

V.5. Gestiunea memoriei

Gestiunea memoriei

Funcții

- alocarea zonelor de memorie către aplicații
- prevenirea interferențelor între aplicații
- detectarea și oprirea acceselor incorecte

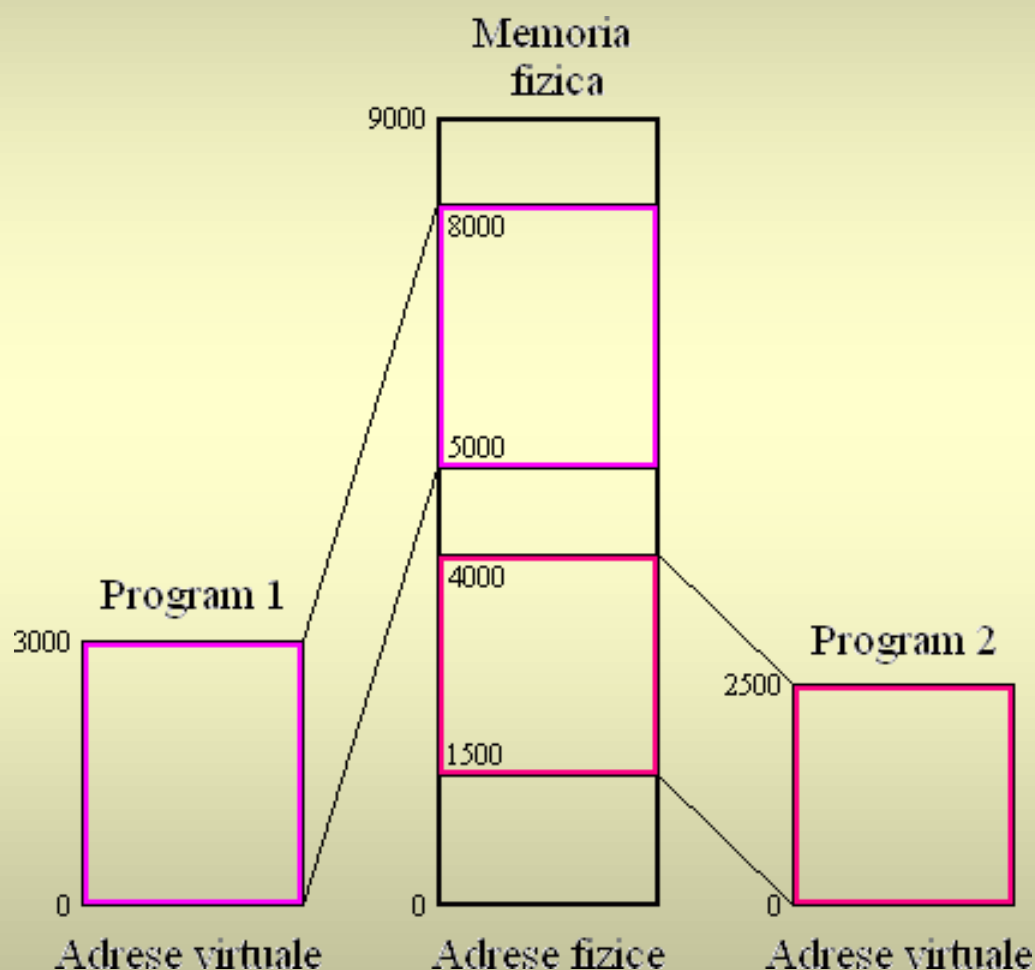
Problema fundamentală

- mai multe aplicații → zone de memorie disjuncte
- fiecare aplicație → anumite zone de memorie; care sunt aceste zone?
 - depind de ocuparea memoriei la acel moment
 - nu pot fi cunoscute la compilare

Soluția

- două tipuri de adrese
 - virtuale - aplicația crede că le accesează
 - fizice - sunt accesate în realitate
- corespondența între adresele virtuale și cele fizice - gestionată de sistemul de operare

Adrese virtuale și fizice (1)



Adrese virtuale și fizice (2)

Gestionarea adreselor fizice și virtuale

- 2 metode diferite
 - segmentare
 - paginare
- pot fi folosite și împreună
- componentă dedicată a procesorului - MMU (*Memory Management Unit*)

V.5.1. Segmentarea memoriei

Principiul de bază (1)

- segment - zonă continuă de memorie
- conține informații de același tip (cod, date etc.)
- vizibil programatorului
- adresa unei locații - formată din 2 părți
 - adresa de început a segmentului
 - deplasamentul în cadrul segmentului (*offset*)

Principiul de bază (2)

- la rulări diferite ale programului, segmentele încep la adrese diferite
- efectul asupra adreselor locațiilor
 - adresa de început a segmentului trebuie actualizată
 - deplasamentul - nemodificat
- problema este rezolvată numai parțial
 - dorim ca adresa să nu fie modificată deloc

Descriptori (1)

- descriptor de segment - structură de date pentru gestionarea unui segment
- informații reținute
 - adresa de început
 - dimensiunea
 - drepturi de acces
 - etc.

Descriptori (2)

- descriptorii - plasați într-un tabel
- accesul la un segment - pe baza indicelui în tabelul de descriptori (selector)
- adresa virtuală - 2 componente
 - indicele în tabelul de descriptori
 - deplasamentul în cadrul segmentului
- adresa fizică = adresa de început a segmentului + deplasamentul

Descriptori (3)

- la rulări diferite ale programului, segmentele încep la adrese diferite
- efectul asupra adreselor locațiilor
 - nici unul
 - trebuie modificată doar adresa de început a segmentului în descriptor
 - o singură dată (la încărcarea segmentului în memorie)
 - sarcina sistemului de operare

Accesul la memorie (1)

- programul precizează adresa virtuală
- identificare descriptor segment
- verificare drepturi acces
 - drepturi insuficiente - generare excepție
- verificare deplasament
 - dacă deplasamentul depășește dimensiunea segmentului - generare excepție

Accesul la memorie (2)

- dacă s-a produs o eroare la pașii anteriori
 - rutina de tratare a excepției termină programul
- dacă nu s-a produs nici o eroare
 - calcul adresă fizică (adresă început segment + deplasament)
 - acces la adresa calculată

Exemplificare (1)

Tabel descriptori (simplificat)

Indice	Adresa început	Dimensiune
0	65000	43000
1	211000	15500
2	20000	30000
3	155000	49000
4	250000	35000

Exemplificare (2)

Exemplu 1:

```
mov byte ptr ds:[eax], 25
```

- $ds = 3 \rightarrow$ adresa început segment = 155000
- $eax = 27348 < 49000$
 - deplasament valid (nu depășește dimensiunea segmentului)
- adresa fizică: $155000 + 27348 = 182348$

Exemplificare (3)

Exemplu 2:

```
add dword ptr ss:[ebp], 4
```

- $ss = 1 \rightarrow$ adresa început segment = 211000
- $ebp = 19370 > 15500$
 - deplasament invalid (depășește dimensiunea segmentului)
 - eroare \rightarrow generare excepție

Cazul Intel (1)

3 tabele de descriptori

- global (GDT - *Global Descriptor Table*)
 - accesibil tuturor proceselor
- local (LDT - *Local Descriptor Table*)
 - specific fiecărui proces
- de întreruperi (IDT - *Interrupt Descriptor Table*)
 - nu este direct accesibil aplicațiilor

Cazul Intel (2)

Segmentele - accesate cu ajutorul selectorilor

Structura unui selector (16 biți)

- primii 13 biți - indicele în tabelul de descriptori
 - maximum 8192 descriptori/tabel
- 1 bit - tabelul folosit (global/local)
- ultimii 2 biți - nivelul de privilegii
 - 0 - cel mai înalt, 3 - cel mai scăzut

Cazul Intel (3)

31	24				19		16	15	14	13	12	8		7	0		
BASE 31-24				G	D / B	0	A V L	LIMIT 19-16		P	DPL	TYPE			BASE 23-16		
BASE 15-0										LIMIT 15-0							

Cazul Intel (4)

- intervin nivelele de privilegii a 3 entități
 1. CPL (*Current Privilege Level*)
 - al procesului - reținut de procesor
 2. RPL (*Requested Privilege Level*)
 - cel solicitat - preluat din selector
 3. DPL (*Descriptor Privilege Level*)
 - cel al segmentului accesat - din descriptor

Cazul Intel (5)

- relațiile dintre aceste nivele de privilegii decid dacă se poate realiza accesul
- condiția pentru realizarea accesului:
 $CPL \leq DPL$ și $RPL \leq DPL$ (simultan)
- orice altă situație indică o încercare de acces la un nivel prea înalt
 - generare excepție

Zone libere și ocupate (1)

Problemă

- crearea unui segment nou → plasare în memorie
- este necesară o zonă liberă continuă suficient de mare
- pot exista mai multe asemenea zone - care este aleasă?

Zone libere și ocupate (2)

Algoritmi de plasare în memorie

- *First Fit* - prima zonă liberă găsită suficient de mare
- *Best Fit* - cea mai mică zonă liberă suficient de mare
- *Worst Fit* - cea mai mare zonă liberă (dacă este suficient de mare)

Zone libere și ocupate (3)



Fragmentare (1)

Fragmentarea externă a memoriei

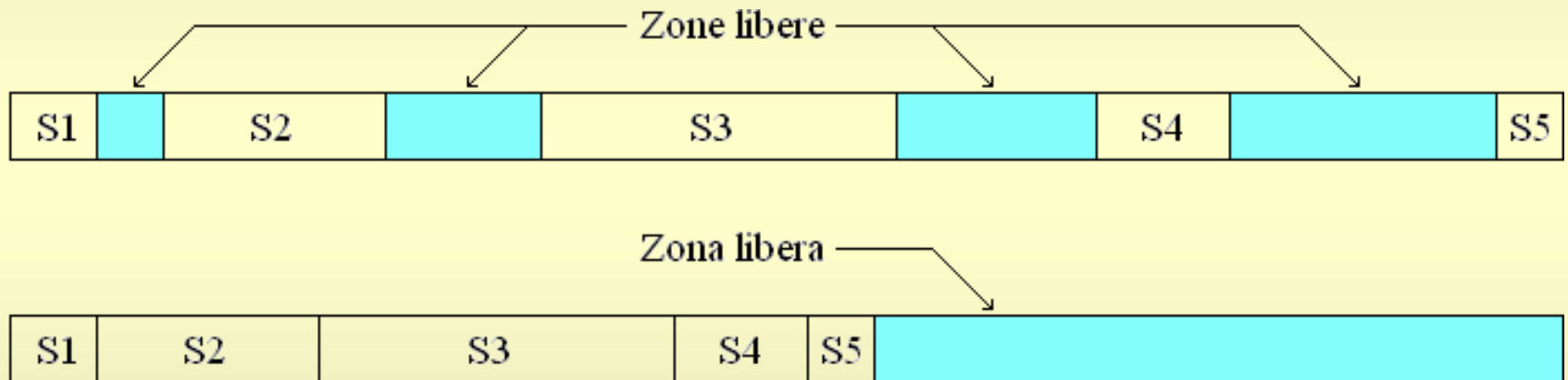
- multe zone libere prea mici pentru a fi utilizate
- apare după un număr mare de alocări și eliberări de segmente
- indiferent de algoritmul folosit
- plasarea unui segment poate eșua, chiar dacă spațiul liber total ar fi suficient

Fragmentare (2)

Eliminarea fragmentării externe

- compactarea memoriei
 - deplasarea segmentelor astfel încât să nu mai existe zone libere între ele
 - se crează o singură zonă liberă, de dimensiune maximă
 - realizată de un program specializat, parte a sistemului de operare

Compactare (1)



Compactare (2)

Rularea programului de compactare a memoriei

- consumă mult timp
 - mutarea segmentelor în memorie
 - actualizarea descriptorilor de segment
- nu poate fi rulat foarte des
- numai când este necesar

Compactare (3)

Situații în care se poate decide rularea programului de compactare

- când plasarea în memorie a unui segment eșuează din lipsă de spațiu
- la intervale regulate de timp
- când gradul de fragmentare a memoriei depășește un anumit nivel

Segmentarea - concluzii

Probleme ale mecanismului de segmentare

- gestiune complicată
 - suprapunerea segmentelor - greu de detectat
- fragmentarea externă - de obicei puternică
 - mult spațiu liber nefolosit
- compactarea consumă timp

V.5.2. Paginarea memoriei

Principiul de bază

- spațiul adreselor virtuale - împărțit în pagini (*pages*)
 - zone de dimensiune fixă
- spațiul adreselor fizice - împărțit în cadre de pagină (*page frames*)
 - aceeași dimensiune ca și paginile
- dimensiune - uzual 4 KB

Tabele de paginare (1)

- corespondența între pagini și cadre de pagină - sarcina sistemului de operare
- structura de bază - tabelul de paginare
- câte unul pentru fiecare proces care rulează
- permite detectarea acceselor incorecte la memorie

Tabele de paginare (2)

- la rulări diferite ale programului, paginile sunt plasate în cadre diferite
- efectul asupra adreselor locațiilor
 - nici unul
 - trebuie modificat tabelul de paginare
 - o singură dată (la încărcarea paginii în memorie)
 - sarcina sistemului de operare

Accesul la memorie

- programul precizează adresa virtuală
- se determină pagina din care face parte
- se caută pagina în tabelul de paginare
 - dacă nu este găsită - generare excepție
- se determină cadrul de pagină corespunzător
- calcul adresă fizică
- acces la adresa calculată

Exemplificare (1)

Tabel de paginare (simplificat)

Pagini	0	1	2	8	9	11	14	15
Cadre de pagină	5	7	4	3	9	2	14	21

Exemplificare (2)

Exemplu 1:

- dimensiunea paginii: 1000
- adresa virtuală: 8039
 - pagina: $[8039/1000]=8 \rightarrow$ cadrul de pagină 3
 - deplasament: $8039 \% 1000=39$ (în cadrul paginii)
- adresa fizică: $3 \cdot 1000 + 39 = 3039$

Exemplificare (3)

Exemplu 2:

- dimensiunea paginii: 1000
- adresa virtuală: 5276
 - pagina: $[5276/1000]=5$
 - nu apare în tabelul de paginare
 - eroare → generare excepție

Restricții

Construirea tabelelor de paginare

- sarcina sistemului de operare
- trebuie să evite suprapunerile între aplicații
- restricții
 - o pagină virtuală poate să apară pe cel mult o poziție într-un tabel de paginare
 - un cadru de pagină fizică poate să apară cel mult o dată în toate tabelele de paginare existente la un moment dat

Fragmentare (1)

Fragmentarea internă a memoriei

- între pagini nu există spațiu → nu apare fragmentare externă
- fragmentarea internă
 - spațiu liber nefolosit în interiorul unei pagini
 - nu poate fi preluat de alt proces
 - nu se poate face compactare
- mai puțin severă decât fragmentarea externă

Fragmentare (2)

Alegerea dimensiunii paginilor

- putere a lui 2 (nu rămân resturi de pagină)
- odată aleasă, nu se mai schimbă
- se stabilește ca un compromis
 - prea mare - fragmentare internă puternică
 - prea mică - mult spațiu ocupat de tabelele de paginare
 - uzual - 4 KB

Chiar atât de simplu?

Procesor pe 32 biți

- spațiu de adrese: 4 GB ($=2^{32}$)
 - dimensiunea paginii: 4 KB ($=2^{12}$)
- tabel cu 2^{20} elemente
- ar ocupa prea mult loc
 - consumă din memoria disponibilă aplicațiilor
- la procesoarele pe 64 biți - și mai rău

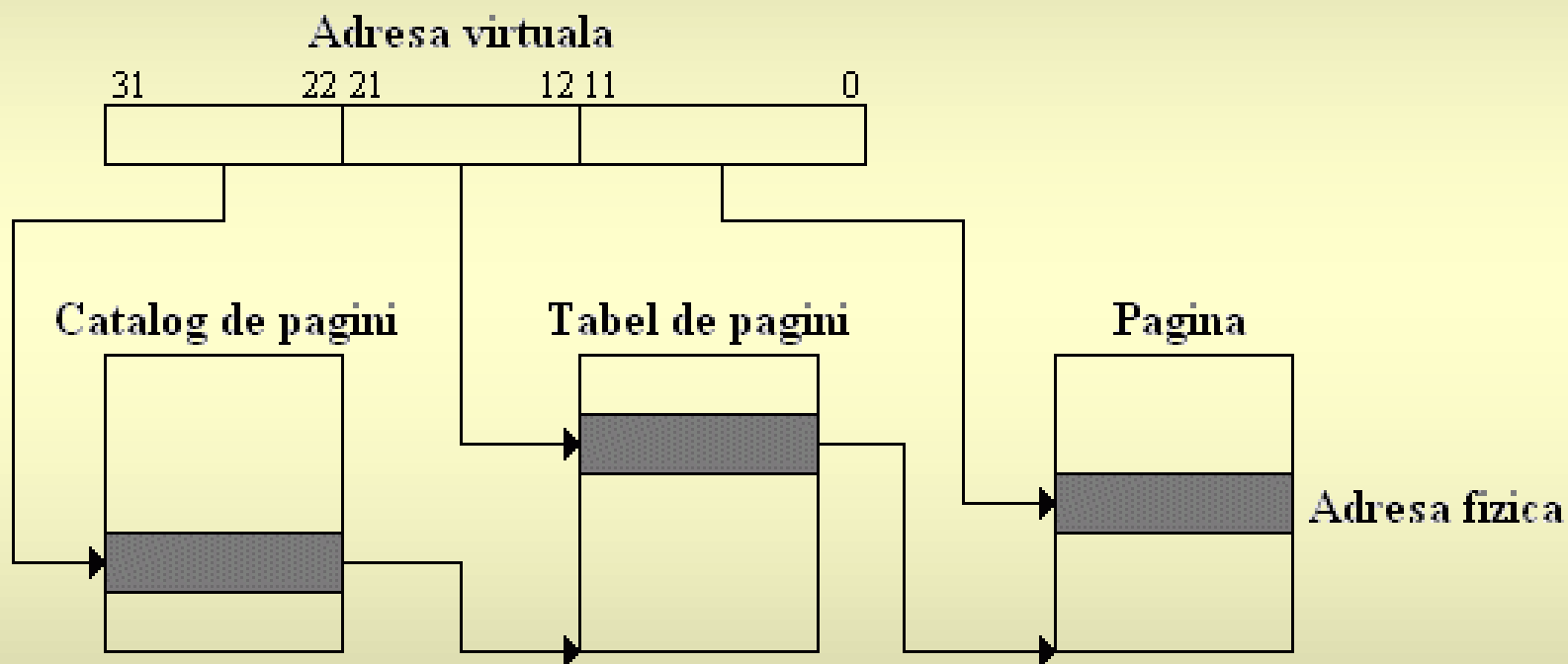
Soluția 1

- tabele de pagini inversate
- nu se rețin toate cele 2^{20} elemente
 - doar paginile folosite
- plasare/căutare în tabel - funcție *hash*
- greu de implementat în hardware
 - viteză
 - evitare coliziuni

Soluția 2

- tabele pe mai multe nivele
- cazul Intel - 2 nivele
 - catalog de pagini (*Page Directory*)
 - tabel de pagini (*Page Table*)
- elementele din catalog - adrese de tabele de pagini
- se alocă doar tabelele de pagini folosite

Structura Intel



Performanța (1)

- cataloagele și tabelele de pagini se află în memorie
 - prea mari pentru a fi reținute în procesor
 - prea multe - specifice fiecărui proces
- efect - performanță scăzută
 - pentru fiecare acces la memorie solicitat de proces - 2 accese suplimentare
- soluția - cache dedicat

Performanța (2)

- TLB (*Translation Lookaside Buffer*)
 - în interiorul procesorului
 - reține corespondențe între pagini virtuale și cadre de pagină fizice
 - ultimele accesate
 - trebuie invalidat atunci când se trece la execuția altui proces

V.5.3. Memoria virtuală

Ideea de pornire

Problema

- aplicațiile - consum mare de memorie
- memoria disponibilă - insuficientă

Cum se poate rezolva?

- capacitatea discului hard - foarte mare
- nu toate zonele de memorie ocupate sunt accesate la un moment dat

Memoria virtuală

Soluția - memoria virtuală (*swap*)

- unele zone de memorie - evacuate pe disc
- când este nevoie de ele, sunt aduse înapoi în memorie

Cine gestionează memoria virtuală?

- sunt necesare informații globale
- sistemul de operare

Fișierul de paginare

- conține zonele de memorie evacuate pe disc
- informații pentru regăsirea unei zone stocate
 - adresele din memorie
 - programul căruia îi aparține
 - dimensiunea
 - etc.

Politica de înlocuire (1)

- problema - aceeași ca la memoria cache
- aducerea unei zone de memorie din fișierul de paginare implică evacuarea alteia
 - care?
- scop - minimizarea acceselor la disc
- politică inefficientă → număr mare de accese la disc → scăderea vitezei

Politica de înlocuire (2)

- set de lucru (*working set*) - zonele de memorie necesare programului la un moment dat
- uzual mult mai mic decât totalitatea zonelor folosite de program
- dacă încapă în memorie - puține accese la disc

Politica de înlocuire (3)

- se va selecta pentru evacuare zona care nu va fi necesară în viitorul apropiat
- nu se poate ști cu certitudine - estimare
 - pe baza comportării în trecutul apropiat
- paginare la cerere (*demand paging*) - evacuare pe disc numai dacă este strict necesar

Implementare

- prin intermediul mecanismelor de gestiune a memoriei, deja discutate
 - dacă un program încearcă să acceseze o locație aflată temporar pe disc, este necesar același tip de detecție
 - memoria virtuală poate fi folosită împreună atât cu segmentarea, cât și cu paginarea
- rolul sistemului de întreruperi - sporit

Accesul la memorie (1)

Cazul paginării

1. programul precizează adresa virtuală
2. se determină pagina din care face parte
3. se caută pagina în tabelul de paginare
4. dacă pagina este găsită - salt la pasul 9
5. generare excepție
6. rutina de tratare caută pagina în fișierul de paginare

Accesul la memorie (2)

Cazul paginării (cont.)

7. dacă pagina nu este în fișierul de paginare - programul este terminat
8. se aduce pagina în memoria fizică
9. se determină cadrul de pagină corespunzător
10. calcul adresă fizică
11. acces la adresa calculată

Reducerea acceselor la disc (1)

- duce la creșterea performanței
- o pagină este salvată pe disc și readusă în memorie de mai multe ori
- readucerea în memorie - copia de pe disc nu este ștearsă
- pagina și copia sa de pe disc sunt identice până la modificarea paginii din memorie

Reducerea acceselor la disc (2)

- evacuarea unei pagini din memorie
 - dacă nu a fost modificată de când se află în memorie - nu mai trebuie salvată
 - util mai ales pentru paginile de cod
- este necesar sprijin hardware pentru detectarea acestei situații
 - este suficient să fie detectate operațiile de scriere

Reducerea acceselor la disc (3)

- tabelul de paginare - structură extinsă
 - fiecare pagină are un bit suplimentar (*dirty bit*)
 - indică dacă pagina a fost modificată de când a fost adusă în memorie
 - resetat la aducerea paginii în memorie
- instrucțiune de scriere în memorie
 - procesorul setează bitul paginii care conține locația modificată