

Sisteme de operare

Laborator 11

Semestrul I 2024-2025

Laborator 11

- comunicare inter-procese (IPC)
- programare client-server TCP/IP
 - clienti TCP
 - clienti UDP

Context

- perspectiva de sistem de operare asupra unor servicii de comunicare inter-proces (IPC)
 - *pipes/FIFOs/cozi de mesaje SysV*
 - *sockets*
- diferenta de paradigma: *shared memory* vs *message passing*
 - *pipe* – construiește un canal de comunicare în memoria kernel (partajată) a aceleiași mașini
 - => consecințe: kernelul asigură sincronizarea proceselor care scriu/citesc din *pipe*, la fel ca și politica de acces la date FIFO
 - *socket* – construiește un canal de comunicare între două mașini diferite legate între ele printr-o rețea de comunicare
 - conectează două endpoint-uri, i.e. perechi (adresa IP, port), o pereche pentru client, o pereche pentru server
 - => consecințe: complexitatea operării rețelei de comunicare (care poate implica implicarea mai multor mașini intermediare pt. efectuarea schimbului de mesaje) impune folosirea unor *protocoale de comunicare*

Ex protocol de comunicatie: TCP/IP

- asambleaza/dezasambleaza mesajele in pachete pt a putea fi transmise peste canale de comunicatie eterogene si asigura livrarea lor in ordine la destinatie (TCP)
- 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 (IP)
- asigura retransmisia automata a pachetelor pierdute/defecte (TCP)
- controleaza congestia retelei (TCP)
- samd

Protocoale de comunicatie

- modelul OSI (Open Systems Interconnection)
 - 7 nivele: *fizic, data link, retea, transport, sesiune, prezentare, aplicatie*
- model alternativ, protocoalele internet (a.k.a. protocoalele Department of Defense, DoD):
 - combina ultimele trei nivele OSI intr-un singur nivel: *aplicatie*
 - combina nivelul fizic si data link intr-un singur nivel: *link*
- conceptual, functioneaza pe principiul *stivei de protocoale*: protocolul de nivel cel mai inalt (*aplicatie*) apeleaza la serviciile protocolului imediat anterior (*prezentare*), samd
 - la nivelul cel mai de jos, datele trimise de protocoalele *data link* sunt trimise peste reseaua fizica
 - la receptie se parcurge calea inversa
- exemple protocoale:
 - data link (retele LAN): Ethernet (CSMA/CD), Token Ring, ATM, Wireless
 - retea: IP
 - transport de date: TCP, UDP, SCTP
 - sesiune: RPC
 - prezentarea datelor: XDR, ASN.1, criptare, compresie
 - aplicatie: DNS, SMTP, HTTP, FTP, SSH, etc

Protocoale de comunicatie (cont.)

- curs de retele de calculatoare: studiul intregii stive de protocoale OSI/internet
- laborator SO: 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 ?

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 apelam 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 gestionate cu comanda *service* (necesita drepturi de root)

\$ service <serviciu> <start | stop | reload>
- modificarile fisierelor de configurare ale serverelor sunt luate in considerare doar dupa “reincarcarea” serviciului (*service reload*)
 - alternativa: *inotify*