Memoria *cache*

- Memoria *cache*
  - Principiul memoriei *cache*
  - Organizarea memoriei *cache*
  - Funcționarea memoriei *cache*
  - Translatarea adreselor
    - Translatarea asociativă
    - Translatarea directă
    - Translatarea cu seturi asociative
  - Strategii de înlocuire
  - Performanța memoriei *cache*

Performanța memoriei *cache* (1)

- O metrică de evaluare a performanței: timpul extins al UCP

$$t_{UCP} = C_{UCP} \times t_C$$

  - C_{UCP} – numărul ciclurilor de ceas ale UCP
  - t_C – durata ciclului de ceas
- S-a introdus numărul mediu al ciclurilor de ceas pe instrucțiune pentru execuție, CPI

$$\mathrm{CPI} = \frac{C_{UCP}}{N}$$

$$t_{UCP} = N \times \mathrm{CPI} \times t_C$$

Performanța memoriei *cache* (2)

- Se va ține cont de numărul ciclurilor de ceas necesare accesului la memorie, M_{UCP}

$$t_{UCP} = (C_{UCP} + M_{UCP}) \times t_C$$

- Numărul mediu al ciclurilor de ceas pe instrucțiune pentru accesul la memorie, CMPI

$$\text{CMPI} = \frac{M_{UCP}}{N}$$

$$t_{UCP} = N \times (\text{CPI} + \text{CMPI}) \times t_C$$

- Presupunem că C_{UCP} include timpul de acces la memoria *cache* în caz de succes

Performanța memoriei *cache* (3)

- M_{UCP} depinde de:
  - Numărul de eșecuri la accesul memoriei, N_e
  - Penalizarea pentru eșec, P_e

$$M_{UCP} = N_e \times P_e$$

- Se introduc valorile medii pe instrucțiune:
  - Eșecuri la memorie pe instrucțiune, EMPI

$$\mathrm{EMPI} = \frac{N_e}{N}$$

  - Accese la memorie pe instrucțiune, AMPI (N_M – numărul total de accese la memorie)

$$\mathrm{AMPI} = \frac{N_M}{N}$$

Performanța memoriei *cache* (4)

$$M_{UCP} = N_e \times P_e = N \times \text{EMPI} \times P_e$$

- Rata de eșec, R_e

$$R_e = \frac{N_e}{N_M}$$

$$M_{UCP} = N_e \times P_e = N_M \times R_e \times P_e$$

$$M_{UCP} = N \times \text{AMPI} \times R_e \times P_e$$

- Exprimarea valorii EMPI

$$\text{EMPI} = \frac{N_e}{N} = \frac{N_M}{N} \times R_e$$

$$\text{EMPI} = \text{AMPI} \times R_e$$

Performanța memoriei *cache* (5)

- Exemplul 3.8
  - Calculator cu valoarea CPI = 1 atunci când pentru toate accesele la memorie se obțin succese în memoria *cache*
  - Datele se accesează prin instrucțiunile *Load* și *Store* $\rightarrow$ 50% din totalul instrucțiunilor
  - Rata de eșec: 1%
  - Penalizarea pentru eșec: 100 cicluri de ceas
  - Care ar fi creșterea vitezei dacă pentru toate accesele la memoria *cache* s-ar obține succese?

Performanța memoriei *cache* (6)

- Dacă C_{UCP} include timpul de acces la memoria *cache* în caz de succes:

$$t_{UCP\,ideal} = (C_{UCP} + M_{UCP}) \times t_C = (C_{UCP} + 0) \times t_C$$

$$t_{UCP\,ideal} = C_{UCP} \times t_C = N \times \mathrm{CPI} \times t_C = N \times t_C$$

- În cazul real, $M_{UCP} \neq 0$

$$\mathrm{AMPI} = 1 + 0{,}5 = 1{,}5$$

$$M_{UCP} = N \times \mathrm{AMPI} \times R_e \times P_e$$

$$M_{UCP} = N \times 1{,}5 \times 0{,}01 \times 100 = N \times 1{,}5$$

$$t_{UCP\,real} = (N + N \times 1{,}5) \times t_C = 2{,}5 \times N \times t_C$$

- Creșterea vitezei: 2,5

Performanța memoriei *cache* (7)

- Exemplul 3.9
  - Calculator cu valoarea CPI = 1 atunci când nu apar întârzieri datorate accesului la memorie
  - Penalizarea pentru eșec: 200 cicluri de ceas
  - Număr mediu de accese la memorie pe instrucțiune: 1,5
  - Număr de eșecuri la 1000 instrucțiuni: 30
  - Cu cât crește timpul de execuție dacă se ține cont de întârzierile datorate accesului la memorie?

Performanța memoriei *cache* (8)

- a) Utilizarea valorii EMPI
  - $\mathrm{EMPI} = 0{,}03$
  - Dacă $\mathrm{CPI} = 1$:

    $$t_{UCP\ ideal} = N \times \mathrm{CPI} \times t_C = N \times t_C$$
  - Dacă apar întârzieri la accesul memoriei:

    $$t_{UCP\ mem} = N \times (\mathrm{CPI} + \mathrm{CMPI}) \times t_C$$

    $$\mathrm{CMPI} = \frac{M_{UCP}}{N} = \frac{N \times \mathrm{EMPI} \times P_e}{N} = \mathrm{EMPI} \times P_e$$

    $$\mathrm{CMPI} = 0{,}03 \times 200 = 6$$

    $$t_{UCP\ mem} = N \times (1 + 6) \times t_C = 7 \times N \times t_C$$
  - Timpul de execuție crește de 7 ori

Performanța memoriei *cache* (9)

- b) Utilizarea ratei de eșec R_e

$$\text{EMPI} = \text{AMPI} \times R_e$$

$$\text{EMPI} = 0{,}03$$

$$\text{AMPI} = 1{,}5$$

$$R_e = \frac{\text{EMPI}}{\text{AMPI}} = \frac{0{,}03}{1{,}5} = \frac{3}{150} = \frac{1}{50} = 0{,}02$$

- Dacă apar întârzieri la accesul memoriei:

$$t_{UCP\,mem} = N \times (\text{CPI} + \text{CMPI}) \times t_C$$

$$\text{CMPI} = \frac{M_{UCP}}{N} = \frac{N \times \text{AMPI} \times R_e \times P_e}{N} = \text{AMPI} \times R_e \times P_e$$

$$\text{CMPI} = 1{,}5 \times 0{,}02 \times 200 = 1{,}5 \times 4 = 6$$

$$t_{UCP\,mem} = N \times (1 + 6) \times t_C = 7 \times N \times t_C$$

Performanța memoriei *cache* (10)

- Altă metrică de evaluare a performanței: timpul de acces mediu
  - Metrică indirectă a performanței
  - Presupunem că timpul de acces la memoria *cache* în caz de succes este inclus în M_{UCP}
  - t_S – timpul de acces în caz de succes

$$t_{A\ total} = N_M \times t_S + N_e \times P_e$$

$$t_{A\ mediu} = \frac{t_{A\ total}}{N_M} = t_S + \frac{N_e}{N_M} \times P_e$$

$$t_{A\ mediu} = t_S + R_e \times P_e$$

Performanța memoriei *cache* (11)

- Exemplul 3.10
  - Memorii *cache* separate
    - Memorie *cache* de instrucțiuni: 3,8 eșecuri la 1000 instrucțiuni
    - Memorie *cache* de date: 40 eșecuri la 1000 instrucțiuni
  - Transferurile de date reprezintă 36% din totalul instrucțiunilor
  - Timp de succes: 1 ciclu de ceas
  - Penalizarea pentru eșec: 100 cicluri de ceas

Performanța memoriei *cache* (12)

- Se ignoră întârzierile la operațiile de scriere
- Care este timpul de acces mediu pentru fiecare memorie *cache*?

- Date inițiale:
  - $\mathrm{EMPI}_{instr} = 0{,}0038$
  - $\mathrm{EMPI}_{date} = 0{,}04$
  - $t_S = 1$ (ciclu de ceas)
  - $P_e = 100$ (cicluri de ceas)
- $t_{A\ mediu} = t_S + R_e \times P_e$

Performanța memoriei *cache* (13)

- Ratele de eșec:

$$R_{e\ instr} = \frac{\text{EMPI}_{instr}}{\text{AMPI}_{instr}} = \frac{0{,}0038}{1} = 0{,}0038$$

$$R_{e\ date} = \frac{\text{EMPI}_{date}}{\text{AMPI}_{date}} = \frac{0{,}04}{0{,}36} = \frac{4}{36} = \frac{1}{9} = 0{,}11$$

- Timpii de acces medii:

$$t_{A\ mediu\ instr} = t_S + R_{e\ instr} \times P_e$$
$$t_{A\ mediu\ instr} = 1 + 0{,}0038 \times 100 = 1 + 0{,}38$$
$$t_{A\ mediu\ instr} = 1{,}38 \text{ (cicluri de ceas)}$$
$$t_{A\ mediu\ date} = t_S + R_{e\ date} \times P_e = 1 + 0{,}11 \times 100$$
$$t_{A\ mediu\ date} = 1 + 11 = 12 \text{ (cicluri de ceas)}$$

Sisteme de memorie

- Ierarhia memoriilor
- Tipuri de memorii
- Memorii semiconductoare
- Memoria cu unități multiple
- Memoria asociativă
- Memoria *cache*
- Memoria virtuală

Memoria virtuală

- Memoria virtuală
  - Principiul memoriei virtuale
  - Translatarea adreselor
  - Paginarea
  - Segmentarea
  - Paginarea combinată cu segmentarea
  - Strategii de înlocuire

Principiul memoriei virtuale (1)

- Permite utilizarea unei memorii cu o dimensiune mai mare decât memoria fizică
- Permite multiprogramarea
  - Memoria principală este partajată între mai multe programe sau utilizatori
- Sistemul de memorie este adresat printr-un set *V* de *adrese logice* sau *virtuale*
  - Nu se referă la același spațiu din memoria fizică la diferitele execuții

Principiul memoriei virtuale (2)

- Memoria fizică este adresată printr-un set *R* de *adrese fizice* sau *reale*
  - Identifică locațiile din memoria fizică
- Adresele virtuale – sunt generate în timpul compilării și sunt translatate în adrese fizice în timpul execuției
- Se utilizează o funcție pentru translatarea adreselor
  - $f : V \rightarrow R$

Memoria virtuală

- Memoria virtuală
  - Principiul memoriei virtuale
  - Translatarea adreselor
  - Paginarea
  - Segmentarea
  - Paginarea combinată cu segmentarea
  - Strategii de înlocuire

Translatarea adreselor (1)

- *Spațiul de adrese virtuale V*: setul de locații abstracte pe care le poate adresa un program
  - Adresele virtuale: specificate explicit sau implicit de identificatorii din program
- *Spațiul de adrese reale R*: definit de memoria fizică din calculator
  - O secvență liniară de numere → corespunde locațiilor adresabile de memorie

Translatarea adreselor (2)

- Translatarea adreselor virtuale în adrese reale
- Translatarea poate fi efectuată:
  - În timpul compilării → compilator
  - La încărcarea programului pentru execuție → program încărcător
  - În timpul execuției programului → unitate de gestiune a memoriei (MMU – *Memory Management Unit*)

Translatarea adreselor (3)

- Translatare *statică*
  - Efectuată la compilare sau la încărcare
  - Spațiul adreselor reale ale programului este fix pe durata execuției
- Translatare *dinamică*
  - Efectuată la execuție
  - Spațiul adreselor virtuale ale programului poate varia în mod dinamic în timpul execuției

Translatarea adreselor (4)

- Un program executabil cuprinde un set de blocuri de instrucțiuni și date
- Un cuvânt *C* are propria *adresă efectivă* A_{ef}:

  $A_{ef} = B + D$

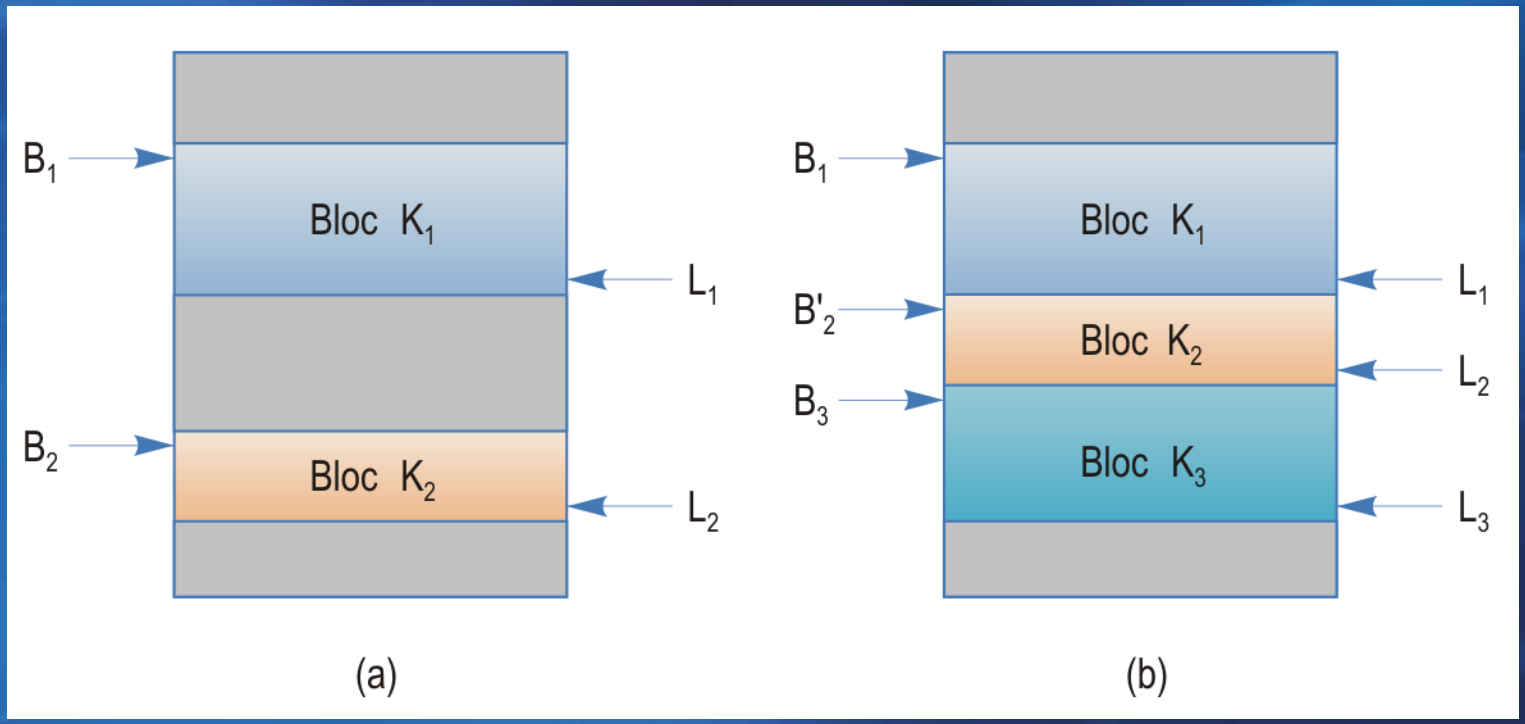
  B – adresa de bază a blocului → furnizează biții de ordin superior ai adresei efective

  D – deplasamentul cuvântului din bloc → furnizează biții de ordin inferior ai adresei efective

Translatarea adreselor (5)

- O metodă de implementare a translatării: păstrarea adreselor de bază într-o *tabelă de adrese ale memoriei*
  - Tabela poate fi păstrată în memorie, în registrele UCP sau în ambele
  - Logica de generare a adreselor din UCP calculează adresa efectivă A_{ef}
  - Blocurile pot fi relocate simplu în memorie
    - Modificarea adreselor de bază

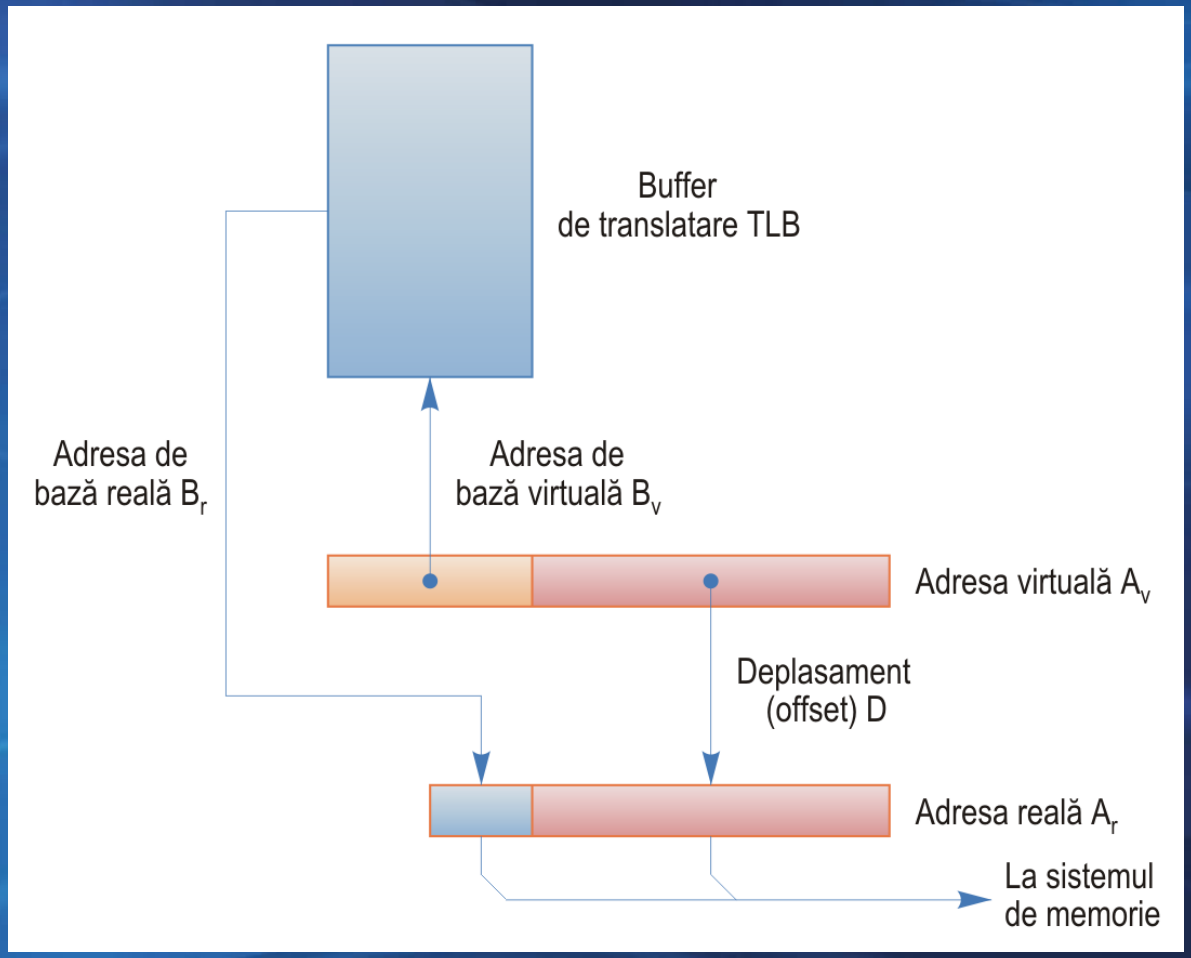
Translatarea adreselor (6)



Translatarea adreselor (7)

- Translatarea dinamică a adreselor:
  - Trebuie controlate referințele efectuate de un bloc la locațiile din afara zonei asignate acestuia
  - Scrierea în afara zonei asignate blocului trebuie prevenită
    - Specificarea adresei limită L_i
    - Specificarea dimensiunii blocului
  - B_i şi L_i sunt memorate în tabela de adrese

Translatarea adreselor (8)



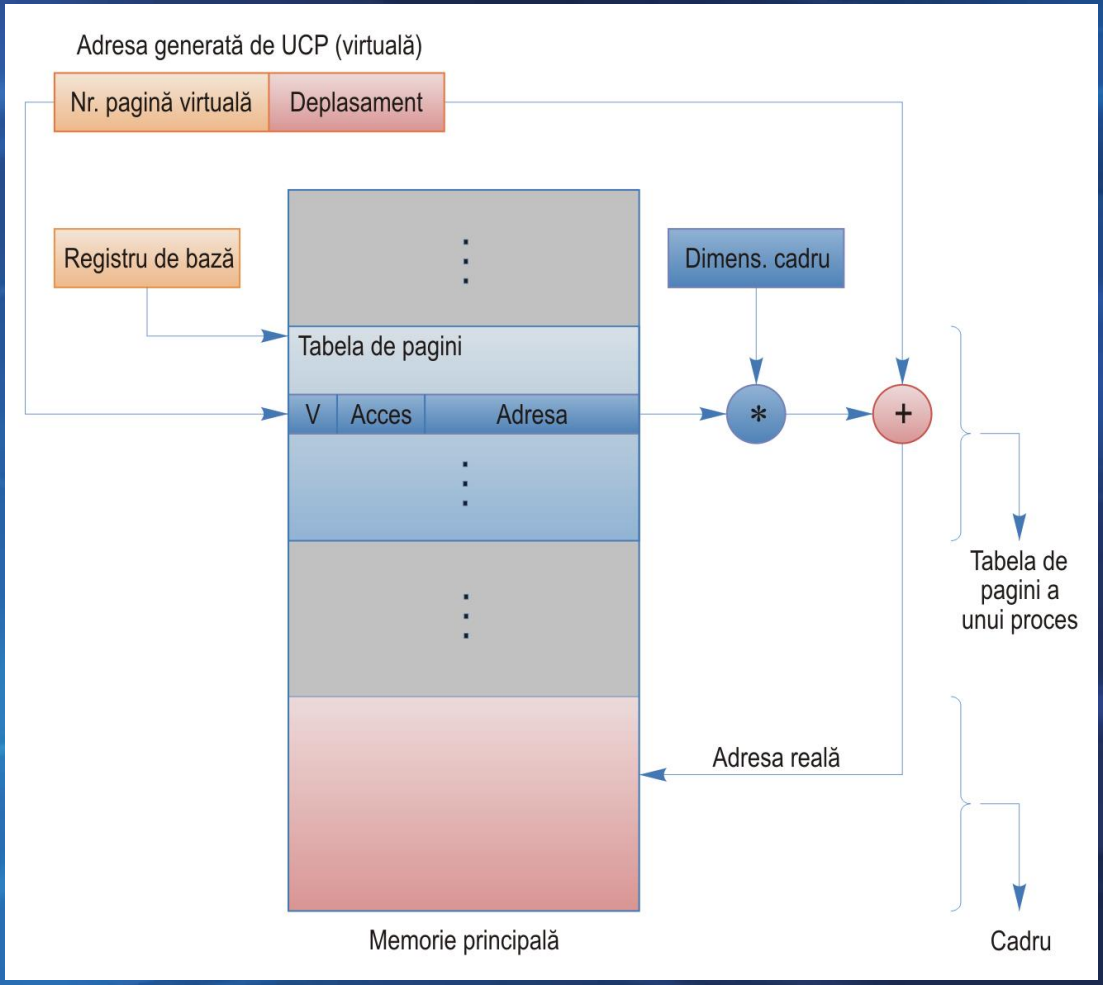
Memoria virtuală

- Memoria virtuală
  - Principiul memoriei virtuale
  - Translatarea adreselor
  - Paginarea
  - Segmentarea
  - Paginarea combinată cu segmentarea
  - Strategii de înlocuire

Paginarea (1)

- Divizarea unui program (proces) în blocuri cu dimensiuni identice
  - Blocurile sunt stocate în memoria secundară → *pagini*
- Paginile sunt încărcate în memoria principală → *cadre de pagină*
- Fiecare proces trebuie să păstreze în memoria principală o tabelă de adrese ale memoriei → *tabelă de pagini*

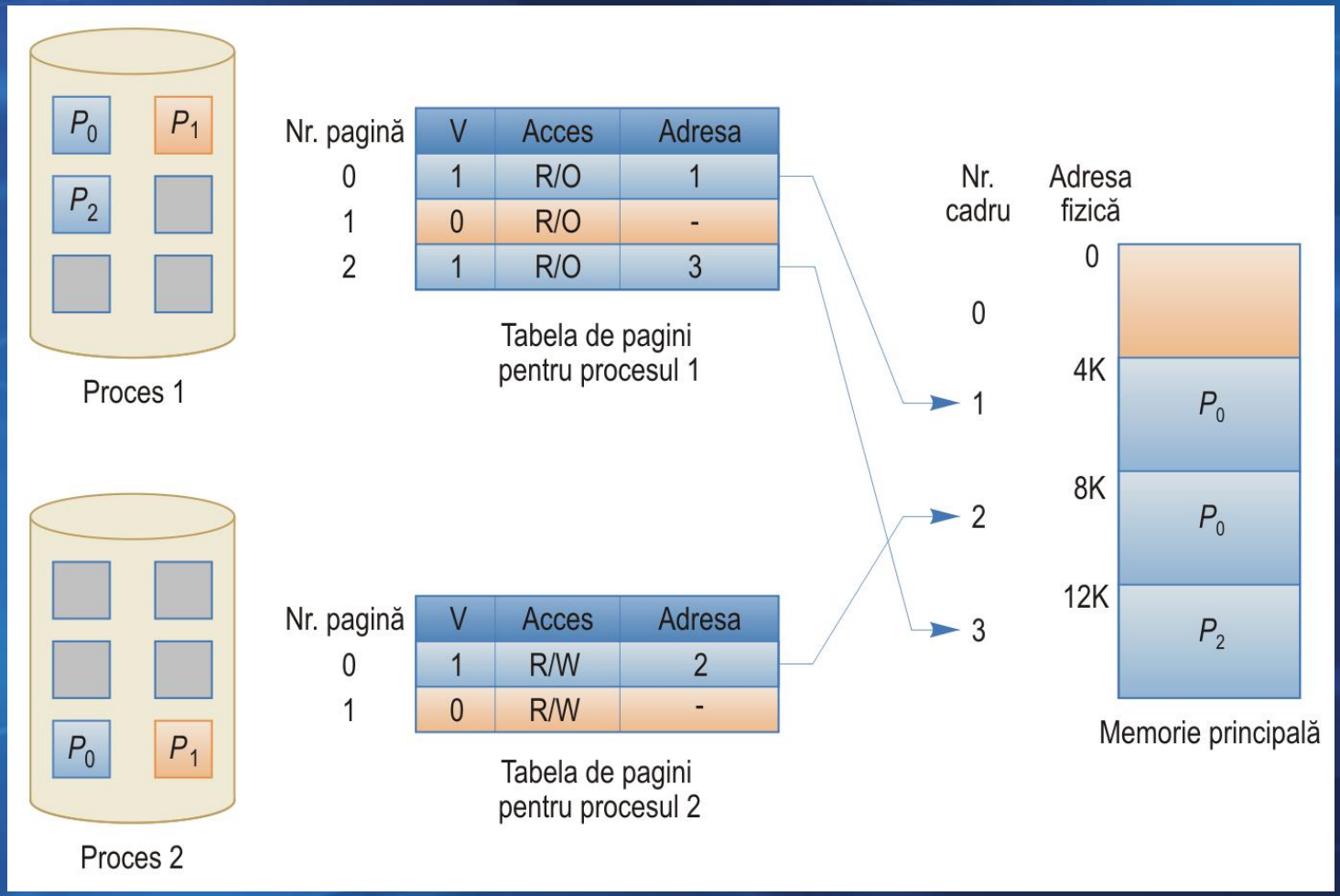
Paginarea (2)


Adresa generată de UCP (virtuală)
Nr. pagină virtuală
Deplasament
Registru de bază
Dimens. cadru
Tabela de pagini
V
Acces
Adresa
*
+
Tabela de pagini a unui proces
Adresa reală
Memorie principală
Cadru

Paginarea (3)

- Accesul unei variabile sau instrucțiuni care nu este încărcată în memorie: *lipsă de pagină* (“*page fault*”)
  - Pagina este depusă într-un cadru de pagină liber
  - Dacă nu există un cadru de pagină liber: trebuie selectată una din pagini pentru a fi înlocuită → *strategie de înlocuire*

Paginarea (4)


P_0
P_1
P_2
Proces 1
Nr. pagină
V
Acces
Adresa
0
1
R/O
1
1
0
R/O
-
2
1
R/O
3
Tabela de pagini pentru procesul 1
Nr. cadru
Adresa fizică
0
0
4K
1
P_0
8K
2
P_0
12K
3
P_2
Memorie principală
P_0
P_1
Proces 2
Nr. pagină
V
Acces
Adresa
0
1
R/W
2
1
0
R/W
-
Tabela de pagini pentru procesul 2

Paginarea (5)

- Exemplul 3.11
  - Sistem de memorie virtuală cu paginare
  - Dimensiunea paginii: 64 KB
  - Adrese virtuale: 40 biți
  - Adrese fizice: 32 biți
  - Intrările din tabela de pagini conțin: un bit de validare; un bit de acces; numărul cadrului de pagină
  - Adresele din memoria secundară nu sunt păstrate în tabela de pagini

Paginarea (6)

- Toate paginile virtuale sunt utilizate
- Care este dimensiunea totală a tabelei de pagini pentru fiecare proces?

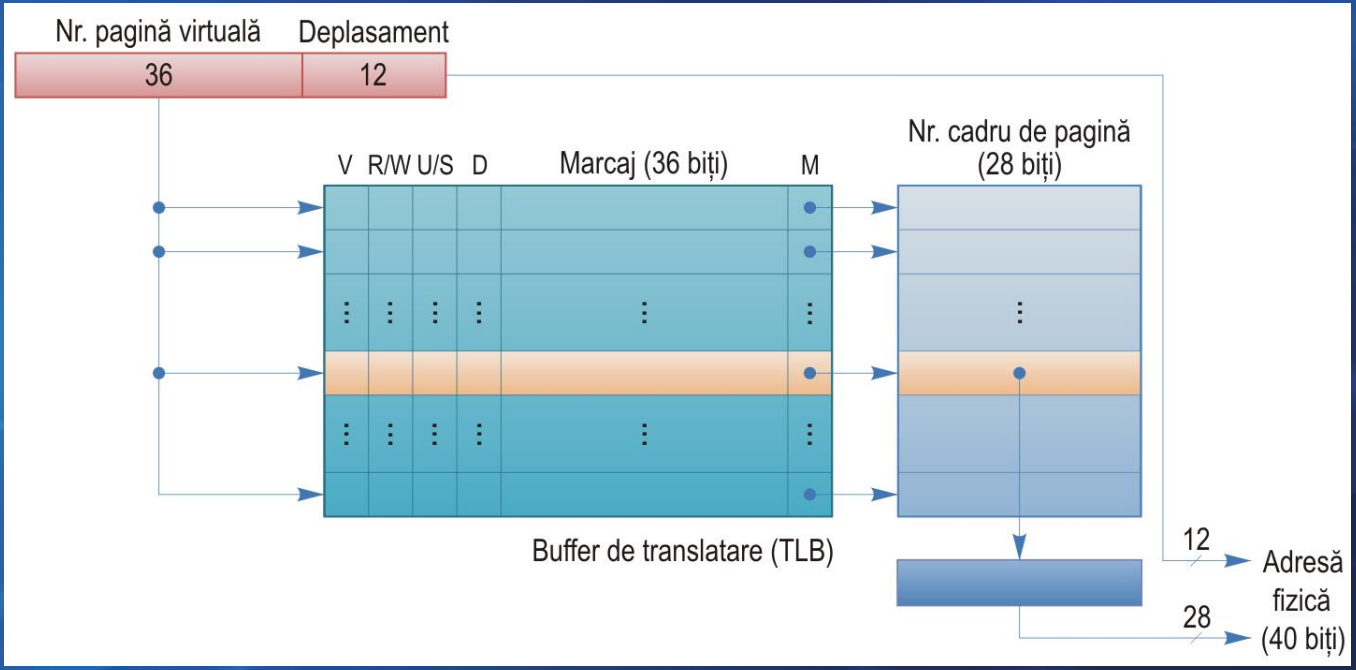
- Numărul de pagini virtuale: 2^{40} B / 64 KB = 2^{40} B / 2^{16} B = 2^{24} $\rightarrow$ intrări în tabela de pagini
- Numărul cadrelor de pagină: 2^{32} B / 64 KB = 2^{32} B / 2^{16} B = 2^{16} $\rightarrow$ 16 biți pentru nr. cadrului
- Dimensiunea tabelei de pagini: 2^{24} x (1+1+16) biți = 16 M x 18 biți = 288 Mbiți = 36 MB

Paginarea (7)

- Creșterea vitezei de conversie a adreselor virtuale în adrese fizice:
  - Se poate utiliza un buffer de translatare TLB (*Translation Lookaside Buffer*)
  - Adresa paginii este încărcată în buffer → este translatată într-un număr al cadrului de pagină
  - Poate fi implementat cu memorie asociativă
  - Exemplu: Bufferul de translatare al unui procesor Opteron

Paginarea (8)

- Memorie *cache* complet asociativă; 40 de intrări
- Biți: V; R/W; U/S (*User/Supervisor*); D (*Dirty*)
- Marcajele conțin numere de pagini virtuale
- Datele conțin numere ale cadrelor de pagină



Paginarea (9)

- Eficiența unui sistem de memorie virtuală depinde de minimizarea numărului lipsurilor de pagină
- Frecvența lipsurilor de pagină (PF – *Page Fault Frequency*): $PF = F / (F + S)$
  - F – numărul lipsurilor de pagină
  - S – numărul de accese cu succes
- Valoarea PF trebuie să fie cât mai redusă pentru a minimiza numărul acceselor la memoria secundară

Paginarea (10)

- Paginarea crește timpul de prelucrare necesar unui proces
  - Pot fi necesare două accese la memoria secundară
  - Execuția unui algoritm de înlocuire
- Numărul acceselor la memoria secundară poate fi redus prin adăugarea unui bit de modificare
  - Dacă acest bit este 0, nu este necesară scrierea paginii în memoria secundară

Memoria virtuală

- Memoria virtuală
  - Principiul memoriei virtuale
  - Translatarea adreselor
  - Paginarea
  - Segmentarea
  - Paginarea combinată cu segmentarea
  - Strategii de înlocuire

Segmentarea (1)

- Un proces este împărțit în secțiuni de lungime variabilă → *segmente*
- Fiecare proces păstrează o tabelă de segmente în memoria principală
- Segmentele au lungimi diferite și ele pot începe în orice zonă din memorie
  - Eliminarea unui segment din memorie nu asigură întotdeauna spațiu suficient pentru un alt segment

Segmentarea (2)

- Segment: set de cuvinte contigue, asociate logic
- Pentru adresarea unui cuvânt (sau octet) dintr-un segment se specifică:
  - O adresă de bază → *adresă de segment*
  - Un *deplasament* în cadrul segmentului
- Un proces și datele sale pot fi considerate ca o colecție de segmente înlănțuite

Segmentarea (3)

- Avantaje ale segmentării:
  - Limitele segmentelor corespund limitelor programului și ale datelor
  - Un segment poate fi modificat și recompilat fără a afecta alte segmente
  - Anumite tipuri de segmente variază în lungime în timpul execuției
- Dezavantaje ale segmentării:
  - Metodă de alocare mai complexă

Segmentarea (4)

- Comparație între paginare și segmentare:
  - Paginarea necesită un sistem mai simplu de alocare a memoriei decât segmentarea
  - Paginile nu au semnificație logică
  - La segmentare, apare fragmentarea externă a memoriei $\rightarrow$ algoritm de compactare
    - Spațiu inutilizabil între zonele ocupate
  - La paginare, poate apare fragmentarea internă: spațiu inutilizabil în interiorul unei pagini

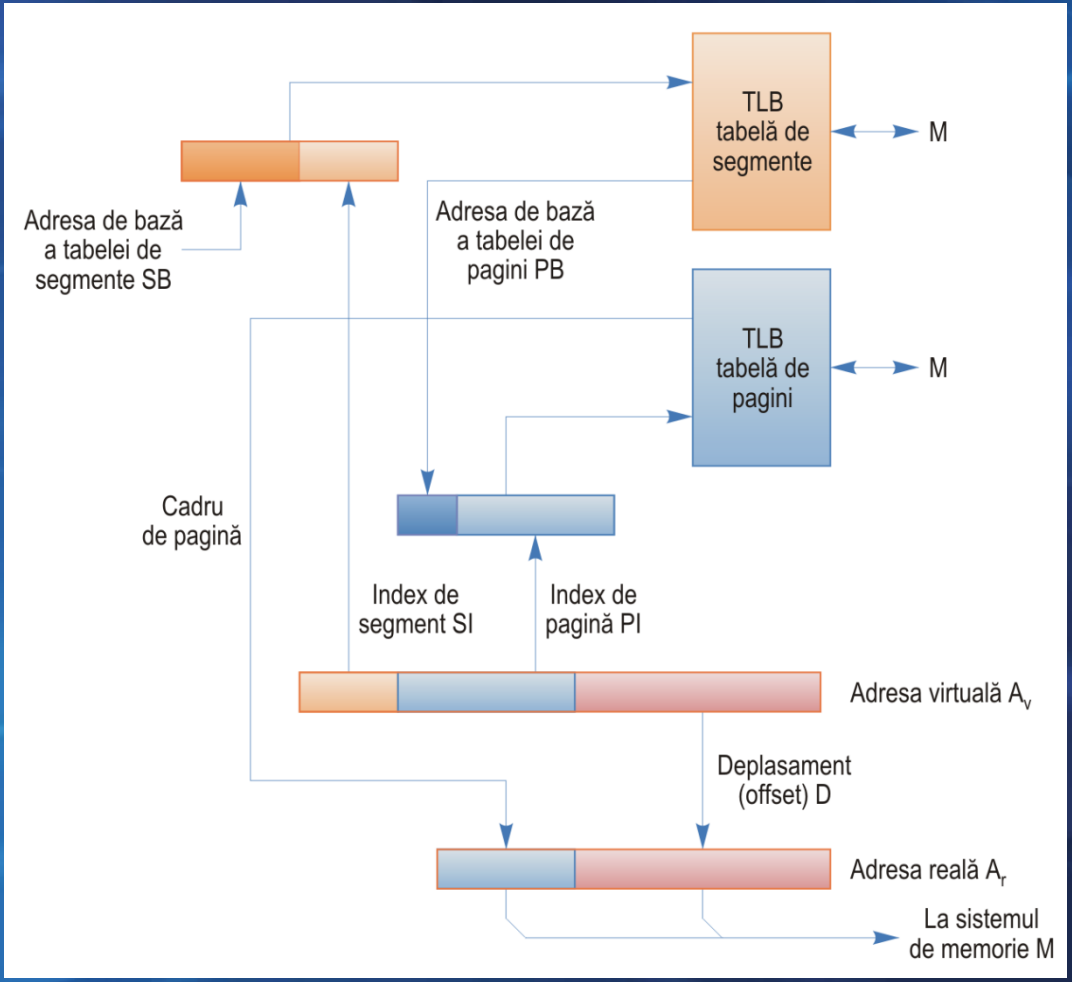
Memoria virtuală

- Memoria virtuală
  - Principiul memoriei virtuale
  - Translatarea adreselor
  - Paginarea
  - Segmentarea
  - Paginarea combinată cu segmentarea
  - Strategii de înlocuire

Paginarea combinată cu segmentarea (1)

- Fiecare segment este împărțit în pagini
- Avantajul: se elimină necesitatea de a plasa segmentul într-o zonă contiguă din memoria principală
  - Este nevoie doar de un număr de cadre de pagină egal cu numărul paginilor în care s-a împărțit segmentul
  - Cadrele de pagină nu trebuie să fie contigue $\rightarrow$ se simplifică amplasarea segmentelor în memoria principală

Paginarea combinată cu segmentarea (2)



Memoria virtuală

- Memoria virtuală
  - Principiul memoriei virtuale
  - Translatarea adreselor
  - Paginarea
  - Segmentarea
  - Paginarea combinată cu segmentarea
  - Strategii de înlocuire

Strategii de înlocuire (1)

- Criteriul pentru selecția unui bloc care va fi înlocuit: strategia de înlocuire
- Scopul: maximizarea ratei de succes a memoriei mai rapide M_1
- Majoritatea strategiilor iau în considerare principiul localității la selectarea unui bloc
- Eficiența: depinde de dimensiunea blocului și numărul de regiuni din memoria mai rapidă M_1

Strategii de înlocuire (2)

- Strategia optimă de înlocuire OPT
- Strategie ideală
- Ar trebui să determine, la momentul t_i , momentul $t_j > t_i$ la care apare următoarea referință la blocul K care se va înlocui
  - Blocul care se va înlocui este cel pentru care $t_j - t_i$ are valoarea maximă t_K
- Poate fi implementată prin efectuarea a două treceri peste programul de executat

Strategii de înlocuire (3)

- Prima trecere: o rulare de simulare
  - Se determină secvența S a adreselor virtuale generate de program
  - Valoarea t_K poate fi calculată din S
  - t_K – utilizată pentru a construi secvența optimă S_{OPT} a blocurilor care vor fi înlocuite
- A doua trecere: rularea pentru execuție
  - Se utilizează S_{OPT} pentru a specifica blocurile care trebuie înlocuite
- OPT nu este o strategie practică

Strategii de înlocuire (4)

- Strategia FIFO (*First In, First Out*)
- Blocul selectat pentru înlocuire: blocul cel mai puțin recent încărcat în memoria M_1
- Avantaj: simplu de implementat
  - Fiecărui bloc i se asociază un contor în lista spațiilor ocupate
  - La transferul unui bloc în memoria M_1, contoarele sunt actualizate

Strategii de înlocuire (5)

- Dezavantaj: nu ia în considerare principiul localității referințelor
  - Poate mări în mod semnificativ timpul necesar pentru execuția unui proces
  - Poate înlocui cu aceeași probabilitate blocuri utilizate intens și blocuri utilizate rar
- Exemplul 3.12A
  - Ilustrarea funcționării strategiei FIFO
  - Determinarea valorii PF

Strategii de înlocuire (6)

Timp	1	2	3	4	5	6	7	8	9	10	11
Secvența referințelor	0	1	2	0	3	2	0	1	2	4	0
n = 3 cadre	0	0	0*	0*	3	3	3	3*	2	2	2*
		1	1	1	1*	1*	0	0	0*	4	4
			2	2	2	2	2*	1	1	1*	0
	F	F	F	S	F	S	F	F	F	F	F

$F = 9,\ S = 2,\ PF = 9 / (9 + 2) = 0{,}81\ (81\%)$

(a)

Timp	1	2	3	4	5	6	7	8	9	10	11
Secvența referințelor	0	1	2	0	3	2	0	1	2	4	0
n = 4 cadre	0	0	0	0	0*	0*	0*	0*	0*	4	4
		1	1	1	1	1	1	1	1	1*	0
			2	2	2	2	2	2	2	2	2*
					3	3	3	3	3	3	3
	F	F	F	S	F	S	S	S	S	F	F

$F = 6,\ S = 5,\ PF = 6 / (6 + 5) = 0{,}54\ (54\%)$

(b)

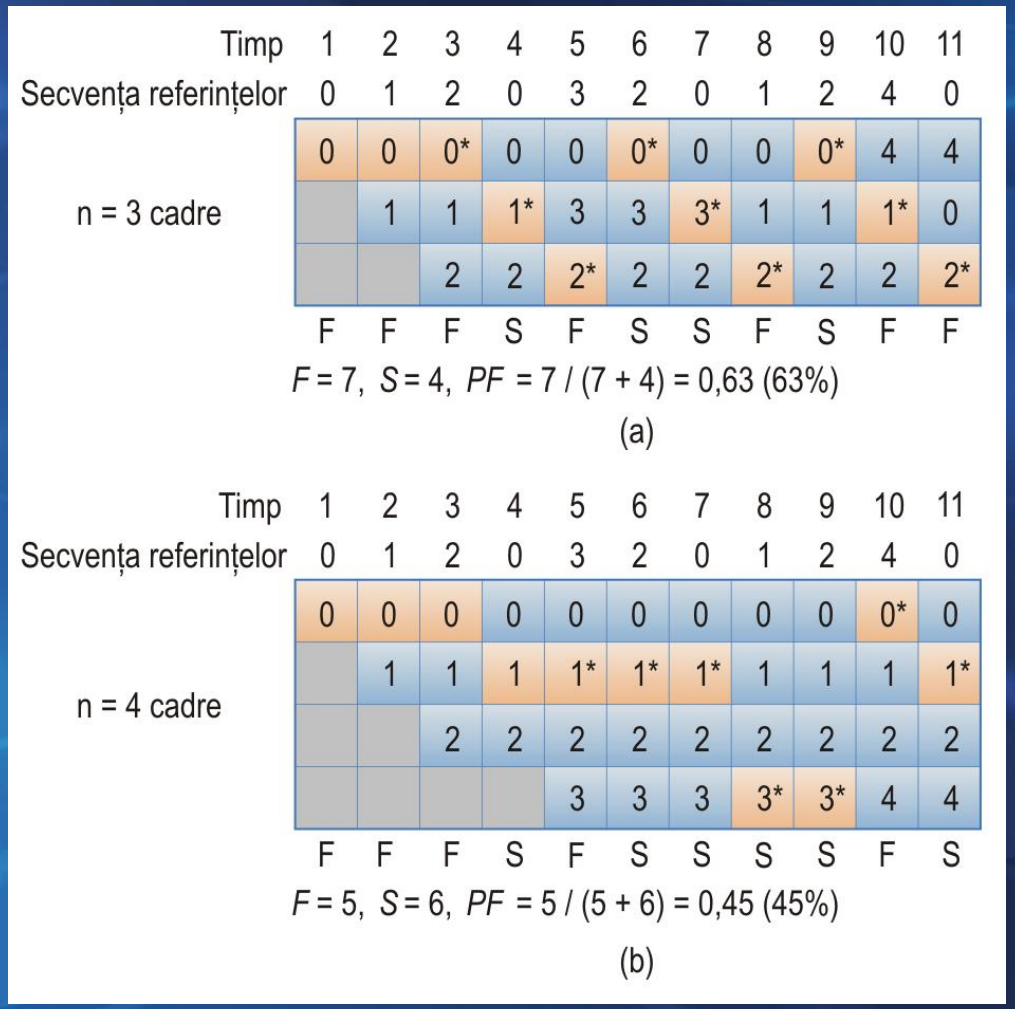
Strategii de înlocuire (7)

- Strategia LRU (*Least Recently Used*)
- Blocul selectat pentru înlocuire: blocul care nu a fost utilizat de cel mai mult timp
  - Presupunere: blocul cel mai puțin recent utilizat este cel mai puțin probabil de a fi utilizat în viitor
  - Evită înlocuirea blocurilor încărcate de mai mult timp, dar frecvent utilizate

Strategii de înlocuire (8)

- Dezavantaj: implementarea mai dificilă
  - Se asociază un contor cu fiecare bloc din M_1
  - La fiecare referire a unui bloc, contorul acestuia este setat la o valoare pozitivă
  - La intervale fixe de timp, contoarele tuturor blocurilor sunt decrementate
  - Se înlocuiește blocul al cărui contor conține valoarea cea mai mică
- Exemplul 3.12B
  - Ilustrarea funcționării strategiei LRU

Strategii de înlocuire (9)



Timp	1	2	3	4	5	6	7	8	9	10	11
Secvența referințelor	0	1	2	0	3	2	0	1	2	4	0
n = 3 cadre	0	0	0*	0	0	0*	0	0	0*	4	4
		1	1	1*	3	3	3*	1	1	1*	0
			2	2	2*	2	2	2*	2	2	2*
	F	F	F	S	F	S	S	F	S	F	F

$F = 7,\ S = 4,\ PF = 7 / (7 + 4) = 0{,}63\ (63\%)$

(a)

Timp	1	2	3	4	5	6	7	8	9	10	11
Secvența referințelor	0	1	2	0	3	2	0	1	2	4	0
n = 4 cadre	0	0	0	0	0	0	0	0	0	0*	0
		1	1	1	1*	1*	1*	1	1	1	1*
			2	2	2	2	2	2	2	2	2
					3	3	3	3*	3*	4	4
	F	F	F	S	F	S	S	S	S	F	S

$F = 5,\ S = 6,\ PF = 5 / (5 + 6) = 0{,}45\ (45\%)$

(b)

Rezumat (1)

- Metrici pentru evaluarea performanței memoriei *cache*: timpul extins al UCP, timpul de acces mediu
- Într-un sistem cu memorie virtuală, memoria principală și cea secundară se prezintă ca o memorie unică, adresabilă direct
- Este necesar un mecanism eficient pentru translatarea adreselor virtuale în adrese fizice
  - Se utilizează o tabelă de adrese ale memoriei
  - Pentru creșterea vitezei de translatare, se poate utiliza un buffer de translatare (TLB)

Rezumat (2)

- Metode pentru implementarea memoriei virtuale: paginarea, segmentarea
  - Paginarea: împărțirea unui proces în pagini cu dimensiuni identice
  - Segmentarea: împărțirea unui proces în segmente cu dimensiuni variabile
- Paginarea combinată cu segmentarea simplifică amplasarea segmentelor
- Strategia de înlocuire: selecția unui bloc care va fi înlocuit în memoria principală
  - Strategii practice de înlocuire: FIFO, LRU

Noțiuni, cunoștințe (1)

- Evaluarea performanței memoriei *cache* utilizând timpul extins al UCP
- Evaluarea performanței memoriei *cache* utilizând timpul de acces mediu
- Tipuri de translatare a adreselor virtuale în adrese reale
- Schema bloc generală pentru translatarea adreselor virtuale în adrese reale
- Principiul paginării
- Utilizarea tabelei de pagini pentru translatarea adreselor virtuale în adrese reale

Noțiuni, cunoștințe (2)

- Principiul bufferului de translatare TLB
- Frecvența lipsurilor de pagină
- Principiul segmentării
- Avantaje și dezavantaje ale segmentării
- Comparație între paginare și segmentare
- Fragmentarea memoriei
- Translatarea adreselor virtuale în adrese reale atunci când paginarea este combinată cu segmentarea
- Strategii de înlocuire pentru memoria virtuală