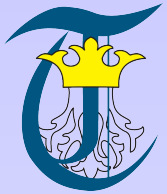

Universitatea *Transilvania* Brașov
Facultatea de Inginerie Electrică și Știința Calculatoarelor
Departamentul de Electronică și Calculatoare

Sisteme de operare

– curs 9 –

2012.04.24

ș.l. dr. ing. Kertész Csaba-Zoltán

3.4. Algoritmi de înlocuire a paginilor

- la apariția unui page fault SO trebuie să decidă care pagină va fi înlocuită în memorie de către noua pagină adusă de pe disc
- SO va trebui să rescrie pagina din memorie pe disc, dacă aceasta a fost modificată
  - dacă nu a fost modificată nu mai este nevoie de rescriere (de exemplu pentru zona text)

3.4.1. Algoritmul de înlocuire optim

- cel mai bun algoritm de înlocuire a paginilor
  - este foarte ușor de descris, dar foarte greu de implementat
- presupunem că la apariția unui page fault, în memorie se află câteva pagini
- una din aceste pagini va fi referită de următoarea instrucțiune, celelalte pagini nu vor fi referite decât după execuția a unui număr mare de instrucțiuni
- fiecare pagină poate fi etichetată cu numărul de instrucțiuni până când va fi referită

Algoritmul optim

- algoritmul efectiv:
  - pagina având eticheta cea mai mare va fi înlocuit
- acest algoritm nu este realizabil
  - la momentul apariției unui page fault, SO nu poate să cunoască momentele la care vor fi referite celelalte pagini
- ar fi posibil implementarea algoritmului:
  - la prima rulare se înregistrează toate referințele la pagini
  - la a două rulare se folosesc informațiile colectate anterior

3.4.2. Algoritmul de înlocuire a paginilor Not Recently Used

- presupunem că există 2 biți asociați cu fiecare pagină:
  - R: este setat ori de câte ori pagina este referită
  - M: este setat când pagina este scrisă
- acești biți fac parte din fiecare linie a tabelei de pagini
- acești biți vor trebui actualizați la fiecare referire la memorie

Algoritmul NRU

- când se activează un proces, biții R și M sunt resetați de către SO
- periodic (la fiecare întrerupere de ceas) bitul R este resetat pentru a distinge paginile care au fost referite recent
- la apariția unui page fault, SO inspectează acești biți pentru fiecare pagină și împarte paginile în 4 clase:
    - 0: nereferite, nemodificate
    - 1: nereferite, modificate
    - 2: referite, nemodificate
    - 3: referite, modificate

Algoritmul NRU

- bitul M nu va fi resetat de SO la apariția întreruperii de ceas deoarece este necesar pentru a decide dacă se rescrie pagina pe disc sau nu
- algoritmul NRU înlocuiește o pagină aleasă aleator din clasa nevidă având cea mai mică etichetă
- este simplu, eficient de implementat
- asigură performanțe acceptabile

3.4.3. Algoritmul de înlocuire a paginilor First-In First-Out

- se înlocuiește pagina care a stat cel mai mult în memorie
- SO va menține o structură de tip coadă care conține paginile rezidente
  - la începutul cozii se află cea mai veche pagină din memorie, la sfârșit cea mai nouă
- la apariția unui page fault pagina cea mai veche va fi eliminată din coadă, iar pagina nouă va fi plasată la sfârșitul cozii
- pagina înlocuită poate fi folosită cel mai intens
- este foarte rar utilizat

3.4.4. Algoritmul de înlocuire a paginilor Second Chance

- este o îmbunătățire a algoritmului FIFO, care evită eliminarea din memorie a paginii utilizate cel mai frecvent
- se va inspecta bitul R
  - dacă R = 0, pagina va fi înlocuită imediat
  - dacă R = 1, bitul R se va reseta, iar pagina se va plasa la sfârșitul cozii (va fi tratată ca o pagină nouă)
- astfel în coadă paginile vor fi sortate în funcție de timpul de când au fost plasate în memorie

3.4.5. Algoritmul de înlocuire a paginilor Clock

- paginile sunt așezate într-o listă circulară
- acest algoritm vine la soluționarea problemei ineficienței FIFO și SC datorate mutării frecvente a paginilor de la începutul cozii la sfârșit
- la apariția page fault, se inspectează pagina curentă la care arată indicatoarea, dacă R=0 va fi înlocuită cu o nouă pagină pe același loc și indicatorul va fi avansat, dacă R=1, se resetează R, iar indicatorul avansează o poziție

3.4.6. Algoritmul de înlocuire a paginilor Least Recently Used

- paginile care au fost frecvent utilizate de ultimele instrucțiuni vor fi utilizate probabil și în continuare
- se va înlocui paginile care nu au fost utilizate de foarte mult timp
- se apropie cel mai mult de algoritmul ideal, dar implementarea este foarte grea
- pentru implementare este necesar să menținem o listă cu toate paginile din memorie, la început pagina cea mai recent utilizată, la coadă pagina mai puțin utilizată

Algoritmul LRU

- probleme de implementare:
  - lista va trebui actualizată la fiecare referință la memorie
  - găsirea unei pagini în listă, eliminarea, mutarea în listă sunt operații consumatoare de timp
  - manipularea unei liste la fiecare execuție de instrucțiune este costisitoare chiar dacă se realizează în hardware

Algoritmul LRU implementare #1

- va trebui implementat un contor C hardware
- C va fi incrementat după fiecare instrucțiune
- fiecare linie din tabela de pagini va avea un câmp pentru memorarea valorii lui C setate la ultima referire a paginii
- la apariția unui page fault, SO examinează toate liniile din tabela de pagini pentru a găsi una având cea mai mică valoare pentru C – aceasta va fi înlocuită

Algoritmul LRU implementare #2

- având n page frame, vom menține o matrice n×n biți inițializate la 0
- la referirea unei pagini k, hardware-ul va seta toți biții de pe linia k la 1 apoi toți biții de pe coloana k la 0
- la un moment dat pagina corespunzătoare liniei având cea mai mică valoare binară este cea mai puțin utilizată
- exemplu: 4 pagini referite în ordinea:

0 1 2 3 2 1 0 3 2 3

3.4.7. Simularea software a algoritmului LRU

- soluțiile pentru LRU prezentate anterior presupun existența unui suport hardware
- o soluție software este implementarea algoritmului **Not Frequently Used**
- vom avea o variabilă contor asociată cu fiecare pagină inițial pe 0
- la fiecare întrerupere de ceas SO scanează toate paginile din memorie
- pentru fiecare pagină se adaugă bitul R la contorul asociat

Algoritmul NFU

- contorul va deține astfel informații despre cât de utilizată a fost pagina asociată
- la apariția unui page fault se va înlocui pagina a cărei variabilă contor este cea mai mică
- algoritmul nu va decrementa contorul niciodată
  - e posibil ca paginile care au fost utilizate frecvent să rămână în memorie chiar dacă nu mai este nevoie de ele, iar cele mai recente să aștepte intrarea în memorie

Algoritmul NFU
problema paginilor vechi

- putem corecta această problemă printr-o mică modificare:
  - contoarele vor fi deplasate la dreapta cu 1 bit înainte de adunarea bitului R
  - R va fi adunat la bitul cel mai din stânga:

aging algorithm

- paginile frecvent utilizate vor avea biți de 1 în contor
- paginile vechi vor elimina treptat biții de 1
- când apare un page fault se va elimina din memorie pagina cu cel mai mic contor

Algoritmul NFU

	clock 0	clock 1	clock 2	clock 3	clock 4
R	101011	110010	110101	100010	011000
pg 0	10000000	11000000	11100000	11110000	01111000
pg 1	00000000	10000000	11000000	01100000	10110000
pg 2	10000000	01000000	00100000	00010000	10001000
pg 3	00000000	00000000	10000000	01000000	00100000
pg 4	10000000	11000000	01100000	01110000	01011000
pg 5	10000000	01000000	10100000	01010000	00101000

- diferență față de LRU
  - pg. 3 și 5 nu au fost referite în ultimele două tacte: LRU alege oricare dintre ele, NFU alege numai 3
  - mărimea contorului este finită (pentru un clock de 20ms, 8 biți sunt de obicei suficienți)