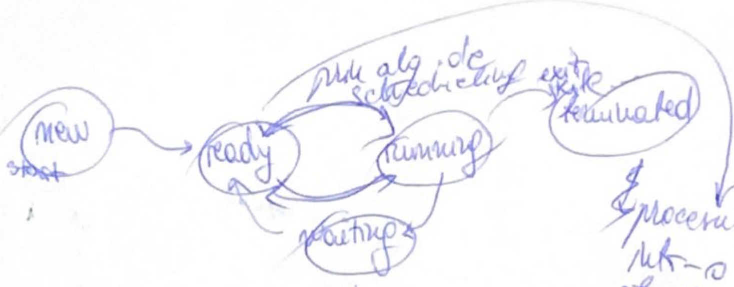
19. 01. 2024

Sisteme de operare
curs 14

Recapitulare

ch3, 3.8. procese



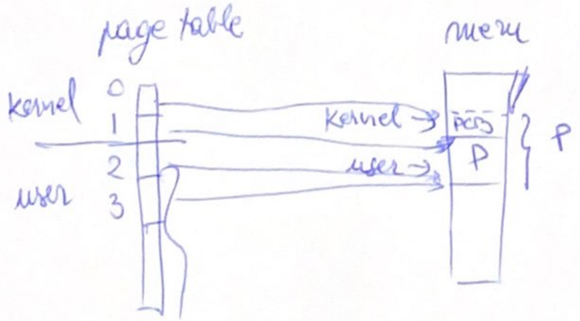
procesul e pus într-o coadă pt procesor

semafor cu wait

procese ce pot fi întrerupte preemptive
/non-preemptive

Alg. de citire pe disc : SCAN
Look
CLook

3.9 PCB : toate datele despre un proces



3.11 fire de execuție = Threads
un fel de procese, dar ;

P_0^0 pthread create

$\mathcal{D}_0$, T_0, EIP_0 (PC_0), $stack_0$
$\mathcal{D}_1$, T_1, EIP_1 (PC_1, $stack_1$

program counter

partaj. heap-ul
stiva nu e aceeași

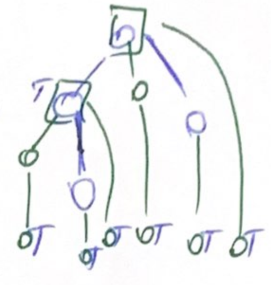
3.22 arborescența de procese

fork()

alt program: exec()
↳ transcrie procesul

EXEMPLU 1

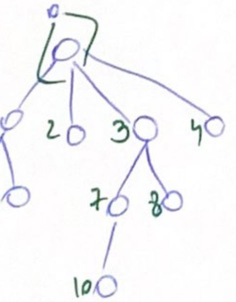
```
for (i=0, i<2; i++) {
  if (!fork())  ←
     pthread_create(); T copilul
  if (i==1)
     break;
  if (fork())    Toată lumea
     fork();  ──────→ pid-ul copilului creat 27
}
```



~~i=1~~

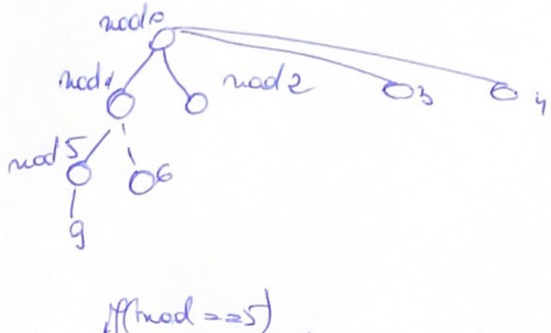
∇ am presupus ca fork ul nu copiază thread-urile părintelui.
cu T urile
O

NOD=0 EXEMPLU 2:



~~for(i=0; i<4; i++)~~
~~if(!fork())~~

```
nod=0;
{ if(nod==0)
     if(!fork())
        nod=1;
  if(nod==0)
     if(!fork())
        nod=2;
  if(nod==1)
     if(!fork())
        nod=5
  if(nod==1)
     if(!fork())
        nod=6
```



```
if(nod==5)
   if(!fork())
      nod=9
```

5.7 Race condition

MUTEX
SEMAFOR
BARIERE.

```
do {
    entry
    CS
    exit
    remainder
} while (true)
```

excluziune mutuală

progres → ast. multi-ș alegerea celui

așteptare finita

↳ fiecare proces va ajunge în zona critică într-un timp FINIT

care intră are loc în timp finit

Dacă un proces așteaptă ∞ => STARVATION

test_and_set 5.18 → întoarce val veche schimbă val. cu TRUE?

compare_and_swap 5.20 ↳ similar, dar pt. întregi

5.19

5.25 semafoare
deadlock → toate proc. sunt blocate

5.22 implem. cu timp finit de așteptare

6.10 CPU scheduling SJF,

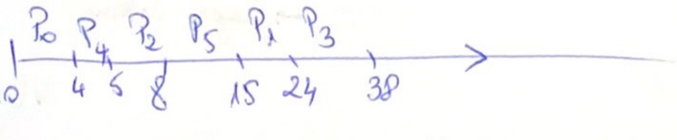
prioritatea e doar pentru departajare

EXEMPLU 1

Proces	Arrival	Burst	Priority
P_0	0	4	3
P_1	2	9	5
P_2	4	3	1
P_3	2	14	2
P_4	3	1	4
P_5	6	7	3

SJF non preemptive = nu întrerupem pe nimeni

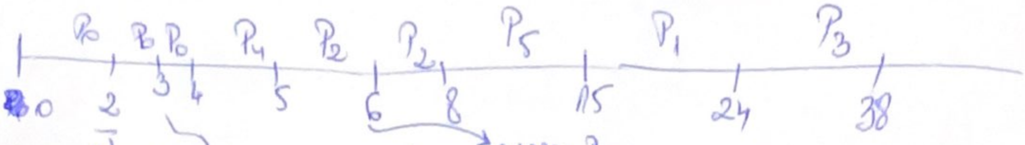
Diag. Gant cu ordinea în care se vor executa:



Timpul mediu de așt: dif dintre Arrival și când intră (medie)

0+1+ __ +

· acelaşi ex, dar preemptive ;



ajunge P1 şi P3 9, 14

P4 vine cu 4 P0 are 3 → ~~P4~~ P0

apare P5 dar cont. cu P2

P0 mai are 2

---

ch8 ; Main Memory ; segmentare

fragmentare < ext / int

8.32 schema

8.34 paginarea

4 3 | 2 1 0
p | d

Exemplu! 4 3 | 1 0 p | d procesor pe 5 biţi

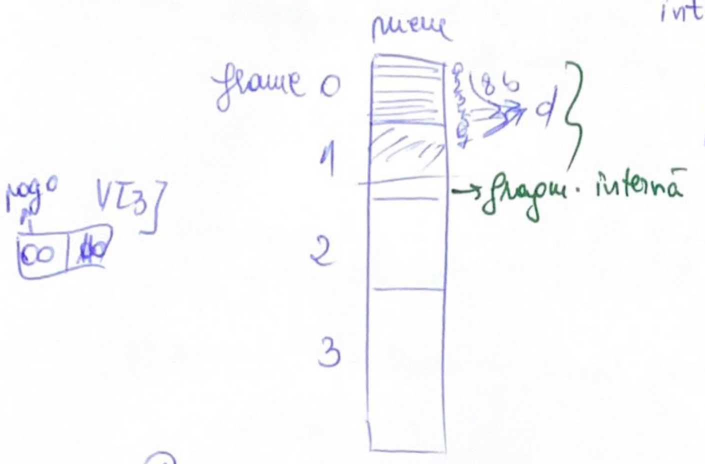
# pagini → 4 00 01 10 11

dim. unei pag. → 8 000 ⋮ 111

dim. memoriei totale → bytes

$2^5 = 4 \times 8 = 32$ bytes !

Cum arată?



int v[6]

size of(int) = 2

6×2 = 12 bytes

1 pg = 8 ⇒ avem nevoie de 2 pag.

⇒ 16 bytes

V[4] este în al doilea frame.

0 1 2 3 4

ptd → | 01 | 000 |

Putem găsi orice v[ ] în acest mod ↗

10.20 FCFS

10.21 SSTF

10.23 SCAN își păstr. un sens → liftul

10.25 C-SCAN

10.27 CLOOK (CONT)
LIFT.

EXEMPLU:

disk, 3000 cilindri

coada: 501, 42, 2023, 2999, 1024

head: 500

prev: 314 (înainte de 500)

FCFS:

$$(501 - 500) + (501 - 42) + (\overset{2023-42}{\cancel{2023-2999}}) + \ldots$$

SCAN:

314 → 500

501 → 1024 → 2023 → 2999 → 42 →

$$(501-500) + (1024-501) + \cancel{(202} \ldots + (\cancel{2999}-42)$$

$$(\cancel{1+300}\ 2999-42)$$

ch 12 12.19.

examen: - cu materiale

- nu capcane/trucuri

2.01.2024

12 ian
curs 13

Curs Sisteme de operare

s.urm. Recapitulare

- ch. 9 virtual memory
  +8 main memory

pag + segm.

→ 8.32 al câtelea segm ⇒ tabela de segmente limit base

cpu | s | d |

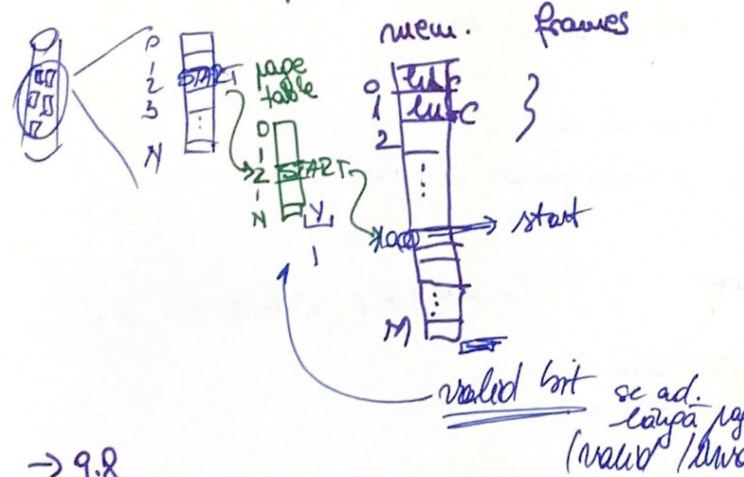
↳ deplasament ⊕ la bază

→ 8.36 8.35

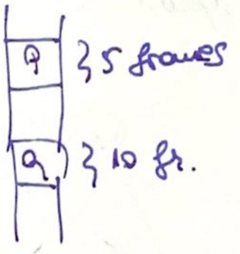
paginare : proces în blocuri egale

memoria virtuală → lene

Desch. o apl : main-ul / entry point / rutina de start



Intel
encodarea unei instr.
1 byte → 20 byte



→ 9.8

→ 9.9 putem refolosi

Q → copie pt. P → Q e un proces cu un pointer către aceeași tabelă de pagini până când se apucă să modif.

fork()

→ 9.10 lazy swapper → nu șt. tabla

pag. read only ⇒ nu o mai ducem înapoi pe disk

dirty bit care spune dacă s-a scris în fișier

9.15 schema compl. Ne putem da seama de [illegible]

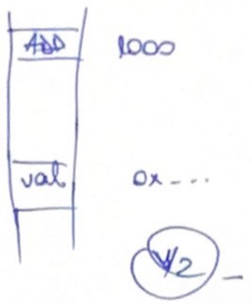
→ tabela de pagini e ținută în PCB al procesului

9.17 → o instr. acc. mai multe pagini ↴

```
1000 : JMP label(loop)
   | dist > 4k
loop :
5000 :
   instr.
   instr
   instr
```

V1

ADD EAX ,[0x DEADBEEF]



9.28 → page replacement (avem algoritmi simpli şi tâmpi)

9.32 ALG. FIFO slide 9.32

Belady's Anomaly

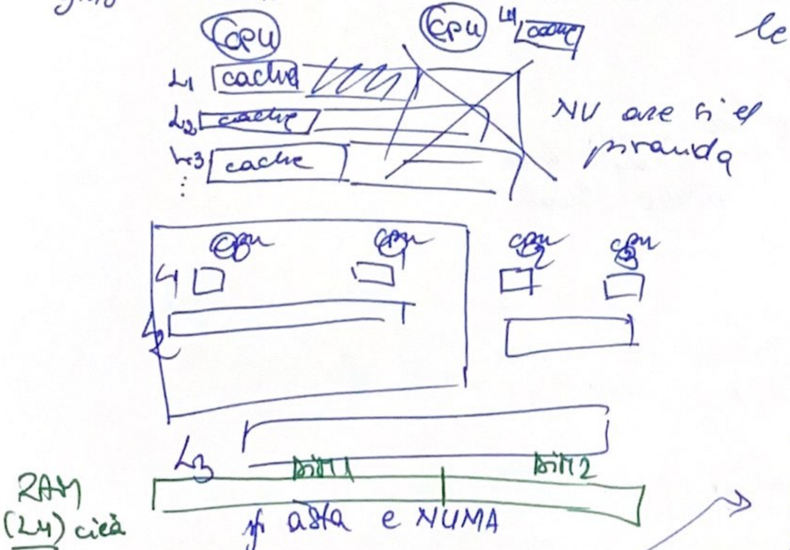
9.34 ALG OPTIM → vedem viitorul şi înloc. pagina ce va fi fol. cel mai târziu

↓ NU SE POATE ATINGE

9.35 LRU → ca optim doar ne uităm în urmă

9.47 → global replacement } luăm de la cn. nu fol. şi dăm unde e nevoie

9.48 NUMA



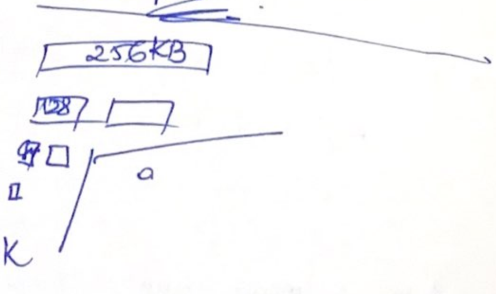
le dau L1 şi împ. L2, L3

NU are şi el prandă

şi asta e NUMA

P se duce pe CPU0 ⇒ va vrea tot pe CPU0

pe CPU3 se va comporta mai prost.



9.63 sist. BUDDY (arbore)

9.66 SLAB ALL. →

9.72 → examen.

128 × 128 data[i][j]

128 data[j][i]

each . row = one page

TLB