12.10.2022

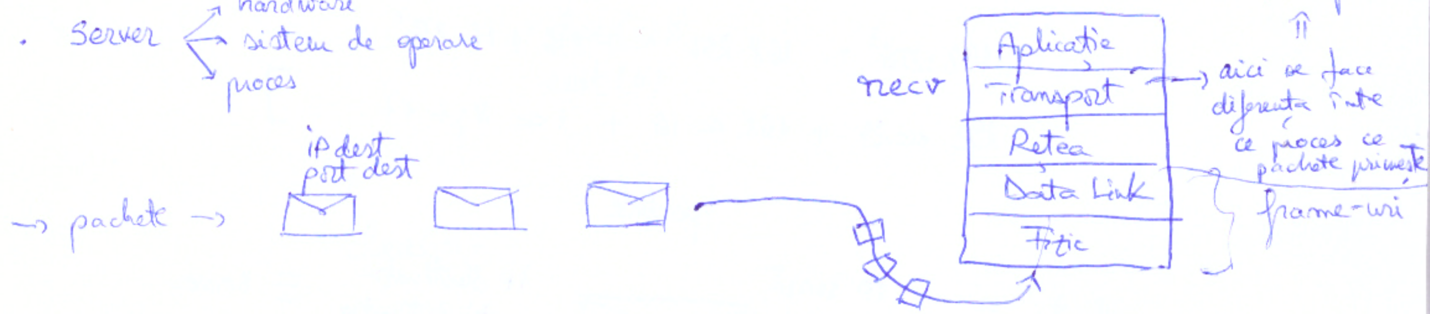
Cursul 2

CLIENT ⟶ SERVER

ip, portul serverului

↓ identificarea procesului cu care se comunică

- Server ⟨ hardware / sistem de operare / proces



CLIENT

c = socket(,)

↳ e vorba de protocoalele:

- Transport – TCP
- Rețea – IPv4

if (connect (c, {– ip server, – port server}, sizeof (structură)) < 0)

- {– ip server, – port server} ↓ sub forma unei structuri (– tip structură*)
- < 0 ↓ eșuează

* tip structură : AF_INET

send (c, &i, sizeof (i), 0)

recv (c, &j, sizeof (j), MSG_WAITALL)

close (c)

SERVER

s = socket(,)

→ pagina următoare

SERVER

s = socket (,)

if(bind (s, {- INADDR_ANY (→ 0.0.0.0), - port}, sizeof(structură)) < 0)

listen (s, 5) ← această îi conferă atributul de server

["192.168.1.17"

inet_addr
inet_ntoa

$192 \cdot 256^3 + 168 \cdot 256^2 + 1 \cdot 256^1 + 17 \cdot 256^0$

$192 \ll 24 + 168 \ll 16 + 1 \ll 8 + 17$]

pachete

client ≡ ip sursă, port sursă

ip destinație, port destinație ≡ server

roluri interschimbabile

⇒ socket-urile sunt bidirecționale

→ pentru a verifica ce porturi sunt ocupate: netstat

bind → la client este opțional (în general, nu se face), pentru că portul clientului nu este important și, de asemenea, dacă s-ar face bind la client există riscul de a ajunge pe un port deja ocupat

```
while (1) {
    c' = accept (s, {- ip client, - port client}, "sizeof")
    recv (c'
    send (c'
    close (c')
}   close (s)
```

deservirea clientului respectiv

} un server iterativ

! Obs: ordinea pt. send și recv, semantica + alte detalii / reguli de comunicare → RFC

→ transformarea serverului iterativ în server concurent:

```
while(1)
{
    c' = accept
    if( fork() == 0)
    {
        recv
        send
        close(c')
        exit(0)
    }
}
close(s)
```

înainte de trimitere

htons	ntohs
htonl	ntohl
	după primire

fie send la client → send(c, &i, sizeof(i), 0)

recv la server → ↳ | E8 | 03 | 00 | 00 |

este necesar ca arhitecturile să fie aceleași (dif. little endian vs. big endian)

- datele, în rețea, circulă în format big endian