inserire il secondo dobbiamo spiazzare il puntatore a + sizeof(int), per non sovrascrivere il valore precedente.

Analizzando il traffico con wireshark notiamo come ci siano solo due scambi di messaggi, uno dove il client comunica al server i due operandi, ed un altro dove è il server a rispondere con il risultato.

Perchè non inviamo i dati singolarmente?

Perchè non inviamo prima un operando e poi l'altro, ma invece li inviamo in un unica volta? Gli operandi vengono inviati in un'unica soluzione perchè per ogni datagrama abbiamo un'intestazione, e quindi delle informazioni di controllo; dal punto di vista di canale di comunicazione, qualora inviassimo i dati singolarmente, invieremmo più dati.

---

fine lezione 7