Curs 9

Gestiunea memoriei

Controleaza partile din memoria RAM utilizate si neutilizate, alocand si dezalocand memorie pentru procese.

Gestioneaza spatiul de swap => schimb de segmente de memorie intre disc si memoria principala atunci cand aceasta nu poate tine toate procesele rezidente

Observatie: in absenta multiprogramarii si a swapping-ului un singur proces este rezident in memoria principala la un moment dat si aproape toata memoria principala e disponibila procesului

Programul trebuie adus de pe disc in memorie si plasat unui proces pentru a fi rulat, si de accea trebuie sa ne asiguram ca un proces poate accesa doar adrese permise.

OBSERVATIE: kernelul are accest nerestrictionat la intreaga memorie.

Legarea adreselor si a datelor din memorie are loc la compilare, la load time si la executie.
Adresa logica = generata de CPU = adresa virtuala
Adresa fizica = adresa pe care o vede memory unit
Adresa logica == adresa fizica la compilare si la load time, LA EXECUTIE DIFERA
Spatiu de adresa logice = multime ce contine toate adresele logice (la fel fizice)

Memory-Management Unit (MMU)

Hardware device care in timpul executiei mapeaza adresele virtuale de cele fizice.

Swapping

Un proces poate fi temporar scos din memorie si adaugat intr-o zona de depozitare oarecum, iar mai apoi adus inapoi in zona de memorie si executia sa fiind automati continuata. Acest procedeu poate creste gradul de multiprogramare.

Backing store = spatiul de depozitare cu un spatiu destul de mare care sa poata permita copii pentru toti userii

Roll out, roll in = o varianta de swapping folosita pentru algoritmii de programare a proceselor ce se bazeaza pe prioritati. Astfel, procesele cu prioritate mica vor fi date la o parte, iar cele cu prioritate mare vor fi incarcate in memorie si executate.

O parte importanta din procesul de swap ar fi timpul de transfer, fiind direct proportional cu cantitatea de memorie swapped.

O buna parte din sistemele de operare apeleaza la swapping clasic doar in cazuri exceptionale (utilizeaza prea multa memorie de exemplu**). Dar, unele versiuni de swapping modificate sunt apreciate deoarece sunt capabile sa faca swap doar cand memoria libera este foarte putina sau sa faca swap doar parti ale procesului in sisteme care implementeaza memorie virtuala.**

Pe langa timpul de swap, intervine si timpul alocat context-switchului, insa acest timp poate fi redus daca se cunoaste exact cata memorie este intr-adevar utilizata. **(request_memory() sau release_memory())**

Alocarea memoriei cu partitii fixe

Memoria principala e impartita intr-un numar fix de partitii, fiecare partitie continand un singur proces (deci gradul de multiprogramare e limitat de numarl partitiilor).

Procesele care sosesc in sistem sunt adaugate intr-o coada de asteptare pentru partitii libere, partitia trebuie sa fie suficient de mare pentru a contine programul. Odata disponibila, programul se incarca si se executa, iar la terminarea sa partitia se elibereaza si devine accesibila pt alt proces.

Problema: dimensiunea fixa a partitiilor implica existenta unui spatiu neutilizat de catre proces pe durata rularii:

1. Solutie 1: cozi separate pt fiecare partitie
2. Solutie 2: coada unica si se alege cel mai mic proces care incape

Dynamic Storage – Allocation Problem

1. First-fit -> pune primul element care incape
2. Best-fit -> pune cel mai mic element care incape
3. Worst-fit -> pune cel mai mare element
4. Next-fit -> ca first-fit dar continua din pct in care a ramas ultima data
5. Quick-fit -> mentine liste spearate pt cele mai frecvente dimensiuni

Fragmentarea

Fragmentarea externa = memoria totala care exista pentru a indeplini un request (nu e contigua)

Fragmentarea interna = memoria alocata putin mai mare decat memoria ceruta (diferenta de marime este memoria intarna a unei partitii, dar nu e folosita)

Analiza fragmentarii externe

- f = fractiunea de memorie aferenta gaurilor
- s = dimensiunea medie a *n* procese
- k * s = dimensiunea medie a gaurilor, k > 0
- m = dimensiunea memoriei in biti

⇒ *n/2* gauri de memorie ocupa *m – n*s* biti
⇒ (n/2)*k*s = m – n*s
⇒ m = n*s*(1 + k / 2)
⇒ f = ((n / 2) *k*s) / m = k / (k + 2)
⇒ pt k = ½ (dimensiunea gaurii e ½ din dimensiunea medie a procesului)
f = 20% pierdere de memorie
pt. k = ¼ => f = 11%

OBS: In general fragmentarea externa se rezolva prin alocarea necontigua a spatiului logic (VIRTUAL) de adrese. Ca solutii pot fi segmentarea/paginarea, insa paginarea este preferata deoarece rezolva si problema fragmentarii interne.

Paginarea

➔ Spatiul adreselor fizice unui proces pot fi non-contigue. Procesului ii este alocata memorie fizica ori de cate ori se poate.
➔ Se divide memoria fizica in blocuri de marime fixa numite FRAME-URI (marime fixa = puteri de 2 intre 512 bytes si 16 mbytes)
➔ N pages => N free frames
➔ Se seteaza un page table pentru a traduce adrese logice in adrese fizice
➔ La incarcarea paginilor in memorie nu e nevoie de relocare

➔ fiecare proces are tabela de pagini proprie spatiului sau de adrese virtuale
➔ spatiul de adrese virtual = contiguu, alocare memorie fizica = necontigua
➔ permite implementarea sistemelor de memorie virtuala (spatiul virtual de adresa e mai mare decat spatiul de adrese fizice)
➔ o parte din pagini sunt rezidente in memorie, restul se afla pe disc
➔ se potriveste multiprogramarii
➔ page-fault = acces la pagina nerezidenta in memorie = determina oprirea procesului pana la incarcarea paginii de pe disc in memorie (prilej pt a oferi procesorului alt proces)

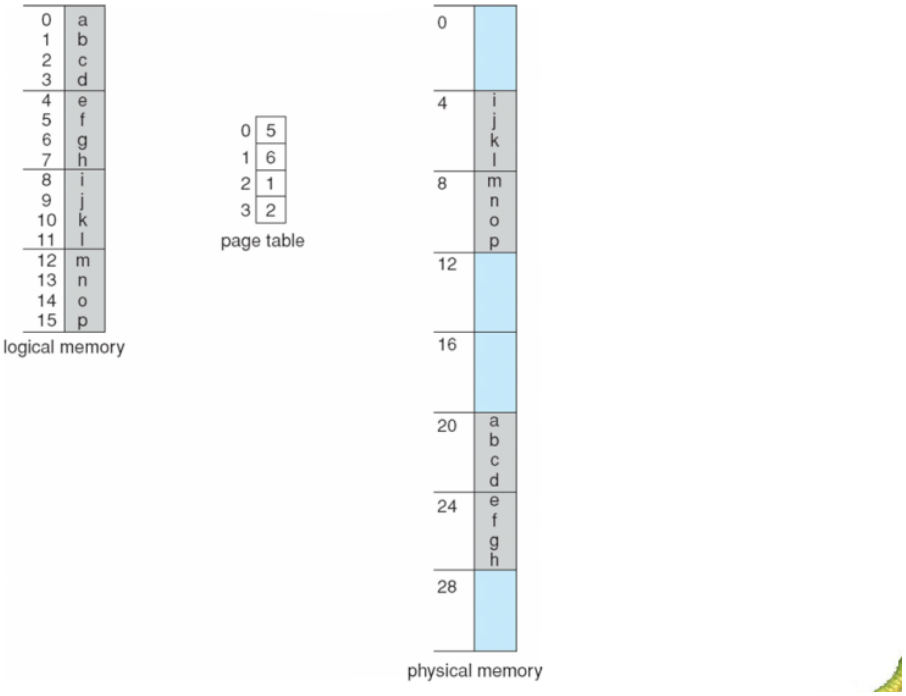
Page number (p) = se utilizeaza ca un index intr-o tabela de pagini care contine adresa de baza a unui frame f in memoria fizica

Page offset(d) = combinat cu adresa de baza pentru a defini adresa fizica ce este trimisa catre memory unit

Frame-ul f = base register, iar limita e data de dimensiunea paginii/frame-ului

Exemplu:

- Logical address: n = 2 and m = 4. Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages)



OBSERVATII:

Paginarea exclude fragmentarea externa (orice frame liber poate fi alocat unui proces care are nevoie de el)

Paginarea nu elimina complet fragmentarea interna. Exemplu:

- Page size = 2,048 bytes
- Process size = 72,766 bytes
- 35 pages + 1,086 bytes
- Internal fragmentation of 2,048 - 1,086 = 962 bytes

- Worst case fragmentation = 1 frame – 1 byte
- On average fragmentation = 1 / 2 frame size
- So small frame sizes desirable?
- But each page table entry takes memory to track
- Page sizes growing over time
  - Solaris supports two page sizes – 8 KB and 4 MB

➔ pagini micii => fragmentare interna mica, dar tabela de pagini mare
➔ pagini mari => tabela de pagini mai mica, dar si I/O mai eficient cu discul
➔ s = dimenisunea medie a procesului in bytes, p = dim paginii in bytes
➔ creste p => scade dimensiunea tabelei de pagini si creste fragm interna
➔ scade p => scade fragm. interna si creste dimensiunea tabelei de pagini

OBS:

pt a determina valoarea optima a lui *p*, derivam si rezolvam ecuatia

$$-\frac{s*e}{p^2}+\frac{1}{2}=0$$

$$\Rightarrow \quad p=\sqrt{2se}$$

ex:

s = 512 KB si *e* = 4 bytes => $p=\sqrt{2*2^2*2^{19}}=2^{11}=2KB$

in general, se iau in considerare si alti factori (eg, viteza discului) pt stabilirea dimensiunii optime a paginii

Idei principale ale paginarii

Separarea clara intre perspectiva programatorului asupra memoriei si memoria fizica. Programul nu este stocat intr-o zona de memorie contigua, ci este imprastiat in memoria fizica, paginile sale amestecandu-se cu paginile altor programe.

MMU => mapeaza adresele logice/virtuale in adrese fizice (pagini => frame-uri)

Tabela de pagini e per proces deci aceasta nu poate accesa alte pagini decat paginile sale, iar sistemul de operare trebuie sa mentina contabilitatea frame-urilor alocate si respectiv a celor libere, in general se foloseste o **tabela de frame-uri.**

!!! tabela de pagini apartine PCB – ului (deci paginarea afecteaza timpul de context switch) !!!

Page-table base register = PTBR = pointeaza la tabela de pagini

Page-table length register = PTLR = tine dimensiunea tabelei de pagini

Pentru eficienta se foloseste TLB (tranlsation look-aside buffer = memorie asociativa)

TLB

Cache de mare viteza din MMU care metine copiile celor mai des folosite intrari din tabela de pagini. O intrare in TLB contine:

1. Numarul de pagina si intrarea coresp din tabela de pagini
2. Un bit de validitate care specifica daca intrarea e in uz sau nu

Functionarea TLB:

- ➔ Intrarile TLB sunt referite nu prin adresa ci prin continut
- ➔ TLB compara in paralel numarul paginii asociate de intrarea in TLB cu toate numerele de pagina din toate intrarile sale
- ➔ Daca mai sus s-a gasit o potrivire atunci intrarea curenta din tabela de pagini se foloseste de catre MMU pentru mapare
- ➔ Daca nu se gaseste nicio potrivire, atunci MMu cauta in tabela de pagini din memoria principala o mapare valida si daca o gaseste, inlcouieste o intrare din TLB cu noua mapare
- ➔ Daca intrarea inlocuita = pagina modificata => MMU marcheaza intrarea din tabela de pagini

Consecintele multiprogramarii asupra TLB-ului:

- ➔ La context-switch uri se schimba tabelele de pagini (maparile din TLB nu mai sunt valide) (ca solutie ar fi invalidarea intregului TLB => performante ff slabe SAU folosirea PID-ului la indexarea in TLB)

Memory protection

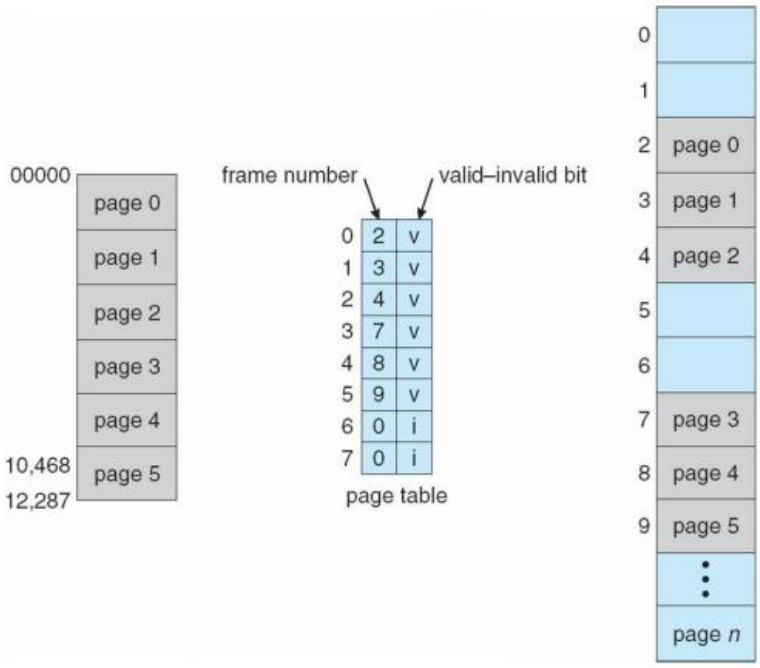
Se asociaza un bit de protectie pentru fiecare frame pentru a indica daca este read-only sau read-write.

Valid-invalid bit atasat de fiecare intrare in tabela de pagini:

VALID = pagina asociata e in spatiul de adrese logice a procesului = pag legala

INVALID = pagina ilegala = nu se regaseste in spatiul de adrese logice a proc

Exemplu:



Shared pages

Shared code = copie a codului impartit de catre procese
Private code and data = fiecare proces tine o copie pentru cod si date, iar paginile pentru acestea pot aparea oriune in spatiul adreselor logice

Structura tabelei de pagini

Daca se folosesc doar metode clasice => dimensiune enorma (solutie: paginare ierarhizata, hashed page tables, ineverted page tables)

Hierachical page tables

Imparte spatiul adreselor logice in mai multe tabele de pagini (precum 2-level, 3-level, 0-level page tables)

Zero-level paging

Este intalnita la unele procesoare RISC. (exemplu MIPS R3000 implementeaza doar TLB si nu are HW dedicat pentru cautarea in tabelele de pagini)

Hashed page tables

Numarul virtual al paginii este hashuit intr o tabela de pagini ce contine o serie de elemente care hashuiesc aceeasi locatie.
Fiecare element contine:

1. Virtual page number
2. Valoarea pentru fiecare page frame mapat
3. Un pointer catre urmatorul element

Se compara virtual page numbers, iar daca se gaseste un match, atunci este extras frame-ul fizic respectiv.

Inverted page tables

Se vor gestiona toate paginile fizice si va exista o intrare pentru fiecare pagina reala de memorie. O intrare consta in adresa virtuala a paginii stocata in acea locatie din memorie reala, alaturi de informatia despre procesul care are acea pagina.

OBSERVATII

Scade memoria necesara pentru a stoca fiecare tabela de pagina, dar creste timpul necesar pentru a verifica daca un „apel” catre o pagina are loc.
Se foloseste hashed table pentru a se limita intrarile in page-tables (TLB poate accelera accesul)

Pentru a se implementa memoria partajata, se implementeaza o mapare a adresei virtuale catre adresa fizica partajata. Referintele la adresele virtuale nemapate in tabela inversata vor rezulta in page-faults!

Segmentarea memoriei

Paginarea ofera un tip de memorie virtuala unidimensionala.
Spatiul logic/virtual de adrese devine o colectie de segmente:

1. Fiecare segment are un nume si o lungime
2. Adresare: nume segment + offset
3. Simplificare: adresa logica/virtuala devine <nr segm, offset>

Suport HW pentru segmentare:

Adresa logica/virtuala = bidimensionala, adresa fizica = liniara
HW mapeaza adresa logica – fizica folosind o tabela de segmentare.
Fiecare intrare are segment base si limit
Numarul de segment indexeaza in tabela, iar offsetul din adresa logica trebuie sa fie intre 0 si limita.

Avantajele segmantarii

Componentele logice pot avea dimensiuni arbitrare si chiar dinamice.
Spatiile acesto componente logice (segmentele) sunt independente unele de altele (nu exista interferente).
Se faciliteaza partajearea de componente intre programe.

Caracteristici

Segment = secventa liniara de adrese de la 0 la o val maxima
= fiecare spatiu de adrese virtuale independent
= au dimensiuni diferite

Paginare VS segmentare

Programatorul este constient de segmentare, dar nu si de paginare
Paginarea are loc intr-un spatiu de adrese virtuale
Spatiul de adrese virtual poate depasi dim memoriei fizice (ambele cazuri)
Paginarea nu distinge intre continutul paginilor (nu asigura protectie)
Paginarea nu permite modificarea dinamica a spatiului de adrese
Paginarea nu faciliteaza partajarea componentelor logice ale programului in niciun fel.

Paginare = se foloseste in principal pt a incarca si rula programe mai mari decat memoria fizica
Segmentare = se poate imparti programul in componente logice distincte pe care le aloca in spatii virtuale independente, pe care le poate proteja si partaja