Informatiile conținute de i-noduri (continuare)

- **data creării**
- **data ultimei deschideri**
- **data ultimei modificări a i-nodului**
- **data ultimei modificări a conținutului**
- **numărul de octeți conținuți în fișier**

Observație: fișierele **makefile** sunt fișiere text; conțin numele obiectelor de construit și lista dependințelor, precum și comanda de construcție.

Fişierele speciale de tip link simbolic

Ne amintim că o legătură fizică se creează cu ajutorul unei comenzi de forma:

ln sursă destinație

unde **sursă** reprezintă un fișier existent, iar **destinație** este directorul (și noul nume al legăturii) în care apare o nouă intrare care are același număr de i-nor precum **sursă.**

Un link simbolic se creează cu o comandă de forma:

ln -s sursă destinație

Diferența este că această comandă creează un i-nod nou care nu are informație proprie; informația din **destinație** reprezintă o trimitere la fișierul **sursă.**

Linkurile simbolice, spre deosebire de legăturile fizice, pot fi folosite și pentru directoare. Un dezavantaj al linkurilor simbolice ar fi că în cazul ștergerii fișierului **sursă**, **destinație** nu este șters. Dacă se încearcă deschiderea fișierului **destinație** vom primi eroarea 'File Not Found' pentru **destinație**,nu **sursă.**

Structura sistemului de fișiere în memorie

Un atribut al proceselor este lista fișierelor deschise. Aceasta este reținută într-o *tabelă de descriptori proprie fiecărui proces*. Procesul are acces doar la numărul intrării în tabela de descriptori. Fiecare proces are la creare următoarele fișiere deschise: **stdin (0); stdout(1); stderr(2)**. Fiecare descriptor pointează către o intrare într-o altă tabelă, cea de fișiere deschise în memorie. În această *tabelă de descriptori a fișierelor deschise în memorie* se memorează poziția curentă în fișier.

Observatie: Se reține o intrare nouă și un descriptor nou pentru fiecare deschidere de fișier, chiar dacă este vorba de același fișier.

O a treia tabelă este *tabela de i-noduri în memorie*. Fiecare fișier deschis are o singură intrare în această tabelă. Fiecare intrare din tabela de descriptori a fișierelor deschise în memorie pointează către o intrare din tabela de i-noduri în memorie. Mai departe, fiecărui i-nod îi corespunde o tabelă numită t*abela de blocaj asupra i-nodului*.

Apeluri sistem pentru gestiunea fișierelor

struct stat

Unul din câmpurile acestei structuri ce reține atributele unui i-nod este **int st_mode.** In acesta se memorează pe 4 biți tipul fișierului, pe 9 biți drepturile de acces și pe 3 biți drepturile speciale (set_uid, set_gid, sticky bit); există constante speciale cu care se poate testa conținutul.

ex.: if (s.st_mode & _S_IFDIR)
{ ...*este director...* }
if (S_ISDIR (s.st_mode))
{ ...*este director...* }

Asemenea constante există și pentru drepturile de acces și biții speciali.

- **int stat (char * nume, struct stat * s)**
  - încarcă în structura desemnată de **s** atributele i-nodului fișierului **nume**
  - dă eroare, dacă nu există fișierul **nume**
  - o problemă a acestui apel este că dacă **nume** este un fișier de tip link simbolic, atunci în **s** se trec atributele fișierului **sursă**

- **int lstat (char * nume, struct stat * s)**
  - funcționează la fel ca **stat**, dar dacă **nume** este un fișier de tip link simbolic, atunci în **s** sunt trecute atributele acestui fișier, nu ale fișierului sursă.

Exercițiu: Simulați comanda **ls -l**

Apeluri sistem pentru exploatarea fișierelor

Există două metode pentru exploatarea fișierelor în C sub UNIX: utilizând biblioteca standard C sau (și – mai rar) utilizând apeluri sistem de nivel jos.

Biblioteca sistem UNIX pentru exploatarea fișierelor

- **int open (char * nume , int mod , int dr)**
  - apelul returnează -1 în caz de eroare, iar în caz de succes întoarce indicele în tabela de descriptori;
  - **nume** reprezintă numele fișierului
  - **mod** reprezintă accesul la fișier și este construit prin disjuncție pe biți din constantele:
    - una și numai una din: **O_RDONLY, O_WRONLY, O_RDWR**
    - se pot alege opțional și: - **O_CREAT** (dacă nu există fișierul nu dă eroare și îl creează)
      - **O_EXCR** (se folosește numai în prezența lui O_CREAT; dă eroare dacă fișierul deja există)
      - **O_TRUNC** (trunchiere de fișier, dacă fișierul există, conținutul este șters)
      - **O_APPEND** (toate scrierile se fac la sfârșit)
      - **O_SYNC** (scriere sincronizată, altfel se scrie întâi în cache)
      - **O_NONBLOCK , O_NDELAY** (eventualele operații de

citire/scriere care ar duce la un apel blocant, să nu fie operații blocante)

- **dr** reprezintă drepturile de acces și are efect doar când **open** creează fișier nou, dacă lipsește, se consideră drepturile implicite; este construit prin disjuncție pe biți din constante din structura **stat**

Observatie: Nu se poate anticipa că prin apelul **open** se va obtine cel mai mic descriptor liber.

Observatie: **dr** poate lipsi, deci apelul poate avea 2 sau 3 parametrii

- **int close (int fd)**
  - **fd** este un descriptor de fișier valid (are același rol cu un file pointer)
  - apelul eșuează doar când **fd** este invalid

- **int read (int fd, void * p, int dim)**
  - **fd** este un descriptor de fișier valid
  - **p** este un pointer alocat unde se vor pune datele citite
  - **dim** reprezintă numărul maxim de octeți ce vor fi citiți
  - returneaza -1 în caz de eroare sau numărul de octeți citiți

- **int write (int fd, void * p, int dim)**
  - scrie **dim** octeți de la adresa **p** în fișierul cu descriptoul **fd**
  - returnează numărul de octeți scriși efectiv

Un dezavantaj al bibliotecii sistem UNIX pentru exploatarea fișierelor este că nu avem la dispoziție scrieri/citiri formatate. De aceea existe metode de trecere dintr-o bibliotecă în alta prin apelurile:

- **FILE * fdopen (int fd , char * mod)**
- **int fileno (FILE * fp)**

Duplicarea descriptorilor

- **int dup (int fd)**
  - **fd** este un descriptor valid
  - în caz de succes creează o nouă intrare în tabela de descriptori a procesului, care va pointa către aceeași intrare în tabela de descriptori a fișierelor deschise în memorie (prin urmare partajează poziția curentă)

Observatie: Duplicarea se face în cel mai mic descriptor liber.
Aplicație: Redirectarea fișierelor standard

< stdin | > stdout | >> stdout

ex.:

$ > titi este echivalentă cu

```
int fd;
fd = open ( ”titi” , O_WRONLY | O_CREAT | O_TRUNC );
close (1);
dup (fd);
close (fd);
```