# Sistemul de Fişiere din Unix

Mihai BUDIU — budiu@cs.cornell.edu, mihaib@pub.ro
http://www.cs.cornell.edu/Info/People/budiu/budiu.html

iulie 1996

**Subiect :** Sistemele de fişiere in Unix.

**Cuvinte cheie:** fişier, partiţie, i-nod, director, legătură.

**Cunoştinţe necesare:** noţiuni elementare despre: un sistem de operare, structuri de date, shell-ul Unix.

## Contents

Chiar şi cei mai naivi utilizatori ai unui calculator au de a face cu *sistemul de fişiere* în majoritatea timpului (termenul este consacrat în jargonul informatic); o mai apropiată privire asupra sa este salutară.

## 1 Terminologia

"Sistem de fişiere" (filesystem sau file system) este un termen folosit în mai multe accepţiuni distincte. O preliminară revizie a acestora poate ajuta eliminarea unora din neclarităţile care ar surveni atunci cînd în discuţie sensurile se schimbă ne-explicit.

O primă accepţiune a "sistemului de fişiere" este cea a unei *structuri de date* creată pe un suport de memorie, de obicei permanent, cum e discul, în care utilizatorii memorează informaţii într-un mod care le face mai uşor de stocat, manipulat sau regăsit.

O a doua accepţiune este cea a unui *set de operaţii* care se pot face asupra structurilor de date de mai sus: crearea, ştergerea, scrierea şi citirea de fişiere.

În fine, ultima accepţiune într-un sens le înglobează pe cele două anterioare: astfel sistemul de fişiere este o bucată dintr-un sistem de operare (adică o parte dintr-un program, de fapt); această bucată este vizibilă utilizatorului ca un set de *proceduri* (*apeluri de sistem*, pentru a fi mai precişi) care operează cu fişiere, manipulînd o structură de date pe un suport permanent de memorie.

Sistemul de fişiere văzut ca o colecţie de proceduri nu trebuie musai să fie parte a unui sistem de operare, însă de obicei sistemele moderne conţin un sistem de fişiere, pentru că este extrem de util; chiar dacă în anumite sisteme "micro-nucleu" sistemul de fişiere nu este parte a nucleului, el poate fi asimilat în sistemul de operare.

Vom încerca în secţiunile următoare să vedem cum arată sistemul de fişiere în Unix. Pe cel de la MS-DOS îl vom folosi cîteodată pentru a exemplifica o vedere alternativă.

# 2 Cu ce obiecte lucrăm

## 2.1 Partiţia

Pentru a simplifica discuţia vom presupune că sistemul de fişiere este plasat pe un disc. Variantele pe bandă, în memorie (Ram-Disk) sau prin reţea nu sunt neapărat fundamental diferite.

Discul este la nivelul cel mai de jos pentru software o instalaţie extrem de complexă: el este gestionat de un procesor propriu, numit controler de disc, care ştie să facă oarece operaţii, dar care trebuie programat pentru asta. Controlerul vede discul ca o sumă de *partiţii* independente. (Vom folosi cîteodată termenul de "disc" pentru a înţelege "partiţie", pentru că partiţia este un "disc logic".) Natura fizică a unui disc îl împarte în cilindrii, fiecare cilindru e împărţit în piste, fiecare pistă în sectoare, fiecare sector conţinînd mai mulţi octeţi.

Pentru a fi utilizabil, un disc trebuie să fie *formatat*. Operaţia de formatare (realizată de obicei cu programul `fdisk`) înscrie pe o partiţie o sumedenie de informaţii: de exemplu pe fiecare pistă este marcat numărul ei de ordine; în acest fel cînd se încearcă citirea uneia se poate verifica dacă se află capul de citire unde trebuie (detectînd erorile datorate mecanicii deficiente).

Deşi subiectul este realmente fascinant, nu vom insista asupra relaţiilor între aceste părţi, pentru că sistemul de fişiere se construieşte de obicei pe o structură ceva mai simplă, care este oferită de *driverul de disc*. Driverul este o colecţie de proceduri care ştie să manipuleze partiţiile, pistele şi toate alea, însă transformă discul într-un *vector de blocuri*, fiecare cu mai mulţi octeţi. Pe scurt, driverului îi indici un număr de bloc iar el decide pe care sector, pistă, cilindru se găseşte acest bloc. Dacă vrei să citeşti ceva din bloc, indici blocul şi numărul octetului din bloc iar driverul găseşte unde se află octetul pe discul fizic şi îl ia de acolo.

De ce blocuri şi nu octeţi? Pentru că raportul duratelor între a citi un octet şi a citi 100 de octeţi *consecutivi* nu este de 100 la 1 ci undeva între 1 şi 2. Asta pentru că foarte mult durează să găseşti primul octet; citirea următorilor este destul de rapidă. De asemenea, s-a observat că lumea rareori citeşte fişierele complet aleator: un octet aici altul colea; in general sunt accesaţi împreună mai mulţi consecutivi. De aceea driverul face toate operaţiile pe disc în termeni de blocuri de mărime egală.

Cît de mari? Depinde de sistem. De obicei între jumătate de kilo octet şi 8k.

Iată deci cum este plasat sistemul de fişiere în cadrul sistemului de operare:

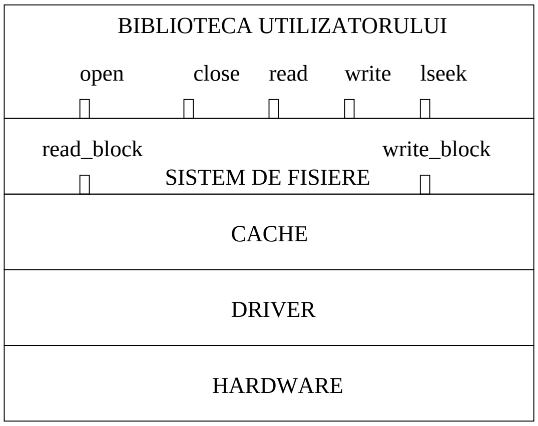


Figura 1: Sistemul de Fisiere in Sistemul de Operare

Sistemul de fişiere are deci la dispoziţie mai multe partiţii. Fiecare partiţie este văzută ca un array de blocuri independente. Citirea/scrierea unui bloc este realizată de driverul de disc, de serviciile căruia sistemul de fişiere se foloseşte. (În realitate din motive de eficienţă între sistemul de fişiere şi driver de obicei se mai plasează un nivel: *cache*-ul de disc, care memorează blocurile mai frecvent utilizate în memorie). În mod normal pe fiecare partiţie se construieşte cîte un sistem de fişiere independent; din această cauză vom simplifica discuţia presupunînd că avem la dispoziţie una singură.

În DOS partiţiile devin "discuri logice" distincte: C: D: E:, etc. În Unix ele sunt specificate prin "fişiere speciale" aflate în directorul `/dev`, dar pentru folosire sunt *montate* una peste alta. De ajuns pentru moment despre partiţii.

### 2.2 Fişierele

Sistemul de fişiere trebuie să ne ofere operaţii de: creare şi distrugere a fişierelor, numire a fişierelor (de fiecare dată cînd ai de-a face cu mai multe obiecte de acelaşi fel se pune problema de a le distinge prin nume diferite), de modificare şi inspectare a conţinutului fişierelor.

Modurile în care se pot realiza aceste operaţii sunt nenumărate, influenţate fiind de felurite proprietăţi pe care le dorim. De exemplu sistemul de operare CP/M (din care a fost derivat şi MS/DOS 1.0) are un singur *director* (vom vedea mai jos un pic ce este un director în realitate) în care se află toate fişierele. Din cauza asta oricare două fişiere sunt distinse numai prin numele lor, deci nu putem avea pe acelaşi disc două fişiere cu nume identice.

Unix, şi MS-DOS (care a preluat de la el ideea) folosesc o *ierarhie* (un arbore) de directoare, fiecare avînd un nume la rîndul lui. Fiecare fişier "se află" într-un director, în directoare diferite putînd exista fişiere la fel numite. Identificarea unui fişier se face indicînd pe lîngă numele lui, numele directoarelor prin care se ajunge la el (*calea* — *path* pînă la el).

Am aflat cam cum *numim* fişierele. Mai trebuie să ştim ce putem *face* cu ele. Întrebarea nu e chiar banală; de exemplu limbajul Pascal oferă un model de fişier "structurat": fiecare fişier este un array de lungime teoretic nelimitată de obiecte identice (record de obicei). Se pot imagina cu uşurinţă modele mai complicate de fişiere (unele foarte utile în aplicaţii de baze de date); de exemplu fişiere de "perechi" în care fiecare dată are o "cheie" iar accesele se fac după valorile cheii: "găseşte cîmpul cu cheia 100", "incrementează cîmpul cu cheia **Bibi**".

Unix oferă probabil cel mai simplu model de fişier: un array de octeţi arbitrari. Acelaşi lucru îl face şi DOS-ul, distingînd cîteodată între fişiere text şi fişiere "oarecare".

Deşi ar mai fi multe de spus, să ne concentrăm acum asupra modului în care sunt realizate aceste operaţii. Deşi felurite versiuni şi variante ale Unix-ului pot proceda în realitate în mod diferit, vom vedea cum arată implementările "clasice". Mai important decît să ştim cum se fac lucrurile în detaliu este să ştim cum le-am putea face noi.

Cum s-a ajuns la structura pe care o vom desfăşura mai jos din cerinţele pe care le-am schiţat mai sus, nu vă va fi foarte limpede pînă nu veţi încerca să rezolvaţi aceeaşi problemă în alt fel: atunci veţi vedea cum se justifică unele construcţii care par artificiale.

O ultimă observaţie preliminară: din comunicările prezentate la conferinţa USENIX (a utilizatorilor de Unix), peste 40% sunt legate de sisteme de fişiere: implementări, optimizări, arhitecturi. Deci domeniul este în plină evoluţie.

## 3 Structura de date

Sistemul de fişiere foloseşte ca unitate de bază blocul de pe partiţie. Cînd este nevoie de spaţiu se alocă un bloc întreg, chiar dacă e nevoie de mai puţin. Asta simplifică gestiunea spaţiului liber. Iată cum sunt folosite blocurile de pe o partiţie pentru a crea un sistem de fişiere:

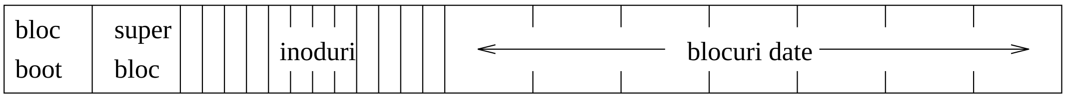


Figura 2: Structura unei partiţii Unix cu sistem de fişiere

### 3.1 Blocul de boot

Nucleul însuşi al sistemului de operare este de obicei ţinut sub forma unui fişier. Însă procedurile care manipulează fişiere sunt parte a nucleului, aşa că pentru a porni (boot) sistemul de operare însuşi nu se pot folosi operaţiile pe fişiere, căci ele nu sunt încă citite de pe disc! Din această cauză nucleul trebuie să fie accesibil şi direct, fără a fi nevoie să fie citit ca fişier.

Blocul de boot conţine tocmai informaţiile necesare pentru a citi nucleul: de obicei o listă a blocurilor de pe disc ocupate de fişierul-nucleu. Cînd calculatorul porneşte un program mic (din ROM de exemplu) citeşte nucleul în memorie folosind aceste informaţii.

### 3.2 Superblocul

Superblocul este rezervat pentru a descrie cum sistemul de fişiere foloseşte partiţia. Informaţiile pe care le cuprinde sunt:

- un *număr magic* (o constantă rezervată) pentru a indica tipul sistemului de fişiere (ex.: sistem de fişiere SCO Unix);
- mărimea partiţiei;
- numărul de i-noduri (fişiere);
- numărul de i-noduri ocupate;
- numărul de blocuri libere;
- lista blocurilor libere;
- starea partiţiei (este cache-ul salvat pe disc?);
- i-nodul directorului rădăcină;
- etc.

Superblocul dă o descriere a informaţiilor *globale* referitoare la întreaga partiţie care conţine un sistem de fişiere. Le vom pricepe cînd vom şti de fapt despre ce este vorba.

h

| |
|---|
| este i-nodul liber? |
| posesorul fişierului (uid) |
| grupul care posedă fişierul (gid) |
| drepturile asupra fişierului (rwx rwx rwx) |
| atribute suplimentare ale fişierului (tss) |
| tipul fişierului (director, ordinar, special) |
| data (inclusiv ora) creării fişierului |
| data ultimei modificări |
| data ultimului acces |
| lungimea fişierului |
| lista blocurilor pe care le foloseşte |
| numărul de "legături" (reference count) |

Figura 3: Inod.

### 3.3 I-nodurile

În Unix numărul de fişiere care se pot afla la un moment dat pe un sistem de fişiere este limitat precis, încă de la crearea sistemului de fişiere (vedeţi secţiunea despre `mkfs`)! Fiecare fişier are un *i-nod* (i-node sau inode în engleza, abreviere de la Information Node) asociat, care conţine aproape toate informaţiile legate de acel fişier, mai puţin *conţinutul* şi *numele*. Unele i-noduri sunt nefolosite la un moment dat.

Chiar dacă pe un disc mai există blocuri libere, în momentul cînd au fost folosite toate i-nodurile nu se mai pot crea noi fişiere. Discul este plin.

Să ne uităm mai aproape la un i-nod:

#### Cîmpurile i-nodului

Iată o scurtă descriere a semnificaţiei fiecăruia din cîmpurile de mai sus (nu uitaţi: anumite implementări pot avea şi altele). Cele mai multe cîmpuri pot fi văzute cu ajutorul comenzii `ls`. `ls` are ca argumente nume de fişiere, nu i-noduri, dar cum fiecare fişier are un singur i-nod (vom vedea cum se găseşte ce i-nod corespunde unui anumit nume mai încolo), `ls` ştie de unde să ia informaţiile.

Pentru a vedea ce i-nod corespunde unui fişier folosiţi `ls -i`; de exemplu la mine acasă:

```
$ ls -i /bin/b*
 22363 /bin/bash      22905 /bin/binmail
```

Deci fişierul `/bin/bash` are i-nodul cu numărul 22363, iar fişierul `/bin/binmail` are i-nodul 22905.

#### este i-nodul liber?

Aici este marcat un i-nod liber. La crearea unui nou fişier un i-nod liber este alocat, marcat ocupat, şi cîmpurile celelalte înscrise corespunzător. La ştergerea unui fişier i-nodul devine liber (atenţie: "ştergerea" în Unix este un fenomen destul de complicat!).

**uid (User IDentifier)**

Fiecare fişier în Unix aparţine cuiva. Un identificator numeric asociat celui care posedă fişierul (de obicei cel care l-a creat) este marcat aici. Al treilea cîmp de la `ls -l` indică utilizatorul. Dacă vreţi valoarea numerică a UID daţi `ls -ln`.

Identificatorul acesta ridică oarecari probleme cînd un sistem de fişiere este folosit în reţea (ca sistemul NFS — network file system), pentru că pe un calculator utilizatorul cu codul 10 poate fi cineva, iar pe alt calculator altcineva! (Aceasta este o deficienţă a Unix-ului care nu a fost proiectat originar pentru a opera în reţea — cîrpelile nu merg prea bine.)

```
$ ls -l /bin/bash
-rwxr-xr-x   1 root     bin        279556 Feb  7  1994 /bin/bash
               ^^^^     ^^^
               UID      GID
```

**gid (Group ID)**

În Unix utilizatorii fac parte din unul sau mai multe *grupuri*. Fiecare fişier aparţine şi el unui anume grup, care este indicat aici. Consideraţii asemănătoare cu cele de la UID se aplică şi aici. Al patrulea cîmp din `ls -ln` este GID, sub formă numerică.

**drepturile**

Puteţi vedea drepturile în coloanele 2-10 din ceea ce tipăreşte `ls -l`.

```
$ ls -l /bin/bash
-rwxr-xr-x   1 root     bin        279556 Feb  7  1994 /bin/bash
 ^^^^^^^^^
 drepturile
```

Drepturile vin sub forma a trei triplete de valori Da/Nu. Primul triplet arată ce poate face cu fişierul *posesorul* lui (un proces al unui utilizator cu UID egal cu cel al fişierului). Al doilea arată ce pot face utilizatorii din grupul fişierului. Al treilea triplet arată ce poate face restul lumii.

Fiecare grup de trei valori Da/Nu arată aşa:

| | | |
|---|---|---|
| r | w | x |
| Read | Write | eXecute |
| Citeşte | Scrie | Execută |

Ele arată dacă fişierul poate fi citit, modificat sau respectiv executat (pentru fişiere de tip director *căutat*) de persoana respectivă. Pentru un exemplu concret să luăm un fişier cu drepturile:

```
r-x-w---x
\_/\_/\_/
  \  \  \ restul lumii
   \  \__grupul
    \___posesorul
```

Asta înseamnă că:

- posesorul fişierului poate: citi şi executa acest fişier, dar *nu* modifica;
- cei din grupul fişierului: pot modifica acest fişier, dar *nu*-l pot citi sau executa;
- ceilalţi: pot doar executa fişierul, dar *nu* citi sau modifica.

**atributele tss**

Fişierele mai au trei atribute de forma Da/Nu (biţi). Acestea se numesc: bitul SUID, bitul SGID şi bitul *sticky* (lipicios). Multă informaţie este condensată în aceşti trei biţi.

- bitul SUID (Set User ID): un fişier cu acest bit pus pe Da arată faptul că un execuţia acestui fişier dă naştere unui proces care este posedat de posesorul *fişierului* şi nu de posesorul *procesului* care execută fişierul (discuţia depăşeşte cadrul sistemelor de fişiere aşa că ne oprim aici cu explicaţiile; poate vom reveni cu o altă ocazie). Un bit SUID este indicat la `ls -l` printr-un `s` în loc de `x`:

```
$ ls -l /bin/mount
-rwsr-xr-x   1 root     bin         21508 Jan 28  1994 /bin/mount
   ^
   SUID
```

  Cînd utilizatorul `mihai` va executa acest fişier, procesul va avea UID `root`, şi nu `mihai`, din cauza bitului SUID.

- bitul SGID (Set Group ID) are o funcţie similară, acţionînd asupra grupului procesului:

```
$ ls -l /usr/bin/convfont
-rwsr-sr-x   1 root     root         2272 Mar 16  1994 convfont
   ^  ^
   SUID si SGID
```

  Cum se vede şi din descriere, aceşti biţi nu au sens decît dacă fişierul este executabil. Pentru fişiere ne-executabile ei sunt folosiţi pentru a indica faptul că folosirea fişierului se poate face *numai* după *încuierea (lock)* lor. (Acesta este un mecanism care nu permite accesul simultan a două procese la un singur fişier.) Aceşti biţi se vad cu litere mari `S`.

  Biţii SUID la directoare indică faptul că toate fişierele create în acele directoare moştenesc drepturile de la director şi nu de la procesul care le crează.

  Ar mai fi cîte ceva de spus despre lock-uri, dar ar merita un articol în sine. Să mergem mai departe cu i-nodul.

- bitul Sticky (lipicios), notat cu `t`

  La fişiere executabile iniţial acest bit indica nucleului să optimizeze folosirea fişierului: odată executat (deci citit în memorie) nu mai trebuie scos de acolo, chiar dacă procesul se termină, pentru că probabil va fi folosit din nou în curînd.

  Pentru directoare el indică o calitate foarte interesantă: în directoarele în care oricine poate scrie (`-------w-`), cum e `/tmp`, teoretic oricine poate şterge orice fişier (după cum vom vedea un pic mai jos). Un bit lipicios pe un astfel de director va permite însă ştergerea unui fişier *numai* de către posesorul lui (de către un proces avînd acelaşi UID cu i-nodul).

```
$ ls -ld /tmp
drwxrwxrwt   3 root     root         1024 Jul 26 00:10 /tmp/
                  ^
                  bit sticky
```

Modificatorul `d` al lui `ls` este necesar pentru a lista drepturile directorului tmp şi nu ale tuturor fişierelor cuprinse în el.

### tipul fişierului

`ls -l` va scrie tipul unui fişier în prima literă afişată. Aceasta poate fi într-un Unix normal:

| | |
|---|---|
| d | pentru un director |
| `-` | pentru un fişier ordinar |
| l | pentru o legătură simbolică (*symbolic link*) |
| s | pentru o *mufă (socket)* |
| c | pentru un fişier special de tip caracter |
| b | pentru un fişier special de tip bloc |
| p | pentru o *ţeavă cu nume (named pipe)* |

Cele mai multe dintre aceste tipuri de fişiere sunt de fapt doar nişte nume asociate unor părţi ale nucleului, şi nu vor fi discutate în acest articol. Primele trei au într-adevăr legătură cu sistemele de fişiere, dar numai despre primele două vom vorbi în acest articol, pentru că altfel iese prea mare.

### timpii

Fiecare i-nod înregistrează în formatul tipic Unix (numit `time_t`, de obicei un întreg lung reprezentînd numărul de secunde scurse de la 1 ianuarie 1970) data (şi ora) creării, accesului şi modificării fişierului. Timpii se pot obţine astfel cu `ls`:

| | |
|---|---|
| `ls -l` | timpul creării |
| `ls -ul` | timpul ultimului acces |
| `ls -lc` | timpul ultimei modificări |

**lungimea fişierului**

în octeţi este indicată de asemenea de `ls -l`:

```
$ ls -l /bin/bash -rwxr-xr-x 1 root bin 279556 Feb 7 1994 /bin/bash
                                        ^^^^^^ bash are 273K
```

**lista blocurilor folosite**

nu poate fi văzută cu `ls`. Ea este ţinută într-o formă foarte ingenioasă care permite accesul extrem de rapid în orice punct al fişierului. Această listă indică pentru fiecare bloc din fişier ce bloc din partiţie corespunde (fiecare prim bloc din fiecare fişier ocupă alt loc în partiţie, fireşte).

Pentru a indica ce blocuri compun un fişier se folosesc *pointeri* la blocuri. Un pointer la un bloc este pur şi simplu numărul de pe partiţie al unui bloc. Pentru că blocul 0 nu poate fi al vreunui fişier (fiind blocul de boot) avem la dispoziţie şi o valoare care poate indica *lipsa* unui bloc. (Vom vedea că in Unix un fişier poată avea "găuri": blocuri lipsă în mijloc!)

Schema este extrem de ingenioasă, dar un pic mai complicat de descris în cuvinte. Pe scurt i-nodul conţine:

- pointeri spre primele 10 blocuri ale fişierului;
- dacă fişierul e mai lung de 10 blocuri, un pointer spre un bloc (numit *blocul simplu indirect*) în care se pun pointerii spre următoarele blocuri. Cît de mulţi pointeri sunt în acest bloc depinde de mărimea blocului şi de mărimea pointerului (de obicei 32 de biţi, dar poate mai mult!).
- dacă fişierul are mai multe blocuri decît se pot reprezenta astfel, urmează un pointer la un bloc în care sunt pointeri la blocuri care ţin pointeri la blocurile fişierului: indirectare dublă.
- în fine, dacă nici atît nu ajunge, urmează o indirectare triplă.

O schemă aproximativă ar arăta aşa:

Cîte blocuri se pot indica astfel? Dacă într-un bloc încap X pointeri avem lungimea maximă pentru un fişier:

10 + X + X*X + X*X*X blocuri

Pentru nişte valori tipice: bloc de 1024 octeţi şi pointer pe 4 octeţi avem X=1024/4=256. Atunci valoarea de mai sus devine

(10 + 256 + 256*256 + 256*256*256) * 1K = (10 + 256 + 65536 + 16777216) K >= 16 Giga octeţi.

Pînă de curînd asta era destul de mult pentru un singur fişier. Oricum, introducerea unui al patrulea nivel de indirectare ar ridica valoarea la 4 Tera octeţi, ceea ce ar trebui să ajungă pentru o vreme…

Din virtuţile acestei scheme: fişiere scurte rapide şi compacte (oricare din primele 10 blocuri se accesează imediat), dimensiune maximă foarte mare, acces rapid la orice bloc (maximum trei blocuri citite de pe disc pentru indirectări), fişiere care pot avea găuri.

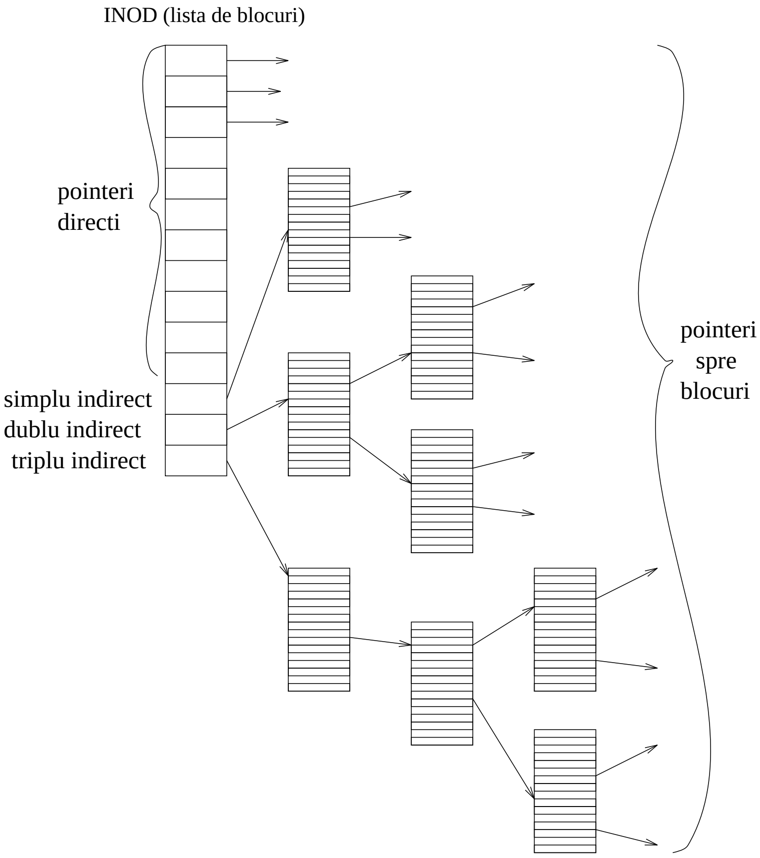


Figura 4: Pointerii spre blocuri

#### numărul de legături

În Unix un fişier se poate "afla" simultan în mai multe directoare! Cîte apariţii distincte are el este indicat de numărul de legături. De exemplu:

```
$ ls -ld /usr/bin
drwxr-xr-x   2 root     bin          7168 Jul 23 12:45 /usr/bin
             ^
             numarul de legaturi este 2!
```

Cu alte cuvinte directorul `/usr/bin` este prezent în două directoare! Vom vedea mai jos de ce.

Cu asta am epuizat structura i-nodului clasic.

### 3.4 Zona de date

Să revenim la studiul partiţiei: organizarea ei în sistem de fişiere. După blocurile în care se ţin i-nodurile toate blocurile care urmează sunt alocate pentru a servi la memorarea conţinutului fişierelor. Aceste blocuri sunt la început toate libere: nici unul nu este indicat de i-nodul vreunui fişier ca aparţinînd acelui fişier. Pe măsură ce se scriu date în fişiere şi acestea cresc, se iau blocuri libere şi se adaugă fişierelor, făcînd i-nodurile să puncteze spre ele.

Cînd fişierele sunt şterse (sau trunchiate) blocurile revin pe lista blocurilor libere.

### 3.5 Lista de blocuri libere

Sistemul de fişiere ţine şi o listă a blocurilor libere, pentru a se aproviziona rapid cînd are nevoie să crească fişierele. Cînd lista este goală discul este plin (în alt fel decît atunci cînd nu mai sunt i-noduri).

Lista blocurilor libere este şi ea structurată într-un mod inteligent: în superbloc există un pointer la un prim bloc liber. Acesta este plin cu pointeri la alte blocuri libere. Ultimul dintre blocurile arătate de el conţine la rîndul lui pointeri spre blocuri libere ş.a.m.d.

```
superbloc--->primul bloc liber /-->bloc liber /-->bloc liber
                     | | | | | |   | | | | |  |   | | | | |
                     v v v v \-/   v v v v \--/   v v v v ...
                    pointeri spre blocuri libere
```

Această listă arată ca un arbore dezechilibrat: numai nodurile de pe ramura cea mai din dreapta au fii; toate celelalte sunt frunze. De ce schema asta? Pentru că modificarea listei se face de la început: se iau din blocurile libere punctate de primul, iar cînd toate s-au terminat mai puţin ultimul (din cele punctate de primul), chiar primul bloc liber se alocă şi superblocul punctează spre cel care era ultimul. Se procedează invers pentru eliberarea de blocuri.

Schema este mai eficientă decît o listă simplu înlănţuită pentru că nu trebuie citit decît un bloc liber (primul) pentru a aloca X pointeri: cei pe care îi conţine.

### 3.6 Crearea unui sistem de fişiere

Deşi utilizatorii de DOS pot avea această impresie, nu orice partiţie trebuie să conţină un sistem de fişiere! Ba chiar Unix-ul foloseşte în mod curent o partiţie (de *swap*) pentru a crea mecanismul de memorie virtuală. Pe acea partiţie nu se află fişiere.

Operaţia prin care pe o partiţie formatată se crează un sistem de fişiere se numeşte în Unix `mkfs` (MaKe FileSystem) iar în DOS `format` (după cum se vede în DOS distincţia între partiţie şi sistem de fişiere nu este prea clară nici la nivel de nume al programelor utilitare).

Programul `mkfs` foloseşte driverul de disc pentru a crea blocul de boot, superblocul şi a iniţializa toate i-nodurile şi blocurile de date ca fiind goale. Abia după ce el este executat poate fi folosit sistemul de fişiere de pe partiţie, *montîndu*-l (cu comanda-apel de sistem `mount`).

## 4 Operaţiile pe fişiere

După ce am aflat atîtea despre structura de date cu care se descrie un fişier putem să ne imaginăm cu uşurinţă cum se implementează operaţiile de citire/scriere în fişiere. Să modelăm operaţia de scriere, care este mai complicată.

Să zicem că vrem să scriem octetul nr. 100000 cu valoarea 1 într-un anumit fişier.

Prima etapă, de care încă n-am spus nimic, este de a găsi i-nodul fişierului cînd i se dă numele. (Secţiunea despre directoare a articolului de faţă numai cu asta se ocupă.) Această etapă verifică şi dacă am permisiunea de a face operaţia cerută pe acest fişier. Dacă nu am voie, operaţia nu se face şi apelul de sistem care a invocat-o se termină cu o eroare.

După ce am găsit i-nodul, trebuie să găsim blocul pe disc în care e octetul nr. 100000. Ştim cîţi octeţi sunt într-un bloc, aşa că aflăm al cîtulea bloc al fişierului este cel care are octetul 100000. Presupunînd, ca mai sus, că un bloc are 1024 de octeţi, obţinem blocul $100000/1024 = 98$ (rotunjind în sus). Ca să ajung la blocul 98 trebuie să citesc pointerii din blocul de pointeri simplu indirecţi (el conţine pointerii pentru blocurile 11-266). Şi anume trebuie să caut al 88-lea pointer (nu uitaţi că primele 10 blocuri au proprii pointeri). Deci mă uit la blocul cu pointerii spre blocuri.

Dacă fişierul a fost mai scurt de 10 blocuri pînă acum, acest bloc (cel simplu-indirect) nu exista! În acest caz îl aloc de pe lista de blocuri libere şi îl umplu cu pointeri zero, ca să arăt că e gol.

După ce am găsit blocul simplu-indirect mă uit la al 88-lea pointer din el. Dacă e nenul, înseamnă că blocul 98, pe care vreau să-l modific, făcea parte deja din fişier. Altfel trebuie să îl aloc tot eu. O fac dacă e nevoie, înscriind în blocul simplu-indirect la pointerul 88 adresa blocului 98, proaspăt alocat. Blocul nou alocat îl umplu cu zerouri.

În fine, blocul la care am ajuns conţine octetul cu nr. 100000. Al cîtulea este? Aflu făcînd restul împărţirii 100000 la 1024 = 672. Deci în acest bloc scriu în octetul 672 valoarea 1, după care pun blocul la loc pe disc.

Dacă lungimea fişierului era mai mică de 100000 (trecută în i-nod) o aduc la această valoare. Pun şi timpul accesului şi modificării fişierului la ora curentă.

Dacă a fost nevoie pe drum şi nu s-a putut aloca un bloc liber, operaţia iar a eşuat.

Aţi prins schema? Încercaţi să o urmăriţi pe hîrtie cu creionul.

Observaţi că nu este obligatoriu ca blocurile 1-97 să existe! Un fişier poate avea doar octetul 100000, ocupînd pe disc doar două blocuri (cel simplu indirect şi cel cu octetul) în loc să aibă 99 de blocuri cu zerouri!

Aceasta este semantica operaţiei de scriere în fişier în Unix. Citirea unui octet care nu există se soldează cu citirea unui zero.

# 5 Directoarele

Observaţi un lucru interesant: dacă ştii i-nodul unui fişier ştii toate informaţiile importante despre el şi ai acces la tot conţinutul lui.

Directoarele în Unix au un singur rol: să asocieze fiecărui *nume* de fişier un i-nod. Se spune despre directoare că "leagă nume de i-noduri". Directoarele sunt din multe privinţe nişte fişiere. Deosebirea esenţială însă este că directoarele sunt nişte fişiere care au *o structură*: sistemul de fişiere interpretează el însuşi octeţii care formează conţinutul unui director şi nu permite utilizatorului manipularea lor arbitrară.

Pe scurt un director este un vector de legături (links). Orice director arată aşa:

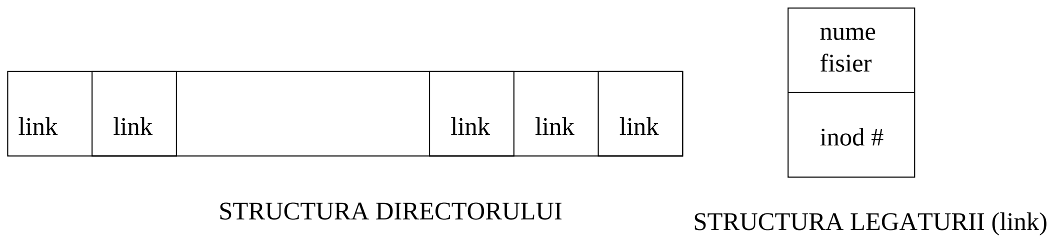


Figura 5: Link şi Director

Pe lîngă un bit care distinge link-urile folosite de cele libere (link-uri libere apar din cauza lui `rm` — ştergerea), link-ul mai conţine doar două cîmpuri: un nume de fişier (care poate conţine aproape orice caractere, cea mai notabilă excepţie fiind `/`), şi un număr de i-nod. O restricţie este ca toate numele aflate în link-urile dintr-un director sa fie distincte. Nici unul din caracterele care compun numele unui fişier nu are vreo semnificaţie specială (cum de exemplu în DOS are punctul).

Spunem că fişierul `bash` se află în directorul `bin` dacă fişierul `bin`, de tip director, conţine un link al cărui nume este `bash`.

Directoarele pot conţine la rîndul lor directoare: pur şi simplu i-nodul dintr-un link este un i-nod de tip director.

După cum vedeţi de fapt fişierele nu sunt *incluse* în directoare, ci directoarele ţin nişte *pointeri* (numărul de i-nod) spre fişierele propriu-zise, asociindu-le şi un nume. O primă întrebare care se poate ivi: nu poate un i-nod să fie prezent în două linkuri diferite? Răspunsul este da: pot exista link-uri multiple la un acelaşi i-nod, în acelaşi director sau în directoare diferite!

Un director ar putea arăta aşa:

```
nume  | vi  | ex  |     |
i-nod | 100 | 100 | ... |
```

Asta înseamnă că un *acelaşi* fişier este prezent de două ori, odată cu numele `vi`, altă dată cu numele `ex`. Modificarea fişierului numit `vi` se reflectă prin modificarea fişierului `ex`.

## 5.1 Linkurile . şi ..

Observaţi că datorită faptului că nu fişierul este în director ci un arătător spre fişier, *arborele* de directoare devine un graf: putem avea cicluri (directoare care se conţin reciproc) sau fişiere în mai multe directoare simultan.

Prin definiţie un director nu este niciodată gol (deşi la o adică putem să-l forţăm), ci conţine întotdeauna cel puţin două link-uri. Unul se numeşte `.` (punct) şi este un link la i-nodul directorului însuşi, iar altul se numeşte `..` (punct punct) şi este un link la directorul *tată* (în care a fost creat cel despre care vorbim). Să vedem un ipotetic exemplu:

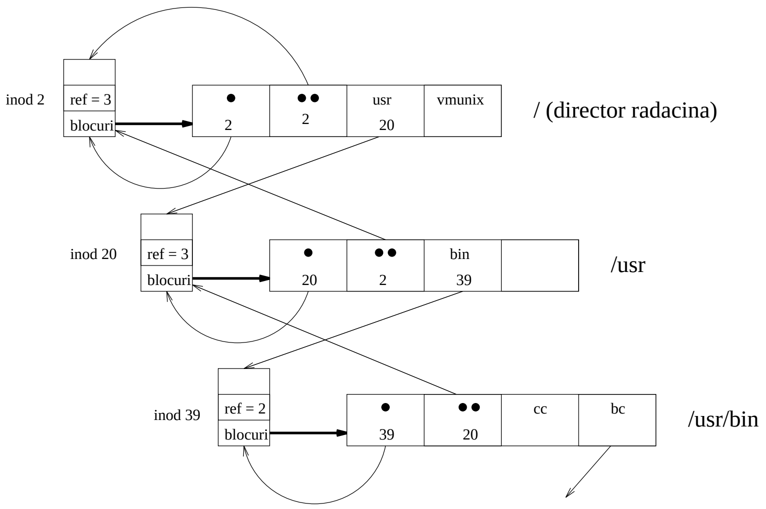


Figura 6: Legaturi intre directoare

După cum vedeţi în directorul `/usr`, care are i-nodul 20, se află un link cu numele `bin` şi i-nodul 39. Directorul cu i-nodul 39 conţine un link cu numele `.` spre i-nodul 39 (el însuşi) şi un link cu numele `..` spre directorul cu i-nodul 20 ("tatăl" lui).

Dacă mai ţineţi minte, i-nodul are un cîmp care arată cîte link-uri arată spre el însuşi. Din cauza asta mai sus

```
$ ls -ld /usr/bin
```

```
drwxr-xr-x   2 root     bin          7168 Jul 23 12:45 /usr/bin
             ^
```

are două legături: una aflată în `/usr` (cu numele `bin`) şi una în el însuşi cu numele `.` (punct).

Dacă în `/usr/bin` ar mai fi fost subdirectoare, fiecare din ele ar fi avut un link cu numele `..` spre `/usr/bin`.

Comanda `pwd` (print working directory), care afişează directorul curent al procesului care o execută, lucrează astfel: urmăreşte link-urile `..` pînă la directorul rădăcină şi caută i-nodul de la link-ul `.` în directorul `..` pentru a afla numele (mai citiţi odată fraza, pe-ndelete).

### 5.2 Directorul rădăcină

Orice sistem de fişiere are un director privilegiat, care are link-ul `..` spre el însuşi. Acesta este *directorul rădăcină* al sistemului de fişiere. Toate căutările de fişiere care încep cu semnul `/` caută întîi în acest director. Mai exact, sa vedem cum se fac

## 6 Operaţiile pe directoare

### 6.1 Găsirea i-nodului unui fişier

Principala operaţie pentru care se folosesc directoarele este pentru a găsi i-nodul unui fişier cînd ştiu directoarele care "duc" la el (*calea: path*).

Orice proces în Unix are ceea ce se cheamă un *director curent* (în MS-DOS există un singur director curent pentru tot sistemul de operare). Acesta este cunoscut procesului prin i-nodul său (şi se poate schimba cu apelul de sistem `chdir`). Cînd un proces specifică o operaţie pe fişiere el indică *întotdeauna* un nume de fişier ca o listă de directoare separate prin semnul `/` (în MS-DOS acesta a fost înlocuit din motive obscure cu `\`). Dacă primul semn dintr-un path este `/` atunci primul director în care se caută este directorul rădăcină. Altfel primul director în care se caută este cel curent.

Să vedem acum, pe un exemplu, cum nucleul găseşte i-nodul fişierului `/usr/bin/cc`.

Primul semn este `/`, deci deschide directorul rădăcină. I-nodul lui este marcat în superbloc, deci conţinutul lui este accesibil.

Primul nume este `usr`. Caută deci printre toate link-urile din directorul rădăcină unul cu numele `usr`. Dacă l-a găsit, se uită la numărul de i-nod al lui `usr`. Citeşte apoi acest i-nod. Pentru că mai există nume în path după `usr`, i-nodul curent trebuie să aibă tipul director.

Presupunînd că aşa este, nucleul deschide fişierul aflat (`usr`) şi caută un link cu numele `bin`. Găseşte i-nodul lui, citeşte fişierul (`bin`) şi caută link-ul `cc`. Odată găsit, a aflat i-nodul lui `cc`, deci poate face orice operaţie cu acesta.

Mai trebuie spus că orice operaţie de deschidere a unui director pentru căutare este precedată de verificarea dreptului de a face această căutare. Există două feluri de drepturi pentru directoare:

- bitul `r` la un director arată dreptul de a citi conţinutul directorului (un array de link-uri). Deci dreptul de a afla ce i-nod corespunde la un nume.
- bitul `x` (em search!) arată dreptul de a citi conţinutul unui i-nod indicat de un link din acest director.

Dacă un director este accesibil are de obicei drepturi `r-x`.

Dacă are numai drepturi `r--` atunci se poate afla ce fişiere sunt în el, dar ele nu pot fi deschise prin aceste link-uri, şi nici i-nodurile lor nu pot fi accesate. Puteţi face `ls` sau `ls -i` dar nu `ls -l` pe un astfel de director.

Dacă are numai drepturi `--x` *nu* puteţi vedea ce fişiere conţine, dar dacă *ştiţi* unul din nume, îl puteţi deschide!

Dacă are drepturi `---` pentru dumneavoastră, atunci uitaţi de el.

## 6.2 Crearea şi distrugerea fişierelor

Pentru a crea un fişier în primul rînd trebuie alocat un i-nod şi apoi construit un link. În Unix *toate* operaţiile asupra fişierelor specifică numele fişierului, şi niciodată i-nodul. Nu poţi accesa un fişier al cărui i-nod îl cunoşti: trebuie să ştii un path spre el. Din cauza asta funcţionează securitatea în Unix: dacă nu există nici un path în care un utilizator să aibă dreptul de a citi toate directoarele componente, atunci el nu va putea ajunge niciodată să citească fişierul indicat de acel path.

Crearea fişierelor se poate face prin mai multe apeluri de sistem: `open`, `creat`, `mknod`, `mkdir`. Toate specifică un path. Dacă procesul care le execută are dreptul de scriere în penultima componentă a path-ului (care este directorul în care se crează fişierul) atunci fişierul este creat, alocîndu-se un i-nod liber şi un link în acel director.

Fişierele nu pot fi şterse în Unix! În Unix se pot doar şterge link-uri. Singurul apel de sistem pentru aşa ceva este `unlink`. Cînd este şters un link, i-nodului spre care acesta punctează i se decrementează numărul de referinţe. Dacă acest număr devine 0, înseamnă că acest fişier nu mai este legat în nici un director. *Dacă* acest fişier nu este în acea clipă deschis de nici un proces, atunci toate blocurile de date pe care el le posedă sunt trecute pe lista de blocuri libere, iar i-nodul lui este eliberat. (Dacă este deschis, atunci el va fi eliberat numai cînd toate procesele vor închide fişierul).

Nu faceţi confuzie între operaţia de ştergere a unui link, care are nevoie de drept de scriere în directorul din care se şterge link-ul, şi operaţia de *trunchiere* a unui fişier (la lungime zero, de pildă) care are nevoie de drept de scriere *doar* în acel fişier.

```
$ ls -la
drwxr-x--x   7 mihai    users        1024 Jul 29 21:46 ./
drwxr-xr-x   6 root     root         1024 Jul 26 22:50 ../
-rw-rw-rw-   1 mihai    users        3735 Jul 29 19:18 ss
```

Un alt utilizator decît `mihai` *nu* poate face `rm ss`, dar poate face `cp /dev/null ss`, cu care îl şterge!

În fine, o ultimă operaţie pe fişiere pe care o discutăm este cea de legare. Se poate crea o legătură nouă la un fişier existent, cu apelul de sistem `link` (sau comanda shell-ului `ln`, care se foloseşte de acest apel).

```
$ ln /bin/bash ./bb
```

procedează astfel:

1. află i-nodul lui `/bin/bash`;

2. crează în directorul . un link cu numele `bb` (dacă are drept de scriere);

3. pune în acest link i-nodul aflat mai-nainte.

În mod normal numai "super-user-ul" poate face legături la un director.

Pentru că o legătură conţine un număr de i-nod, iar fiecare partiţie are propria ei numerotare independentă de a celorlalte pentru i-noduri, nu se pot face legături pe o partiţie care să arate la un fişier de pe alta. Pentru asta au fost inventate legăturile simbolice, despre care vom vorbi altădată.

# 7 Programul `fsck`

Un sistem de fişiere se poate strica din felurite motive (cel mai comun fiind că se stinge calculatorul printr-o procedură brutală şi anumite informaţii nu sunt salvate din cache). Ca un filesystem să fie corect, anumite relaţii trebuie să fie adevărate între componentele sale. De exemplu un bloc nu trebuie să fie simultan în două fişiere.

Un program sofisticat este oferit pentru a verifica integritatea unui sistem de fişiere. `fsck` (File System ChecK) este rulat de obicei la fiecare pornire a calculatorului şi încearcă să determine toate discrepanţele care arată că un sistem de fişiere nu este integru, şi să le repare cu minimum de alteraţii ghicind ce s-a stricat de fapt.

Acest program *nu* lucrează cu fişiere şi directoare, ci analizează direct partiţia la nivel de bloc. El nu trebuie să fie niciodată executat cînd partiţia este "montată" pentru că s-ar putea ca fişierele să fie modificate de procesele care rulează în timp ce sunt verificate, deci verificarea să iasă prost deşi totul este în regulă.

# 8 Un experiment

Ca să verificaţi că aţi înţeles cum funcţionează sistemul de fişiere din Unix (deşi multe lucruri au fost lăsate în suspensie, v-am dat toate informaţiile necesare), încercaţi să rezolvaţi următorul exerciţiu pe un Unix care vă oferă mai multe "ferestre":

1. Deschideţi cîte un shell în două ferestre;

2. În prima tastaţi:

   ```
   $ sh
   $ cd
   $ mkdir bb
   $ cd bb
   $ ls
   $ pwd
   ```

3. În a doua fereastră tastaţi:

```
$ cd
$ rm -r bb
$ ls -a
```

4. Reveniţi în prima şi tastaţi:

```
$ ls
$ pwd
$ cd ..
$ cd /
```

Explicaţi ce se întîmplă.

## Răspuns:

- un shell (sh) face din `bb` directorul sau curent.
- alt shell şterge toate link-urile spre `bb` (cu `rm -r`)
- procesul `sh` nu poate face
  - `ls` pentru că acesta lucrează cu link-ul `.` din directorul curent (fost `bb`), care nu mai există
  - `pwd` pentru că acesta încearcă să urmeze link-ul `..` din `bb`, care nu mai există
  - `cd ..` nu merge din aceleaşi motive