

2

Sistemul de fișiere – interfața de user-space

2 martie 2010

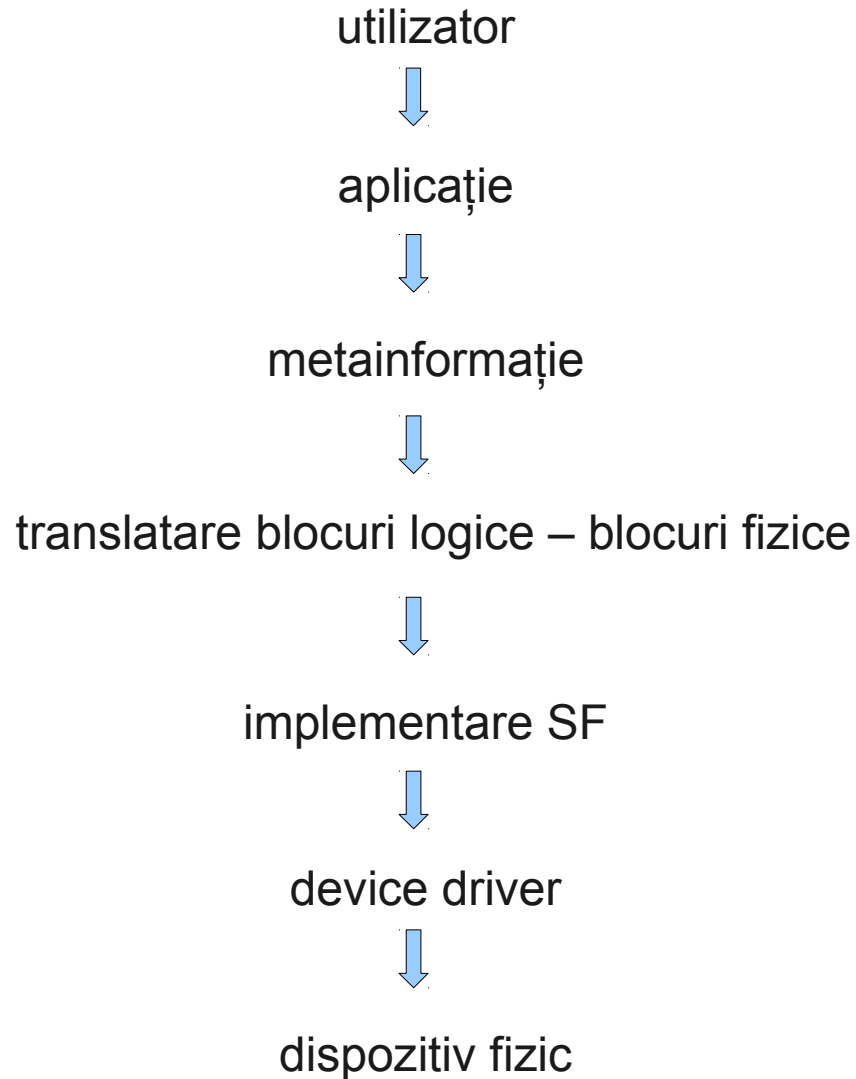
- OSC
 - Capitolul 10 – File-System Interface
 - Capitolul 11, Secțiunea 11.1 – File-System Structure
- MOS
 - Capitolul 6 – File System Implementation
 - secțiunile 1 și 2
- LSP – Capitolul 2 (parte programatică)
- WSP – Capitolul 2 (parte programatică)

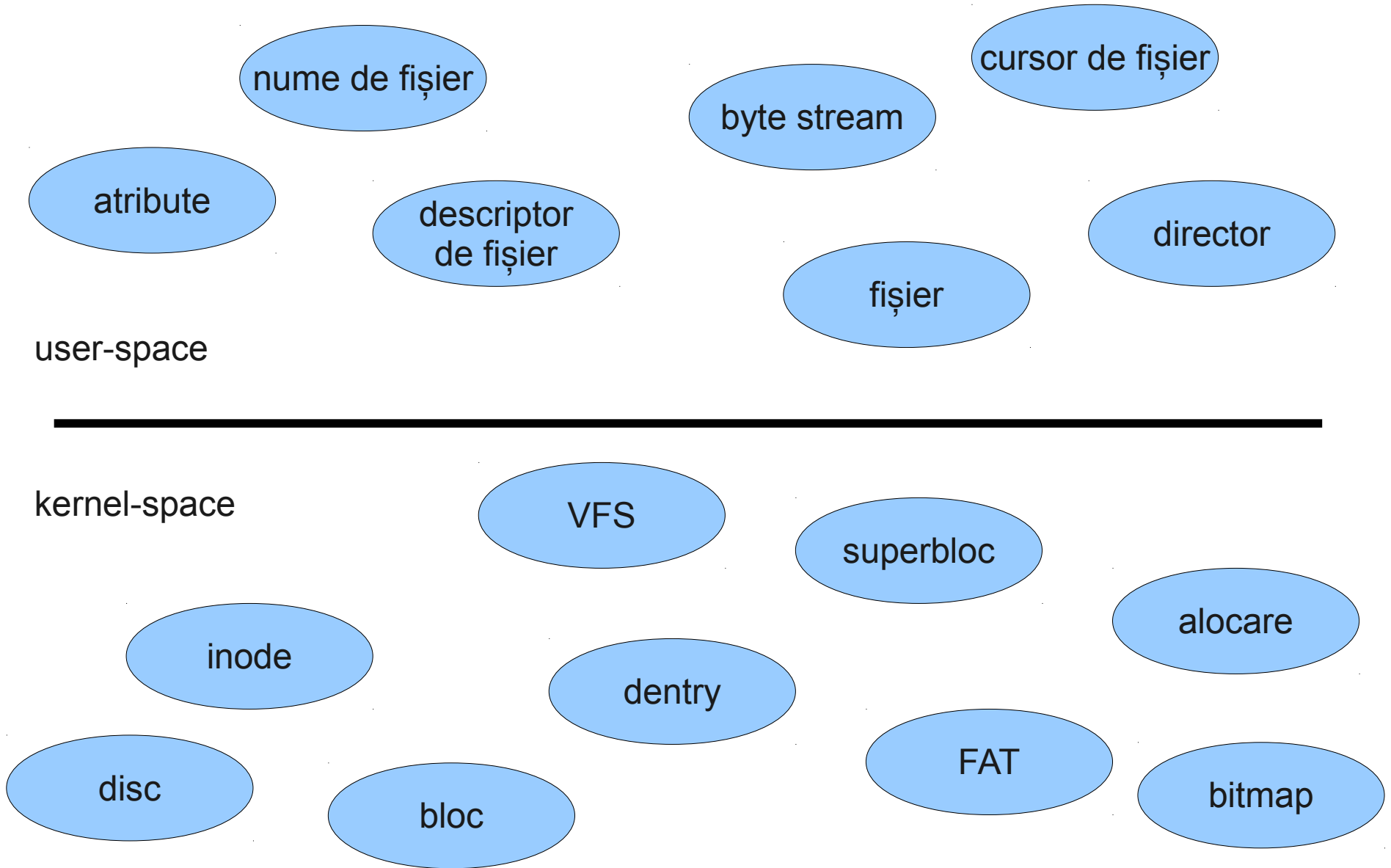
- Sisteme de fișiere: definiții, concepte
- Fișiere. Atribute ale fișierelor
- Operații cu fișiere
- API pentru lucrul cu fișiere
- Directoare. Operații cu directoare
- Tipuri speciale de fișiere
- Montarea sistemelor de fișiere

- mod de organizare a informației (persistente) pe un suport fizic
- scalabilitate și acces facil
- unitatea de bază este fișierul
- din punct de vedere al utilizatorului:
 - o interfață de acces la informație formată din directoare și fișiere (în general ierarhică)
- din punct de vedere al sistemului de operare
 - structuri de date și algoritmi pentru organizarea informației pe disc

- necesitatea unui suport persistent
- suport de dimensiune mare
- acces facil la informație (prin intermediul fișierelor)
- sistemul de fișiere mapează fișierele pe suportul fizic

- ext3
- ReiserFS
- HFS
- NTFS
- FAT32
- ISO9660
- UDF

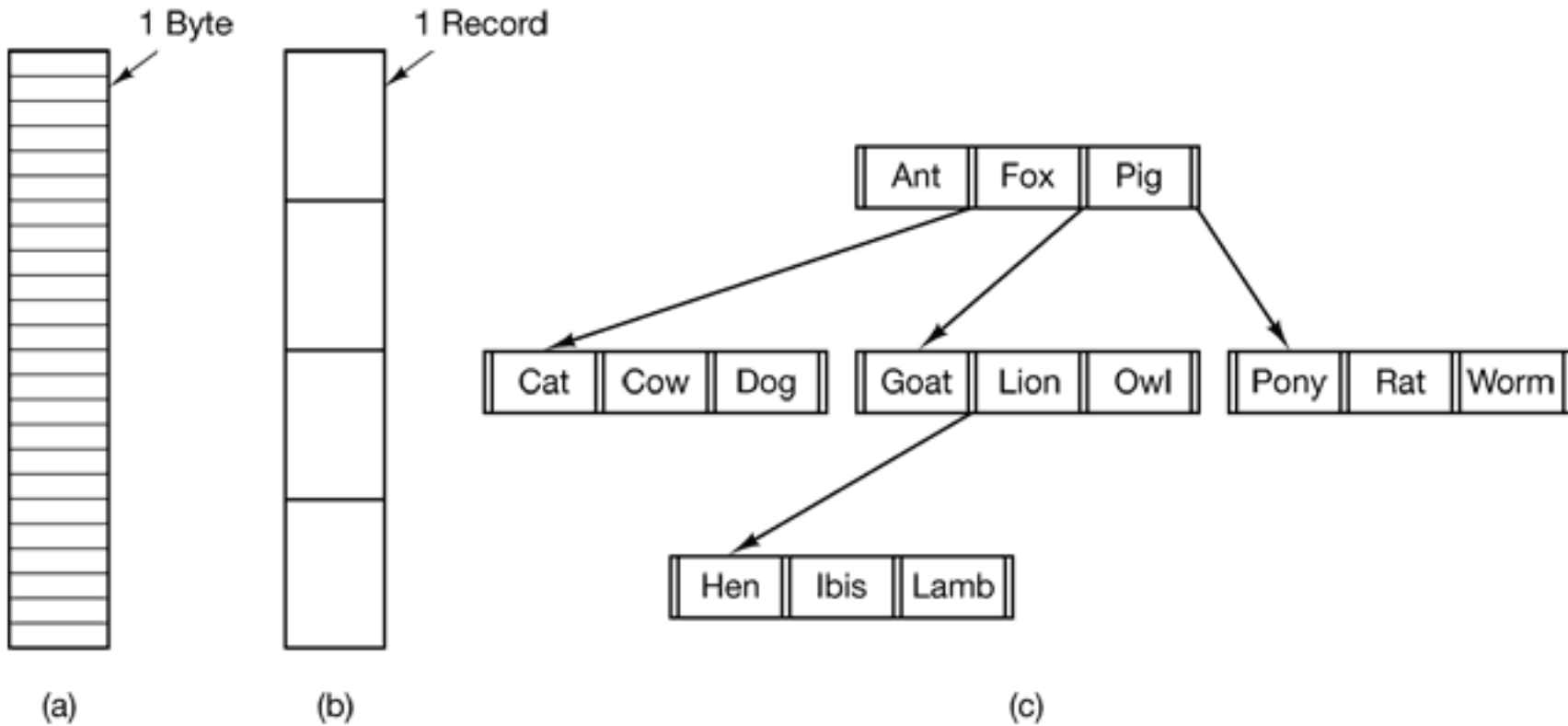




- unitatea de bază a sistemului de fișiere
- în lumea SO există două abstracții fundamentale
 - fișierul – abstractizează informația/datele
 - procesul – abstractizează efectuarea de acțiuni
- în sistemele Unix-like, totul este un fișier
 - de fapt, aproape totul (Plan9 este foarte aproape)
 - aceleași attribute și tipuri de operații cu fișiere sunt corelate cu diverse interfețe ale sistemului de operare

- fișiere simple – regular files
- directoare – directories
- legături (simbolice) – hard/soft links
- dispozitive tip caracter – char devices
- dispozitive tip bloc – block devices
- pipes/FIFOS
- socket-uri UNIX
- cum aflați tipul unui fișier în UNIX?

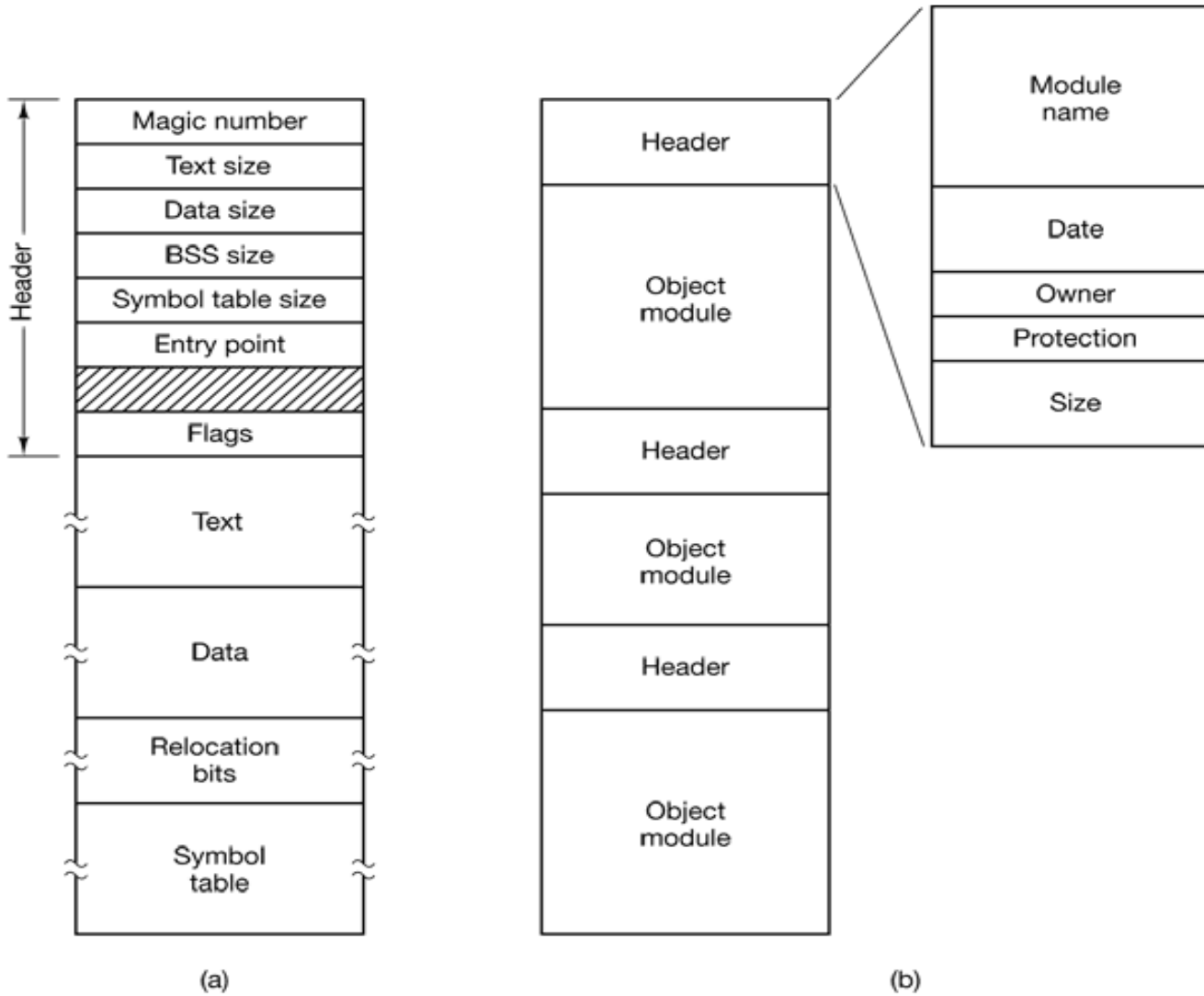
- binare, text
- extensie
- organizare/structură
 - record
 - șir de octeți
- attribute
 - drepturi
 - dimensiune
 - utilizator
 - timestamps



a) byte-stream

b) record file

c) tree



a) fișier executabil

b) arhivă

- creare
- deschidere
- citire
- scriere
- poziționare în cadrul fișierului
- trunchiere
- închidere
- ștergere

- shell

```
touch /path/to/file
```

- comenzile care scriu într-un fișier îl și creează

- ISO/ANSI C

```
FILE *f = fopen ("/path/to/file", "rw");
```

- Unix

```
int fd = open ("/path/to/file", O_CREAT | O_EXCL, 0644);
```

- Windows

```
HANDLE fileHandle = CreateFile ("/path/to/file", ..., CREATE_NEW);
```

- ISO/ANSI C

```
FILE *f = fopen("/path/to/file", "rt");
```

- Unix

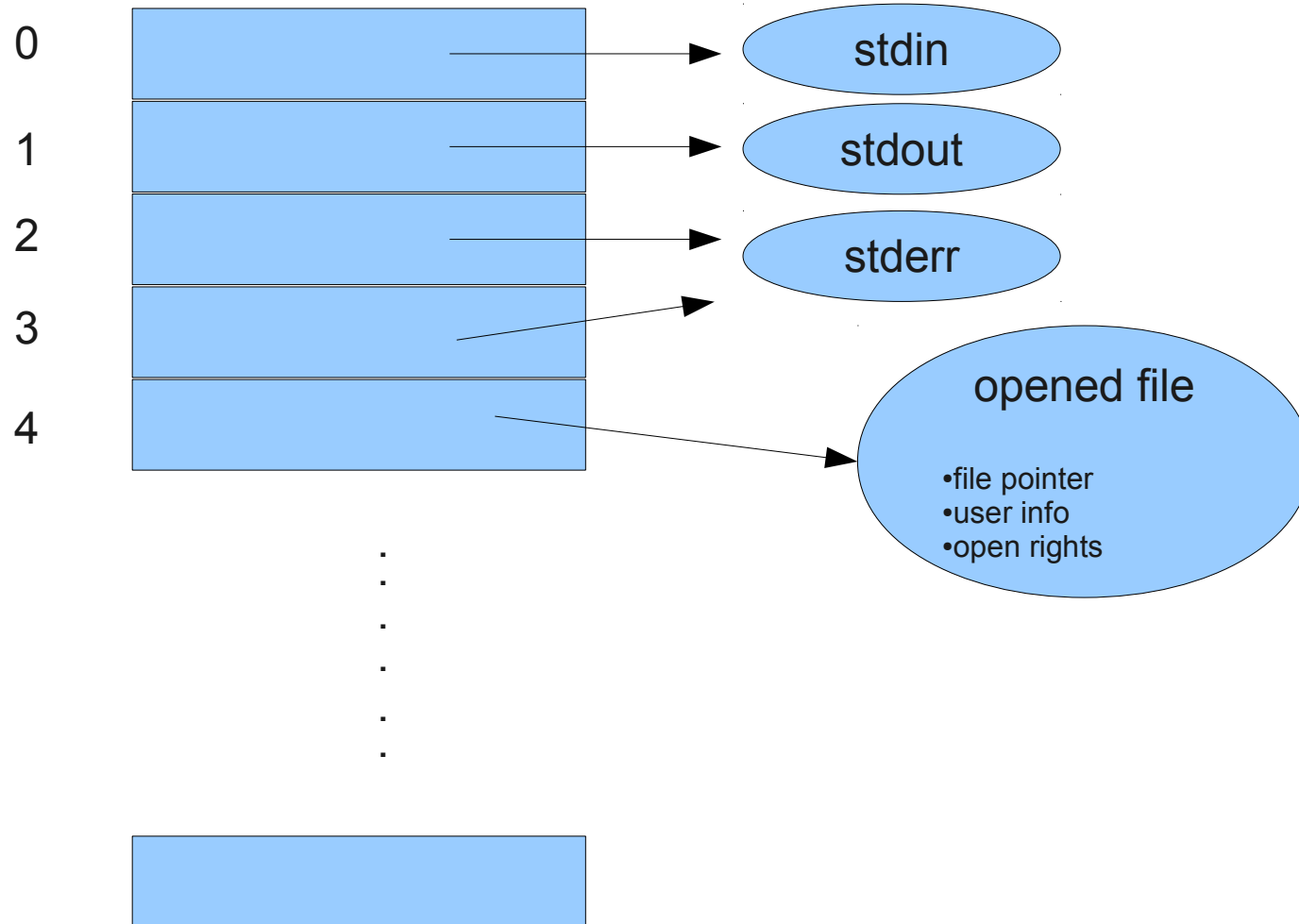
```
int fd = open("/path/to/file", O_RDONLY);
```

- Windows

```
HANDLE f = CreateFile("/path/to/file", ...,  
    OPEN_EXISTING);
```


- identificare
 - handle, descriptor
 - de ce nu nume?
- file pointer, cursor de fișier
- drepturi de deschidere
- contor de utilizare (file-open count)

- un întreg ce identifică o instanță de fișier deschisă în cadrul unui proces
- care este asocierea între un descriptor și un fișier? (unu la unu, mai multe la mai multe?)
- fiecare proces are o tabelă de descriptori de fișier
- 0, 1, 2 sunt descriptori speciali – stdin, stdout, stderr
- FILE (ISO C)
- HANDLE (Win API)



```
/* /usr/include/libio.h */  
  
struct _IO_FILE {  
    [...]  
    int _fileno;  
    [...]  
}  
  
/* /usr/include/stdio.h */  
  
typedef struct _IO_FILE FILE;
```

- stocarea informației într-un buffer
- avansul cursorului de fișier
- ISO C

```
n_recs = fread(buf, RECSZ, N_RECS, fin);  
/* se foloseste cu feof(), ferror() */
```

- Unix

```
n_read = read(fd, buf, BUFSIZ);  
/* se foloseste obligatoriu intr-un while */
```

- Windows

```
ReadFile(fHandle, buf, bytesToRead, &bytesRead, NULL);
```

- scrierea informației dintr-un buffer
- avansul cursorului de fișier
- ISO C

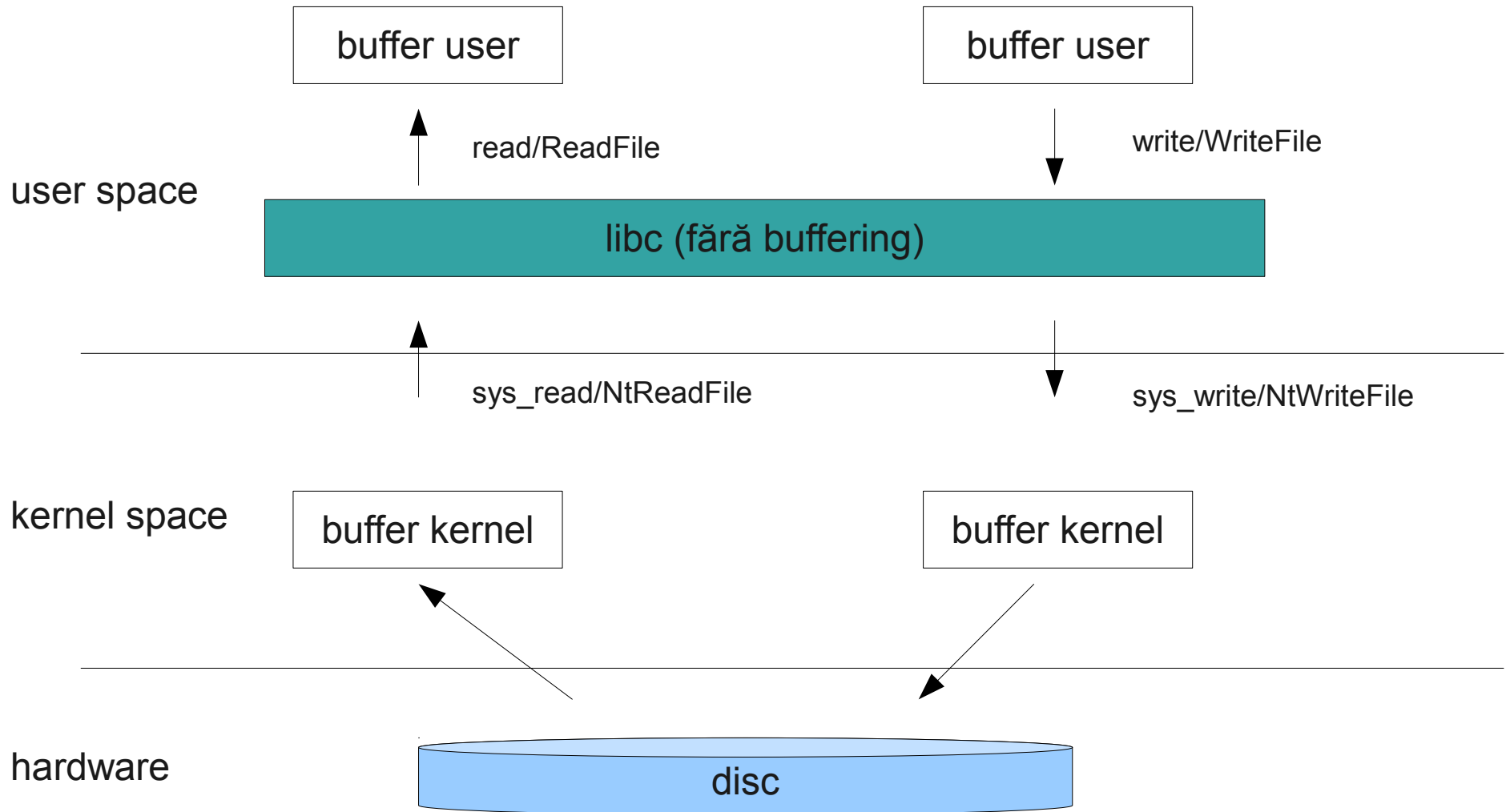
```
n_recs = fwrite(buf, RECSZ, N_RECS, fin);  
/* scrie garantat, sau eroare de scriere */
```

- Unix

```
n_written = write(fd, buf, BUFSIZ);  
/* se foloseste obligatoriu intr-un while */
```

- Windows

```
WriteFile(fHandle, buf, bytesToWrite, &bytesWritten, NULL);
```



- modificare la read și write

- inițializare la deschidere

- modificare:

- ISO C

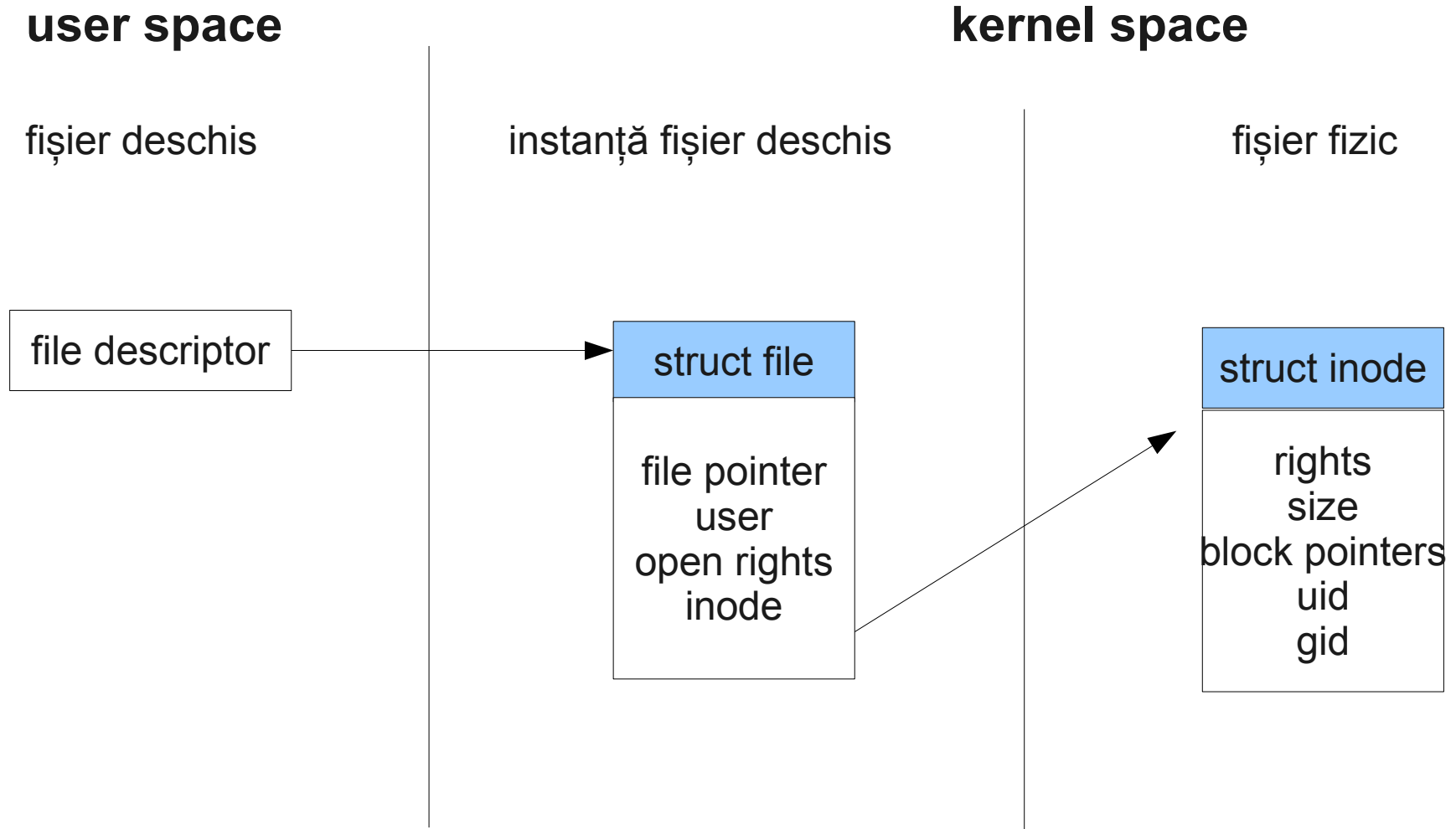
```
fseek(f, offset, SEEK_SET);
```

- Unix

```
lseek(fd, offset, SEEK_END);
```

- Windows

```
SetFilePointer(hFile, distanceToMove, NULL, FILE_BEGIN);
```

- utilizatorul recunoaște fișierul prin nume
- utilizatorul folosește un descriptor
- descriptorul indexează tabela de descriptori
- un element al tablei este un pointer la o structură dinamică (instanță de fișier deschis)
- structura dinamică are referința la structura ce descrie fișierul fizic
- ce asociere există între cele trei structuri? (file descriptor, structură dinamică, fișier fizic)

- eliminarea conținutului fișierului
- cursorul de fișier este poziționat pe 0

- un fișier poate fi deschis și trunchiat

```
open("/path/to/file", O_RDWR | O_TRUNC);  
truncate("/path/to/file", size);
```

- un fișier se poate trunchia după deschidere

```
ftruncate(fd, size);  
SetEndOfFile(hFile);
```

- se pierde accesul la fișier
- este eliminată intrarea din tabela de descriptori

```
close(fd);
```

```
CloseHandle(fHandle);
```

- duplicarea unui descriptor în alt descriptor
- cei doi descriptori lucrează asupra aceluiași fișier

```
newfd = dup(oldfd);  
dup2(oldfd, newfd);
```

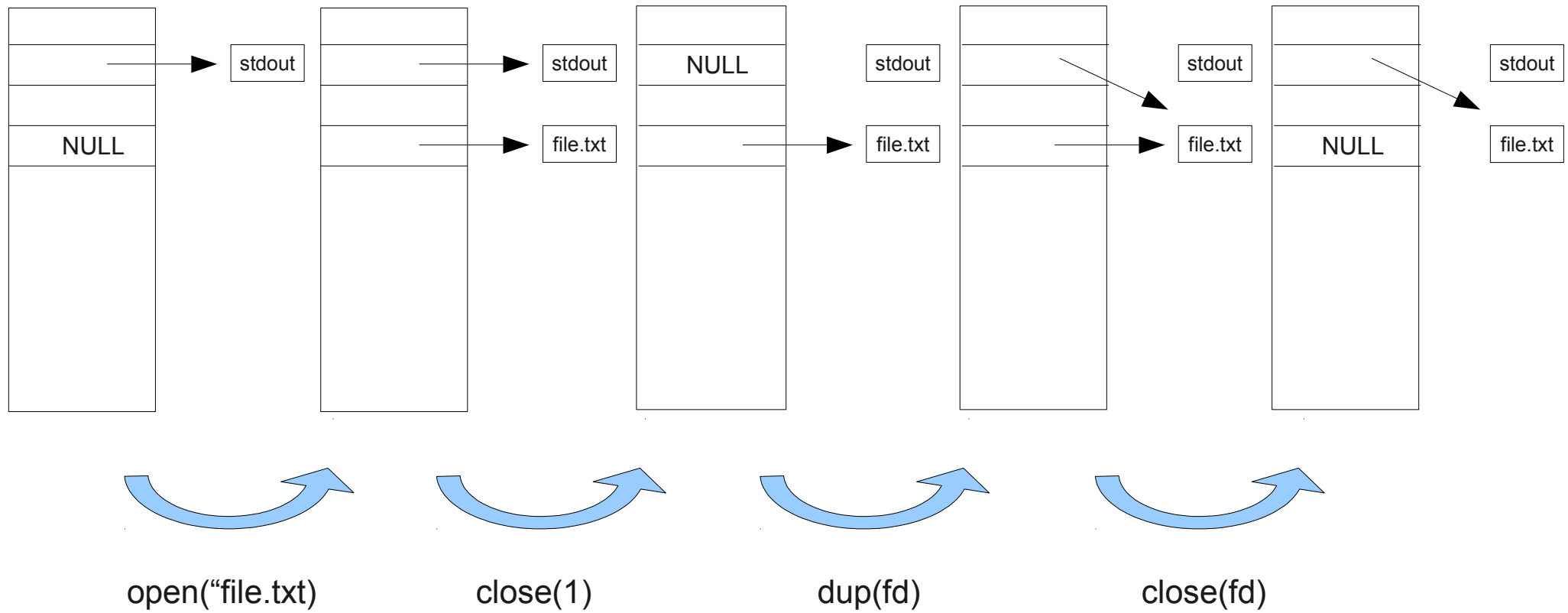
```
DuplicateHandle(...);  
SetStdHandle(...);
```

- în shell

```
echo "mesaj" > file_out.txt  
grep "cuvant" < file_in.txt
```

- programatic

```
fd = open ("file_out");  
close (STDOUT_FILENO);  
dup (fd);  
close (fd);
```



- eliminarea fișierului din sistemul de fișiere

- shell

```
rm file.txt
```

- ISO/C

```
remove("file.txt");
```

- UNIX

```
unlink("file.txt");
```

- Windows

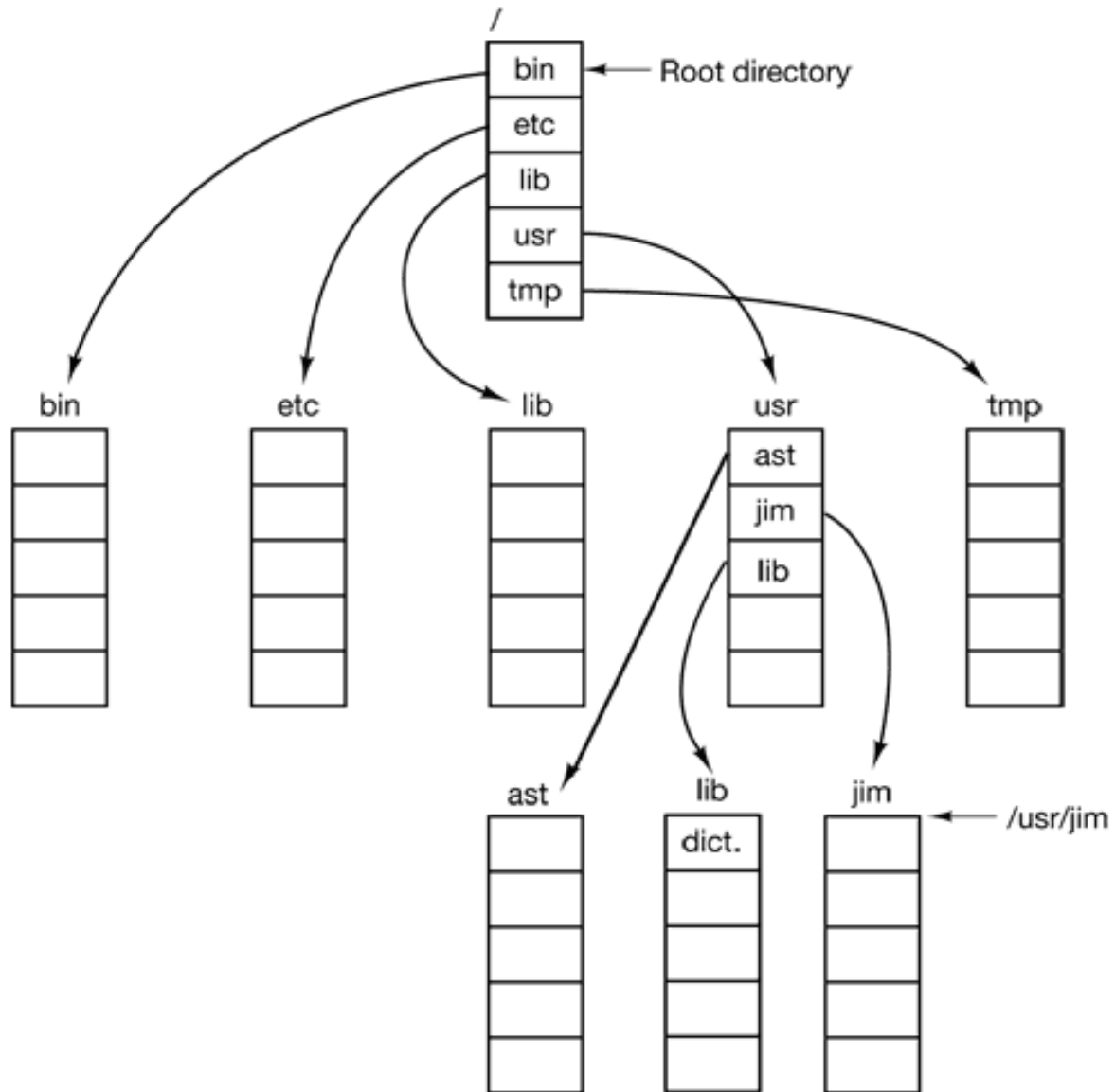
```
DeleteFile("file.txt");
```


- colecție de intrări în sistemul de fișiere
 - folder/catalog

- de ce directoare?
 - organizarea fișierelor
 - acces facil al utilizatorului

- organizarea cea mai întâlnită este cea ierarhică
 - generalizat – graf orientat aciclic (DAG)

- Înlănțuire de intrări în sistemul de fișiere
- orice sistem de fișiere are un director rădăcină (/ pe Unix)
- separator de cale (/ pe Unix, \ pe Windows)
- cale absolută
- cale relativă
- intrările . și .. sunt speciale
 - . (dot) - referință către directorul curent
 - .. (dot-dot) - referință către directorul părinte



- crearea/ștergerea unui director (mkdir, rmdir)
- crearea unei intrări (touch, mkdir, ln, mknod)
- eliminarea unei intrări (rm, rmdir, unlink)
- listarea conținutului (ls)
- parcurgerea unei căi (cd)
- redenumirea unei intrări (mv)

- UNIX
 - DIR *, struct dirent
 - mkdir, chdir, rmdir
 - opendir, closedir, readdir, rewinddir

- Windows
 - HANDLE
 - CreateDirectory, RemoveDirectory, SetCurrentDirectory
 - FindFirstFile, FindNextFile, FindClose

- legături (links) (ln, link)
- char device (mknod, mknod)
- block device (mknod, mknod)
- FIFO (mknod/mkfifo, mknod/mkfifo)
- socketi UNIX (N/A, socket/mknod)

- folosite pentru referirea unui fișier deja existent
- symlink (soft link)
- hard link
- soft link pot referi intrări din alt sistem de fișiere

```
$ ls -l /usr/bin/cc
```

```
lrwxrwxrwx 1 root root 20 Oct  6  2005 /usr/bin/cc -> /etc/alternatives/cc
```

```
lrwxrwxrwx 1 root root 22 Apr 22  2007 /etc/alternatives/cc -> /usr/bin/gcc
```

- de tip caracter (char device)

- parcurgere secvențială
- asociate cu dispozitive seriale, mouse, tastatură

```
$ ls -l /dev/ttyS0
```

```
crw-rw---- 1 root dialout 4, 64 Feb 26 2005 /dev/ttyS0
```

- de tip bloc (block device)

- acces aleator
- citire în blocuri; discuri, CD-ROM, USB Flash Drive

```
$ ls -l /dev/sda
```

```
brw-rw---- 1 root disk 8, 0 Feb 26 2005 /dev/sda
```


- socketii Unix sunt o formă de comunicare inter-proces – comunicare locală
- au asociată intrare în sistemul de fișiere

```
# ls -l /var/run/cups/cups.sock  
srwxrwxrwx 1 root root 0 2008-03-03 11:10  
/var/run/cups/cups.sock
```
- FIFO (pipe cu nume) sunt mecanisme de bază de comunicare inter-proces
- un proces scrie, alt proces citește

```
$ ls -l mypipe  
prw-r--r-- 1 razvan razvan 0 2008-03-03 21:14 mypipe
```

- Standard I/O – fopen, fread, fwrite, fclose
 - buffered I/O
 - double-buffering problem
 - acces high-level pentru lucrul cu șiruri
- Low Level I/O – open/CreateFile, read/ReadFile
 - wrappare peste apeluri de sistem
 - cea mai mare flexibilitate
 - nu sunt portabile

- la crearea fișierului
- la deschiderea fișierului

- în Unix

- user, group, others
- read, write, execute

```
chmod("/path/to/file", 0644); /* drepturi în octal */
```

- în Windows

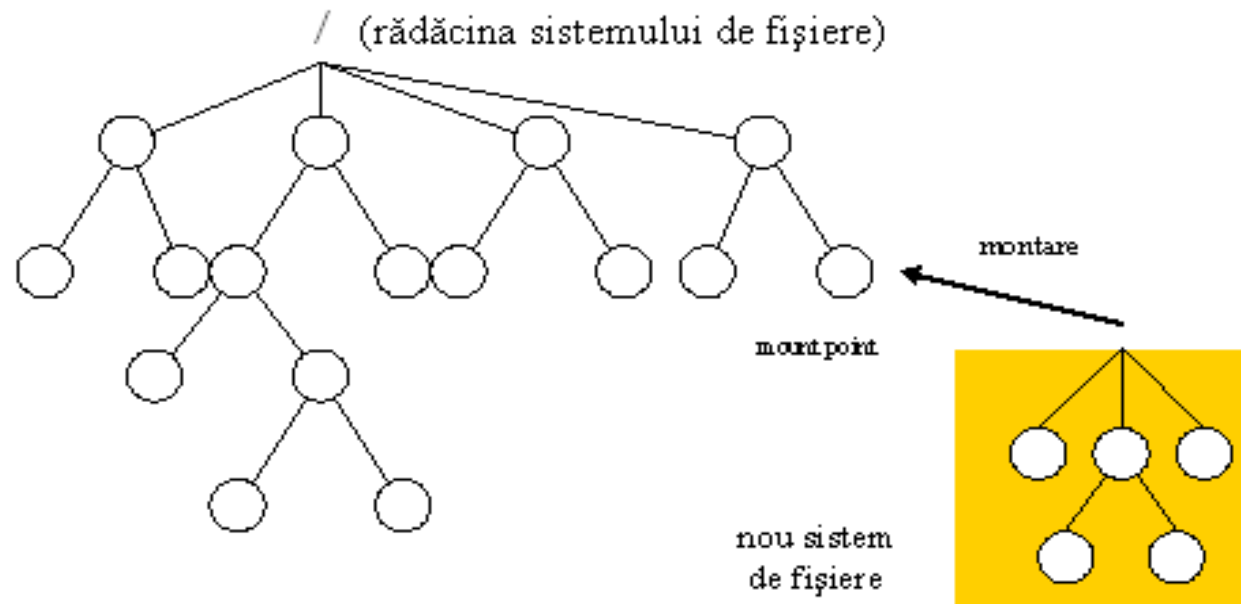
- citire, scriere, listare, traversare, creare sub-fișier, etc.

```
SetNamedSecurityInfo(...);
```

- atașarea unui punct de intrare în sistemul local de fișiere (mount point)

```
mount("/dev/hda5", "/mnt/hda5", "ntfs", ...)
```

```
SetVolumeMountPoint("C:\\mycd\\", "D:\\");
```



- sistemele de fișiere virtuale nu au un suport fizic
 - procfs montat în /proc
 - sysfs montat în /sys
 - ce suport au?
- există și intrări de fișiere specializate (fișiere fără suport fizic – de obicei char device-uri)
 - /dev/null
 - /dev/zero
 - /dev/urandom

- sistem de fișiere
- fișier
- tipuri de fișiere
- byte-stream
- descriptor de fișier
- cursor de fișier
- tabelă de descriptori
- redirectare
- director
- cale
- ierarhie
- rădăcină
- fișiere speciale
- standard I/O
- low-level I/O
- drepturi de acces
- montare SF

- Ce afișează următoarea secvență de cod?

```
int fd1 = open("file.txt"), fd2 = open("file.txt"),  
    fd3 = dup(fd1), off;  
  
lseek(fd1, 100, SEEK_CUR);  
lseek(fd2, 100, SEEK_CUR);  
off = lseek(fd3, 100, SEEK_CUR);  
printf("%d\n", off);
```

- Care sunt asocierile corecte (unu la unu, unu la mai multe, mai multe la unul, mai multe la mai multe) între următoarele concepte?
 - descriptor de fișier – fișier (pe disc)
 - descriptor de fișier - cursor de fișier
 - proces - descriptor de fișier
 - proces - tabelă de descriptori

- Pentru fiecare dintre opțiunile de creare/deschidere a fișierelor în Windows, care este formatul echivalent Linux?

Windows

OPEN_ALWAYS
OPEN_EXISTING
CREATE_ALWAYS
CREATE_NEW
TRUNCATE_EXISTING

Linux

O_CREAT
O_TRUNC
O_EXCL

<p>CREATE_ALWAYS 2</p>	<p>Creates a new file, always.</p> <p>If the specified file exists and is writable, the function overwrites the file, the function succeeds, and last-error code is set to ERROR_ALREADY_EXISTS (183).</p> <p>If the specified file does not exist and is a valid path, a new file is created, the function succeeds, and the last-error code is set to zero.</p> <p>For more information, see the Remarks section of this topic.</p>
<p>CREATE_NEW 1</p>	<p>Creates a new file, only if it does not already exist.</p> <p>If the specified file exists, the function fails and the last-error code is set to ERROR_FILE_EXISTS (80).</p> <p>If the specified file does not exist and is a valid path to a writable location, a new file is created.</p>
<p>OPEN_ALWAYS 4</p>	<p>Opens a file, always.</p> <p>If the specified file exists, the function succeeds and the last-error code is set to ERROR_ALREADY_EXISTS (183).</p> <p>If the specified file does not exist and is a valid path to a writable location, the function creates a file and the last-error code is set to zero.</p>
<p>OPEN_EXISTING 3</p>	<p>Opens a file or device, only if it exists.</p> <p>If the specified file or device does not exist, the function fails and the last-error code is set to ERROR_FILE_NOT_FOUND (2).</p> <p>For more information about devices, see the Remarks section.</p>
<p>TRUNCATE_EXISTING 5</p>	<p>Opens a file and truncates it so that its size is zero bytes, only if it exists.</p> <p>If the specified file does not exist, the function fails and the last-error code is set to ERROR_FILE_NOT_FOUND (2).</p> <p>The calling process must open the file with the GENERIC_WRITE bit set as part of the <i>dwDesiredAccess</i> parameter.</p>

