# Recapitulare lab 1 & 2

- perspectiva de sistem de operare asupra unor servicii de comunicatie inter-proces (IPC)
  - *pipes*
  - *sockets*

- diferenta de paradigma: shared memory vs message passing
  - *pipe* – construieste un canal de comunicatie in memoria kernel (partajata) a aceleiasi masini

  => consecinte: procesele care scriu/citesc din pipe au sincronizarea accesului la canalul de comunicatie asigurata de kernel, la fel ca si politica de acces la date FIFO
  - *socket* – construieste un canal de comunicatie intre doua masini diferite legate intre ele printr-o retea de comunicatie

  => consecinte: complexitatea operarii retelei de comunicatie (care poate pp implicarea mai multor masini intermediare in efectuarea schimbului de mesaje) impune folosirea unor *protocoale de comunicatie* care:
  - asambleaza/dezasambleaza mesajele in pachete pt a putea fi transmise peste canale de comunicatie eterogene si asigura livrarea lor in ordine
  - rezolva automat probleme de rutare a pachetelor pe cai de comunicatie alternative intre nodurile sursa si destinatie atunci cand anumite masini intermediare nu sunt disponibile
  - asigura retransmisia automata a pachetelor pierdute/defecte
  - controleaza congestia retelei
  - samd

# Protocoale de comunicatie

- modelul OSI (Open Systems Interconnection)
  - 7 nivele: aplicatie, prezentare, sesiune, transport, retea, data link, nivelul fizic
- model alternativ, protocoalele internet (DoD):
  - combina primele trei nivel OSI intr-un singur nivel: aplicatie
  - combina nivelul fizic si data link intr-un singur nivel: link
- la curs: studiul intregii stive de protocoale OSI/internet
- la laborator: ne marginim la protocoale de comunicatie de nivel aplicatie si, concret, folosim modelul internet
- consecinta: vom folosi abstractia de *socket* pentru a crea cai de comunicatie bazate pe protocoalele de nivel transport ale internetului: TCP/UDP/SCTP

# TCP sockets

- creeaza un canal de comunicatie (o conexiune) intre doua endpoint-uri (adr IP, port)
- canalul de comunicatie garanteaza livrarea corecta si in ordine a mesajelor intre sursa si destinatie
- mesajele pierdute se retransmit automat
- canalul de comunicatie are o semantica asociata de tip stream de octeti (analog pipe)
- ca atare, din socket-ul TCP se citesc octeti, nu mesaje intregi !
- se creeaza folosind constanta predefinita SOCK_STREAM in apelul sistem *socket*
- un socket TCP trebuie intotdeauna conectat folosind apelul sistem *connect*

# UDP sockets

- creeaza un  canal de comunicatie *fara garantii* intre doua endpoint-uri
- mesajele (numite in acest caz *datagrame*, ca la protocolul IP) se pot pierde, nu se retransmit automat, nu se livreaza in ordine
- mesajele se citesc in intregime, i.e. o datagrama la un moment dat (spre deosebire de TCP)
- se creeaza folosind constanta predefinita SOCK_DGRAM in apelul sistem *socket*
- in general, un socket UDP nu trebuie conectat
  - exista primitive specifice bibliotecii de sockets pt transmisia/receptia mesajelor (datagramelor): *sendmsg/sendto* respectiv *recvmsg/recvfrom*
  - aceste apeluri sistem includ endpoint-ul destinatie printre parametri
- daca se foloseste totusi apelul sistem *connect*, se creeaza o asociere intre endpoint-urile client si server care permite folosirea apelurilor sistem de file I/O *read/write* (si variantele lor)
  - momentan, vom folosi UDP sockets conectati pentru a putea folosi *read/write*

# Exemplu de protocol de comunicatie de nivel aplicatie

```
1       $ telnet mail7.imar.ro 25
2       Trying 193.226.4.17...
3       Connected to mail7.imar.ro.
4       Escape character is '^]'.
5       220 mail7.imar.ro ESMTP Postfix
6       HELO <nume host>
7       250 mail7.imar.ro
8       MAIL FROM: <sender's email address>
9       250 2.1.0 Ok
10      RCPT TO: <receiver's email address>
11      250 2.1.5 Ok
12      DATA
13      354 End data with <CR><LF>.<CR><LF>
14      Subject: test
15
16      Acesta este corpul mailului. Textul trebuie incheiat cu un "." singur pe linie, ca mai jos.
17      Subiectul trebuie sa fie pe prima linie dupa raspunsul serverului la comanda DATA si contine
18      cuvantul cheie "Subject:"
19
20      .
21      250 2.0.0 Ok: queued as ABC6B6A022D
22      QUIT
23      221 2.0.0 Bye
24      Connection closed by foreign host.
```

# Exemplu HTTP

- telnet <host> 80
- apoi comanda

  GET / HTTP/1.1 (sau GET /index.html HTTP/1.1)

  Host: <host>

  <cr/lf>

  <cr/lf>

- telnet fmi.unibuc.ro 80 ?
- telnet www.imar.ro 80 ?

# Alte exemple simple

- *daytime*, *echo*, *time*
- comenzile de tip *telnet host port* emuleaza activitatea clientului
- pentru partea de server, momentan folosim servicii predefinite
  - ulterior, vom scrie propriile programe server
- pentru a porni serverele standard am apelat la *inetd/xinetd*
  - server cu structura particulara care multiplexeaza mai multe servicii diferite
  - serviciile multiplexate de server se configureaza cu ajutorul unor fisiere de configurare aflate in /etc/xinetd.d, specifice fiecarui serviciu
  - in general, serverele au fisiere de configurare, clientii au doar parametri de apel in linia de comanda
    - diferenta e dictata de complexitatea diferita a celor doua tipuri de programe
- serviciile standard in Linux se gestioneaza cu ajutorul comenzii *service* (necesita drepturi de administrator pentru a fi executata)

  service <serviciu> <start|stop|reload>

- dupa orice modificare a fisierului de configurare al unui server, e necesara “reincarcarea” serviciului (service reload) pentru a fi luate in considerare modificarile