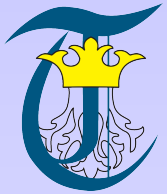


Universitatea *Transilvania* Brașov
Facultatea de Inginerie Electrică și Știința Calculatoarelor
Departamentul de Electronică și Calculatoare

Sisteme de operare

– curs 10 –

2012.05.15

ș.l. dr. ing. Kertész Csaba-Zoltán

3.5. Modelarea algoritmilor de înlocuire a paginilor

- intuitiv se pare că dacă vom avea mai multe page frame-uri în memorie, numărul page fault-uri va fi mai mic

nu este întotdeauna așa

- în 1969 Belady a descoperit un contraexemplu
- astfel fiecare algoritm de înlocuire a paginilor trebuie atent modelat pentru a măsura performanțele
- s-a dezvoltat o întreagă teorie a algoritmilor de paginare

3.5.1. Anomalia lui Belady

- algoritmul FIFO cauzează mai multe page fault-uri dacă se folosesc 4 frame în loc de 3 frame-uri
- exemplu: un program cu 5 pagini virtuale, paginile vor fi referite în ordinea 0 1 2 3 0 1 4 0 1 2 3 4
  - FIFO cu 3 pagini cauzează 9 page fault, FIFO cu 4 pagini cauzează 10 page fault

3.5.2. Algoritmi de tip *stivă*

- fiecare proces generează o secvență de referințe la memorie
- fiecare referință la memorie corespunde unei pagini virtuale
  - putem caracteriza referințele la memorie ale unui proces printr-o listă de numere de pagini, listă numită **reference string**
- pentru simplitate considerăm cazul unui singur proces care rulează pe mașină
  - astfel pentru fiecare mașină avem un singur reference string

Algoritmi de tip stivă

- un sistem de paginare poate fi caracterizat prin trei parametri
  - reference string pentru procesul care se execută
  - algoritmul de înlocuire a paginilor
  - numărul m al page frame-urilor disponibile
- să ne imaginăm un interpretor care lucrează astfel
  - menține un masiv M în care memorează starea memoriei (dimensiunea M este n = numărul de pagini virtuale pe care le utilizează procesul)

Modelul algoritmilor de tip stivă

- M este împărțit în două:
  - o parte va conține numărul paginilor care sunt în memorie (m)
  - a doua parte de dimensiune n-m conține paginile virtuale care nu sunt în memorie
- inițial M este vid
- de exemplu utilizăm algoritmul LRU pentru 8 pagini virtuale și 4 page frame

Modelarea algoritmilor

reference string	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	4
	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	4
		0	2	1	3	5	4	6	3	7	4	7	7	3	3	5	3	3	3	1	7	2	3
			0	2	1	3	5	4	6	3	3	4	4	7	7	7	5	5	5	3	1	7	2
				0	2	1	3	5	4	6	6	6	6	4	4	4	7	7	7	5	3	1	7
					0	2	1	1	5	5	5	5	5	6	6	6	4	4	4	4	5	4	1
						0	2	2	1	1	1	1	1	1	1	1	6	6	6	6	4	4	5
							0	0	2	2	2	2	2	2	2	2	2	2	2	2	6	6	6
									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page fault	p	p	p	p	p	p	p		p					p			p				p		p
distance string	∞	∞	∞	∞	∞	∞	∞	4	∞	4	2	3	1	5	1	2	6	1	1	4	7	4	6

modelul lucrează la fel de bine și cu alți algoritmi

Analiza modelului

- există o clasă de algoritmi care prezintă interes
- acești algoritmi au proprietatea:

$$M(m,r) \subseteq M(m+1,r)$$

  - m – numărul de page frame-uri
  - r – index în reference string
- mulțimea paginilor aflate în partea superioară a lui M pentru o memorie cu m page frame după r referințe la memorie este inclusă în mulțimea aflată în partea superioară a lui M pentru o memorie cu m+1 page frame-uri

Analiza modelului

- clasa de interes:
  - dacă mărim dimensiunea memoriei cu încă un page frame și reexecutăm procesul, la fiecare moment al execuției toate paginile care au fost prezente la prima rulare vor fi prezente și la a doua rulare plus încă o pagină adițională
  - algoritmul LRU are această proprietate, FIFO nu
- algoritmii care au această proprietate se numesc algoritmi de tip stivă
- acești algoritmi nu suferă de anomalia Belady

3.5.3. Distance string

- pentru algoritmi de tip stivă este convenabil să reprezentăm reference string într-un mod mai abstract
- o referință la o pagină va fi specificată prin distanța de la vârful stivei până la locul unde se află respectiva pagină în stivă
- paginile care nu au fost încă referite și deci nu se află în stivă distance string = ∞
- distance string depinde nu numai de reference string dar și de algoritmul de înlocuire a paginilor utilizate

3.5.4. Predicția ratei page fault-urilor

- putem utiliza distance string pentru a estima numărul de page fault-uri care vor apare pentru memorii având diferite dimensiuni
- algoritmul de estimare:
  - se scanează distance string pagină cu pagină
  - se memorează numărul de apariții pentru fiecare pagină
  - fie C_i numărul de apariții pentru pagina i
  - calculăm vectorul F cu formula:

$$F_m = \sum_{k=m+1}^{n} C_k + C_{inf}$$

Predicția ratei page fault-urilor

- pentru exemplul considerat anterior

$C_1 = 4$ $\quad F_1 = C_2 + C_3 + C_4 + \ldots + C_\infty = 20$

$C_2 = 2$ $\quad F_2 = C_3 + C_4 + C_5 + \ldots + C_\infty = 18$

$C_3 = 1$ $\quad F_3 = C_4 + C_5 + C_6 + \ldots + C_\infty = 17$

$C_4 = 4$ $\quad F_4 = C_5 + C_6 + C_7 + \ldots + C_\infty = 13$

$C_5 = 2$ $\quad F_5 = C_6 + C_7 + \ldots + C_\infty = 11$

$C_6 = 2$ $\quad F_6 = C_7 + \ldots + C_\infty = 9$

$C_7 = 1$ $\quad F_7 = C_8 + \ldots + C_\infty = 8$

$C_\infty = 8$ $\quad F_8 = 8$

- F_m reprezintă numărul de page fault-uri care vor apare pentru m page frame-uri

3.6. Considerații privind proiectarea sistemelor de paginare
3.6.1. Modelul Working Set

- când un proces se lansează în execuție nici o pagină de a sa nu se află în memorie
- CPU va încerca să încarce prima instrucțiune => page fault => se aduce pagina în memorie
- urmează alte page fault-uri până când majoritate paginilor necesare se află în memorie
- după care vor apare foarte puține page fault-uri

Modelul Working Set

- această strategie se numește **demand paging** deoarece paginile sunt încărcate numai când sunt necesare și nu în avans
- majoritatea proceselor accesează numai o mică fracțiune din numărul paginilor sale (**localitatea referințelor**)
- mulțimea paginilor pe care un proces le utilizează la un moment dat se numește **working set**

Modelul Working Set

- dacă Working Set este în memorie, procesul va determina apariția unui număr redus de page fault-uri până va trece într-o altă fază al execuției (alt Working Set)
- dacă Working Set nu poate fi ținut complet în memorie => page fault-uri dese
- un program care determină apariția unui page fault după câteva instrucțiuni în mod repetat se spune că este **thrashing**

Modelul Working Set

- în sistemele time sharing se încearcă memorarea informației despre working set, iar la comutarea proceselor înainte de lansa în execuție un proces, se va încărca în memorie working space respectiv
  - astfel se reduc page fault-urile
- aceste sisteme se mai numesc și **sisteme cu prepaginare**
- working set poate fi extras de exemplu din contorul metodei aging: dacă în primi n biți există un 1, pagina aparține working setului

3.6.2. Politici de alocare locale și globale

- alocarea memorie proceselor într-un sistem multi-tasking:
  - dacă se rulează mai multe procese și unul generează page fault ce pagini se consideră pentru înlocuire (numai paginile procesului care a generat page fault sau toate)
  - dacă algoritmul consideră numai paginile procesului: **algoritm local**
  - dacă algoritmul consideră și paginile celorlalte procese: **algoritm global**

Algoritmi locali

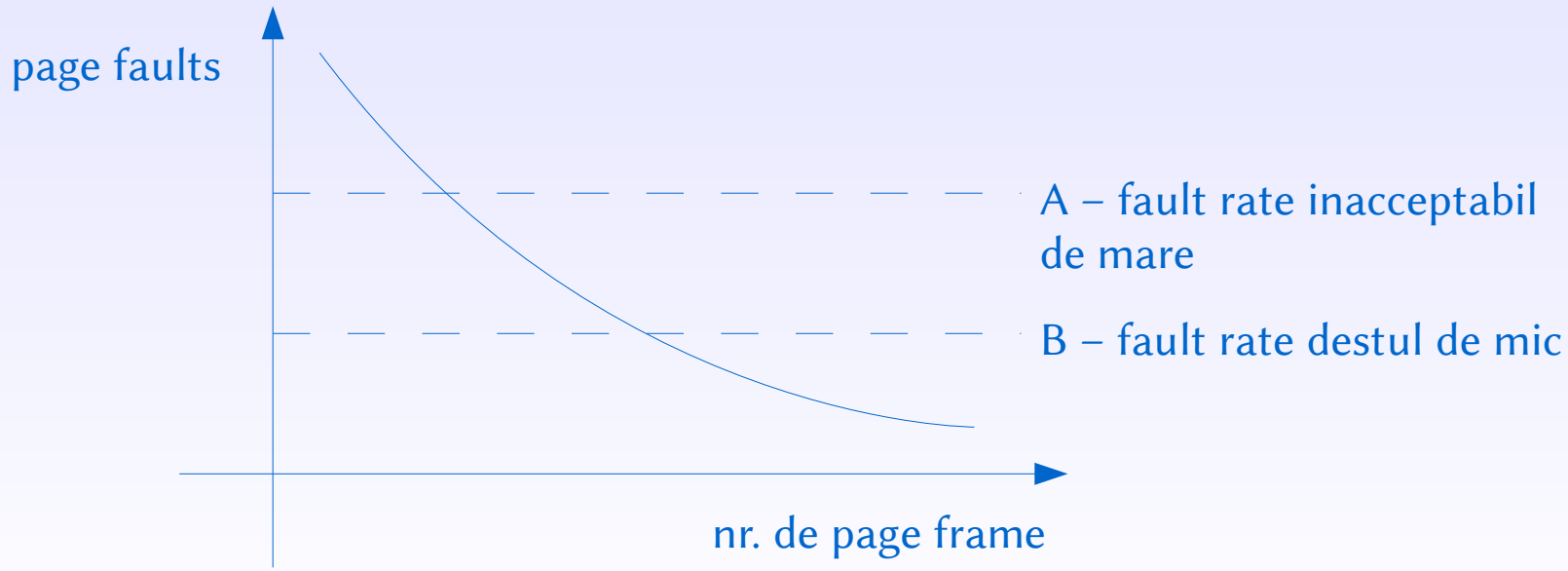
- fiecărui proces va trebui alocat o cantitate de memorie fixată
- aceasta poate rezulta într-o eficiență mai mică față de algoritmii globali (unde memoria poate fi alocată dinamic)
  - de exemplu dacă working set-ul unui proces se mărește, se vor genera multe page fault-uri, iar în memorie sunt page frame-uri libere
- o îmbunătățire este folosirea unui algoritm de alocare a page frame-urilor pentru procese

Algoritmi locali

- nu este eficient să alocăm același număr de page frame-uri unor procese de diferite dimensiuni
- vom folosi un algoritm care alocă memoria proporțional cu dimensiunea procesului
- nici această variantă nu rezolvă eficient problema
- o metodă mai eficientă este utilizarea algoritmului de alocare Page Fault Frequency

Algoritmii PFF

- în cazul algoritmilor de tip stivă fault rate descrește cu creșterea numărului de page frame-uri alocate
- algoritmii PFF vor aloca frame-uri astfel încât să mențină fault rate între limitele A și B



3.6.3. Dimensiunea paginilor

- chiar dacă hardware-ul implementează pagini de anumită dimensiune, SO poate lucra cu altă dimensiune de pagini
- alegând un program aleator, zonele de text, date și stivă nu vor umple complet un număr întreg de pagini – în medie o jumătate din ultima pagină nu va fi utilizată
- având n segmente în memorie și mărimea paginii p byte => $np/2$ byte vor fi pierduți prin fragmentare

Dimensiunea paginilor

- spațiu pierdut mic => **dimensiuni mici**
- pentru anumite sisteme tabela de pagini trebuie încărcată în regiștrii, la fiecare comutare de procese
- pentru o comutarea rapidă tabela de pagini trebuie să fie cât mai mică
- tabelă de pagini mică => **dimensiuni mari**

Analiza matematică

- s – mărimea medie a proceselor
- p – mărimea paginii
- e – dimensiunea unei linii din tabela de pagini
- numărul de pagini / proces = s/p
- dimensiunea tabelei / proces = se/p
- mărimea memoriei neutilizate / proces = $p/2$
- overheadul datorat tabelei de pagini și fragmentării interne = $se/p + p/2$

Analiza matematică

- primul termen este mare când pagina este mică
- al doilea termen este mare când mărimea paginii este mare
- derivând: $-se/p^2 + ½ = 0$
- mărimea optimă a paginii este: $p = \sqrt{2\,se}$
- exemplu: s = 128k, e=8byte => p=1448 ≈ 1kB
- în sistemele curente paginile sunt 512B – 8kB