ALGORITMI DE GESTIUNE A PAGINILOR DE MEMORIE

Ceausu Nicolae Bogdan

Aparitia “erorii de pagina”

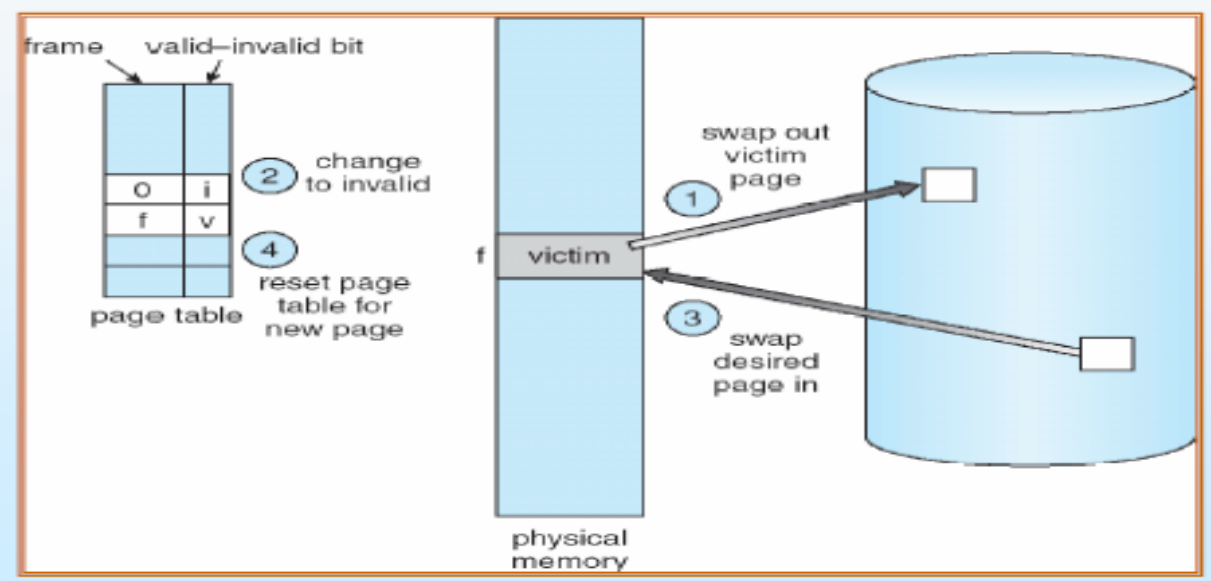
Operatia de inlocuire a paginilor este necesara atunci cand apare o asa-numita “eroare de pagina”(in engleza “page fault”). Trebuie mentionat ca aceasta se intampla in cazul memoriei virtuale, cand este utilizata paginarea. In acest context are loc operatia de translatare a adresei virtuale oferite de procesor intr-o adresa fizica.

Eroarea de pagina apare cand programul incearca sa acceseze o pagina virtuala care nu este prezenta in memoria fizica. Orice intrare in tabela de pagini are in componenta sa un bit de absent/prezent(valid/invalid). El tine evidenta paginilor din spatiul adreselor virtuale care sunt prezente in memoria fizica. Astfel, daca acest bit are valoarea 0, inseamna ca pagina virtuala careia ii acoerspunde intrarea in tabela nu se afla in memoria fizica; o referinta la aceasta pagina va avea ca rezultat o capcana- eraorea de pagina.

Cand o asemenea eroare are loc, este important rolul jucat de bitii Referit, Modificat (si ei facand parte din structura unei intrari intabela de pagini). Bitul Modificat este actualizat (setat) atunci cand datele dintr-o pagina incarcata in memorie se schimba.Necesitatea lui este evidenta cand sistemul de operare are nevoie de cadrul de pagina din memorie. Astfel, pentru M=1, continutul paginii trebuie rescris pe disc, pentru a salva schimbarile survenite. Daca M=0, inseamna ca nu au aparut modificari, iar cadrul poate fi evacuat direct, fara a salva informatia pe disc.

Bitul R(Referit) va fi setat atunci cand pagina este referita pentru scriere sau citire. Utilitatea sa : situatia in care sistemul de operare trebuie sa decida care cadru de pagina urmeaza a fi sacrificat, pentru a incarca de pe disc pagina dorita. Astfel, este de preferat sa se aleaga un cadru nereferit in locul unuia care a fost folosit, dupa cum se va vedea in continuare.

Atunci cand apare o eroare de pagina, sunt posibile 2 cazuri: fie avem cadre de pagina libere, fie nu. In cea de-a doua situatie, sistemul de operare trebuie sa elimine pagina cea mai putin probabil sa fie folosita in curand, realizand procesul de inlocuire. Acest schimb de pagini intre memoria principala si disc, realizat doar la aparitie unei erori de pagina, poarta numele de paginare la cerere.



- Se alege un cadru de pagina care este determinat , conform unui algoritm, ca fiind cel mai putin necesar in viitor;
- Daca a fost modificat (M=1), se scrie pe disc, daca nu, pur si simplu se evacueaza
- Se incarca noua pagina de pe disc
- Se actualizeaza tabelele de pagini, bitul Prezent(invalid inlocuit cu valid) este setat
- Se restarteaza instructiunea care a cauzat eroarea de pagina.

Schimbul de pagini realizat de sistemul de operare intre memoria principala si disc este un proces transparent aplicatiei.

Bibliografie: 1,2,4,9,10,12

Politici de inlocuire

➢ *Algotitmi de inlocuire ficsi/statici*

Pentru acesti algoritmi, fiecarui proces i se aloca un numar fix de pagini de memorie la inceperea executiei, el nesolicitand spatiu de memorie suplimentar mai tarziu.

■ **Algoritmul lui Belady de inlocuire optimala a paginilor (OPRA)**

Aceasta politica de inlocuire alege pentru eliminare cadrul care nu va fi folosit cea mai lunga perioada de timp (acela pentru care exista cel mai mare numar de instructiuni executate pana cand pagina noastra sa fie referita din nou). Pentru a implementa algoritmul ar trebui sa cunoastem numarul respectiv, ceea ce este imposibil in practica, neputand sa privim in viitor. De aici rezulta si inutilitatea sa intr-un sistem de operare real.

Deorece, in teorie, acest algoritm reprezinta solutia optima, cu cea mai mica rata de erori (fault-rate), el este folosit ca un punct de reper pentru a masura performantele celorlalti algoritmi de inlocuire. Astfel, daca algoritmul lui Belady nu este cu mult superior ca performanta unui algoritm dat, se poate spune ca acesta din urma este destul de bun. Pentru a face o comparatie, se considera un anumit program cu un set de date de intrare particula, pe care se ruleaza algoritmul; rezultatele rularii se vor compara cu cele de la rularea altui algoritm pe acelasi program si aceleasi date.

In continuare este prezentat un mic exemplu de functionare, cu o memorie ce contine 4 cadre de pagina:

Bibliografie: 2,5,6,11

- **Random Replacement (Inlocuire Aleatoare)**

Dupa cum arata si numele, algoritmul alege pagina de eliminat in mod aleator, fiecare pagina avand sanse egale de a fi inlocuita. Astfel, nu mai trebuie sa tinem cont de sirul de referiri. Insa prezinta mari dezavantaje, ca natura sa imprevizibila, aparitia unui numar mare de erori de pagina si a fenomenului de trashing, despre care se va discuta mai tarziu.

Drept urmare, s-a renuntat la folosirea sa inca din anii ’60.

Bibliografie: 6

- **Algoritmul NRU (Not Recently Used- neutilizat recent)**

In acest algoritm un rol important il joaca bitii Referit/Modificat. Astfel, la inceperea unui proces, bitii R/M ai paginilor sale sunt resetati. La aparitia unei erori de pagina, bitul R(referit) este setat, se actualizeaza tabela de pagini si se restarteaza instructiunea. Pentru o modificare ulterioara asupra paginii respective, este setat bitul M.

La un anumit interval de timp (o intrerupere de ceas, 20 ms), bitul R este resetat , pentru a tine evidenta paginilor care au fost referite recent. Folosindu-se de valorile bitilor R si M, algoritmul clasifica paginile din memorie in 4 categorii (clase):

- Clasa 0(R=0 -nereferita, M=0 - nemodificata);
- Clasa 1(R=0 -nereferita, M=1 - modificata);
- Clasa 2(R=1 -referita, M=0 - nemodificata);
- Clasa 3(R=1 -referita, M=1 - modificata);

Se observa ca in clasa 1 avem pagini nereferite, dar modificate. Acest lucru este posibil datorita resetarii periodice a bitului R, ceea ce muta o pagina din clasa 3 in clasa 1.

Algoritmul NRU alege pagina de inlocuit in mod aleator din cea mai joasa clasa ce contine elemente.Astfel este preferata o pagina din clasa 1(care a fost modificata, dar nereferita recent) in dauna uneia din clasa 2(nemodificata, insa folosita recent).

Ca avantaje, trebuie precizat ca acest algoritm este usor de implementat si inteles, oferind o adesea o performanta adecvata; un minus ar fi necesitatea parcurgerii bitilor R si M.

Bibliografie: 1,9,11

- **Algoritmul FIFO (first in, first out)**

Acest algoritm este la fel de usor de implementat ca Random Replacement, insa, ca performanta, este cel mult la fel de bun. Functionarea sa se bazeaza pe mentinerea unei liste (o coada) a paginilor care se afla incarcate in memorie, cu proprietatea ca pagina incarcata de cel mai mult timp se afla pe prima pozitie, iar pagina incarcata cel mai recent pe ultima pozitie. La aparitia unei erori de pagina, se elimnia pagina de pe prima pozitie, iar noua pagina se incarca la capatul cozii. Se poate observa ca pentru implementare este suficient un singur pointer

Cum nu exista nici un fel de informatie care sa ne indice daca pagina cea mai veche din memorie este una folosita sau nu, exista posibilitatea de a inlatura o pagina referita frecvent, cea ce reprezinta un minus pentru algoritm. Astfel, este folosit rar in practica in aceasta forma.

FIFO are o proprietate negativa: sufera de anomalia lui Belady. Acest lucru inseamna ca, la cresterea numarului de cadre de pagina, exista posibilitatea de crestere a numarului de

erori de pagina, dupa cum se poate vedea si in exemplul de mai jos:

Page Requests	3	2	1	0	3	2	4	3	2	1	0	4	3 pages
Newest Page	*3*	*2*	*1*	*0*	*3*	*2*	*4*	4	4	*1*	*0*	0	
		3	2	1	0	3	2	2	2	4	1	1	
Oldest Page			3	2	1	0	3	3	3	2	4	4	
Page Requests	3	2	1	0	3	2	4	3	2	1	0	4	4 pages
Newest Page	*3*	*2*	*1*	*0*	0	0	*4*	*3*	*2*	*1*	*0*	*4*	
		3	2	1	1	1	0	4	3	2	1	0	
			3	2	2	2	1	0	4	3	2	1	
Oldest Page				3	3	3	2	1	0	4	3	2	
(red *italics* indicates page fault)													

-pentru 3 cadre avem un numar de 9 erori de pagina;

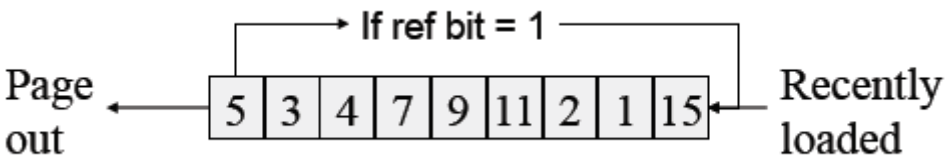
- pentru 4 cadre avem 10 erori de pagina.

Bibliografie: 2,6,11

- **Second-chance Replacement. Clock Replacement**

Second-Chance a derivat din FIFO prin aduagarea a inca unui pas la algoritm. Atunci cand apare o eroare de pagina si se verifica cea mai veche pagina din memorie, se evalueaza si bitul sau de Referit (I se acorda o ”a doua sansa”). Daca acesta este 0, atunci pagina nu a fost referita, deci poate fi eliminata. In cazul R=1, pagina noastra este folosita si are loc urmatoarea operatie: se reseteaza bitul R, iar pagina este mutata pe ultima pozitie in lista inlantuita, ca si cum tocmai ar fi fost incarcata de pe disc. Opreatia de cautare a unei pagini pentru a fi eliminata continua cu uramtoarea pagina “veche” din memorie.

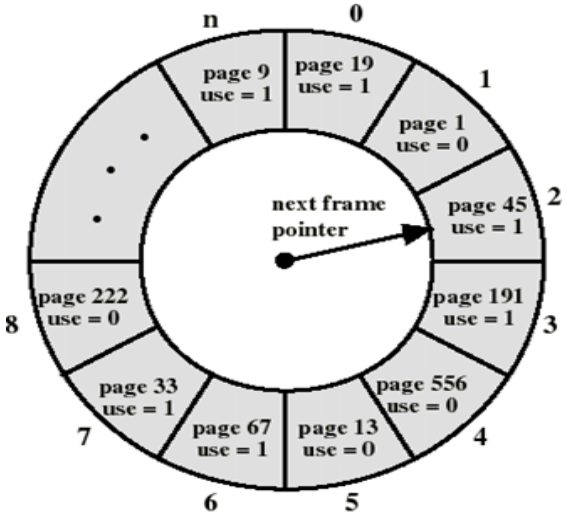
In cazul extrem cand toate paginile au bitul referit 1, deci sunt folosite, algoritmul le va parcurge pe rand, resetandu-le bitul R si mutandu-le la capatul cozii; vom ajunge asadar tot la prima pagina verificata, insa de data aceasta cu botul R=0 , putand fi eliminata; deci algoritmul este achivalent acum cu un FIFO. Mai jos avem un exemplu de functionare:



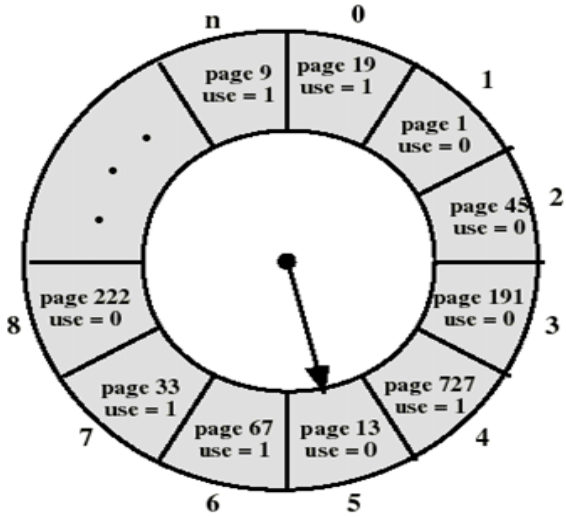
Un algoritm asemanator Second-chance este **Clock Relacement(First In, Not First Used Out)**. Ca si primul, el provine din FIFO; practic, se obtine acelasi rezultat ca si la Second-chance, doar implementarea fiind cea care difera.

Ceea ce schimba algoritmul Clock este incovenientul pe care il prezinta Scond-chance: mutarea paginilor in interiorul listei. El evita acest lucru folosind o lista circulara ce contine paginile si un pointer care indica cea mai veche pagina din memorie. Cand se ajunge la o eroare de pagina, algoritmul verifica bitul R al paginii indicate. Daca acesta are valoarea 0,

pagina este eliminata si in locul ei este incarcata o noua pagina in memorie, pointerul mutandu-se la pagina uramtoare din lista; daca bitul R=1, acesta este resetat, iar pointerul avanseaza pana cand gaseste o pagina cu R=0 care poate fi inlocuita.



(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

In cazul in care pointerul se “plimba” foarte incet, memoria este suficient de mare, nu apar multe erori de pagina sau pagina de inlocuit este gasita rapid-ceea ce reprezinta o situatie favorabila.

Pe de alta parte, cand pointerul se deplaseaza pe ceas foarte rapid, insemna ca memoria nu este suficienta, iar trashing-ul este prea ridicat, sau exista multe pagini cu bitul R setat.

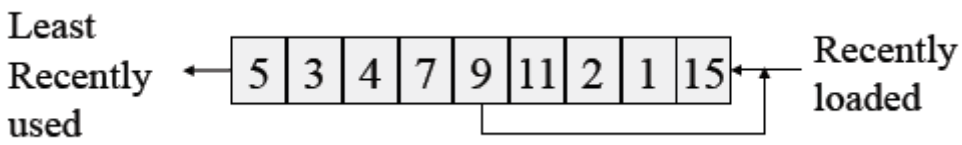
Bibliografie: 8,9,10,12

- **Algoritmul LRU (Least Recently Used= Cel mai putin recent utilizat)**

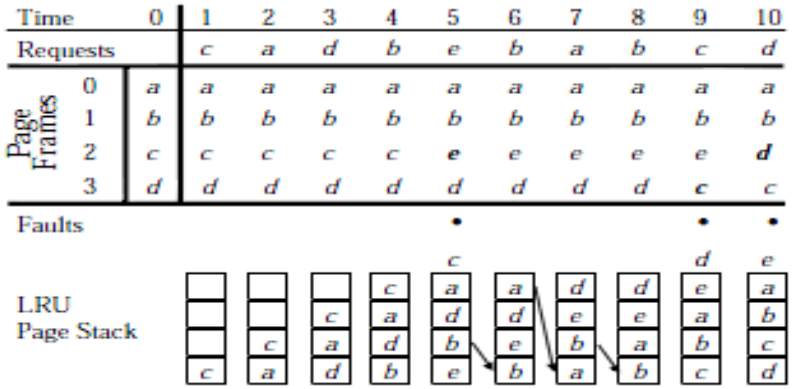
Acest algoritm este o aproximare a algoritmului optimal al lui Belady; el porneste de la ideea ca paginile ce nu au mai fost folosite de mult timp au sanse mari sa nu fie folosite nici de acum inainte, de aceea ele capata prioritate cand apare o eroare de pagina si trebuie eliberat un cadru (se alege pagina care nu a mai fost referita de cel mai mult timp).

Implementarea LRU poate fi costisitoare. Ea presupune existenta fie a unei liste inlantuite(figura a), fie a unei stive(figura b) in care pagina referita cel mai recent sa fie plasata pe prima pozitie, iar cea nefolosita de cel mai mult timp pe ultima pozitie. Dificultatea apare atunci cand are loc fiecare referire, intru-cat pagina respectiva trebuie cautata in lista si mutata din pozitia sa pe prima pozitie:

-figura (a)



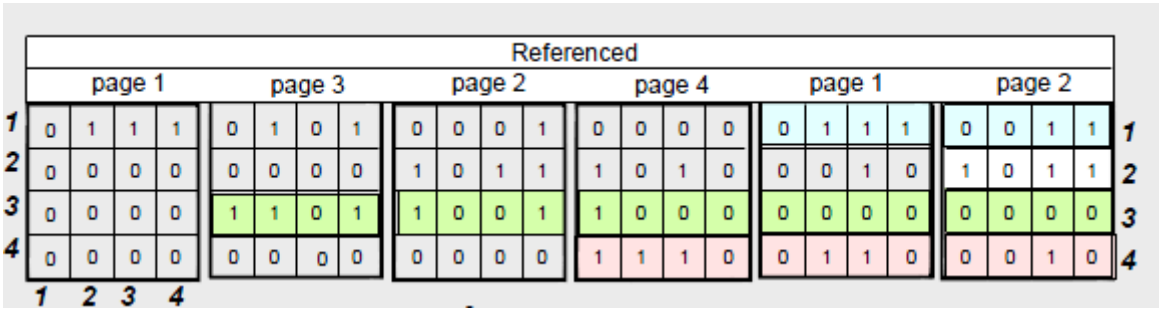
-figura (b)



O implementare la fel de costisitoare este cea care foloseste un registru pentru fiecare pagina ce va memora ceasul de sistem la fiecare referire a paginii respective. La inlocuire, se vor parcurge aceste registre si se va alege pagina cu cel mai vechi ceas. Costul este ridicat pentru un numar mare de pagini in memorie.

O variatie a acestei implementari presupune existenta unui counter de 64 biti, incrementat dupa fiecare instructiune. La fiecare referire a unei pagini, valoarea numaratorului trebuie retinuta in intrarea corespunzatoare din tabela de pagini. Pagina inlocuita va fi cea cu valoarea memorata minima.

Mai exista si implementarea hardware care, pentru o memorie de n pagini, foloseste o matrice patratica cu n*n biti, initializati cu 0. Cand este referita pagina k, are loc setarea bitilor de pe linia k, urmata de resetare bitilor de pe coloana k. Linia care va avea valoarea in binar maxima va corespunde paginii cel mai recent utilizate, iar cea cu valoare minima paginii nereferite de cel mai mult timp.



Bibliografie: 5,7,9, 10,12

- **Aproximari software ale LRU**

Implementarile prezentate mai sus pentru LRU presupun existenta unui hardware. Astfel pentru sistemele de operare ce ruleaza pe masini ce nu detin hardware-ul respectiv se pot folosi algoritmi care aproximeaza LRU.

Algoritmii **Second-chance** si **Clock**, prezentati mai sus, sunt 2 algoritmi ce aproximeaza LRU. Acestie aleg pentru inlocuire cea mai veche pagina din memorie pe care o gasesc nefolosita.

Un alt algoritm este **NFU (Not Frequently Used=neutilizat frecvent)**. Aici se foloseste un counter pentru fiecare pagina din memorie, care este initializat cu 0. La un anumit interval de timp (tact de ceas), se verifica bitul Referit pentru fiecare pagina, iar daca este 1 (pagina a fost folosita) counter-ul va fi incrementat cu o unitate. Astfel, la aparitia unei erorii de pagina, putem verifica numaratorul fiecarei pagini memorate, stiind ca cel cu valoarea minima corespunde paginii cel mai putin accesate. Aceasta va fi aleasa pentru inlocuire.

Algoritmul NFU are un minus: in cazul in care un program isi schimba setul de pagini active sau se termina si este inlocuit de catre un alt program, frecventele de accesare ale vechiului set de pagini vor duce la inlocuirea imediata a paginilor din noul set, deoarece frecventele lor de referire vor fi mult mai mici. Prin urmare, se pot inlatura din memorie pagini folosite la momentul curent, in detrimentul unor pagini folosite in trecut, inutile acum. Cum paginile inlocuite vor fi necesare in noul context, fenomenul de trashing isi va face simtita prezenta.

Algoritmul **Aging** este cel care, printr-o modificare, rezolva acest incovenient al NFU. El va tine cont atat de frecventele de referire ale paginilor din memorie (ca si NFU), cat si de faptul ca au fost folosite recent, acordand prioritate acestora din urma. Pentru a realiza acest lucru, se utilizeaza un counter de 8 biti pentru fiecare pagina. Ca si la NFU, la fiecare intrerupere de ceas, se verifica bitii R ai paginilor; dupa ce se deplaseaza continutul counterului cu o pozitie la drepata, eliberandu-se bitul cel mai semnificativ, acest bit va lua valoarea 0 sau 1, in functie de bitul R al paginii.

Fiecare counter, denumit si octet de referinta, va pastra informatii referitoare la folosirea paginii sale in ultimele 8 intreruperi de ceas. De exemplu, 000000 insemna ca pagina nu a fost referita deloc in ultimele 8 perioade de ceas, 11111111- a fost referita in fiecare perioada,iar 11000100 a- fost folosita mai recent (ultimile 2 tacturi de ceas si al 6-lea in urma) decat 01110111 sau 00111111, fiind preferata in schimbul lor, chiar daca a fost referita mai rar. La o eroare de pagina, se va alege pagina cu valoarea minima a counter-ului.

Deosebirea principala dintre aging si LRU este ca primul are o istorie scurta, limitata la ultimele 8 tacturi de ceas de dimensiunea (8 biti) a counter-ului. Chiar daca nu poate privi mai mult in urma, in practica 8 biti sunt suficienti, obtinandu-se o aproximare buna a LRU, cu performante ridicate.

Bibliografie: 4,6,7,8,9,12

Algoritmi cu ”proprietatea de stiva”

Algoritmii din aceasta categorie sunt acei algoritmi de inlocuire ficsi care au proprietatea ca, o data ce numarul de pagini de memorie creste, performantele lor se imbunatatesc. Astfel de algoritmi sunt OPT, LRU, NFU.

Ei se deosebesc de alti algoritmi statici, precum FIFO, care sufera de asa-numita anomalie a lui Belady (posibila crestere a numarului de erori de pagina, pe masura ce creste si memoria).

Bibliografie: 6

Fenomenul de Trashing

Fenomenul de trashing se manifesta atunci cand cantitatea de memorie este insuficienta pentru numarul de pagini necesare procesului/proceselor sa ruleze. Sistemul va inlocui pagini importante, pe care va trebui sa le aduca ulterior inapoi. Astfel, isi va petrce cea mai mare parte a timpului realizand incarcarea si scrierea paginilor pe disc, fara a se realiza executia propriu-zisa a procesului, ceea ce reprezinta o importanta problema de performanta. Se va creea impresia ca memoria este la fel de inceata ca si discul, ci nu invers.

De exemplu, in cazul algoritmilor Random sau NFU, exista riscul eliminarii din memorie a unor pagini importante la momentul curent.

In cazul sistemelelor cu multiprogramare, pe masura ce numarul de procese in executie creste, va creste si numarul de pagini ce trebuie incarcate in memorie. La un moment dat, cum memoria nu va mai face fata , procesele incep sa cauzeze erori de pagina; cum tot timpul este consumat pe paginare, activitatea procesorului cunoaste o scadere, iar rata de erori de pagina creste drastic.

Bibliografie: 2,4,10

- *Algoritmi de inlocuire variabili/ dinamici*

In cazul algoritmilor de inlocuire ficsi, exista posibilitatea fie ca programul sa foloseasca doar o parte din memoria alocata lui sau, din contra, sa necesite mult mai multa memorie decat ii este alocata. In cel de-al 2-lea caz, programul incepe sa inlocuiasca pagini din memorie in mod repetat, manifestandu-se fenomenul de trashing.

Astfel, trebuie folosita o solutie sub forma unor algoritmi variabili, care controleaza numarul de pagini alocate fiecarui proces in functie de localitatea programului.

- **Algoritmul de inlocuire Working Set**

De regula, in comportamentul lor, procesele respecta principiul localitatii, enuntat de Peter Denning: la un anumit moment dat, un proces refera doar un mic subset din paginile sale. Acest principiu cunoaste doua forme:

-localitatea temporala-conform careia, pentru o pagina, o data accesata, exista mari sanse sa fie accesata din nou in viitorul apropiat;

-localitatea spatiala-daca a fost accesata o pagina, este foarte probabila si accesarea unei pagini vecine ei.

In acest mod, Peter Denning a dezvoltat modelul Working Set (al setului de lucru). Acesta contine deci paginile referite de proces intr-un anumit pas al executiei sale. In functie de capacitatea memoriei de a pastra toate paginile din setul de lucru, poate sa se manifeste sau nu fenomenul de trashing.

Acest model se afla in stransa legatura cu operatia de prepaginare. Spre deosebire de paginarea la cerere, facuta numai la aparitia unei erori de pagina, prepaginarea presupune incarcarea paginilor necesare unui proces anterior ca acesta sa isi inceapa executia, astfel incat sa se reduca numarul de erori de pagina . Faptul acesta este util in cadrul multiprogramarii, cand procesele beneficiaza pe rand de resursele procesorului, iar momentul sosirii unui nou proces duce la aparitia erorilor de pagina. Astfel, pe baza paginilor din setul de lucru de la momentul suspendarii executiei se determina paginile necesare procesului la restartare, avand loc preapaginarea lor.

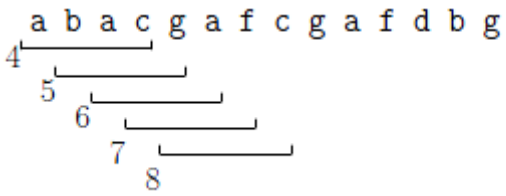
Marimea setului de lucru variaza o data cu localitatea procesului: atunci cand procesul nu respecta principiul localitatii sau are localitate slaba, noi pagini sunt incarcate in memorie, iar working set-ul creste ca marime. In cazul respectarii localitatii, evident ca sunt referite putine pagini de memorie in afara celor deja existente in setul de lucru.

Setul de lucru este definit ca functie de momentul actual de timp t si ultimele k referiri la memorie astfel: w(t,k)= multimea de pagini care au fost accesate in ultimele k referiri la memorie. Aceasta functie are proprietatea de a varia foarte incet in timp.

Exemplu:

-secventa de referiri: a b a c g a f c g a f d b g

-k=4 referiri:



-la fiecare moment, setul de lucru va contine:

1	a	8	acgf
2	ab	9	acgf
3	ab	10	acgf
4	abc	11	acgf
5	abcg	12	agfd
6	acg	13	afdb
7	acgf	14	fdbg

-

In practica, pentru a implementa algoritmul, se prefera inlocuirea lui k cu o perioada de timp, T, astfel incat setul de lucru la un moment dat w(t,T) sa contina paginile care au fost accesate in perioda t-T,t. Acest mod de implementare se bazeaza pe informatia din bitul R al fiecarei pagini. Procedeul este urmatorul: al aparitia unei erori de pagina, se parcurg paginile; daca bitul R=1, insemna ca pagina a fost referita in ultimul tact de ceas (el fiind resetat periodic de sistem), iar pagina face parte din setul de lucru. Totodata, timpul virtualcurent este copiat intr-un camp special din intrarea in tabela de pagini. Pentru o pagina cu R=0, se verifica timpul din intrarea sa in tabela: daca este mai mic decat parametrul T, pagina face parte din setul de lucru; daca este >T, atunci pagina este aleasa pentru a fi inlocuita. In cazul in care nu sunt gasite pagini in afara ferestrei de timp T, se alege pagina din setul de lucru cu R=0 si cea mai luga perioada de nefolosire. De asemenena, pagina tocmai incarcata in memorie este adugata la setul de lucru.

Aceasta varianta are dezavantajul de a parcurge intrarile din tabela de pagini la aparitia fiecarui page fault.

Bibliografie: 2,3,9,11

- **Working Set Clock**

Algoritmul Workin Set Clock este o varianta modificata lui Clock, in care se aplica ideile modelului ”setul de lucru”.

Ca si la algoritmul Clock static, paginile din memorie sunt mentinute intr-o lista circulara, parcursa cu ajutorul unui pointer. Informatiile necesare in acest model sunt bitii R si M, precum si timpul ultimei folosiri (asemanator implementarii working set de mai sus). Toate aceste informatii se vor gasi in intrarea din tabela de pagini, pentru fiecare pagina in parte. Facand diferenta intre timpul curent si timpul ultimei folosiri se obtine vechimea paginii, care se va compara cu fereastra de timp T a setului de lucru.

Astfel, la aparitia unei erori de pagina, se verifica bitul R al paginii indicate de pointer. Daca acesta este 1, pagina a fost accesata, facand parte din setul de lucru. Bitul sau R va fi resetat, se va copia timpul virtual curent in campul sau “timpul ultimei folosiri”, iar pointerul se deplaseaza mai departe in lista.

Daca bitul R=0, sunt posibile 2 situatii. Fie pagina are vechimea mai mica decat fereastra T, ea face patre din setul de lucru, iar pointerul va merge mai departe in lista.

Fie vechimea ei este mai mare, se evalueaza bitul M, pentru a determina daca pagina este una murdara. In cazul M=1 (pagina a fost modificata), se programeaza scrierea ei pe disc pentru a salva modificarile si se trece mai departe. Daca se gaseste o pagina o pagina curata (M=0), aceasta va fi aleasa pentru inlocuire.

Este posibil sa apara situatia in care, dupa o parcurgere a listei, sa nu se fi ales o pagina pentru inlocuire. Daca au fost totusi programate scrieri ale unor pagini, se parcurge lista in continuare pana se gaseste o pagina curata. In cazul in care nu s-au programat scrieri, toate paginile au fost curate, si cum nici una nu a fost aleasa, insemna ca toate fac parte din setul de lucru; pentru acest caz particular, se alege o pagina din setul de lucru, cum ar fi cea cu cea mai apropiata vechime de fereastra T.

Bibliografie: 7,9,11

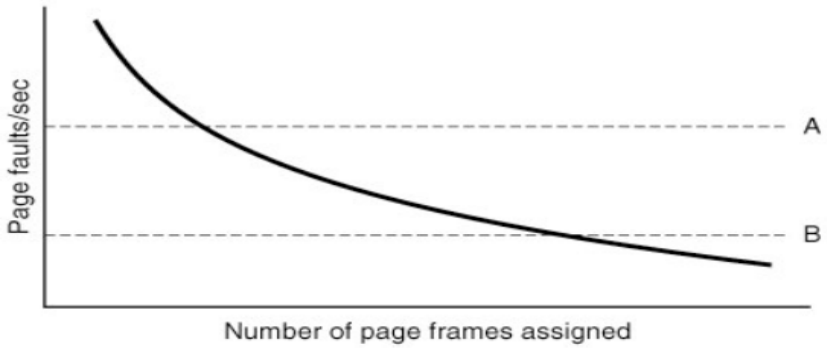
- **Alocarea cadrelor. Algoritmul Page faut frequency (PFF)**

Atunci cand folosim un algoritm fix, fiecarui proces i se aloca un numar fix de cadre de memorie. Situatiile ce se pot ivi: fie procesul nu foloseste toate cadrele alocate, irosind spatiul memoriei; fie, la o schimbare de localitate, setul de lucru se mareste, iar memoria alocata devine insuficienta, rezultand trashing-ul. Algoritmii de inlocuire variabili aloca unui proces un numar variabil de cadre de pagina, cu scopul de a evita fenomenul de trashing.

Problema care se pune este cum sa determinan cate cadre trebuie alocate fiecarui proces. Se contureaza 2 modalitati. Fie alocam un numar egal de cadre fiecarui proces, fie alocam numarul de cadre proportional cu marimea procesului astfel: pentru s marimea unui proces, S marimea totala a proceselor in executie si m numarul de cadre disponibile, rezulta un numar de (s/S)*m cadre alocate procesului nostru.

Algoritmul PFF incearca sa previna fenomenul de trashing, alocand fiecarui proces un numar de cadre astfel incat rata de erori de pagina a procesului sa scada. Se incepe prin a aloca initial fiecarui proces un numar de cadre proportional cu marimea sa. Pe parcursul executiei, trebuie supravegheata intr-un anume fel rata de erori de pagina pentru proces. Acest lucru se poate face in mai multe moduri: fie inregistrand timpul dintre 2 erori de pagina (eventual media pe mai multe perioade) si calculand fecventa ca 1/t, fie masurand numarul de erori pe secunda.

Implementarea PFF trebuie sa tina cont de 2 limite: un nivel peste care rata de erori devine prea mare (caz in care procesului i se aloca mai multe cadre de pagina, pentru a reduce rata) ; respectiv un nivel sub care rata devine suficient de mica astfel incat putem elibera cadre (procesului ii este alocata mai multa memorie decat are nevoie). Daca rata de erori a procesului este in intervalul dintre cele 2 limite atunci, la o eroare de pagina, inlocuirea se face apeland la alt algoritm. Asta deoarece PFF nu specifica ce pagina trebuie inlocuita, doar determina numarul de cadre ce trebuie alocate procesului.



In cazul in care rata de erori creste, insa nu avem cadre libere, e necesara suspendarea unui proces; cadrele astfel obtinute se redistribuie intre procesele cu frecvente mari ale erorilor de pagina.

Bibliografie: 3,4,11

- **Politica de curatare. 2 hand Clock**

Cand apare o eroare de pagina, un algoritm de inlocuire trebuie sa gaseasca “victima” care, eventual, trebuie salvata pe disc(in cazul in care a fost modificata). Politica de curatare inlatura acest inconvenient prin pastrarea unei zone de cadre “libere” din memorie, in care, la fiecare page fault, noua pagina poate fi alocata.

Pentru mentinerea listei de cadre “libere”se foloseste un asa-numit “daemon”, un program ce, in mod periodic, verifica daca aceste cadre sunt in numar suficient;daca este necesar, “daemon”-ul apeleaza la un algoritm de inlocuire pentru a alege ce pagini sa fie introduse in lista. De asemenea, continutul lor este actualizat pe disc, in cazul in care au fost modificate.

Astfel, atunci cand o noua pagina trebuie incarcata in memorie, ea poate fi alocata intr-unul din acele cadre, stiind ca informatia anterioara a fost salvata. Daca se intampla ca una dintre paginile aflate in lista respectiva sa fie referita din nou de proces, iar continutul sau inca sa se afle in cadru, pur si simplu este inlaturat acest cadru din lista cadrelor libere.

Algoritmul 2hand Clock este o modificare adusa algoritmului clock clasic; vor fi 2 pointeri: primul corespunde daemon-ului (parcurge lista, daca M=1 pentru o pagina,este resetat iar pagina salvata pe disc). Celalalt pointer are acelasi rol ca si pointerul de la algoritmul Clock clasic, fiind folosit la inlocuire.

Bibliografie: 8,11

Comparatie intre algoritmii de inlocuire a paginilor

Algoritmul optimal al lui Belady este neimplementabil in practica, insa este folosit pentru a pentru a masura performantele diversilor algoritmi; astfel, acestia se ruleaza pe o anumita secventa de referinte (pentru un proces particular, cu un set particular de date de intrare). Rezultatele se compara la final cu cele obtinute pentru rularea lui OPRA.

Algoritmul NRU ar avantajul de a fi usor de inteles si de implementat, insa este un algoritm crud.

Algoritmii FIFO sau Random Replacement sunt inutilizabili in practica datorita rezultatelor imprevizibile pe care le produc.

Chiar si in situatia in care FIFO duce la rezultate bune, este mai recomandata folosirea variantelor sale modificate, Second-chance sau Clock(FINUFO). Acestea difera doar prin implementare, conducand in cele din urma la selectia aceleiasi pagini pentru inlocuire. Algoritmul Clock Replacement este la fel de necostisitor ca si FIFO, insa se obtin performante comparabile cu ale LRU, reprezentand de fapt o aproximare software a acestuia.

Ca performante, LRU se apropie simtitor de algoritmul optimal al lui Belady. Partea negativa este ca presupune existenta unui hardware special (un camp in intrarea din tabela a paginii ce va retina o valoare), necesitand, la randul sau, aproximari:

Una ar fi algoritmul NFU care, ca si LRU, este un ”algoritm de stiva”. El determina cu precizie ce pagini din memorie sunt sau nu folositoare, insa este sensibil la schimbarile de localitate; drept care se foloseste o varianta imbunatatita a sa, algoritmul Aging. Acesta din urma reprezinta o solutie optima.

Algoritmul Working Set, desi performant , este dificil de implementat. Din aceasta cauza este preferat WSClock- se obtin preformante,in plus, este si eficient.

Bibliografie: 6,11

Marimea paginii de memorie

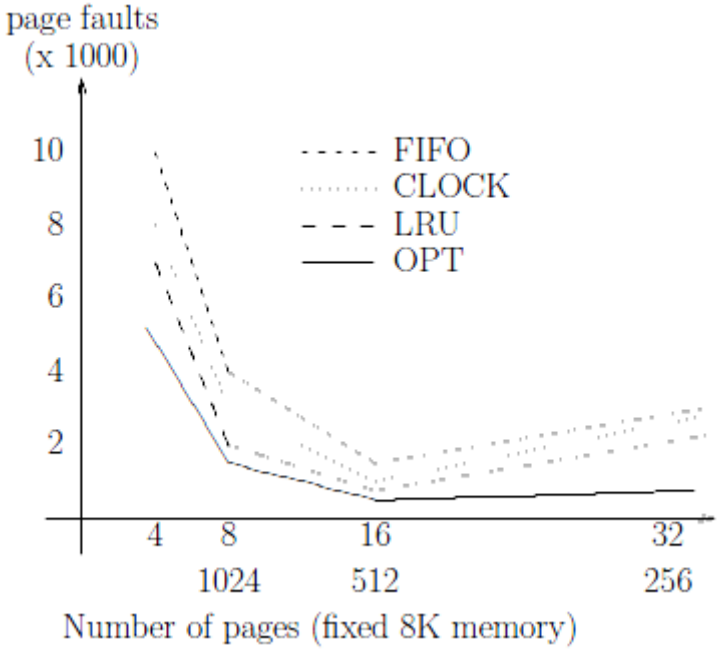
Pentru a stabili o marime optima a paginilor de memorie, trebuie luate in considerare mai multe aspecte.

In primul rand, apare fenomenul de fragmentare interna, atunci cand ramane spatiu neutilizat intr-o pagina de memorie.In medie, pentru o anumita data/cod ce ocupa mai multe pagini,de dimensiune p octeti fiecare, spatiul irosit va fi jumatate din dimensiunea ultimei pagini ocupate, p/2. Astfel, o marime mai mica a paginii de memorie duce la mai putin spatiu pierdut.

Insa , pagini de memorie de dimensiune mica determina, pentru o memorie anume, un numar mai mare de pagini, deci un nmar mai mare de intrari in tabela de pagini. Astfel, pentru un proces de s octeti si o dimensiune a intrarii intabela de pagini de e octeti, procesul nostru ocupa (s/p) *e octeti in tabela de pagini, o valoare invers proportionala cu dimensiunea paginii de memorie,p.

Pentru a tine cont de ambele aspecte si a gasi un optim al lui p, se deriva relatia se/p+p/2 si se egaleaza cu 0 (totul in functie de p). Rezultatul este dimensiunea optima a paginii de memorie p=$\sqrt{(2se)}$.

Faptul ca, pana la un moment dat, o marime mai mica a paginii de memorie este mai eficienta este indicat in figura urmatoare, ce ilustreaza cazul a 4 algoritmi de inlocuire:



Se poate observa ca o marime mare a paginii, ce determina numarul de 4 pagini de memorie, influenteaza negativ performantele algoritmilor. Cresterea numarului de pagini prin micsorarea dimensiunii paginii este benefica pana cand se ajunge la un compromis.

Bibliografie: 3,12

BIBLIOGRAFIE:

1 -G53OPS - Operating Systems, Memory Management,Graham Kendall School of Computer Science & IT,The University of Nottingham
http://www.cs.nott.ac.uk/~gxk/courses/g53ops/powerpoint/004memorymanagement.ppt

2 -CS 537: Operating Systems,Lecture 13,Paging and Page Replacement,Michael Swift,UNIVERSITY OF WISCONSIN-MADISON
http://pages.cs.wisc.edu/~swift/classes/cs537-fa07/lectures/13-pageReplacement.pdf

3 -CS3725 - Computer Architecture,The Memory Architecture,Paul Gillard,Department of Computer Science,Memorial University of Newfoundland
http://web.cs.mun.ca/~paul/cs3725/material/ch7.pdf

4 -Bernard Chen, University of Central Arkansas Operating System, Chapter 9 Virtual Memory

5 -Operating Systems Principles,Lecture 12: Page Replacement Algorithms,Steve Goddard

6 -Virtual Memory Page Replacement Algorithms,Terrance McCraw, CS384 Operating Systems, Milwaukee School of Engineering
http://people.msoe.edu/~mccrawt/resume/papers/CS384/mccrawt_cs384_virtual.pdf

7 -Marvin Paull, Operating Systems Design, Memory Management.Paging replacement algorithms
http://www.cs.rutgers.edu/~paull/OS_MMNG3_REPL.pdf

8 -Operating Systems,Page Replacement Algorithms, Ariel J. Frank-P. Weisberg
http://u.cs.biu.ac.il/~ariel/download/os381/os8-3_vir.ppt

9 -COS 318: Operating Systems,Virtual Memory Paging, Kai Li, Computer Science Department, Princeton University
http://www.cs.princeton.edu/courses/cos318/

10 -Peter Druschel-Demand paging, thrashing, working sets
http://www.mpi-sws.org/~druschel/courses/os/lectures/mem4.pdf

11-Andrew Tanenbaum, Modern Operating Systems

12-Andrew S. Tanenbaum, Operating Systems Design and Implementation, 3rd Edition