Reti di calcolatori - 17 giugno 2024 - Compito A

Scrivere un server che consente di tradurre indirizzi IP in formato numerico nel loro equivalente in formato binario.

Server

Il protocollo implementato dal server prevede un messaggio di richiesta e uno di risposta. Il server dovrà porsi in ascolto sulla porta `1025` in attesa di messaggi da parte del client e rispondere a tali messaggi.

Ogni messaggio è composto da tre campi:

- Tipo, singolo carattere in codifica ASCII, lunghezza 1 byte. Il tipo può valere `Q` per le richieste (Query) o `R` per le risposte
- Lunghezza dei dati, intero senza segno, lunghezza 1 byte. Si tratta di un numero che indica la lunghezza (in byte) del campo successivo
- Dati. Campo a lunghezza variabile che dipende dal tipo del messaggio (stringa per le richieste, numero intero senza segno a 32 bit per le risposte)

Messaggio di richiesta

Il messaggio di richiesta contiene un indirizzo IPv4 in formato dotted decimal e il campo dati contiene un stringa, terminata dal carattere nullo, che rappresenta l'indirizzo IPv4 da convertire. Di seguito alcuni esempi (Nota: le parti tra virgolette indicano valori da intendersi come sequenza di caratteri; il byte nullo come terminatore di stringa è evidenziato come `\0`):

Di seguito alcuni esempi (**Nota**: le parti tra virgolette indicano valori da intendersi come sequenza di caratteri; il byte nullo come terminatore di stringa è evidenziato come `\x00`):

Type	Len	Data
`"Q"`	`10`	`"127.0.0.1"\x00`
`"Q"`	`9`	`"10.1.1.1"\x00`
`"Q"`	`12`	`"192.168.1.1"\x00`

Con riferimento alla prima riga abbiamo:

- il singolo carattere `Q` (che identifica la richiesta)
- il valore numerico 10, a indicare che la stringa nel campo dati conterrà 10 caratteri (incluso il terminatore di stringa)
- la stringa 127.0.0.1 composta da 10 caratteri (incluso il terminatore di stringa)

L'invio del messaggio corrispondente alla richiesta indicata nella prima riga causa la creazione di un pacchetto TCP con il seguente payload (rappresentato in esadecimale): 51 0a 31 32 37 2e 30 2e 30 2e 31 00

- il primo byte (0x51) rappresenta la codifica ASCII della lettera `Q`
- il secondo byte (0x0a) rappresenta il valore intero 10
- i rimanenti byte contengono la codifica esadecimale della stringa 127.0.0.1, terminata da 0x00

Messaggio di risposta

Il messaggio di risposta contiene l'indirizzo IPv4 in formato binario *network*. Esempi di messaggi, basati sugli esempi precedenti di richiesta precedentemente inviata sono riportati di seguito:

Type	Len	Data
`"R"`	`4`	`2130706433`
`"R"`	`4`	`167837953`
`"R"`	`4`	`3232235777`

Si noti che i dati *devono* essere *network-ordered*

Con riferimento alla prima riga abbiamo:

- il singolo carattere `R` (che identifica la risposta)
- il valore numerico 4, a indicare che il campo dati conterrà 4 byte (questo valore è uguale per tutte le risposte)
- la rappresentazione binaria dell'indirizzo IPv4 127.0.0.1 in network order

L'invio del messaggio corrispondente alla risposta indicata nella prima riga causa la creazione di un pacchetto TCP con il seguente payload (rappresentato in esadecimale): 52 04 7f 00 00 01

- il primo byte (0x52) rappresenta la codifica ASCII della lettera `R`
- il secondo byte (0x04) rappresenta il valore intero 4
- i rimanenti byte contengono la codifica binaria dell'indirizzo IPv4 127.0.0.1 (0x7f → 127, 0x0 → 00, 0x0 → 00, 0x01 → 1)

Elementi di valutazione

- Gestione dell'input dal client
- Conversione degli indirizzi in formato binario
- Trasmissione degli indirizzi in formato network-ordered
- Aderenza al protocollo di comunicazione indicato
- Qualità del codice

La verifica del corretto funzionamento del software sviluppato è basata sul client di esempio allegato al testo del compito.

Informazioni aggiuntive

- Il tempo per la prova è di 1 ora.
- Il server deve gestire più richieste (non deve terminare dopo l'invio di una risposta) ma può operare in modo sequenziale.
- *Non* è necessario validare l'input. Si assume che i dati inviati dal client siano corretti.
- Ogni tipo di strumento di generazione automatica del codice (incluso, ma non limitato a ChatGPT e Copilot) è vietato e *comporta l'annullamento della prova.*

- Ogni forma di comunicazione con altre persone è vietata, comporta *l'annullamento della prova* ed eventuali altre sanzioni, inclusa l'esclusione da appelli di esame successivi (si veda il Regolamento Studenti emanato dal Rettore con decreto rettorale 229/2023 in vigore dal 1/3/2023).
- È consentito l'accesso ad appunti, materiale didattico, documentazione cartacea o reperibile via Internet.

Sugerimenti

- Il software può essere implementato sia in C che in Python, ma si consiglia l'uso del linguaggio C.
- Si ricorda l'esistenza delle funzioni `hton`, `htons`, `ntoh`, `ntohs` per gestire la conversione in/da network format.
- Si rimanda all'esempio di codice per la conversione da IP in unsigned int
- Notare che la lunghezza massima di un IP in formato stringa è di 15 byte. La lunghezza massima di un messaggio di richiesta è di 17 byte. Le risposte invece hanno una lunghezza fissa di 6 bytes

Codice del client da usare per il testing del server

Il client effettua tre richieste di prova (le stesse indicate nel testo del compito), riceve le risposte dal server che dovete implementare e verifica che siano uguali al risultato atteso.

```
import socket

HOST='localhost'
PORT= 1025

def send_request(sock, req)
    print(f'Sending request {req}')
    sock.sendall(req)

def check_response(sock, resp):
    recv_resp = sock.recv(1024)
    if recv_resp == resp:
        print(f'OK: Received response {recv_resp}')
        return True
    else:
        print(f'ERROR: Expecting {resp}, got {recv_resp}')
        return False

data=[(b'Q\x0a127.0.0.1\x00',   b'R\x04\x7f\x00\x00\x01'),
        (b'Q\x0910.1.1.1\x00',    b'R\x04\x0a\x01\x01\x01'),
        (b'Q\x0c192.168.1.1\x00', b'R\x04\xc0\xa8\x01\x01')]

if __name__ == '__main__':
    for req, resp in data:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect((HOST, PORT))
```

```
        send_request(sock, req)
        rv = check_response(sock, resp)
        sock.close()
        if not rv: break
```

Funzione C per convertire IPv4 in unsigned integer

```
unsigned int ip2uint(char *ip){
    unsigned int ip3, ip2, ip1, ip0;
    sscanf(ip, "%d.%d.%d.%d", &ip3, &ip2, &ip1, &ip0);
    return (ip3<<24) + (ip2<<16) + (ip1<<8) + ip0;
}
```