Laboratorio di Reti - Indirizzamento IPv4 in C

Introduzione

Gli indirizzi IPv4 (Internet Protocol version 4) sono identificativi a 32 bit, solitamente rappresentati in formato decimale puntato (es. `192.168.1.1`). Di quei 32 bit la prima parte identifica la rete di appartenenza **(NetID),** mentre i restanti bit individuano uno specifico **host** (HostID) di quella subnet. Ma come capiamo dove terminano i bit del NetID e dove cominciano quelli di HostID?

Trova la risposta in questo video di NetworkChuck **qui**.

…ok è simpatico, ma **qui** troverete una spiegazione un po' più tecnica.

E se avete ancora dubbi, potete guardarvi queste dispense qui.

Proposta di Studio

Prima di procedere con gli esercizi di programmazione provate a risolvere questi esercizi (se avete ancora dubbi, fatevi aiutare dall'AI)

1. **Struttura di un IPv4**: Data la seguente configurazione di rete `172.20.30.147` `255.255.0.0`:
    - Convertire l'indirizzo in binario.
    - Determina la notazione CIDR per la subnet mask e convertire in binario.
    - Trova la NetID (indirizzo di rete) e la HostID (porzione host) in binario e decimale puntato.
    - Quanti Host può contenere questa rete?
2. **CIDR**: Dato l'indirizzo IPv4 `192.168.110.147/19` :
    - Convertire l'indirizzo in binario.
    - Determina la subnet mask in decimale puntato e binario.
    - Trova la NetID (indirizzo di rete) e la HostID (porzione host) in binario e decimale puntato.
    - Quanti Host può contenere questa rete?

Consegna

- Implementare almeno gli esercizi 1, 2, 3 e 4 nella libreria myNetLib.h
- Creare un menu interattivo che permetta di scegliere quale funzione eseguire.
- Testare il codice con diversi input e commentare le funzioni.

Esercizi

Esercizio 1: Conversione da Decimale Puntato a Binario

Scrivere una funzione `char* ipToBinary(char* ip)` che accetta un IPv4 in formato stringa (es. `"192.168.1.1"`) e stampa la rappresentazione binaria a 32 bit (NB. se riuscite fate ritornare il puntatore ad una stringa allocata dinamicamente con malloc() in cui avrete salvato la rappresentazione binaria come stringa).

Esempio:

Input: `"192.168.1.1"` → `11000000 10101000 00000001 00000001`

Suggerimento:

- Usare `strtok()` per separare i singoli ottetti.
- Convertire ogni ottetto in un intero e poi in binario.
- se tornate il puntatore al buffer dinamico, ricordate di deallocare nel chiamante con free().

Esercizio 2: Verifica Validità di un IPv4

Scrivere una funzione `int isValidIPv4(char* ip)` che controlla se una stringa è un IPv4 valido. Un indirizzo è valido se:

1. Ha 4 ottetti separati da punti.
2. Ogni ottetto è un numero tra 0 e 255.

Esempio:

Input: `"256.1.2.3"` → Output: `false` (perché 256 > 255).

Suggerimento:

- Usare `sscanf()` per estrarre i numeri.

Esercizio 3: Calcolo Network Address

Scrivere una funzione `void calculateNetworkAddress(char* ip, int cidr)` che, dato un IPv4 e un CIDR (es. `192.168.1.10/24`), stampi l'indirizzo di rete.

Esempio:

Input: `"192.168.1.10"`, `24` → Output: `192.168.1.0`

Suggerimento:

1. Convertire l'IP in un intero a 32 bit (es. `uint32_t`).
2. Applicare una maschera di bit per estrarre la rete.

```c
network_address = ip_integer & (0xFFFFFFFF << (32 - cidr));
```

Esercizio 4: Generazione Indirizzi Host

Scrivere una funzione `void generateHostAddresses(char* networkIp, int cidr)` che, dato un network address (es. `192.168.1.0/24`), stampi tutti gli indirizzi host possibili (da `192.168.1.1` a `192.168.1.254`).

Suggerimento:

- Usare un ciclo per iterare da `network_address + 1` a `broadcast_address - 1`.

- conviene implementare la funzione `char* uint2ip(unsigned ip_32b, char* ipOctet)` che riceve un indirizzo IPv4 come variabile unsigned a 4 Byte e lo converte in stringa, nella notazione ad ottetti. In questo caso lo ritorna sul buffer passato dal chiamante nel secondo parametro, ma anche dal puntatore di ritorno (così potete utilizzarla direttamente in una funzione di stampa: per es. puts())
