

Curs 2 → (au trept)

Cât timp un fișier e pe hdd → = fișier
→ pt. a fi exec:

↓
e încărcat în RAM → devine
= **PROCES**

system interrupts =

un dispoz. periferic
o comp. hardware
transmite CPU-ului că are nevoie de atenția sa
↓
0,1% - 0,5% CPU usage.

→ polling ^{CPU}: înseamnă teate dispoz. dacă necesita
attenția lui

frecvența RPM
3300
3333 MHz

- vectorized

are un tabel cu instructionuri

pt. ce se întâmplă
când e activat fișier
periferic

DRIVER

↳ identifică hardwareul

locul unde se salvează în software

- disp nou \rightarrow driver \Rightarrow înreg. în lista de disp.
conectate la PCI

driver = spune CPU-ului cum să
comunice cu hardware-ul conectat

firmware \Rightarrow BIOS / UEFI

low-level software

shared mem.

DMA = direct memory access
~~RAM -> direct access~~

von Neumann
architecture

o altă linie bus HDD comunica direct cu RAM
pe place de baza

VS
Harvard
architect.

↳ disp. comun. simultan pe bus \Rightarrow
bottleneck : \downarrow
↳ queuing

↓
se pun la coadă
separate storage

PCI \rightarrow bus principal

bus controller \rightarrow tip
tehnologie
highspeed

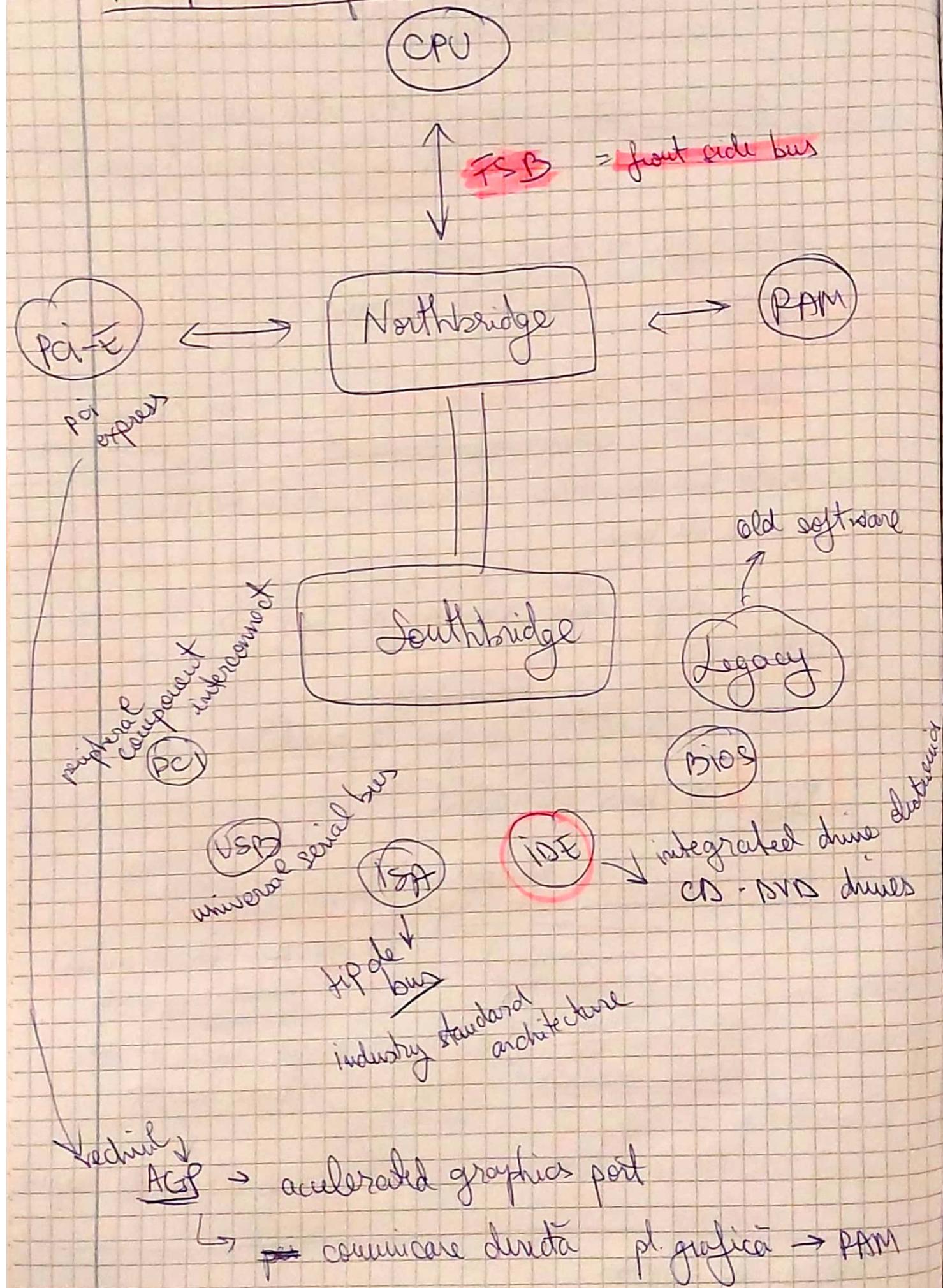
PCI express GPU
SATA (hard drives)
hot-swappable HHD

busuri paralele { North bridge? \Rightarrow să ocupe CPU-ul
 South bridge? mai mult
 \downarrow I/O ... \downarrow acces extrem de rapid

PCIe vs SATA

> rapid \rightarrow conectarea direct la placă
 & consum mai scăzut (de obicei fol la pl. grafica)

Arh. placă debogă



sistem call → comunicare cu OS
↳ forma de interrupere

device-status-table → type
 state
 address → driver

nivele de cache
→ cel mai rapid

l1 → cache

l2 → Roata

l3 → core-wire

↓ important

intre mai multe procesorii (CPU)

= preluarea ~~păstrării~~ în memorie
să precizeze ce date are noului CPU - ue
în viitor și le stocăază → ~~stocare~~
→ mai rapid, decât să cante mereu în
RAM.

⇒ 2 matrice

$\begin{matrix} i & j & k \\ j & i & r \end{matrix}$ → mai rapid → fol. cache-
eficient

vit-calcul.
lent → face saltele mari
în memorie

$i+j$

mai eficient decât
rapid

$j+i$

ob. se increme.

si se ret

se crează o var. temp.

returnată ⇒ după e increm.

HDD de 1000 ori mai lent decât RAM

SSD — 100 — — —

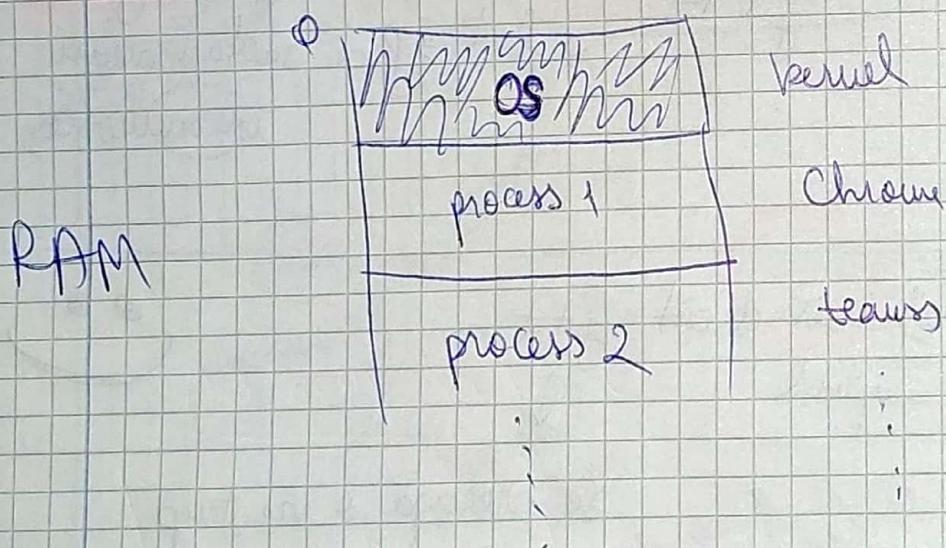
→ Symmetric Multiprocessing

2 Asymmetric . . . (dc. au CPU si GPU , de ex)

CPU → machine Learning
→ cad (computer-aided design)
(proiectare asistata pe calculator)

→ mining

OpenCL → system hibrid



Sixap

MS DOS

black / blue screen ⇒ bopcode greșit în assembly
↓
panic message către OS

Curs 3 (an treut)

Q3x_t read (int fd, void* buf, size_t count)

signed

max-name

tabel de syscall

system call interface API

Hello
Top

file descriptor = int care identifică unu un
fișier deschis într-un OS \leftrightarrow id

prea multe arg pt syscall \rightarrow cream o struct pe care o pasăm
memoriei pe stivă

push după
pop

ca la Risc

SCARPG

\rightarrow la syscall pasăm * căre

parcurg
pt.
Struct.
num.

Mid layer în kernel

! disp. hard sunt răspite drept fișiere !

\rightarrow la windows ~~fișier~~ syscall sep. pt. fișiere

se
fișier normale

ex.: fis. imprimentă → o serie în el = o comunica
cu dineroare disp.

→ a trimit ^{de date} bytes la impr.

pipelining

partajare a memoriei → mai multe procese
au acces la un fisier deschis în
memorie

→ utilizarea sist. operare.

std.in

⇒ terminalul

std.out

std.error

Disk - Op. Sys

MS - DOS

libc

- os fără memoria

wind 98 pornea peste MS-DOS

segmentation fault → încearcă să accesezi adresa
inaccesibil.

round-robin

listă circulară de procese cu priorități
arbore
divide în subprocese

în lista parc. unu slui. arbitr. nu e ef.
în vector e

↓
tot parcurg tot pără astăzii

⇒ • pentru suntare e ef. arborele

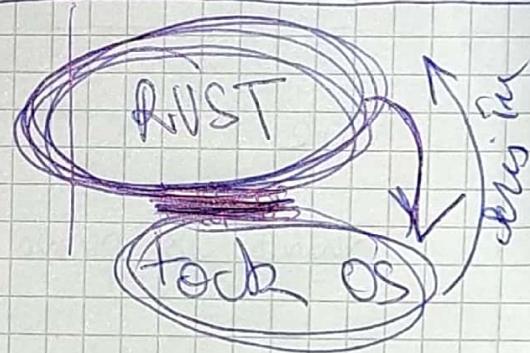
? CPU → core, multiprocess

{ Policy → cum să facem
Mechanism → cum facem. } user, group, others
 rwx

chmod

rwx rwx rwx
rwx rwx rwx
+ + +

Mag. & op. sunt în C.
+ assembly
~~nu e bine să căuta~~



memory management (nu are verificarea acelașului memoria)

Rust
python, java

| nu are aceeași verificare.

Seg-fault → OS. omoară procesul.

"cel mai sigur sistem este cel powered off." :))

driverile sunt în kernel = modul
L pot fi de nu
sunt imp. C nu comunică cu CPU) doar kernel
supraveghere

Mach microkernel → minimalist, căt mai mult
cum cei progs. prin message passing

? self? nu pică nicio dată
demonstrat sigur formal
→ soft. foarte mult (discursiv)

Solaris OS
(SUN Microsystems)
→ cumpărat de ORACLE

Mac OS X

+ BSD
Mach # ~~BSD~~
pr kernel

că în kernel se modulă
Android runtimes

Dalvik bytecode
java modif.

Dalvik
Virtual machine

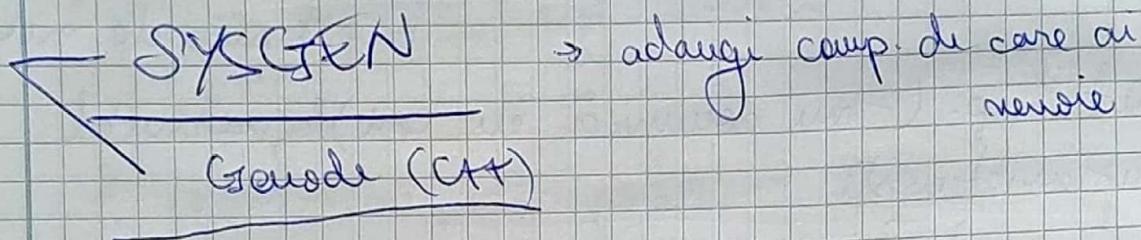
infectant

Linux kernel

Andr. → apl. nativ exec → exec pe

kernelul de Linux

cod c compilat în bibliotecă



EEPROM → ~~locates~~ the kernel

Grub - bootstrap loader locates the kernel
↓ &
loads it into RAM

→ redirectare, terminalul Linux

Laboratorul 4 (an trei)
(3)

\$ în shelluri = variabili

echo \$variable

Curs 4

pg. C \rightarrow python compilat \rightarrow pt. rapiditate

Web assembly \rightarrow apl. web. compilata

Procese

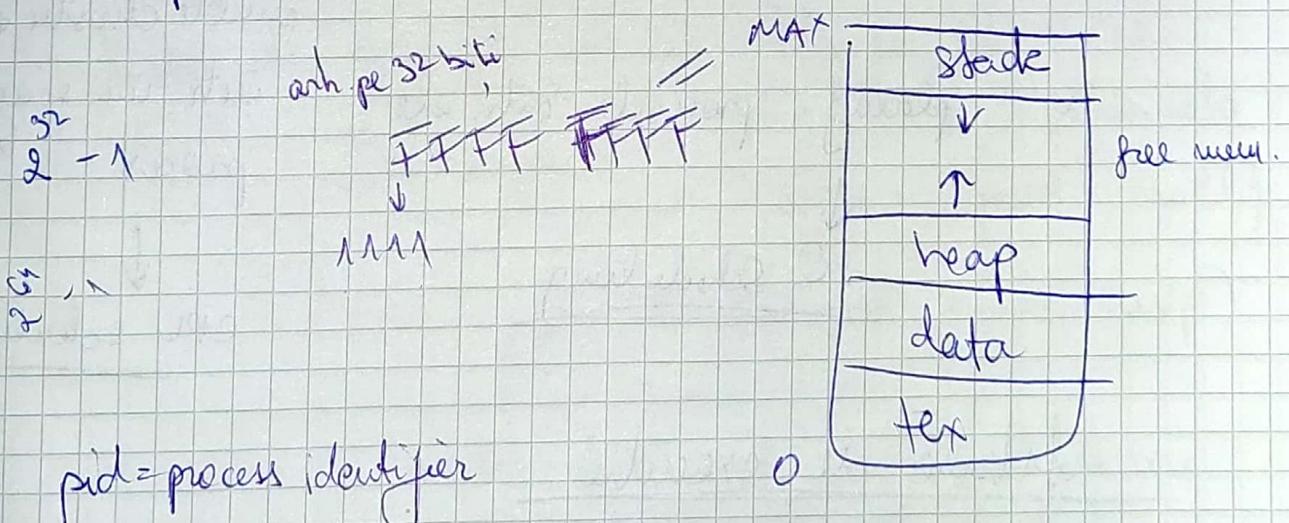
= un progr. aflat în execuție (e în RAM)

program code, text section

program counter = ult. instr. exec.

stack \rightarrow var. locale aloc. statică \rightarrow mem. mica

heap \rightarrow aloc. dinamică \rightarrow mem. mare



pid = process identifier

Kill -9 pid \Rightarrow suicid garantat

\downarrow
fără flag \rightarrow poate să nu fie inclus procesul

std.in
std.out
std.err

↳ fixare mereu deschise

un proces rulășește pe CPU

memoria RAM îne → salvarea datelor procesului

în timp ce unu. CPU-ul este

benchmarking → blochează progr. nefol.

pt. ca să calculeze performanța cum trebuie

un CPU execută o dată un singur proces

→ proces pe mai multe threaduri → CPU rul. mai
multe chestii simultan
într-un singur proces

SO planif. procesele între ele

↓
Scheduling

↓
CPU scheduling

Algoritme de execuție

→ pipelining

→ data hazards

Process control block PCB

time ls

→ cât durează un proces

numpy = bibliotecă din pytho care face operații array-urilor eficiente.

prin calc. conținutul în se ordinează nr. reale
IEEE st. → ad. (iii) este associativă

mp. sum

Laborator 4 | (5 puncte)

listă de arg. pt. execuție tbs. să fie NULL

↓
dătă că par. terminată
args - h "pwd", Null}

execve(path, args, envp)

nu tbs. să am o să căte argumente are
7 null și să nu înceapă cu underscore.

execve → înlocuști procesul apelant

cu procesul obținut ca parametru.

⇒ suprascrie

Poate există după (iii) se execută

Curs 5

instruction pointer
program counter

PCB

process control block

new
ready
waiting
running
terminated

CPU / context / state switch



init

pid = 1

! dc. exec dă eroare → controlul revine
progr. nostru initial
sau este acă

+ intarice flag - 1, sau
alterea nr. fără
 $\neq 0$

`fprintf(stderr, "Fork failed");`

!

thread pool

gcc main.c -pthread -w -o main

↓
fără management

htop

↔ task manager

pthread-create
pthread-join
pthread-cancel

* (int *) arg
detachable ↓
castere
+ după

nu pot accesa din exterior o variabilă locală

↓
segmentation fault

Struct

clone

→ nu fork mai cu mai

multi optimi

păstrează spațiul de adrese
pe asta se bazează

implementarea threadere,

2. 3 + 3

J

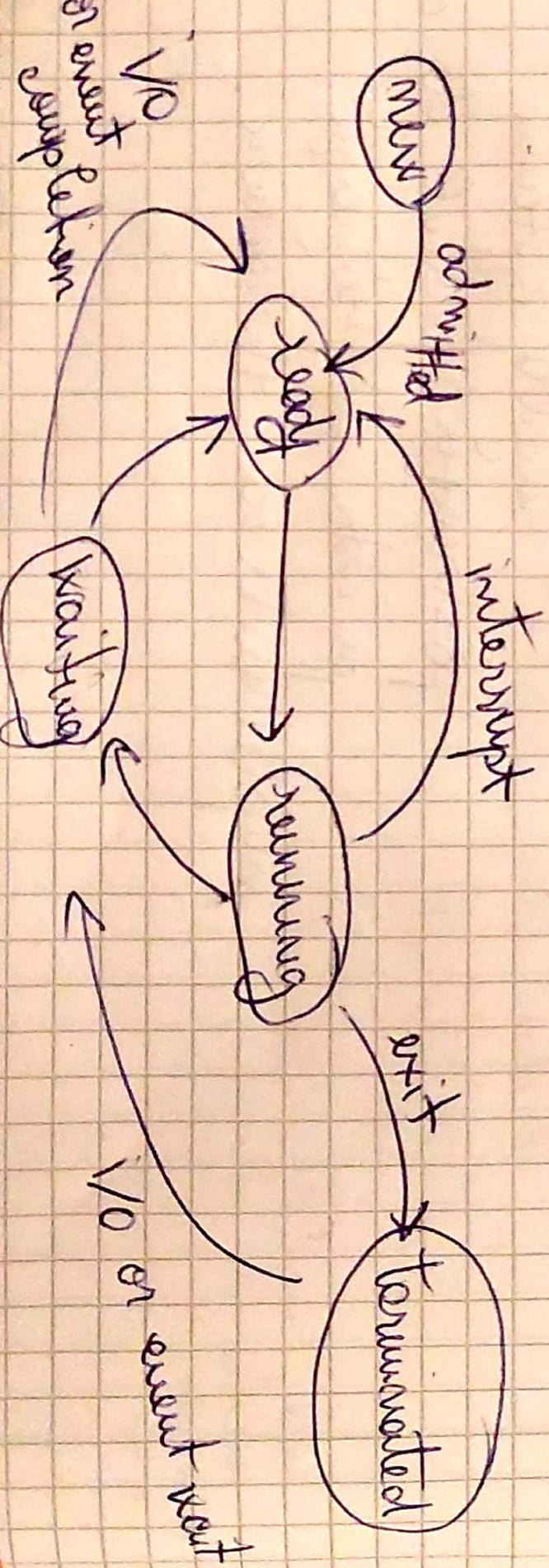
Processor states

new

ready

running

terminated

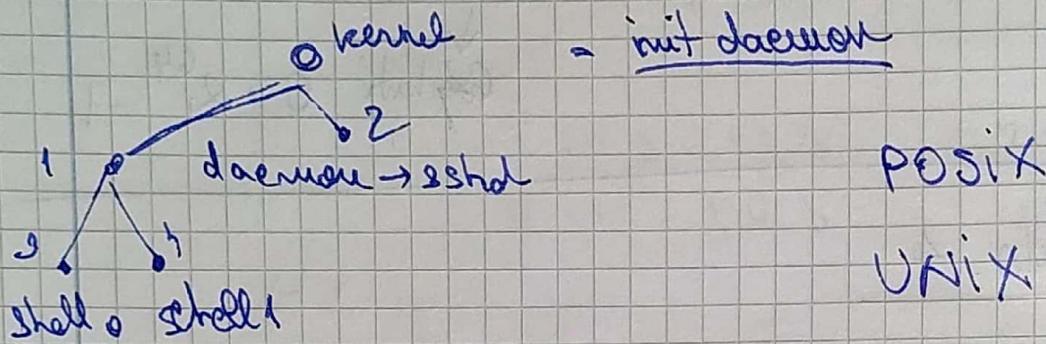


Curs 5

FP → file pointer → fizicele deschise

→ un proces nu poate accesa memoria unui alt proces

* CPU / context / state switch.



execvp → procesul copil devine programul assignat
= execve

În procesul parinte ⇒ fork return pidul copilului
copil ⇒ fork return 0

? wait(& variabilă int) → vedem codul de return
~~stări~~) al copilului

exit → închidere copil

abort → ucidere copil

Curso 5

FIFO → file pointer → fizicale descriere

Proces zombie = copie fără parințe care
→ copiază acțiunile unui alt proces
să iedea

→ nu poate avea memoria unui alt proces

* CPU / context / state switch.

○ kernel - init daemon

1. daemon → shell

2. shell

3. shell

UNIX

execvp → prezintă copia derivate programul original
= execve

În procese copioase ⇒ fără nume

pidul
copiilor

copie → fără nume

0

? wait (& variabile int) → vedere codul de nume
~~stări~~) → acopiator

exit → finalizează copie
abort → uideaza copie

→ copierea unei acțiuni nu are un
loc așteptă copierii
- și le uidează
- copie orphane → parinților ei fermii. Numai ca
dăruind să interacționeze
cascading termination

↓
operă procese fără copii nând pe nând

Windows
+
end process tree

end process tree

Multiprocesor Arhitectură - Chiară Procesor

→ fizicale tab = proces diferenți

→ nume în spatele → Chiară acțiune din

fizicale tab
nume ale numelor diferențiate

↓
securitate

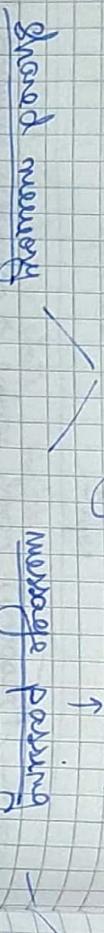
? coherență nu sunt copiate

Jahresprozess Kommunikation (ipc)

prose → independent

→ compensation
four purpose

send (message)
receive (message) |
 | => functions



cond. / (Lab) data w/ message of diet op.

procurar um novo
é tudo o que posso

+ mai existe si alte metode

Measuring Production - Consumption

* *Adonis* (also *Franseria*)

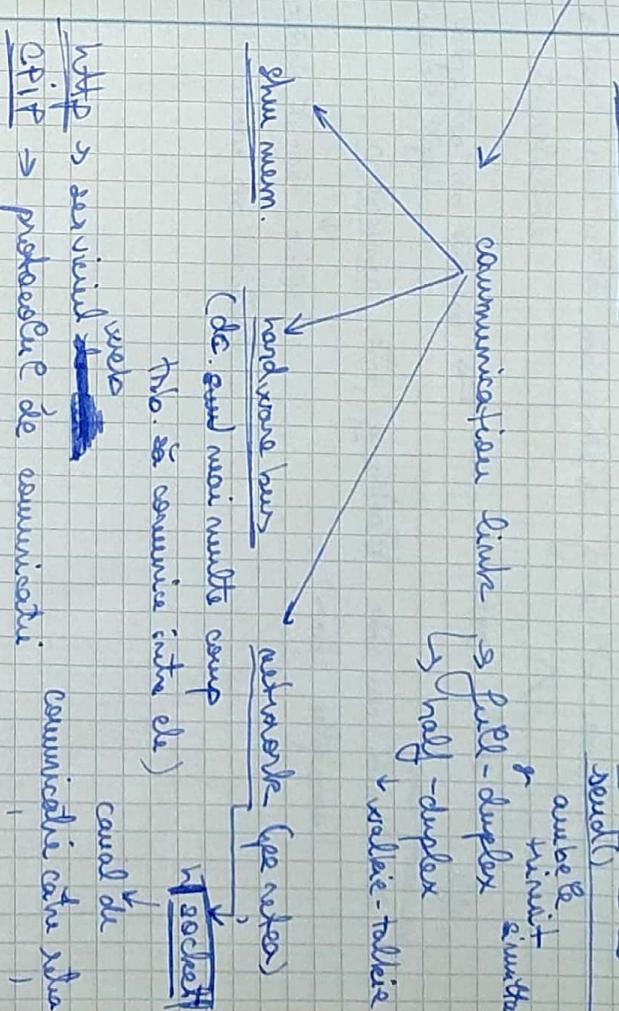
1000

ABOS
district budget showing ad
7-1-19

buffer infusion \Rightarrow circulation

Sound longer → slow move. □.□.□.□.□.

Während et. = Referenten; daher Preise



Direct Communication

pure \rightarrow pick (= ∞ address)

Indirect Communication

→ main
Index
as we easy
be

PEACE

8888
0808
5555
KST

public → https

public → https
443

präzugsrichtung
peripherie des hof

Cine primește mesajul?

broadcast

Sincronitate

→ blocking → până ^{nu} a primit → blocat

→ non-blocking → asincron → trece mai departe
indiferent

send receive

afil
în același cot

⇒ citirea nu este blocking, este asincronă

(exemplu curs)

meniu nu este apărat
deși meniu

nu e blocat, de către
deci nu are nimic de făcut