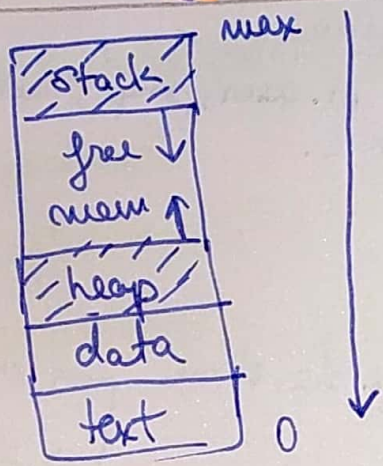
Curs 6 (Sisteme de operare)

HD Program → devine → proces (când este încărcat în memorie)

Întrebări: 1) fork() : procesul copie = clonă a părintelui
! 2) Cum pot comunica procesele între ele? (IPC)

mecanisme: → shared memory
→ message passing (kernelul realiz. managementul cozii (message queue))

Procesul în memorie → reprez.



mecanismul → producător consumator
↳ coadă de obiecte

mecanisme blocking / non-blocking

javascript → realiz. acțiuni într-o manieră care nu este blocantă
apel pt. c, java.

→ comunicarea cu input/output foarte lentă → soluție

Buffering [optimizare a mecanismului de scriere (=)
așteaptă acumularea de informații (Chunking)
↓
i.e. nu scrie literă cu literă

golire la forțată
golirea buffer-ului + scriere → flush

⇒ lucrul cu rețeaua, hd, input-output ⇒ lente ⇒ chunks

Obs
* operația de închidere a unui fișier ⇒ poate fi mai lentă
pt. că ⟶ s.o. realiz. flush înainte (se asigură că buffer-ul a fost copiat complet)

Data trecută: http, https, broadcast, blocking, nonblocking.

IPC System ⟶ POSIX

• POSIX shared mem
0666 ⟶ RW pt. user, grup, restul } drepturi
0777 ⟶ + exec.
↙
baza 8
0x ⟶ baza 16

⟶ Crearea fiș. de shm ⟶ shm_fd (nume, flag, drepturi)
⟶ ftruncate (descriptor, size)
↓
pe Linux NU < 4KB
sprintf (shmptr., format, mesaj)

C ⟶ nu are management al memoriei
↳ garbage collection (python, java, ~~mat~~ .net)

* string \0
atenție la terminatorul de string
↳ când citim fișierul verif. ~~dimensiuni~~ dc.
l-am parc. în întregime ⟶ dimensiunea fiș
↓ din
header.

pointer. poz. pt. ⟶ date binare
unde poate să apară \0

IPC Windows : RPC (LPC)
remote / local procedure call

⇒ + la nivel de rețea
→ port de comunicare

Socket = IP address + Port
⇔ IPC pe rețea ; endpoint

127.0.0.1 (loopback)

IPv4, IPv6

http: 80
https: 443
ssh: 22
ftp: 20; 21
serv. de mail: 25
smtp

≈ 80 particule în univers.

TCP → se asigură că toate pachetele au fost trimise
UDP → connectionless protocol
↳ „fire-and-forget"
↳ mai rapid, dar pachetele pierdute nu sunt recuperate

connection-oriented.

Transmission Control Protocol / User Datagram Protocol.

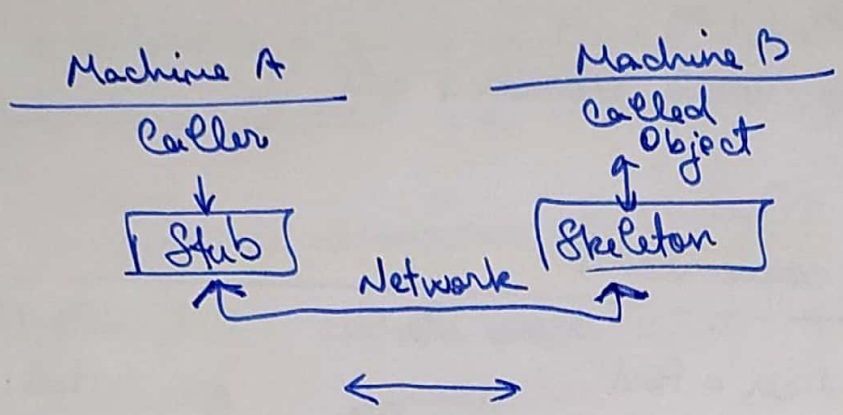
Tipuri de socket:
- connection-oriented (TCP)
- connectionless (UDP)
- multicastSocket → multiple recipients

Remote Procedure Call

→ un client poate exec. o funct. care se află pe un alt calc.

Stub → client-side proxy al procedurii pe server

Marshalling ⇔ **serialization** ⇒ oferă unui obiect din memorie un format care să îi faciliteze transmisia într-o rețea
↓
json
pickle ⇒ python



RPC port = 135

Big-endian : most significant bit first → MSB

Little-endian : least significant bit first → LSB

Pipe-uri

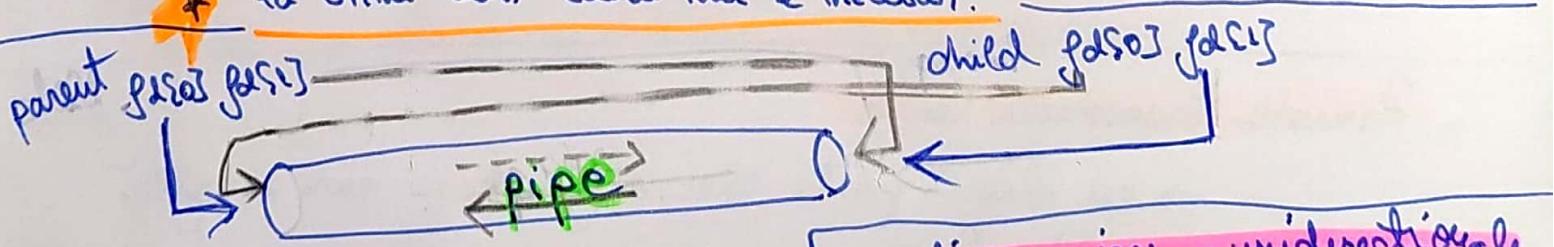
→ copilul transmite părintelui **return value** — îl anunță ca s-a încheiat

→ dc. vrem să transmitem astfel mai multe date ↓ pipe

→ tubele procese comunică prin **pipe**. → coadă de mesaje

care sunt **forkuri** între ele

* la shm acest lucru nu e necesar.



2 fișiere ~~din care~~ → {write / read

↓

2 fd.

named-pipes = bidirecționale

ordinary pipes = unidirecționale

producător — consumator

↓ write end — ↓ read-end.

parent-child

fork

* ordinary pipes → unul scrie, unul citește

named pipes → la citire pointerul este modificat → atenție la comunicarea eronată.

intuitiv
↳ coadă de comunicare full-duplex

half duplex : ambele citesc / scriu → pe rând
full duplex : ——// —— → simultan

Threads

multithread → mai multe fire de exec. → paralelizare

ex: spellchecking.

client → server → thread
dispatcher
worker

→ fiecare thread are copiile regist. + stack individuale
↳ restul datelor progr. este comun tuturor threadurilor

threaduri → + optimizare a memoriei (pt.că) → un singur proces
+ procesarea este paralelă

~~pro~~ fork → clonă a datelor, memoria se copiază