Algoritmi de paginare

- Conceptul de memorie virtuală, permite ca un program în execuţie să nu fie obligatoriu încărcat integral în memoria internă. O justificare a necesităţii acestui aspect, poate rezulta din următoarele exemple:

  - programele conţin părţi de cod în care se testează anumite condiţii de eroare, pentru care se execută anumite proceduri şi care se întâmplă foarte rar;

  - unor structuri de date declarate în program, li se alocă un spaţiu mult mai mare decât cel utilizat efectiv de către procesul respectiv.
- Posibilitatea execuţiei unui program, care este încărcat parţial în memoria internă, oferă o serie întreagă de avantaje, printre care:

  - nu mai există restricţii cu privire la dimensiunea programelor, deoarece cantitatea de memorie virtuală care poate fi alocată proceselor, poate fi oricât de mare;

  - deoarece cantitatea de memorie fizică pe care o foloseşte efectiv un proces scade, creşte gradul de multiprogramare şi, deci gradul de utilizare a CPU;

  - sunt necesare mai puţine operaţii de intrare/ieşire pentru încărcarea/evacuarea programelor de pe/pe disc.
- Există două tipuri de bază de algoritmi de paginare: statici şi dinamici. Algoritmii statici alocă un număr fix de pagini fizice fiecărui proces în timpul execuţiei, pe când în cazul celor dinamici acest număr poate să fie variabil.

Algoritmi statici

- Oricare algoritm de paginare este definit trei politici:

  - politica de extragere (fetch) decide când o pagină va fi încărcată în memoria primară;

  - politica de înlocuire determină care pagină va fi mutată din memoria internă atunci când toate paginile acesteia sunt pline cu informaţii ale proceselor în execuţie;

  - politica de plasare determină unde va fi plasată o pagină extrasă din memoria internă.
- Deoarece numărul paginilor fizice este fixat în cadrul alocării statice, pagina fizică unde va fi adusă o pagină virtuală, va trebui să fie ori o pagină liberă, ori una în care se va înlocui informaţia deja existentă, care fac parte din spaţiul de adrese al procesului respectiv.
- Daca politica de plasare este aceeaşi, celelate două politici diferenţiază algoritmii statici de paginare.

- În descrierea algoritmilor statici, vom folosi următorul **model matematic**.
- Presupunem că N este mulţimea paginilor din spaţiul de adrese virtuale; ω este şirul referinţelor de pagini, fiind o secvenţă de pagini din N, de forma:

$$\omega = r_1, r_2, .., r_i, ...$$

- care sunt referenţiate de către proces în timpul execuţiei sale. Timpul virtual al procesului este indicele din şirul de referinţe de pagini.
- Fie m numărul de pagini din spaţiul de adrese fizice şi $S_i(m)$ mulţimea paginilor fizice încărcate. La momentul 0 avem $S_0(m) = \Phi$ şi la momentul virtual t,

$$S_t(m) = S_{t-1}(m) \cup X_t - Y_t ;$$

X_t, respectiv Y_t fiind mulţimea paginilor extrase, respectiv înlocuite la timpul virtual t.

Politici de extragere

- **Aducerea unei pagini la cerere**, este politica de extragere utilizată; deoarece nu se cunoaşte apriori evoluţia unui proces, nu se poate stabili dinainte o politică de pre-extragere a paginilor.
- Există unele sisteme care utilizează metode de încărcare prin care se aduc **pagini în avans**.
- Astfel, odată cu o pagină se aduc şi câteva pagini vecine, în ipoteza că ele vor fi invocate în viitorul apropiat. O **evidenţă statistică** a utilizării paginilor, poate furniza, cu o anumită probabilitate care vor fi paginile cerute în viitor.
- **Principiul vecinătăţii** afirmă că adresele de memorie solicitate de un program nu se distribuie uniform pe întreaga memorie folosită, ci se grupează în jurul unor centre.
- Apelurile din jurul acestor centre sunt mult mai frecvente decât apelurile de la un centru la altul.
- Acest principiu sugerează o politică simplă de încărcare în avans a unor pagini. Se stabileşte o aşa zisă memorie de lucru compusă din câteva pagini.
- Atunci când se cere aducerea unei pagini virtuale în memoria executabilă, sunt încărcate câteva pagini vecine acesteia. În conformitate cu principiul vecinătăţii, este foarte probabil ca următoarele referiri să fie făcute în cadrul memoriei interne.

Algoritmi de înlocuire

- Dacă considerăm $\omega = r_1, r_2, ..., r_i, ...$ un şir de referire a paginilor şi $\{St(m)\}$ secvenţa de stări ale memoriei interne atunci

$$S_t(m) = S_{t-1}(m) \cup X_t - Y_t$$

relaţie în care X_t şi Y_t au semnificaţia următoare:

- dacă la momentul t procesul cere o pagină r_t care nu se găseşte în memoria fizică şi y_t este pagina care va fi înlocuită atunci $X_t = \{r_t\}, Y_t = \{y_t\}$;
- dacă la momentul t procesul cere o pagină r_t care se găseşte în memoria fizică atunci $X_t = \Phi$, $Y_t = \Phi$.

- Problema care se pune, este cum să fie selectată pagina y_t care va fi înlocuită în memoria fizică.
- **Înlocuirea aleatoare** înseamnă că y_t este aleasă cu o probabilitate $1/m$ dintre oricare paginile fizice alocate procesului. În practică, s-a dovedit că această metodă nu este eficientă.

Algoritmul optimal al lui Belady

- La la un moment t, pentru încărcarea unei pagini r_t, pagina fizică y_t care urmează să fie înlocuită, va fi determinată astfel: dintre paginile r_{t+1}, r_{t+2},..., care fac parte din şirul de referinţe, se va alege acea pagină r_k pentru care diferenţa $k-(t+1)$, $k \geq t+1$, (considerând primele apariţii ale paginilor), este maximă, sau $t=\max\{k/r_k \in S_{t-1}(m)\}$ sau cu alte cuvinte se înlocuieşte pagina care **nu va fi folosită pentru cea mai lungă perioadă de timp**.
- Această metodă nu poate fi aplicată direct, deoarece presupune că se cunoaşte apriori care pagini vor fi solicitate de către procesul respectiv (imposibil, deoarece evoluţia concretă a unui program este determinată de datele concrete asupra căruia operează), dar ea poate fi o sursă de inspiraţie pentru alte metode.

- **Exemplu.** Presupunem că numărul de pagini fizice alocate unui proces este 3. În figura urm este prezentat un şir de referinţe de pagini(prima linie), precum şi paginile fizice în care ele se încarcă, folosind algoritmul lui Belady. a II-a linie coresp. primei pagini fizice alocate, a III-a linie coresp. celei de-a II-a pagini fizice alocate, a IV-a linie coresp. celei de-a III-a pagini fizice alocate.
- Am marcat cu * atunci când are loc fenomenul de lipsă de pagină.

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7*	7	7	2*	2	2	2	2	2	2	2	2	2	2	2	2	2	7*	7	7
	0*	0	0	0	0	0	4*	4	4	0*	0	0	0	0	0	0	0	0	0
		1*	1	1	3*	3	3	3	3	3	3	3	1*	1	1	1	1	1	1

pagini fizice

- Primele 3 pagini din şirul de referinţă sunt încărcate fără probleme în memoria fizică. Când se ajunge la a patra pagină virtuală din şir (pagina 2), va fi înlocuită pagina 7, deoarece, în viitor, ea va fi ultima la care se va face o referinţă, dintre paginile care sunt încărcate în memoria fizică. A cincea pagină virtuală din şir este deja încărcată în memoria internă, deci nu va mai fi necesară înlocuirea unei pagini ş.a.m.d.

Înlocuirea în ordinea încărcării paginilor(FIFO)

- Se crează şi se întreţine o listă a paginilor în ordinea încărcării lor. Această listă se actualizează la fiecare **nouă încărcare de pagină**. Atunci când se cere înlocuirea, este substituită prima (cea mai veche) pagină din listă.
- **Exemplu. În condiţiile aceluiaşi şir de referinţe şi aceluiaşi** număr de pagini fizice alocate unui proces, ca în exemplul ant., în figura urm. este prezentat un şir de referinţe de pagini, precum şi paginile fizice în care ele se încarcă, folosind algoritmul FIFO.
- La momentul 4 (cind se incarca prim data pag. 2) coada contine 7,0,1; dupa incarcare coada va contine 0,1,2.
- La momentul 5 (cind se incarca din nou pag. 0) coada contine1,2,3 ; dupa incarcare coada va contine 2,3.0. La urm. inloc de. pag. , 4 se incarca in locul pag.2 si coada devine 3, 0, 4 s.a.m.d.

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7*	7	7	2*	2	2	2	4*	4	4	0*	0	0	0	0	0	0	7*	7	7
	0*	0	0	0	3*	3	3	2*	2	3	3	3	2	2	2	2	2	0*	0
		1*	1	1	1	0*	0	0	3*	4	4	2*	1*	1	1	1	1	1	1

pagini fizice

- **Metoda celei de-a doua şanse** este o extensie a metodei FIFO.
- Fiecare pagină fizică are asociaţi doi biţi, prin intermediul cărora se va decide pagina de înlocuit.
- Bitul **R**, numit bit de referire, primeşte valoarea 0 la încărcarea paginii. La fiecare referire a paginii, acest bit este pus pe 1. Periodic (constantă de sistem), bitul este pus iarăşi pe 0.
- Bitul **M**, numit bit de modificare, primeşte valoarea 0 la încărcarea paginii respective. El este modificat numai la scriere în pagină, când i se dă valoarea 1. Bitul **M** indică dacă pagina trebuie sau nu salvată înaintea înlocuirii.
- Când se pune problema lipsei de pagină, se testează bitul R al primei pagini din coadă.
- Dacă acesta este 0, atunci pagina este înlocuită imediat.
- Dacă este 1, atunci este pus pe 0 şi pagina este pusă ultima în coadă, având ca valoare a timpului virtual, timpul când se cere încărcarea de pagină.
- Apoi căutarea se reia cu nouă pagină care a devenit prima din coadă.
- Daca toate paginile din coada au bitul R pozitionat pe 1, se aplica FIFO.

Înlocuirea paginii nesolicitate cel mai mult timp (LRU **L**east **R**ecently **U**sed)

- LRU are la bază următoarea observaţie: o pagină care a fost solicitată mult de către ultimele instrucţiuni, va fi solicitată mult şi în continuare şi invers. Problema este cum să se ţină evidenţa utilizatorilor.
- **Numărătorul de accese** se implementează hard. Există un registru, numit contor reprezentat de regulă pe 64 de biţi. La fiecare acces, valoarea lui este mărită cu o unitate. În tabela de pagini, există o locaţie pentru a memora valoarea contorului. În momentul accesului la o pagină, valoarea contorului este memorată în acest spaţiu rezervat din tabela de pagini. Atunci când se impune o înlocuire, este înlocuită pagina cu cea mai mică valoare a contorului.
- In caz de egalitate se aplica metoda FIFO.

- **Exemplul În condiţiile aceluiaşi şir de referinţe** şi aceluiaşi numărul de pagini fizice alocate unui proces, ca în exemplul ant., în figura urm. este prezentat un şir de referinţe de pagini, precum şi paginile fizice în care ele se încarcă, folosind algoritmul LRU.

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7*	7	7	2*	2	2	2	4*	4	4	0*	0	0	1*	1	1	1	1	1	1
	0*	0	0	0	0	0	0	0	3*	3	3	3	3	3	0*	0	0	0	0
		1*	1	1	3*	3	3	2*	2	2	2	2	2	2	2	2	7*	7	7

pagini fizice

- Când este adusă pentru prima dată în memoria fizică pagina virtuală 2, este înlocuită pagina 7, deoarece ea nu a mai fost utilizată de cel mai mult timp; din acelaşi motiv, când este referenţiată pentru prima dată pagina 3, ea înlocuieşte pagina 1 ş.a.m.d.

Metoda NRU(Not **R**ecently **U**sed**)**.

- La fiecare valoare a timpului virtual, biţii **M** şi **R,** amintiti anterior**,** împart paginile fizice în patru clase:

  - clasa 0: pagini nereferite şi nemodificate;

  - clasa 1: pagini nereferite (în intervalul fixat), dar modif. de la înc. lor;

  clasa 2: pagini referite dar nemodificate;

  clasa 3: pagini referite şi modificate.
- Atunci când o pagină trebuie înlocuită, pagina respectivă se caută mai întâi în clasa 0, apoi în clasa 1, apoi în clasa 2 şi în sfârşit în clasa 3. Dacă pagina de înlocuit este în clasa 1 sau în clasa 3, conţinutul ei va fi salvat pe disc înaintea înlocuirii.
- **Observaţie. Combinarea** algoritmilor NRU şi FIFO: se aplică mai întâi NRU, iar în cadrul aceleiaşi clase se aplică FIFO.

- **Alocarea cadrelor (paginilor fizice).**
- În cazul sistemelor care utilizează multiprogramarea, fiecărui proces trebuie să i se asigure un număr de pagini fizice, care nu pot fi utilizate de celelalte procese.
- Prin a**locare egală** se alocă tuturor proceselor acelaşi număr de pagini fizice.
- Prin **alocarea proporţională** se alocă proceselor un număr de pagini fizice direct proporţional cu dimensiunea spaţiului de memorie virtuală utilizat de procesul respectiv.
- O altă metodă de alocare ţine cont de **prioritatea proceselor**, în sensul că numărul paginilor fizice alocate unui proces, să fie direct proporţională cu prioritatea procesului.
- Cele două criterii de alocare proporţională se pot combina.

- **Performanţele sistemului de paginare.**
- Atunci când un proces referenţiază o pagină virtuală care nu este prezentă în memoria fizică, sistemul execută o serie întreagă de operaţii, care au fost amintite.
- Este evident că timpul de execuţie al procesului într-un sistem cu paginare, va fi mai mare decât timpul său de execuţie, în cazul în care va avea la dispoziţie un spaţiu de memorie suficient, astfel încât să nu se confrunte cu situaţia de „pagină lipsă” (pagină referenţiată care nu este încărcată în memoria fizică, şi care trebuie adusă în locul unei pagini încărcate în memoria fizică).
- Dacă apare frecvent situaţia că, la un anumit moment al execuţiei, o anumită pagină este evacuată pe disc, după care, la un interval mic de timp ea este din nou referenţiată, eficienţa utilizării CPU va scade.

Algoritmi dinamici de paginare.

- Prin utilizarea algoritmilor statici de alocare, se poate ajunge la situaţia în care anumitor procese nu li se asigură suficientă memorie fizică, iar alte procese nu folosesc eficient memoria fizică alocată.
- Problema care se pune, este găsirea unei strategii prin care se aduc în memoria fizică numai acele pagini de care procesul va avea nevoie într-un viitor apropiat.
- Numărul de pagini fizice de care va avea nevoie un proces pentru execuţia lui, nu va mai fi o constantă, fiind modificat în funcţie de necesităţi**.**
- **Metoda setului de lucru(WS-W**orking **S**et**)** rezolvă această problemă.
- Ea are la bază modelul localităţii. **Localitatea** este un **set de pagini** care sunt utilizate împreună într-un mod activ. Un program este format din mai multe localităţi, iar execuţia programului presupune trecerea dintr-o localitate în alta.

- Un **exemplu** de localitate este partea de instrucţiuni cod maşină care corespund unui subprogam; aceste instrucţiunui se află în pagini vecine, care după execuţia lor (terminarea execuţiei subrutinei) pot fi evacuate din memorie şi posibilitatea referirii lor într-un viitor apropiat este improbabilă.
- Când un proces trece dintr-o localitate în alta, se modifică atât paginile cerute de către proces, cât şi numărul de pagini fizice necesar procesului.
- Deci, se pune în mod firesc problema, de a considera numărul de pagini alocat procesului ca fiind o entitate dinamică, ce se modifică în timpul execuţiei procesului.
- Algoritmului setului de lucru foloseşte necesităţile curente de memorie fizică ale procesului, pentru a determina numărul paginilor fizice alocate acestuia, în anumite momente ale execuţiei sale, fiind, deci o metodă dinamică de alocare a memoriei interne.

- **Modelul matematic.** Presupunem că există n procese care partajează memoria fizică a sistemului. Fie $m_i(t)$ cantitatea de memorie alocată procesului i la momentul virtual t.
- Avem: $m_i(0)=0$ şi

$$\sum_{i=0}^{n-1} m_i(t) = k$$

k fiind dimensiunea memoriei interne.

Avem , de asemenea

$S_t(mi(t))= S_{t-1}(m_i(t))UX_t-Y_t, \quad S_0(m_0(t))=0.$

$S_t(m_i(t))$ reprezintă starea memoriei la momentul t pentru procesul p_i ($S_t(m_i(0))=\Phi$).

Dacă la momentul t procesul cere pagina r_t şi aceasta este deja încărcată în memoria fizică, atunci $X_t=Y_t=\Phi$;

Dacă pagina r_t nu este încărcară în memorie, atunci

$Y_t = \{y_t\}$, sau pagina y_t va fi evacuată din memorie, dacă numărul de referinţe de la prima referire a paginii y_t, până la cererea paginii curente r_t este mai mare sau egală cu o valoare constantă, notată cu Δ, ce reprezintă dimensiunea unei ferestere logice.

$m_i(t)$ va reprezenta numărul de pagini fizice alocat procesului p_i şi va fi ajustat astfel:

-Daca $Y_t = \Phi$ si $X_t \neq \Phi$,atunci

$m_i(t) = m_i(t-1) + 1$ (se alocă o pagină fizică);

– Daca $X_t = \Phi$ şi $Y_t = \Phi$, atunci

$$m_i(t) = m_i(t-1)$$

- Daca $X_t = \Phi$ şi $Y_t \neq \Phi$

atunci $m_i(t) = m_i(t-1) - 1$

(una dintre paginile fizice este luată procesului).

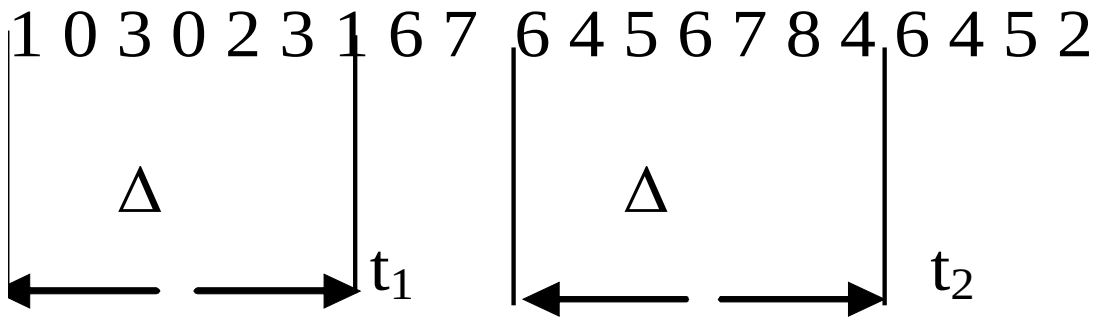
- $S(mi(t))$ rezultat este numit set de lucru al procesului p_i la momentul t, relativ la dimensiunea ferestrei Δ.

- Vom nota cu $W(t,\Delta)$ setul de lucru relativ la parametrul Δ şi la timpul virtual t al procesului.
- $W(t,\Delta)$ este mulţimea de pagini pe care procesul le-a cerut în ultimele Δ unităţi de timp virtual.
- Variabila Δ este un interval de timp virtual (fereastră) în care procesul este observat. Setul de lucru are proprietatea

$$W(t,\Delta+1) \supseteq W(t,\Delta)$$

adică dimensiunea setului de lucru este o funcţie ne-descrescătoare de Δ.

- **Exemplu**. Considerăm $\Delta=7$ şi şirul de referinţe prezentat în figura urm; la valorile timpului virtual al procesului t_1 =7 şi t_2=16, WS(t1)={0, 1, 2, 3}, WS(t2)={4, 5, 6, 7, 8}.



- **Observaţie.**

  1. Cele două seturi de lucru prezentate în exemplul ant. sunt disjuncte, adică ele corespund la localităţi diferite.

  2. Sistemul de operare alocă fiecărui proces un număr de pagini fizice corespunzător dimensiunii setului de lucru asociat; dacă suma dimensiunilor seturilor de lucru corespunzătoare proceselor, este mai mică decât dimensiunea memoriei interne atunci sistemul mai poate executa încă un proces. Reciproc, dacă suma dimensiunilor seturilor de lucru ale proceselor depăşeşte dimensiunea memoriei interne, atunci unul dintre procese va fi suspendat.

  3. Nu există o metodă riguroasă prin care să se determine valoarea parametrului Δ. O valoare prea mică va genera „pagini lipsă”, pe când o valoare prea mare va genera suprapunerea localităţilor.

- **Exemplu:** În figura urm. sunt prezentate diverse moduri de alocare a spaţiului de memorie fizică al procesului pentru diverse valori ale lui Δ. Observăm că în cazul a) numărul de situaţii de „pagină lipsă” este 16, pe când în cazul b) acesta se reduce la 6.
- a) Setul de lucru cu Δ=3.

0 1 2 3 0 1 2 3 0 1 2 3 4 5 6 7

0* 0 0 3* 3 3 2* 2 2 1* 1 1 4* 4 4 7*
1* 1 1 0* 0 0 3* 3 3 2* 2 2 5* 5 5
2* 2 3 1* 1 1 0* 0 0 3* 3 3 6* 6

} pagini fizice

- b) Setul de lucru cu Δ=4.

0 1 2 3 0 1 2 3 0 1 2 3 4 5 6 7

0* 0 0 0 0 0 0 0 0 0 0 0 4* 4 4 4
1* 1 1 1 1 2 1 1 1 1 1 1 5* 5 5
2* 2 2 2 2 2 2 2 2 2 2 2 6* 6
3* 3 3 3 3 3 3 3 3 3 3 3 7*

} pagini fizice

- **Observaţii.**
- - În cursul execuţiei lor, procesele trec dintr-o localitate în alta.
- - Atunci când se află într-o localitate, numărul paginilor utilizate (setul de lucru) tinde să se stabilizeze; atunci când procesele se află într-o stare de tranziţie dintr-o localitate în alta dimensiunea setului de lucru creşte, datorită faptului că se fac noi referinţe către pagini care corespund noii localităţi, care se adauga la paginile deja existente, coresp. vechii localitati.
- Deoarece dimensiunea setului de lucru are o limită, se pune problema înlocuirii unor pagini din setul de lucru curent cu alte pagini.
- Conceptul de set de lucru poate fi folosit pentru a realiza o strategie pentru modificarea mulţimii paginilor rezidente din memoria internă alocate procesului. Astfel:

  - se monitorizează setul de lucru al fiecărui proces,

  - periodic, se scot dintre paginile rezidente ale proceselor, cele care nu mai sunt în setul de lucru, pe baza unei strategii de tip LRU;

  - un proces poate fi executat numai dacă setul său de lucru este în memoria internă, adică mulţimea paginilor rezidente din memoria internă include setul de lucru.

- Deoarece ţine cont de principiul localităţii, această strategie poate fi aplicată, conducând la minimizarea numărului situaţiilor în care apare fenomenul de lipsă de pagină.
- Trebuie rezolvate anumite probleme legate de:

  - dimensiunea setului de lucru şi paginile care fac parte din acesta se schimbă în timp;

  - nu se poate estima setul de lucru al fiecărui proces în timp;

  - nu se poate stabili o valoare optimă a parametrului Δ.
- Următorul algoritm de actualizare a setului de lucru nu se bazează direct pe referinţele de pagină, ci pe dimensiunea numărului de apariţii(rata) a lipsei de pagină al procesului.

- Algoritmul **PFF** (**P**age **F**ault **F**requency) foloseşte un bit pentru fiecare pagină din memorie; acest bit este setat când pagina este accesată.
- 1. Când apare o lipsă de pagină, sistemul de operare observă timpul virtual de la ultima lipsă de pagină a acelui proces. Pentru contorizarea acestuia, se foloseşte o variabilă care contorizează referinţele la pagini.
- 2. De asemenea, se foloseşte un prag notat cu *F*. Dacă valoarea timpului de la ultima lipsă de pagină este mai mic sau egal cu *F*, atunci pagina este adaugată mulţimii de pagini rezidente ale procesului din memorie.
- 3. In caz contrar, se renunţă la toate paginile cu bitul de utilizare setat pe `0` şi se micşorează corespunzător mulţimea paginilor rezidente.
- 4. În acelaşi timp, se resetează bitul de utilizare al paginilor rezidente rămase ale procesului.
- **Observaţii.**

  1. Metoda poate fi îmbunătăţită prin utilizarea a două praguri: un prag superior, utilizat pentru a declanşa o creştere a dimensiunii mulţimii paginilor rezidente şi un prag inferior care este utilizat pentru diminuarea numărului paginilor rezidente.

  2. Metoda prezentată are dezavantajul următor: deoarece paginile nu sunt eliminate din memorie decât după F unităţi de timp virtual, atunci când se trece dintr-o localitate în alta are loc o succesiune de situaţii de lipsă de pagină, corespunzătoare noii localităţi, fără ca paginile corespunzătoare localităţii anterioare să fie eliminate; astfel mulţimea paginilor rezidente din memorie creşte şi unele dintre pagini nu mai sunt necesare.

- Politica **VSWS** (**V**ariable-interval **S**ampled **W**orking **S**et) evaluează setul de lucru la anumite instanţe ale executiei pe baza valorii timpului virtual trecut.
- La începutul intervalului selectat, bitul de utilizare coresp. tuturor paginilor rezidente ale proceselor este resetat; la sfârşit, numai paginile care au fost cerute pe parcursul intervalului vor avea bitul de utilizare setat; aceste pagini sunt reţinute în setul rezident al procesului pe parcursul următorului interval, iar la celelalte se renunţă.
- În decursul intervalului, orice pagină lipsă este adăugată setului rezident; astfel, setul rezident poate să ramână neschimbat sau să crească pe parcursul intervalului.
- Strategia VSWS utilizează trei parametri:

  M: durata minimă a intervalului de selecţie;

  L: durata maximă a intervalului de selecţie;

  Q: numărul situaţiilor de lipsă de pagini care este permis să apară între două instante de selecţie (valori ale timpului virtual, intre care se face verificarea) .

- Algoritmul VSWS este:

  1. Dacă timpul virtual trecut de la ultima instanţă de selecţie atinge L, atunci se suspendă procesul şi se verifica biţii de utilizare.

  2. Dacă, înainte de un timp scurs mai mic sau egal cu L, apar Q lipsă de pagină, atunci:

  a**.** Dacă timpul virtual trecut de la ultima instanţă de selecţie este mai mic decât M, atunci se aşteaptă până când timpul virtual trecut ajunge la valoarea M pentru a suspenda procesul şi a verifica biţii de utilizare.

  b. Dacă timpul virtual trecut de la ultima instanţă de selecţie este mai mare sau egal cu M, se suspendă procesul şi se verifica biţii de utilizare.
- Valorile parametrilor sunt luate astfel încât setarea intervalul de observaţie să fie declanşată după apariţia a Q lipsă de pagină, după ultima verificare.
- **Obs.**
- 1**.** Valorile parametrilor sunt astfel selectate încât intervalul de selectie sa fie modif. la aparitia a Q situaţii de lipsă de pagină, după ultima verificare (cazul 2b).
- 2. Ceilalti doi parametri (M şi L) furnizează limite intre care apare situatia că datorita trecerii dintr-o localitate in alta, are loc fenomenul de lipsa de pagina si se impune renuntarea la anumite pagini si incarcarea altora.

Memoria cu acces rapid(„cache”)

- Memoria **cache** conţine copii ale unor blocuri din memoria operativă. Când CPU încearcă citirea unui cuvânt din memorie, se verifică dacă acesta există în memoria **cache**. Dacă există, atunci el este livrat CPU. Dacă nu, atunci el este căutat în memoria operativă, este adus în memoria cache împreună cu blocul din care face parte, după care este livrat CPU. Datorită vitezei mult mai mari de acces la memoria cache, randamentul general al sistemului creşte.
- Memoria cache este împărţită în mai multe părţi egale, numite **sloturi**. Un slot are dimensiunea unui bloc de memorie, a cărui dimensiune este o putere a lui 2. Fiecare slot conţine o zonă care conţine blocul de memorie operativă depus în slotul respectiv. Problema care se pune, este modul în care se face corespondenţa dintre blocurile din memoria operativă şi sloturile din memoria cache, precum şi politicile de înlocuire a sloturilor din memoria cache, cu blocuri din memoria fizică. Spunem că are loc o **proiecţie** a spaţiului memoriei operative în cel al memoriei cache.

Proiecţia directă

- Dacă c indică numărul total de sloturi din memoria cache, a este o adresă oarecare din memoria operativă, atunci numărul s al slotului în care se proiectează adresa a este s= a mod c.
- Dezavantajul metodei constă în faptul că fiecare bloc are o poziţie fixă în memoria cache. Dacă, de exemplu se cer accese alternative la două blocuri care ocupă acelaşi slot, atunci trebuie efectuate mai multe operaţii de înlocuire a conţinutului slotului care corespunde celor două blocuri.

- **Proiecţia asociativă.** Fiecare bloc de memorie este plasat în oricare dintre sloturile de memorie cache, care sunt libere. Înlocuirea conţinutului unui slot, cu conţinutul altui bloc din memoria operativă, se face pe baza unuia dintre algoritmii NRU, FIFO sau LRU.
- **Proiecţia set-asociativă** combină cele două metode prezentate anterior. Memoria cache este împărţită în i seturi, un set fiind compus din j sloturi.
- Avem relaţia c=i x j.
- Dacă a este o adresă de memorie, numărul k al setului în care va intra blocul, este dat de: K=a mod i.
- Cunoscând numărul setului, blocul va ocupa unul dintre sloturile acestui set, pe baza unuia dintre algoritmii de înlocuire prezentaţi anterior.