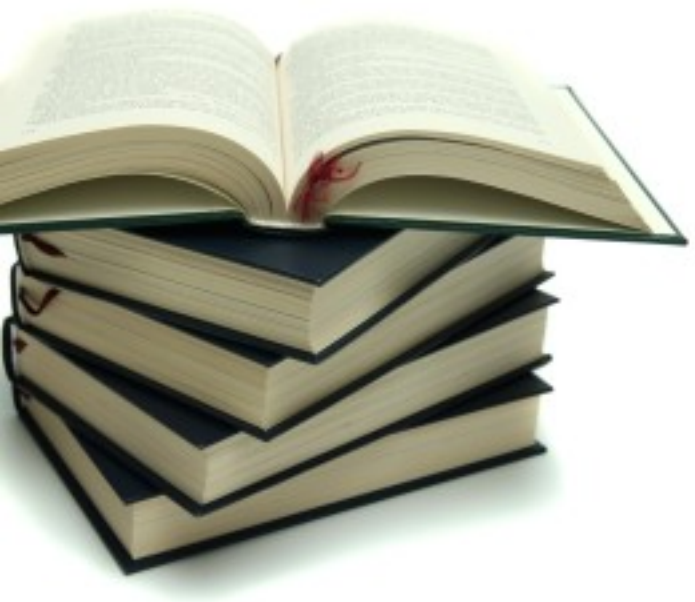
- Presupunând ca mai există doar o singură zona de memorie disponibilă, de 4MB, de câți paşi sunt necesari pentru a efectua apelul **__get_free_pages(GFP_ATOMIC, 2)**?
- Care este complexitatea medie de alocare pentru structura ce descrie un proces (**struct task_struct**)**?**
- Care este numărul minim de pagini ce trebuie alocate în urma apelului

mmap(NULL, 1024*4096, PROT_READ|PROT_WRITE,

MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)

SO2
kernel crunch

Cursul 8



8

Gestiunea fisierelor

22 aprilie 2010

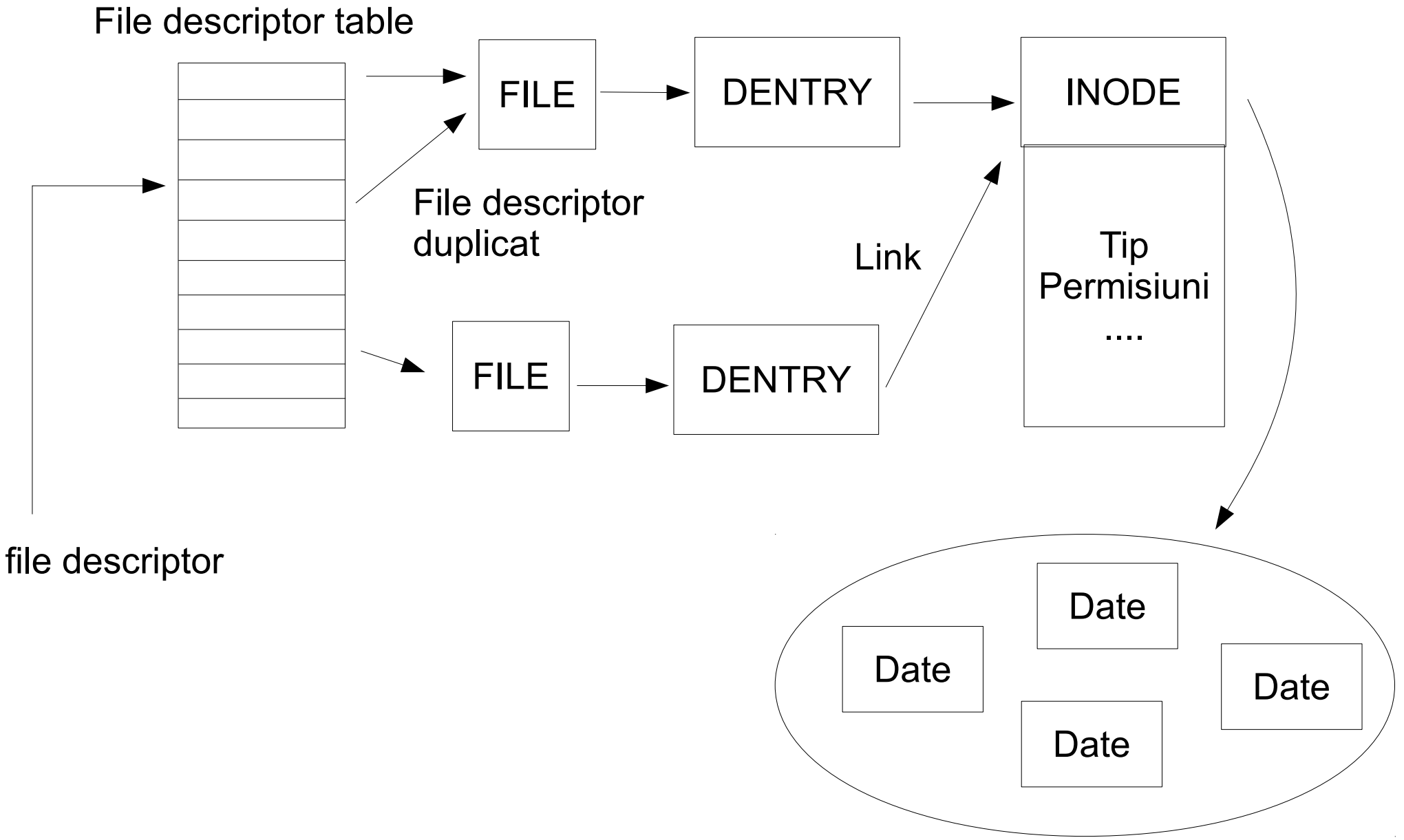
Gestiunea fişierelor

- Gestiunea fişierelor – noțiuni generale
- Gestiunea fişierelor în Linux
  - VFS
  - Inode cache, dcache, page cache
  - Buffer cache
- Bibliografie
  - LKD: capitolele 12, 15

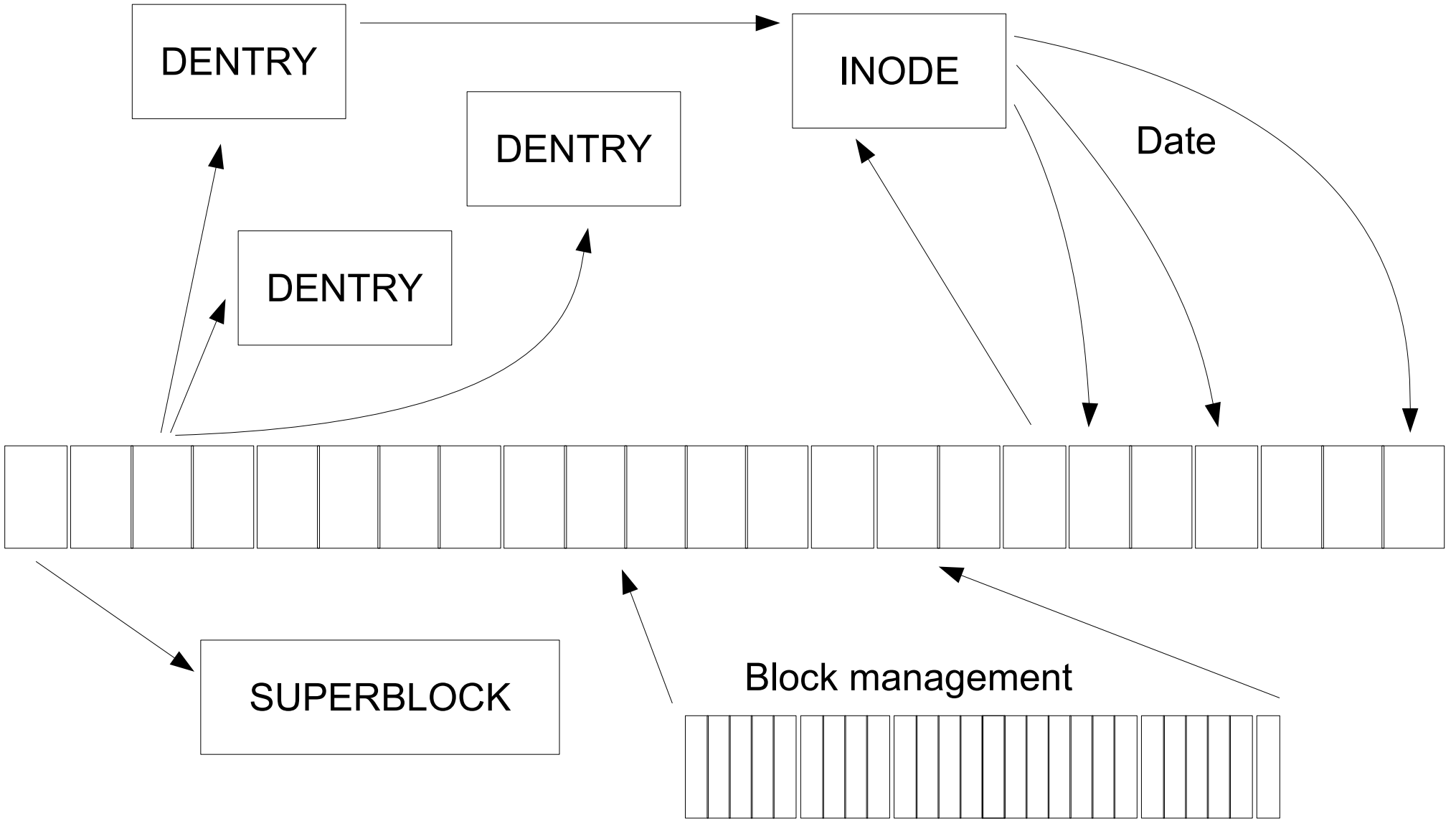
Entități fundamentale

- Superblock – conține informații despre sistemul de fişiere (dimensiunea blocului, inode-ul rădăcină); există atât pe disc cât şi în memorie
- File – structura internă SO ce descrie un fişier deschis; există doar în memorie
- Inode (FCB - file control block) – entitatea ce identifică un fişier în mod unic; există atât pe disc cât şi în memorie
- Dentry- asociază un nume cu un fişier; există atât pe disc cât şi în memorie

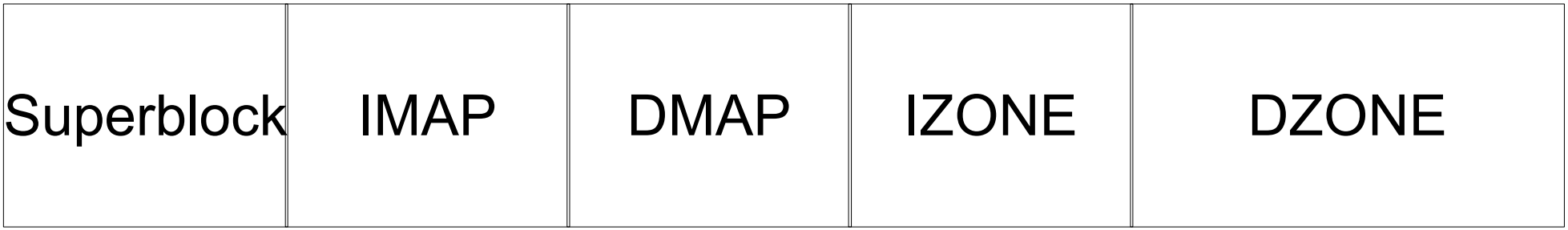
Entități fundamentale – în memorie



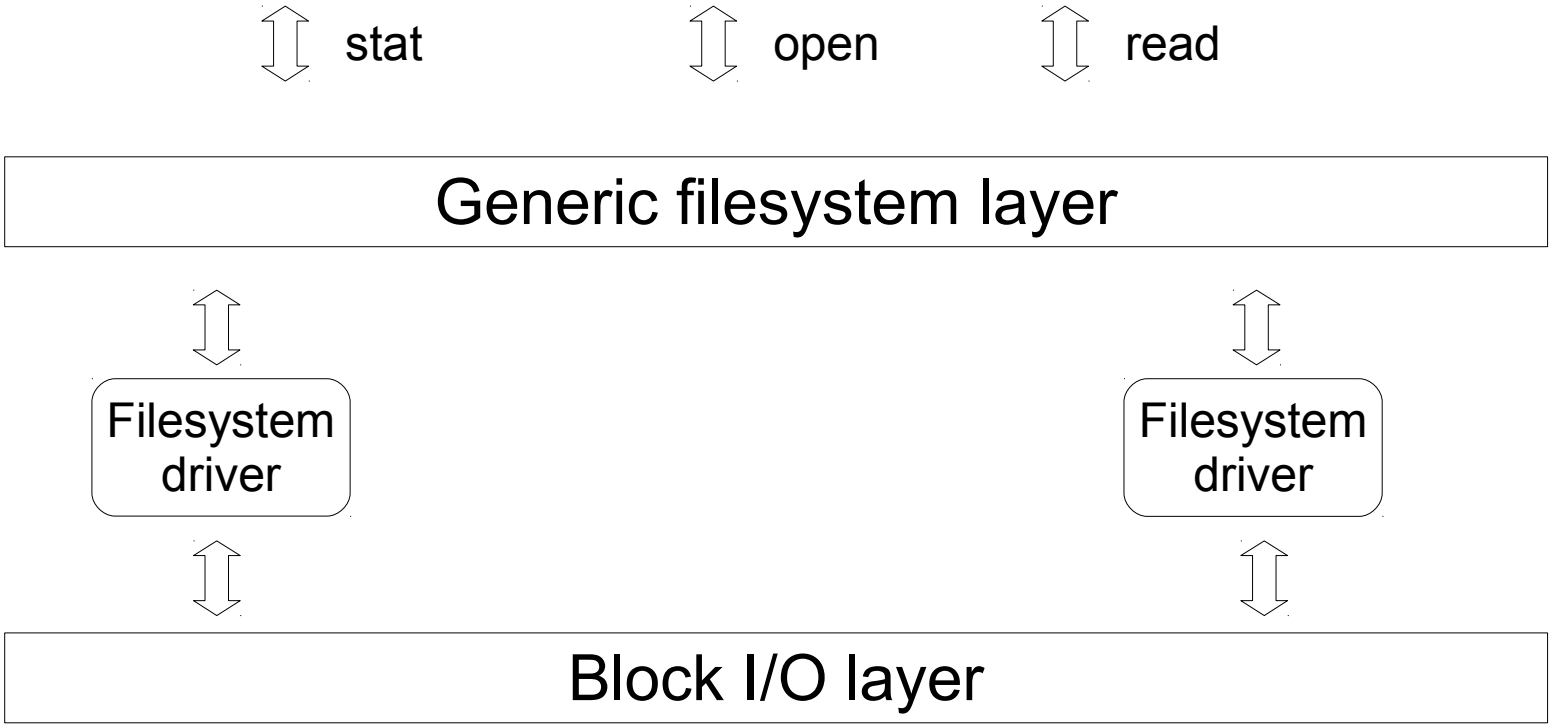
Entități fundamentale – pe disk



Exemplu: PITIX



Gestiunea sistemelor de fişiere în sistemul de operare



Operații fundamentale

- Montarea
- Deschiderea unui fişier
- Determinarea atributelor fişierului
- Citirea de date din fişier
- Scrierea de date în fişier
- Închiderea unui fişier
- Crearea unui fişier
- Ştergerea unui fişier

Montarea sistemului de fişiere

- (sau înregistrarea lui în cadrul sistemului)
- Intrare: un disk (partiție)
- Ieşire: un DENTRY către directorul rădăcină
- Operații: verificare partiție, determinare diverşi parametri, determinare inode-ului rădăcină
- Exemplu PITIX:
  - Se verifică MAGIC-ul
  - Se determină dimensiunea blocului
  - Se citeşte inode-ul rădăcină şi se crează dentry-ul asociat

Deschiderea unui fişier

- Intrare: o cale
- Ieşiere: un file descriptor
- Operații:
  - Determinarea sistemului de fişiere
  - Pentru fiecare nume din cale, determinarea inode-ului asociat cu numele
  - Odată găsit inode-ul ultimului nume din cale se crează structura FILE şi se alocă o intrare în tabela de descriptori de fişier

Determinarea atributelor fişierului

- Intrare: file descriptor
- Ieşiere: atributele fişierului
- Operații:
  - Se accesează inode-ul (file->dentry->inode)
  - Se citesc atributele din inode

Citirea de date din fişier

- Intrare: file descriptor, offset, length
- Ieşire: date
- Operații:
  - Se accesează inode-ul (file->dentry->inode)
  - Se determină blocurile de date
  - Se copiează datele de pe disk în memorie şi sunt apoi trimise utilizatorului

Scrierea de date în fişier

- Intrare: file descriptor, offset, length, date
- Ieşire:
- Operații:
  - Se accează inode-ul
  - Se aloca unul sau mai multe noi blocuri pe disc – se caută blocuri libere, se marchează ca fiind ocupate
  - Se adaugă blocurile alocate la inode
  - Se copieaza datele de la user in bufere interne şi apoi se scriu pe disk

Închiderea unui fişier

- Intrare: file descriptor
- Ieşiere:
- Operații:
  - Se decrementează reference counter-ul structurii FILE
  - Dacă referecen counter-ului structurii FILE ajunge la 0, se şterge structura
  - Se setează pe NULL intrarea din tabela de descriptori de fişier

Directoare

Directoarele sun fişiere ce conțin o înşiruire (vector, câteodată arbore) de DENTRY-uri.

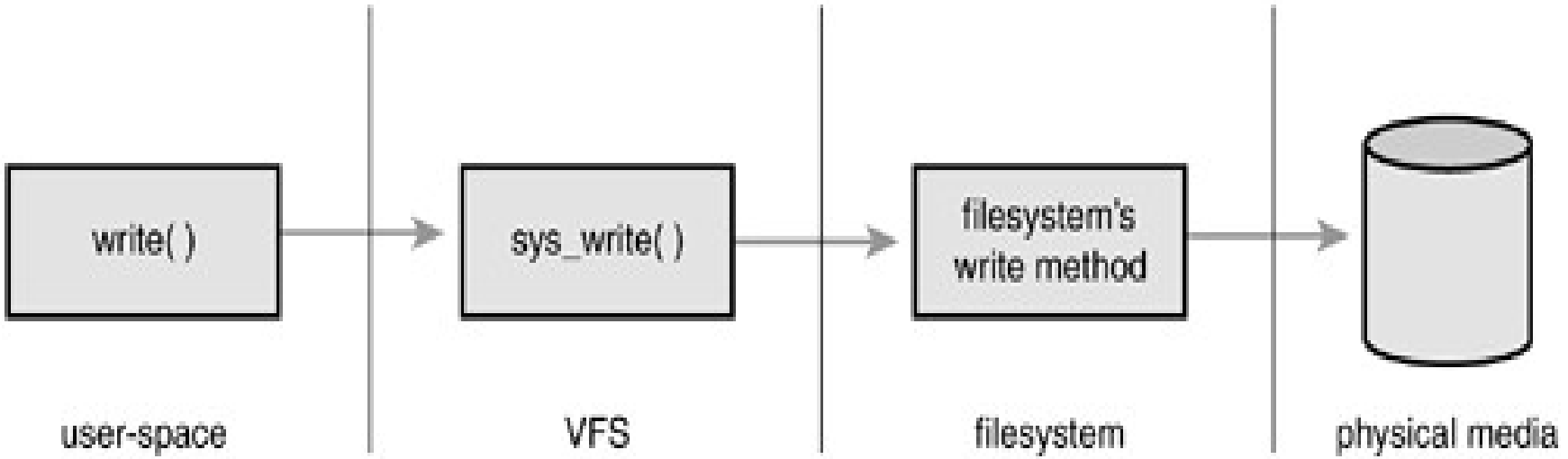
Crearea unui fişier

- Intrare: o cale
- Ieşire:
- Operații:
  - Se determină inode-ul directorului în care trebuie adăugat fişierul
  - Se citesc blocurile de date
  - Se inserează un nou dentry
  - Se scriu pe disc blocurile de date modificate

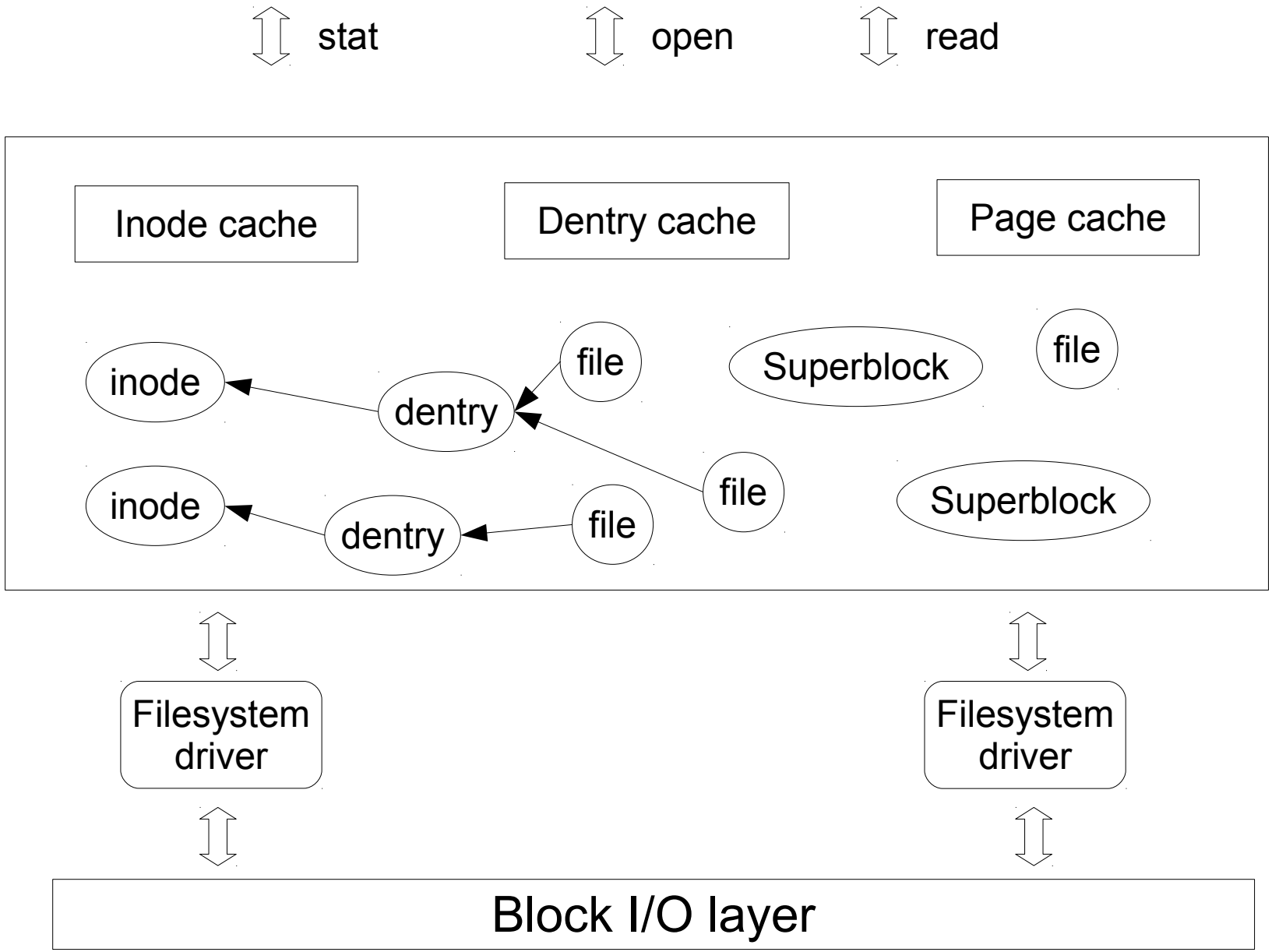
Ştergerea unui fişier

- Intrare: o cale
- Ieşire:
- Operații:
  - Se determină inode-ul directorului în care trebuie adăugat fişierul
  - Se citesc blocurile de date
  - Se caută şi se sterge DENTRY-ul cu numele fişierului (caz special: link-uri)
  - Se scriu pe disc blocurile de date modificate

Gestiunea sistemelor de fişiere în Linux



VFS


stat
open
read
Inode cache
Dentry cache
Page cache
inode
dentry
file
Superblock
file
inode
dentry
file
file
Superblock
Filesystem driver
Filesystem driver
Block I/O layer

Operații superblock

- fill_super
- put_super
- write_super
- read_inode
- write_inode
- delete_inode
- clear_inode
- Statfs
- remount_fs

Operații inode

- Create
- Lookup
- Link
- Unlink
- Symlink
- Mkdir
- rmdir

- Rename
- Readlink
- follow_link
- put_link
- Truncate
- ...

Inode cache

- Operațiile de citire a inode-urilor de pe disc sunt costisitoare
- Un inode citit se va menține în memorie (până când apar condiții de low memory)
- Căutarea inode-ului se face cu ajutorul unei tabele hash
- Funcția de hash (sb, inode_number)

Dentry

- Stări:
  - Used – d_inode este valid şi obiectul dentry este folosit
  - Unused – d_inode este valid dar obiectul dentry nu este folosit
  - Negative – d_inode nu este valid (NULL); fie inode-ul nu a fost încă încărcat fie a fost şters
- Dentry cache
  - Lista de dentry-uri folosite (dentry->d_state == used)
  - Lista celor mai recent folosite dentry-uri (sortată după timpul de access)
  - Un hash table pentru a elimina parcurgerea arborelui

Page cache

- Menține în memorie datele citite de pe disc
- Se foloseşte pentru:
  - read/write
  - Mmap
- Cache-ul se face la nivel de fişier şi nu de block device -> este nevoie de o infrastructură pentru translatarea offset-urilor din fişiere în blocuri pe disc
- Cache-ul se bazează pe un radix tree

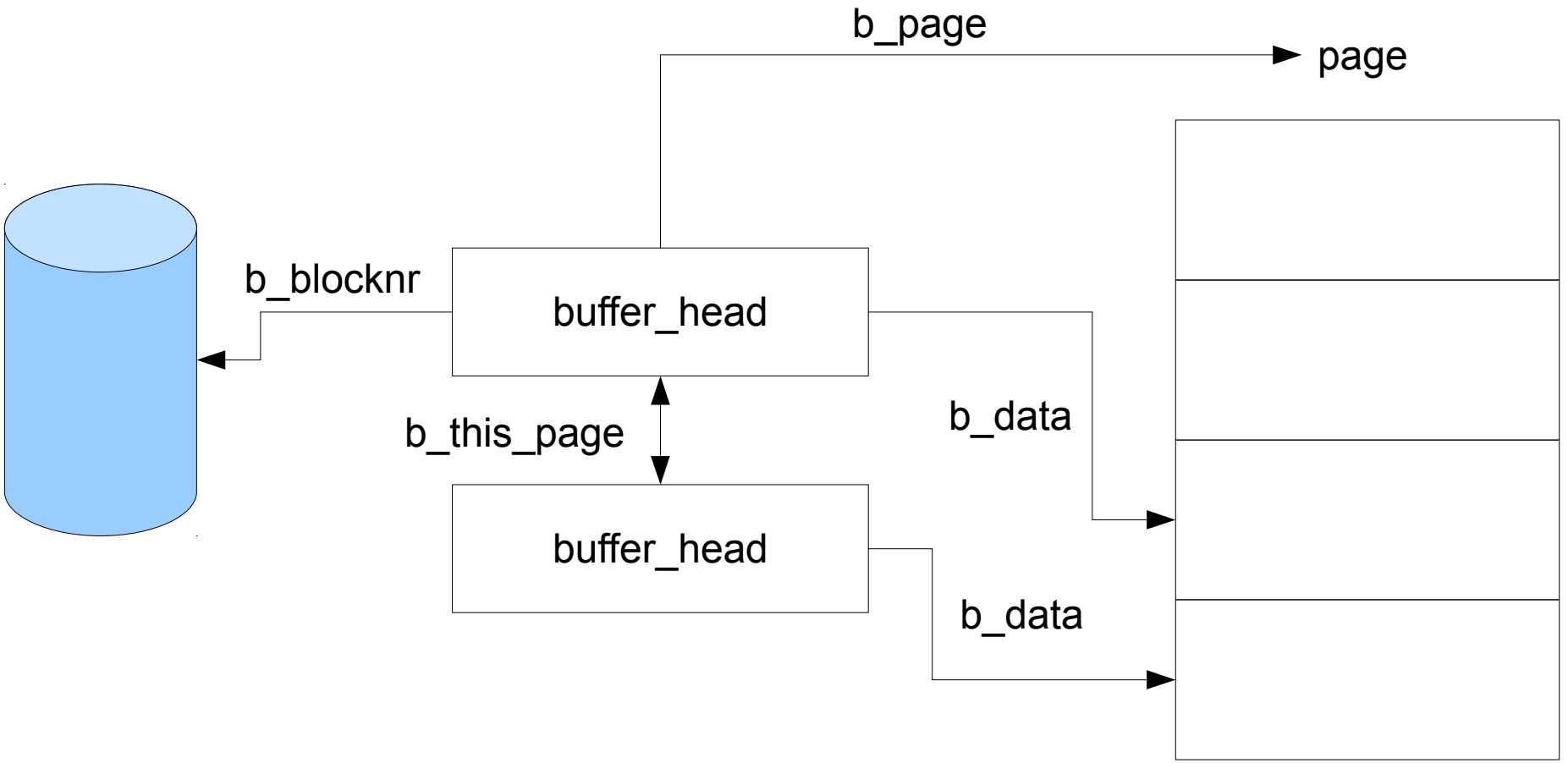
address_space

- Structura care este folosită de page-cache pentru a translata adrese
- Datorită acestei abordări se poate face caching nu doar pe mapări de fişiere
- Operații address_space:
  - write_page, read_page
  - Bmap
  - direct_IO
  - ...

Citirea de date din fişier

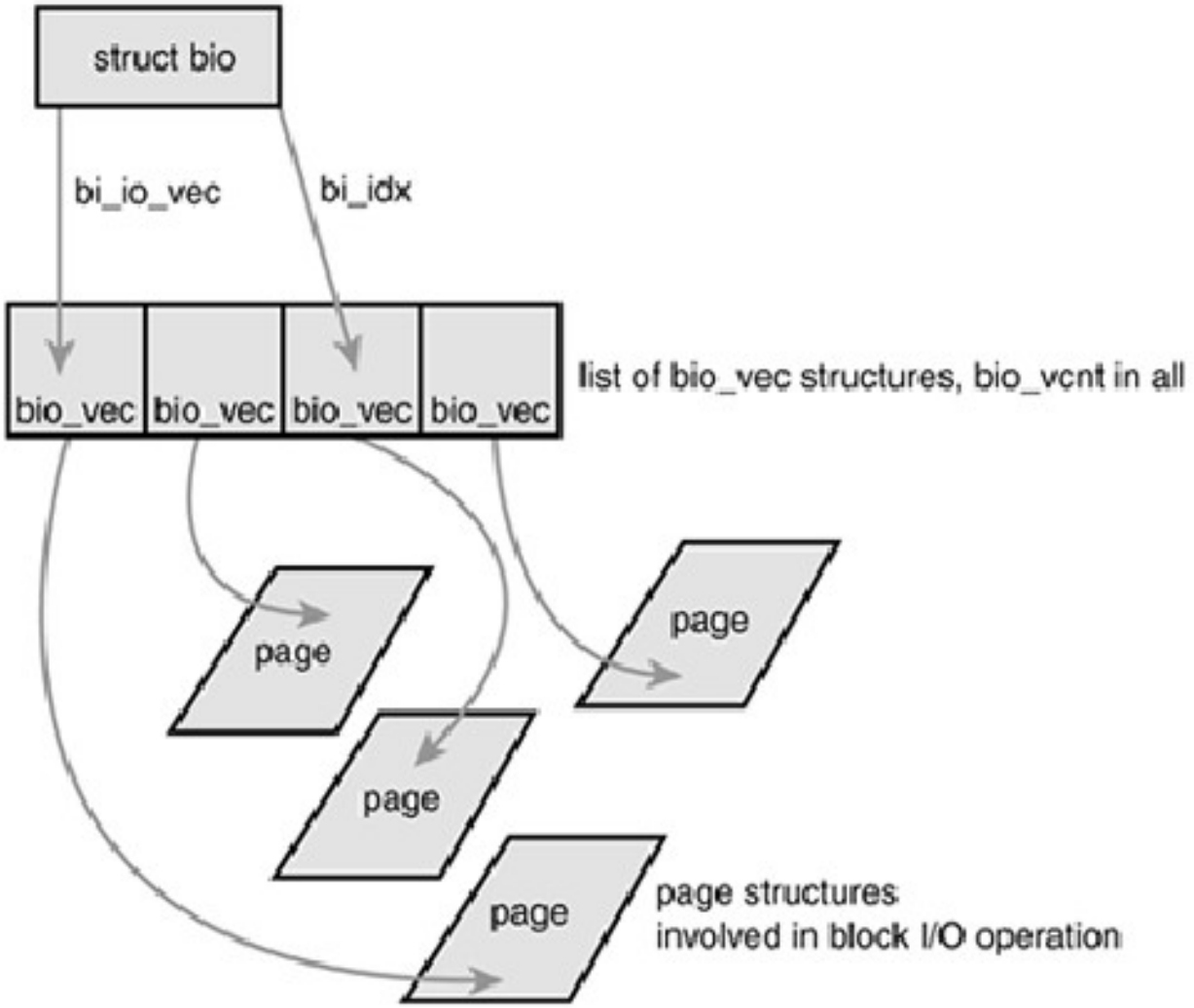
- generic_file_read – implementare generică
  - Se verifică dacă pagina nu există în page cache
  - Daca există se copiază datele din pache cache în buffer-ul utilizatorului
  - Dacă nu există se apelază a_ops->readpage()
- block_read_full_page – implementare generică, foloseşte **a_ops->bmap()**

struct buffer_head



- Mapează sectoare de pe disc în memorie

struct bio



- Comenzi I/O pentru block device drivere

I/O Schedulers

- The Noop I/O scheduler
  - Nu se sortează, doar se coagulează cereri dacă este cazul; folosit în sistemele embedded ce nu folosesc discuri
- The Linus elevator scheduler
  - Algoritm clasic, tip elevator
- The Deadline I/O scheduler
  - Se asociează un deadline cu fiecare cerere; când acesta expiră cererea se va executa indeferent de poziția în lista (sortată)
- The Anticipatory I/O scheduler
  - După ce o cere este servită se aşteaptă o perioadă de timp înainte de tratarea unei cereri (timp în care se aşteaptă cereri similare de la aplicații)

I/O Schedulers (2)

- The Complete Fair Queuing I/O scheduler
  - Fiecare process are propria coadă de I/O şi un timeslice
  - Nucleul inspectează fiecare coadă într-o manineră round-robind şi serveşte cererile din coadă până cand timeslice-ul expiră sau până când nu mai există cereri disponibile
  - Dacă timeslice-ul nu a fost folosit complet se aşteaptă o perioadă de timp înainte de a trece la următoarea coadă
  - În fiecare coadă cererile sincrone (e.g. operațiile de citire) au prioritate față de restul cererilor

Intrebări