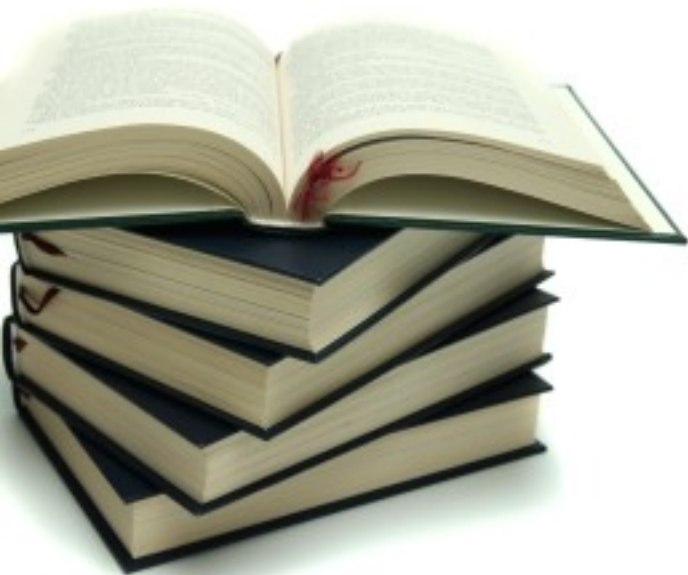


- Creați tabela de pagini (Intel) care să mapeze următorul spațiu de adresă virtual: [0x10000000 – 0x10000400], [0x40000000 – 0x40000400], [0xc0000000 – 0xc0000400]
- Care sunt avantajele folosirii paginilor de 4M?
- De ce este o schimbare de context între două thread-uri mai puțin costisitoare decât o schimbare de context între două procese



7

Gestiunea memoriei (II)

15 aprilie 2010

- Gestiunea memorie fizice în Linux
- Gestiunea memoriei virtuale în Linux
- Bibliografie
 - UTLK: capitolele 8, 9

- Algoritmi și structuri de date ce mențin starea memoriei fizice
- Se face la nivel de pagină
- (Relativ) independentă de gestiunea memoriei virtuale
- Fiecare pagina fizică are asociată un descriptor: **struct page**
- În zona **lowmem** se ține un vector de astfel de descriptori (**mem_map**)
- (descriptorul de pagină conține: un contor de utilizare al paginii, flag-uri, poziția în swap sau în fișier, buferele conținute de pagină, poziția în „page cache”, etc.)

- Zona DMA: 0 - 16Mb
- Zona Normal (LowMem): 16Mb - 896Mb
- Zona HighMem: 896Mb – 4Gb/64Gb
- Non-Uniform Memory Access
 - Memoria fizică este împartită între mai multe noduri
 - există un spațiu comun de adrese fizice
 - accesul la memoria locală este mai rapidă
 - Fiecare nod
 - are procesorul propriu
 - are zone de memorie proprii: DMA, NORMAL, HIGHMEM

- `alloc_pages(gfp_mask, order)`
 - Alocă 2^{order} PAGINI contigue și întoarce un descriptor de pagină pentru prima pagină alocată
 - `alloc_page(gfp_mask)`
- Functii mai folositoare (întorc adresa lineară a paginii/primei pagini)
 - `__get_free_pages(gfp_mask, order)`
 - `__get_free_page(gfp_mask)`
 - `__get_zero_page(gfp_mask)`
 - `__get_dma_pages(gfp_mask, order)`

- `__GFP_WAIT`
 - Kernelul poate bloca procesul curent pentru a aștepta pagini libere
- `GFP_HIGH`
 - Prioritate mare; kernelul are voie să folosească pagini libere rezervate pentru ieșirea din situații critice (atunci când există foarte puțină memorie)
- `__GFP_IO`
 - Kernelul are voie să facă transferuri I/O pe pagini normale pentru a elibera memorie (necesar pentru evitarea deadlock-urilor)
- `__GFP_HIGHIO`
 - Kernelul are voie să facă transferuri I/O pe pagini din high memory pentru a elibera memorie (necesar pentru evitarea deadlock-urilor)
- `__GFP_FS`
 - Kernelul are voie să execute operații VFS low level (necesar pentru evitarea deadlock-urilor)
- `__GFP_DMA`
 - Paginile trebuie alocate din zona DMA
- `__GFP_HIGHMEM`
 - Paginile trebuie alocate din zona HIGHMEM

- GFP_ATOMIC
 - __GFP_HIGH
- GFP_KERNEL
 - __GFP_HIGH | __GFP_WAIT | __GFP_IO | __GFP_HIGHIO | __GFP_FS
- GFP_USER
 - __GFP_HIGH | __GFP_WAIT | __GFP_IO | __GFP_HIGHIO | __GFP_FS

- Algoritmii de alocare sunt lineari și fragmentează memoria
- O posibilă soluție: paging
 - Uneori kernelul chiar are nevoie de memorie fizică contignua (pentru DMA)
 - Dacă s-ar folosi paging, tabela de pagini s-ar modifica => penalizări la accesul la memorie
 - Dacă s-ar folosi paging nu s-ar mai putea folosi paginare extinsă (pagini de 4Mb/2Mb)
 - Anumite arhitecturi (MIPS) mapează o parte din spațiul de adresă kernel linear în memoria fizică (i.e. pe acea zonă nu se folosește paginare)

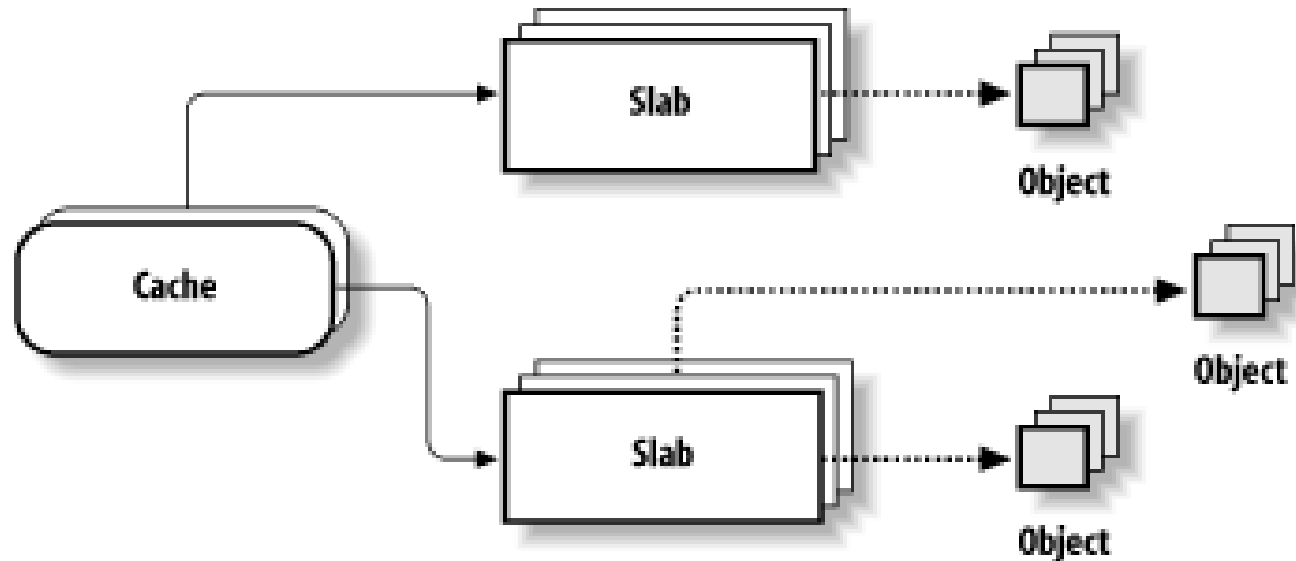
- Blocurile sunt grupate în liste de dimensiune fixă
 - Blocurile au ca dimensiune puterile lui 2, aliniate corespunzător cu dimensiunea
- Alocarea se face numai în blocuri de puteri ale lui 2
- La alocarea unui bloc (dimensiune N)
 - Dacă există un bloc de dimensiune N se alocă
 - Dacă nu, se sparge un bloc de dimensiune $2N$ în două blocuri de dimensiune N , unul se alocă, altul se pune în lista cu blocuri de dimensiune N
- La dealocarea unui bloc (dimensiune N)
 - Dacă există blocuri adiacente în memoria fizică de aceeași dimensiune N , iar primul este aliniat la $2N$, cele două blocuri se “coaguleaza” într-un bloc de dimensiune $2N$
 - Se încearcă iterativ să se coaguleze cât mai multe blocuri

- Sunt folosite 10 liste pentru blocuri de 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 PAGINI
- Fiecare zonă de memorie are alocatorul buddy propriu
- Fiecare zona are asociată un vector de descriptori de blocuri libere, câte o intrare pentru fiecare dimensiune de bloc
- Descriptorul de blocuri libere conține numărul de pagini libere și capul listei de blocuri libere
- Blocurile sunt înlănțuite cu ajutorul câmpului **lru** din descriptorul de pagini
- Paginile libere folosite de buddy au flagul **PG_buddy** setat
- Descriptorul de pagina menține dimensiunea blocului în câmpul **private** pentru a putea face ușor verificările necesare pentru coagularea a două blocuri libere ce sunt adiacente

- Sistemul buddy este folosit în kernel pentru a aloca pagini
- Multe componente din kernel au nevoie de blocuri de dimensiune mult mai mică decât dimensiunea unei pagini
- Soluție: folosirea de blocuri mai mici cu dimensiune variabile
 - Probleme: fragmentare externă
- Soluție: folosirea unor blocuri de dimensiune fixă, dar mai mici
 - Probleme: cât de mica/mare sa fie dimensiunea ?
 - Soluție: crearea mai multor zone cu blocuri de mai multe dimensiuni, distribuite geometric: 32, 64, ..., 131 056

- Zonele de memorie alocate/dealocate sunt văzute ca “obiecte”:
- Fiecare tip de “obiect” are un constructor și un destructor
- Obiectele dealocate sunt păstrate într-un cache, astfel că la următoarea folosire a lor nu mai trebuie reinițializate și nu mai trebuie apelat algoritmul buddy
- În Linux nu (prea) se folosesc constructori/destructori din motive de eficiență

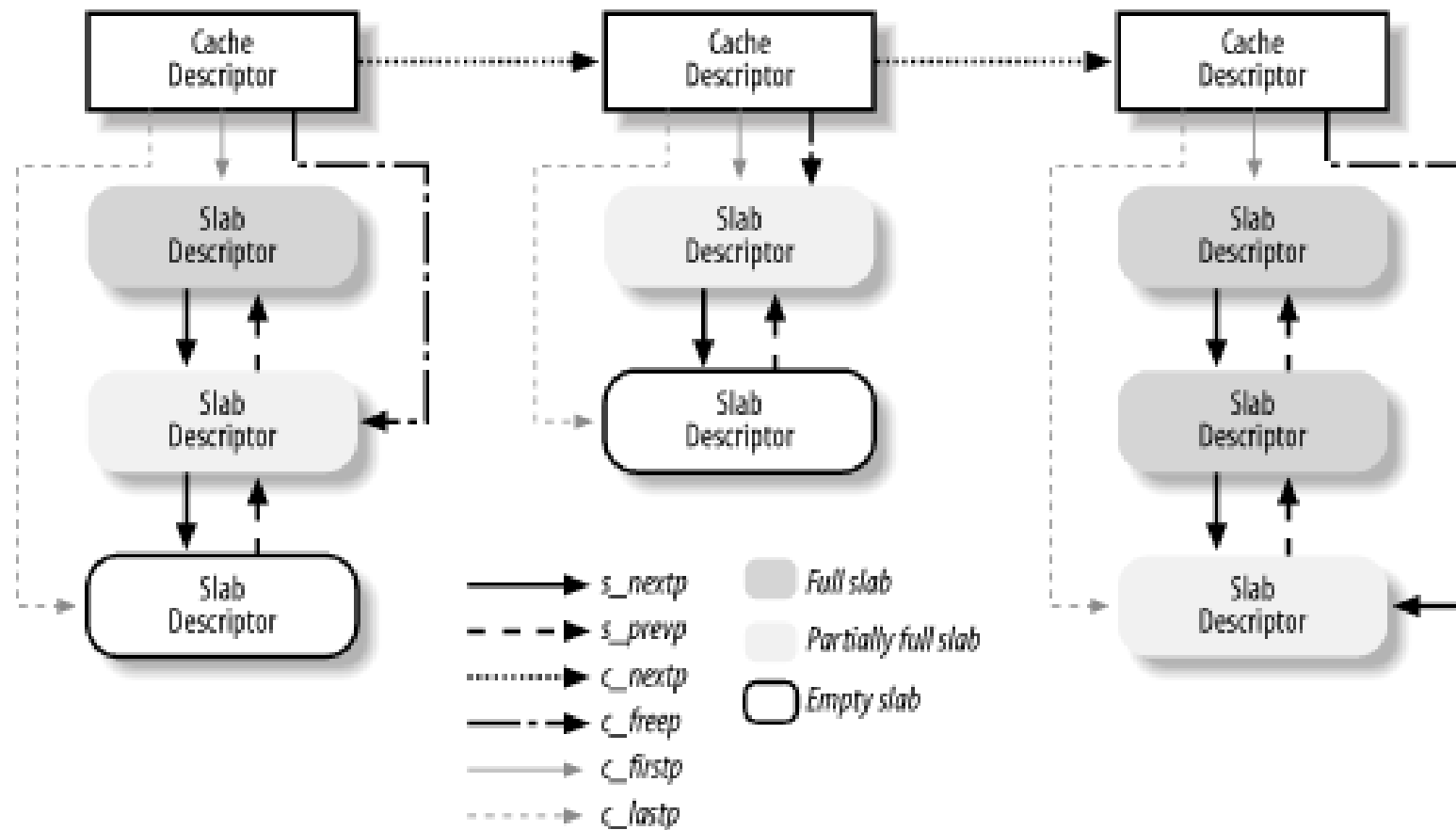
- Kernelul tinde să aloce/dealocare succesiv același tip de structuri de date (de exemplu PCB-uri); folosirea cache-ului din slab reduce frecvența operațiilor (mai costisitoare) de alocare/dealocare
- Multe cereri de alocare frecvente folosesc aceeași dimensiune pentru fiecare alocare: pentru aceste tipuri se pot crea zone speciale, care să conțină blocuri de dimensiunea dorită => se reduce astfel fragmentarea internă
- Pentru cereri ce nu folosesc aceeași dimensiune la alocare (mult mai puțin frecvente) se poate folosi în continuare abordarea geometrică (și cache)
- Reduce foot-print-ul pentru alocare/dealocare pentru că nu se mai caută pagini libere ci sunt luate direct din cache-uri (cache-urile de obiecte sunt concentrate într-o zonă de memorie și sunt mult mai mici decât structurile folosite de buddy)
- Distribuie obiectele în memorie astfel încât să acopere uniform liniile de cache



- Alocatorul slab este format din
 - Mai multe cache-uri
 - Fiecare cache are mai multe slab-uri
 - Fiecare slab menține mai multe obiecte alocate/libere

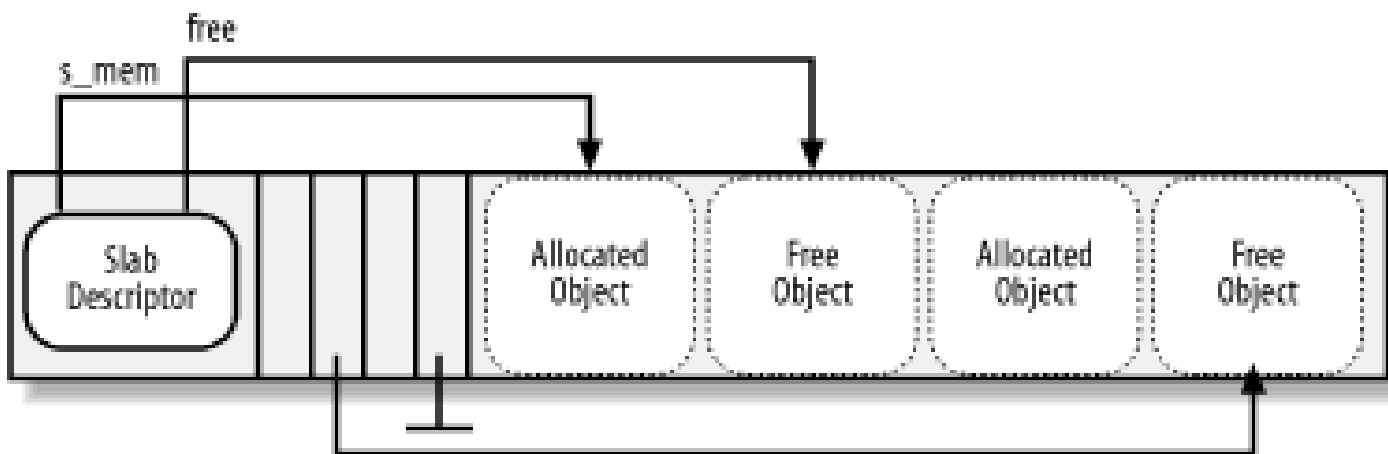
- Un nume folosit pentru informații către user-space
- Funcții pentru inițializarea / deinițializarea obiectelor cache-ului
- Dimensiunea obiectelor
- Diverse flaguri statice / dinamice
- Dimensiunea slab-urilor (structurile care conțin efectiv obiectele) în pagini (puteri ale lui 2)
- Măști GFP pentru alocarea dintr-o zonă specifică
- Mai multe slab-uri: pline, libere sau parțial pline

- Numărul de obiecte alocate
- Zona de memorie în care se țin obiectele
- Zona de memorie către primul obiect liber
- Descriptorii de slab sunt ținuti fie
 - În cadrul slab-ului ce îl descriu (dacă dimensiunea obiectelor e mai mică de 512, sau dacă fragmentarea internă lasa loc pentru descriptorul de slab)
 - În cadrul unor cache-uri generale folosite de către alocatorul slab

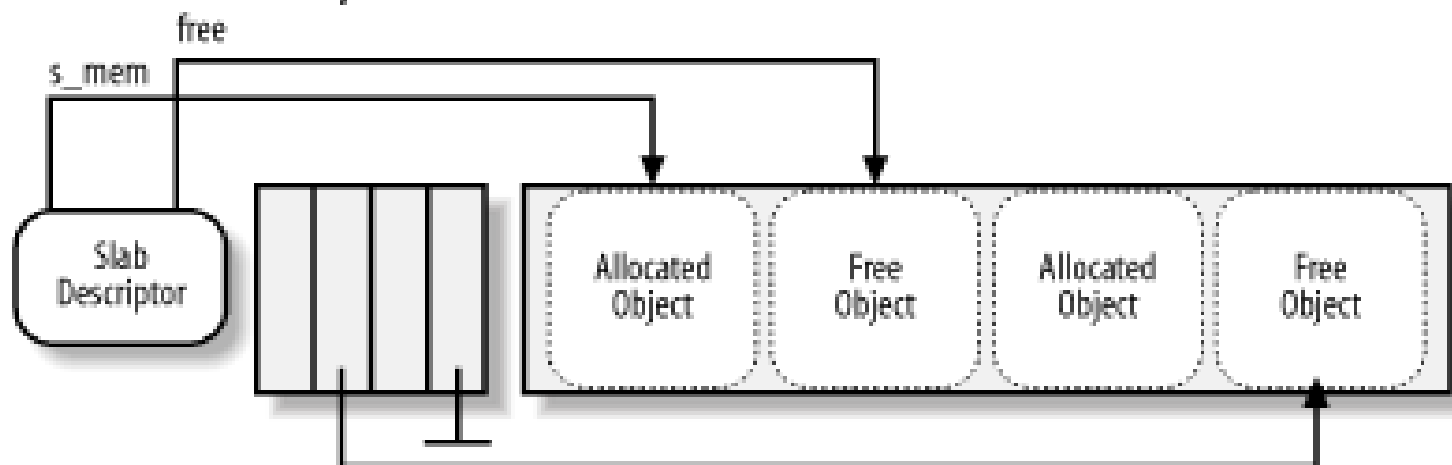


- Cache-urile generale sunt folosite de către alocatorul slab pentru
 - A ține descriptorii de cache pentru celelalte cache-uri
 - A ține 26 de cache-uri generale cu dimensiunea obiectelor de 32, 64, 128, ..., 131 072 (unul pentru zona normală și unul pentru zona DMA); `kmalloc()` alocă memorie din aceste cache-uri
- Cache-urile specifice sunt create de către restul kernelului la cerere

Slab with Internal Descriptors



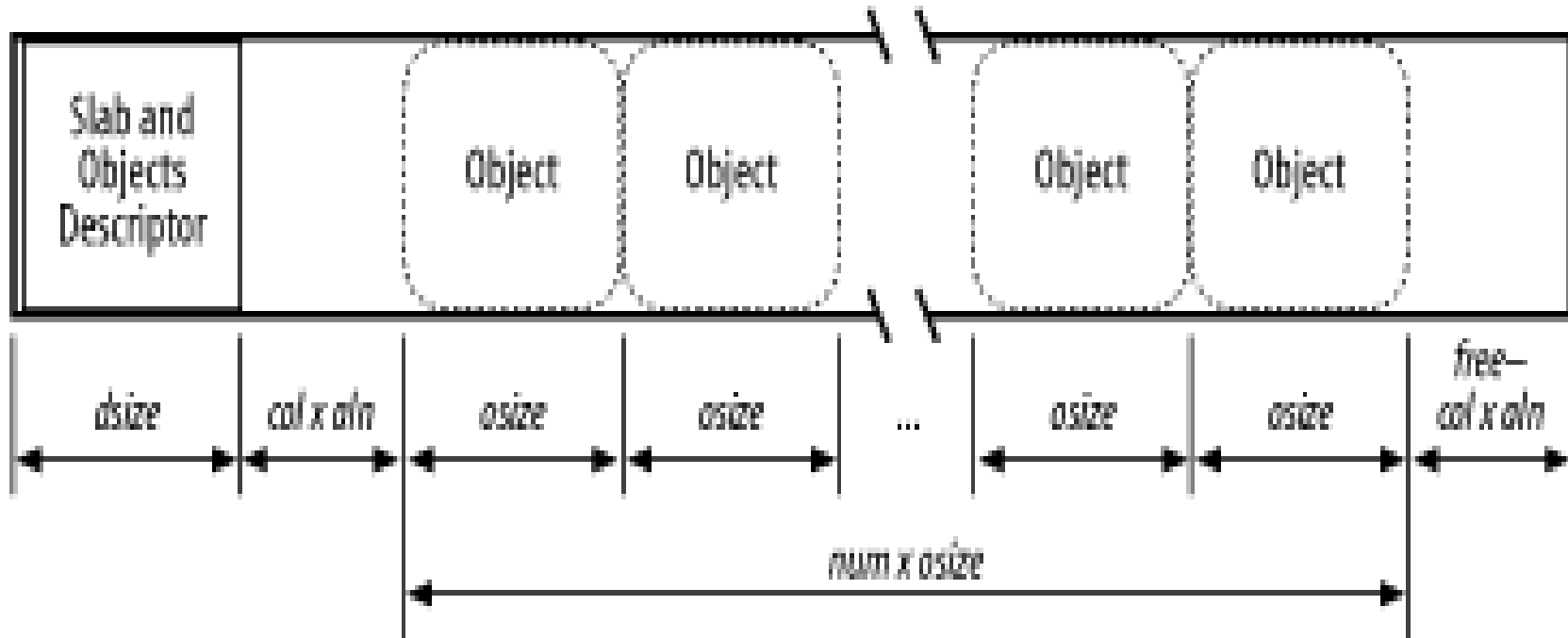
Slab with External Descriptors



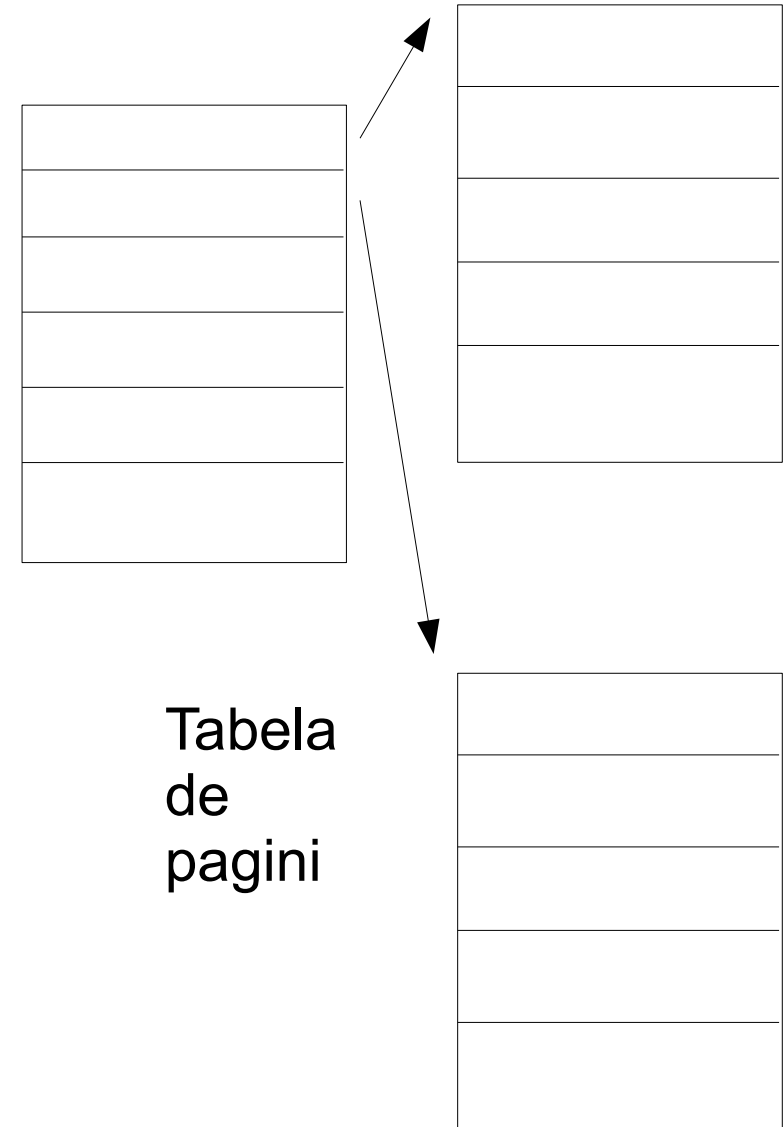
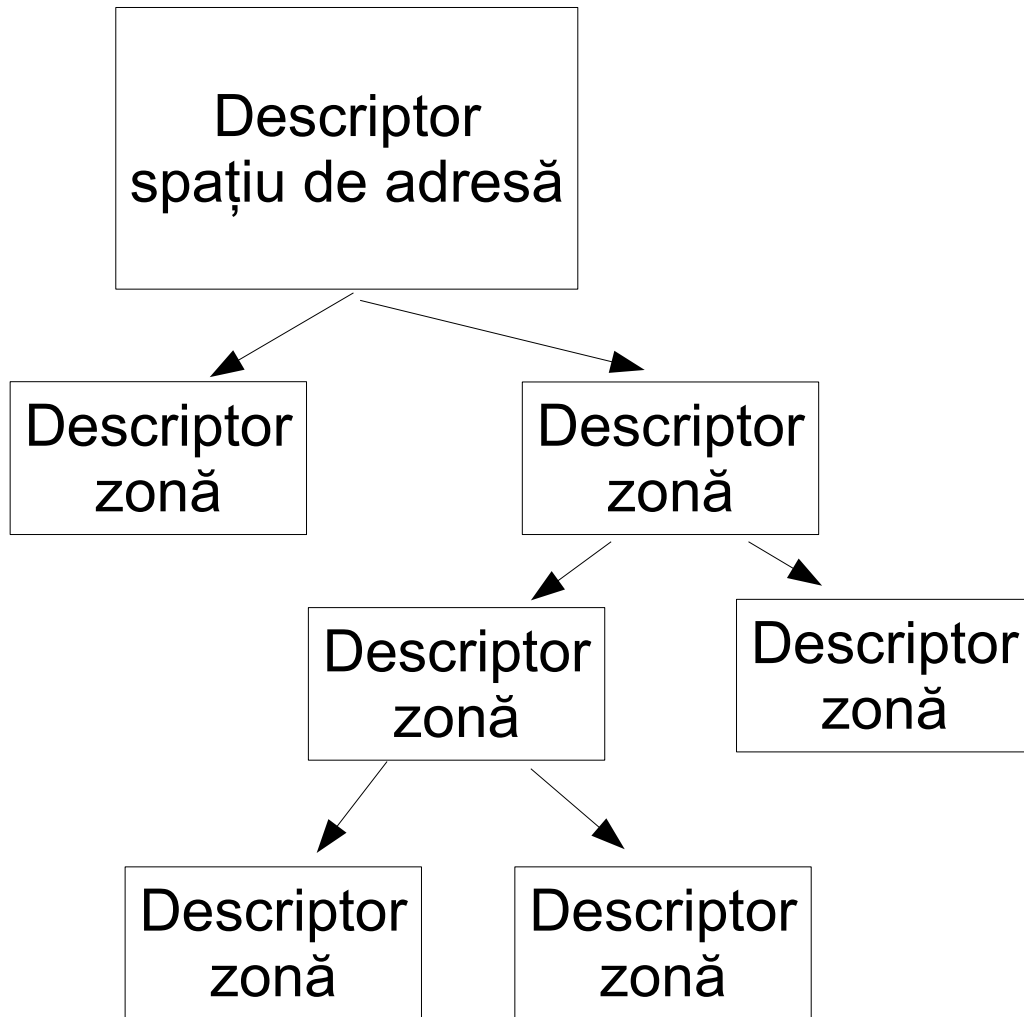
- Descriptorul de obiect este folosit doar atunci când obiectul este liber
- Este de fapt un întreg care indică următorul obiect liber
- Ultimul obiect liber are valoarea `BUFCTL_END`
- Descriptori de obiecte:
 - Interni - ținuți în slab
 - Externi - ținuți în cache-urile generale

- Colorarea este folosită pentru a alinia obiectele din diversele slab-uri din cache la offset-uri diferite (dar constrângerile de aliniere ale obiectelor din cache sunt păstrate)
- Fie
 - Aln - constrângerea de aliniere; adresa obiectului trebuie să fie un multiplu de acest număr
 - Num - numărul de obiecte care pot fi stocate în slab
 - Objsize - dimensiunea obiectelor, inclusiv octeții nefolosiți dar necesari pentru aliniere
 - Dsize - dimensiunea descriptorului slab-ului + dimensiunea descriptorilor de obiecte
 - Free - numărul de octeți neutilizați în slab

- Atunci
 - $\text{slab_length} = (\text{num} \times \text{objsize}) + \text{dsize} + \text{free}$
 - Numărul de culori = $(\text{free}/\text{aln}) + 1$
 - Numărul de culori reprezintă de fapt numărul maxim de posibilități de plasa primul obiect în slab prin mutarea unei bucăți din zona free de la sfârșitul slab-ului la începutul acestuia
 - Cum toate definițiile date sunt comune unui cache, înseamnă că alocatorul slab poate alocă “culori” diferite fiecărui slab



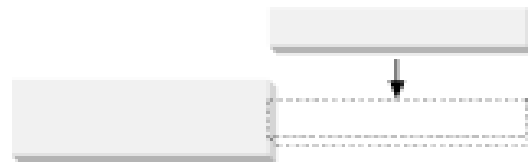
- Memoria virtuală este folosită
 - în user-space
 - în kernel-space
- Alocarea unei zone de memorie în spațiul de adresă (kernel sau user) implică
 - alocarea unei pagini fizice
 - alocarea unei zone din spațiul de adresă
 - în tabelele de pagini
 - în structuri interne, menținute de sistemul de operare



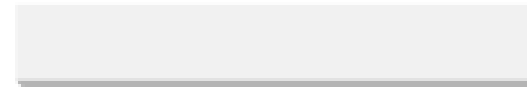
- Tabela de pagini este folosită de procesor sau la procesoarele RISC de către un layer low level (de obicei scris în assembler) care caută o pagină și o adaugă în TLB
- Descriptorul spațiului de adresă este folosit de sistemul de operare pentru a menține informații mai high level
- Fiecare descriptor de zonă specifică dacă zona este mapată peste un fișier, este mapată read-only, este mapată copy-on-write, etc.

1. Căutarea unei zone libere în descriptorul spațiului de adresă
2. Alocarea de pagini fizice pentru descriptorul zonei
3. Inserarea în descriptorul spațiului de adresă a descriptorului zonei
4. Alocarea de pagini fizice pentru tabelele de pagini
5. Inițializarea tabelului de pagini astfel încât să indice către zona alocată
6. Alocarea la cerere de pagini fizice pentru zona alocată (și eventual pentru noi tabele de pagină)

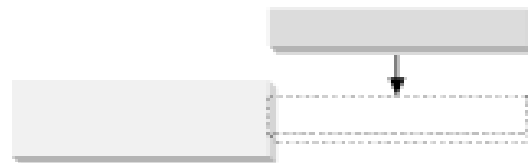
1. Ștergerea descriptorului zonei din descriptorul spațiului de adresă
2. Eliberarea paginii fizice asociată cu descriptorul zonei
3. Invalidarea tuturor intrărilor din tabelele de pagini asociate cu zona de dealocat
4. Invalidarea TLB-ului pentru zona delocată
5. Eliberarea paginilor asociate cu tabelele de pagini construite pentru zona dealocată



(a) Access rights of interval to be added are equal to those of contiguous region



(a') The existing region is enlarged



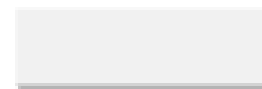
(b) Access rights of interval to be added are different from those of contiguous region



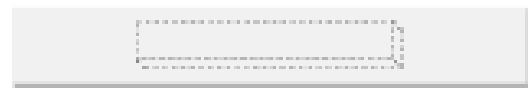
(b') A new memory region is created



(c) Interval to be removed is at the end of existing region



(c') The existing region is shortened



(d) Interval to be removed is inside existing region



(d') Two smaller regions are created

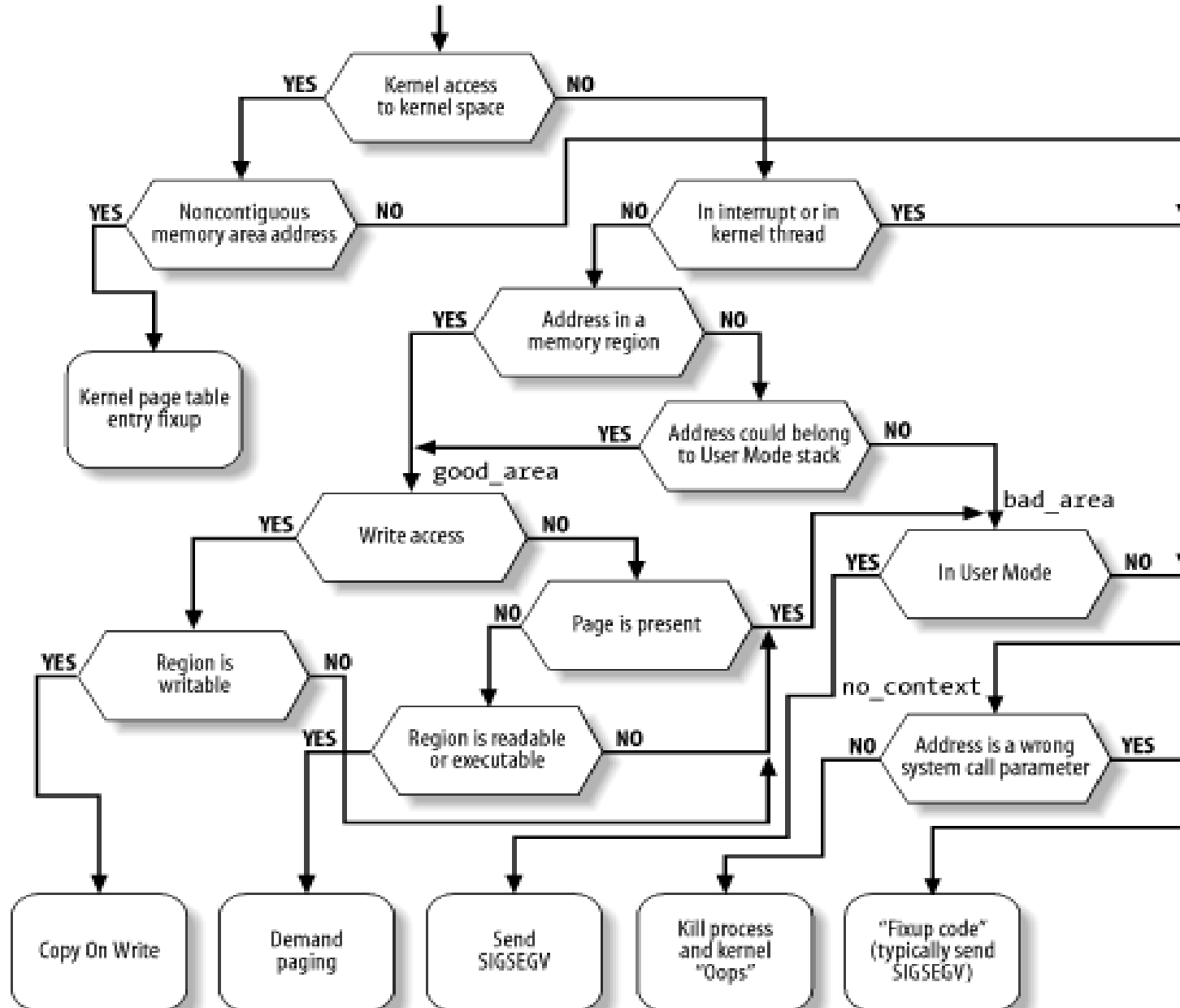


Address space before operation



Address space after operation

- În kernel-space
 - pentru apeluri vmalloc
 - descriptorul de zonă: struct vm_struct
 - descriptorul spațiului de adresă: listă simplu înlănțuită de descriptori de zonă
- În user-space
 - descriptorul de zonă: struct vm_area_struct
 - descriptorul spațiului de adresă: arbore red-black



- Algoritm de tip LRU
 - se mențin două liste: cu paginile active și cu paginile inactive
 - atunci când sistemul are nevoie de pagini, se eliberează pagini, în ordine, din:
 - diversele cache-uri: buffer cache, dcache, icache
 - lista de pagini inactive
 - lista de pagini active
- kswapd
 - kernel thread care evacuează paginile
 - este activat atunci când numărul de pagini libere scade sub o limită

?

