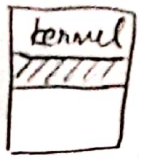
30.05.2019

Curs 13 - SO

(A) - monoutilizator / monotasking
- multiutilizator
  - partiții fixe
    - (B) • absolute
    - (C) • relocabile
  - (D) • partiții relocabile
- virtuală
  - (E) • paginată
  - (F) • segmentată
  - (G) • paginat-segm

A.



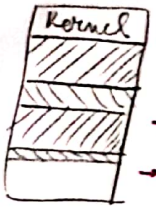
un singur proces o dată

B.

→ compilăm pt. fiecare partiție
→ un singur proces în partiție

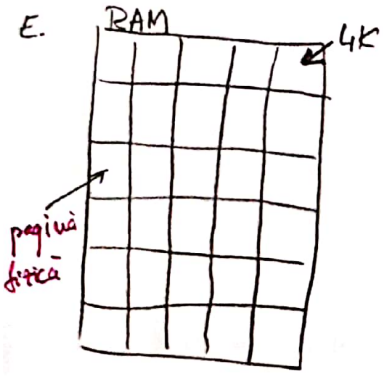
C relocăm — folosim offsetări

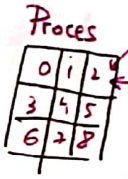
D.



→ alocăm cât spațiu îi trebuie fiecărui proces
→ apare fragmentarea
→ compactarea memoriei libere
- lentă
- câtă e necesară pt. următorul proces

E.





← pagină virtuală

Adresă : (pagină, offset în pagină)

Page table

0	19 · 4K
1	4K
2	22 · 4K
⋮	

F. Segmente ~~cu pa~~ similară cu partițiile variabile ca idee
- fără protecție de acces
- permit gruparea logică a codului
→ risc de fragmentare

G. Pag - segm.
- fiecare segment e împărțit în pagini

H. Flat Memory Model

↑UNDE încărcăm în memorie?

Scanned by CamScanner

Politici de încărcare

CE? CÂND?

— Se încarcă toate paginile procesului de la pornire
A: acces rapid la orice pagină în timpul rulării (totul e deja în RAM)
B: pornire lentă, memorie irosită

— Se încarcă fiecare pagină când e cerută
A: pornire rapidă folosire minimală de RAM
D: acces lent la paginile reîncărcate în timpul rulării

— Încărcare pe principiul vecinătății
- o pagină tocmai încărcată va avea nevoie ~~de~~ de paginile imediat următoare

Politicile de înlocuire

— ce facem când memoria e plină și avem nevoie să încărcăm o pagină?
=> unele pagini (victime) trebuie mutate în SWAP

CARE?

NRU - Not Recently Used

Fiecare pagină are 2 biți care descriu accesul la ea, bitul 0 devine 1 când se scrie în pagină, bitul 1 devine 1 când se citește din pagină. periodic biții sunt resetați la 0.

=> 4 clase de pagini:

0 0	- neatinse recent	1
0 1	- citită recent	3
1 0	- scrisă recent	2
1 1	- c+s recent	4

FIFO → prost

LRU - Least Recently Used

- presupunând că avem N pagini folosim o matrice de N x N biți.
- ori de câte ori o pagină e accesată, linia ei se populează cu 1 și apoi coloana ei cu zero

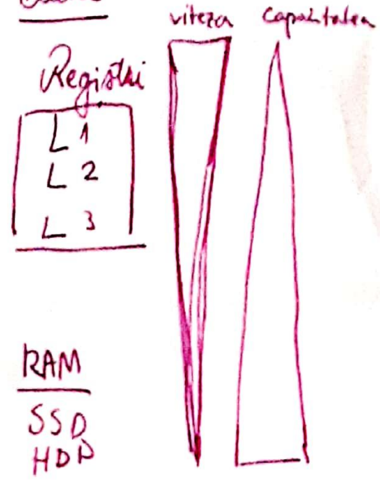
N = 4, secvență de acces pagină 0, 3, 0, 1, 2

0 0 0 0	0 1 1 1	0 1 1 0	0 1 1 1	0 0 1 1	0 0 0 1
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	1 0 1 1	1 0 0 1
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	1 1 0 1
0 0 0 0	0 0 0 0	1 1 1 0	0 1 1 0	0 0 1 0	0 0 0 0

Paginile victimă se aleg în ordinea crescătoare a sumei liniilor respective

Scanned by CamScanner

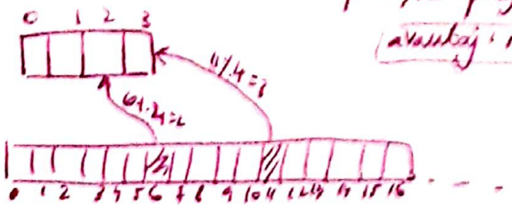
Cache



Cache-ul va conține pagini

Unde stocăm în cache o pagină, astfel încât accesul la ea să fie cât mai rapid?

→ proiecție directă ~ dimensiune cache = c pagini
~ dimensiune RAM = n pagini
~ poziția paginii k în cache = k % c
avantaj: rapid, dezavantaj: cache threshing



0, 3, 1, 7, 11, 0, 2, 19, 27

cache threshing

→ stocăm pagina în primul loc disponibil
A: threshing redus
D: ~~[illegible]~~ lent

cache asociativ

→ b Bănci de pagini (cache set asociativ)

pagina k merge în banca k % B și în bancă se [illegible],