# 3 Comunicaţii între sisteme de operare şi SO distribuite

## *3.1 Adresări în Internet*

### 3.1.1 Adrese IP şi clase de adrese

O adresă IP, folosită de către pachetul TCP/IP, este un număr întreg şi pozitiv, reprezentat pe 32 de biţi. Deci există maximum $2^{32}$ astfel de adrese.
Structura unei adrese IP; clase

Structura generală a unei astfel de adrese este dată în fig. 1.18.

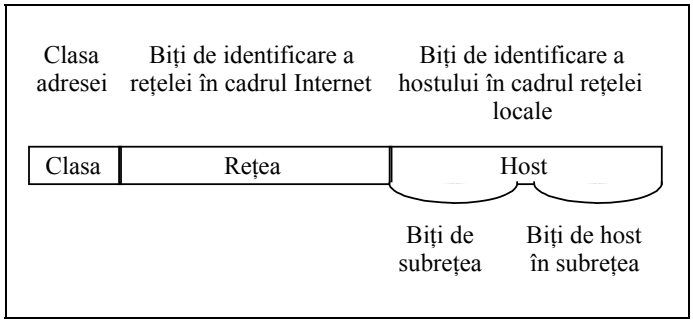



Figura 3.1 Structura unei adrese IP

Din cei 32 de biţi, numărul de biţi afectaţi fiecărui câmp din adresă depinde de clasa adresei. În prezent există 3 clase de adrese: ***clasa A***, ***clasa B*** şi ***clasa C***, prezentate în figurile 1.19, 1.20 şi 1.21. Deocamdată nefolosită, este rezervată şi o a patra, ***clasa D***.

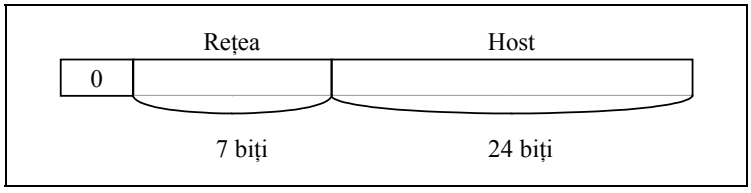



Figura 3.2 Structura adresei de clasă A

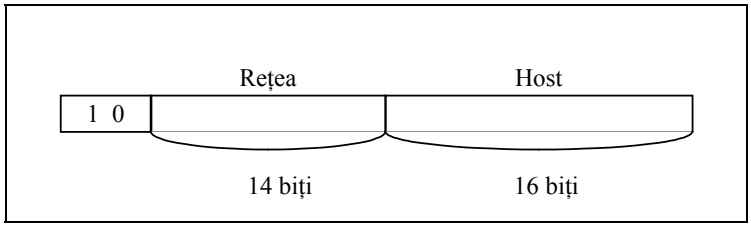



Figura 3.3 Structura unei adrese de clasă B

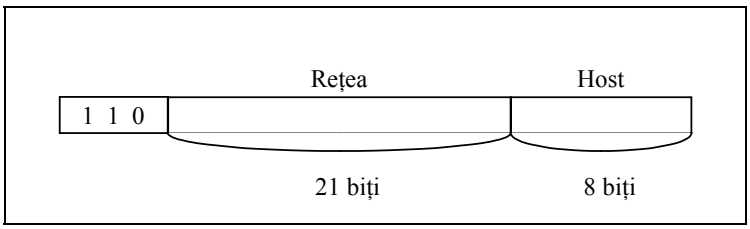


Figura 3.4 Structura unei adrese de clasă C

Adresele de clasă D încep cu şirul de patru biți `1110`, restul de 28 de biți fiind rezervați.
Se poate uşor vedea că există un număr finit de adrese posibile din fiecare clasă şi un număr finit de hosturi în cadrul fiecărei rețele. Dacă se face abstracție de un număr, de altfel foarte mic, de configurații de biți care sunt rezervate în scopuri administrative, limitele maxime de adresare posibile sunt:

Clasa A, maximum $2^7$ rețele a câte maximum $2^{24}$ -1 hosturi fiecare;
Clasa B, maximum $2^{14}$ rețele a câte maximum $2^{16}$ -1 hosturi fiecare;
Clasa C, maximum $2^{21}$ rețele a câte maximum $2^8$ -1 hosturi fiecare.

Aceste limitări au început însă să amenințe funcționarea rezonabilă a comunității Internet. În [34] se arată însă că se intreprind cercetări de extindere a adreselor IP la 48 de biți sau chiar la mai mult, cu modificarea şi proliferarea unei noi versiuni de TCP/IP numită IPV6. Această extindere este necesară atât datorită creşterii exponențiale a numărului de hosturi care se conectează la Internet, cât şi datorită tendințelor de includere în Internet a comunicațiilor clasice mass-media: TV, radio, telefonie etc.

Noi ne rezumăm la prezentarea adreselor actuale. Consorțiul NIC atribuie numai clasa de adresă şi câmpul `Rețea`, lăsând dispoziția administratorilor rețelei să gestioneze câmpul `Host` şi eventual partajarea acestuia în subcâmpul de `subrețea` şi cel de `host în subrețea`, aşa cum sunt ele ilustrate în fig. 1.18.
Atribuirea unei anumite clase de adresă se face în funcție de dimensiunea acesteia, aşa cum s-a arătat mai sus. În consecință, adresele de clasă A sunt atribuite numai rețelelor speciale, de dimensiuni foarte mari. Aceste adrese au fost atribuite mai ales la primele rețele conectate la Internet. În prezent aceste adrese sunt atribuite foarte greu.
Adresele de clasă B sunt rezervate rețelelor de dimensiuni relativ mari, cu relativ multe hosturi în ele.
Majoritatea adreselor atribuite în prezent sunt cele de clasă C. În situația în care o adresă de clasă C nu este suficientă, sunt atribuite mai multe adrese de acest tip.

Reprezentarea internă a unei adrese IP este un şir de 32 de biți, plasați în patru octeți consecutivi.
Reprezentarea externă a unei astfel de adrese se face prin 4 numere întregi separate prin trei puncte. Fiecare număr indică, în ordine, valoarile celor patru octeți (deci nu valorile câmpurilor adresei!). Iată câteva exemple:

Să considerăm adresa de clasă A: "`0 1111101 00001101010010010000111`1". Grupăm în câte 8 biți şi convertim fiecare întreg binar de 8 biți într-un număr în baza 10. Se obține reprezentarea externă: "`125.13.73.15`".

Analog, adresa de clasă B: "`10 10111111011100 0000111011101110`" se scrie în reprezentarea externă: "`175.220.14.238`".

În fine, adresa de clasă C: "`110 00001111000010001010000 00100001`" se scrie în reprezentarea externă: "`193.226.40.33`".

Din exemplele de mai sus se poate desprinde uşor regula după care se poate identifica, din scrierea externă, clasa de adresă. Dacă primul număr este între 0 şi 127, atunci este vorba de o adresă de clasă A. Dacă primul număr este între 128 şi 191 atunci este vorba de o adresă de clasă B. Clasa C are primul număr între 192 şi 223, iar clasa D între 224 şi 255.

Subreţele şi măşti de reţea

Schema de adresare Internet, proiectată pentru sute de reţele, nu a anticipat proliferarea reţelelor LAN. Drept consecinţă, o primă problemă: trebuie, în mod normal, să se atribuie câte o adresă IP la fiecare staţie dintr-o reţea LAN. Aceasta impune existenţa unor tabele de rutare foarte mari şi de aici un efort imens de a întreţine aceste adrese. Evitarea acestei situaţii se face folosind conceptul de subreţea, prin intermediul căruia rutarea este distribuită pe mai multe servere, fiecare gestionând un număr mai mic de adrese.

Mecanismul de subreţele divide zona `Host` în două subzone: subzona de `subreţea` şi subzona de `host în subreţea`. Primii biţi ai zonei, într-un număr impus de necesităţi, sunt rezervaţi subreţelei, iar ultimii pentru hostul în cadrul subreţelei. Nu este obligatoriu ca subzonele să fie constituite din secvenţe contigue de biţi; este de exemplu posibil ca biţii de rang par să desemneze subreţea, iar cei de rang impar să desemneze host. Este, însă indicat ca fiecare subzonă să fie specificată printr-o succesiune de biţi consecutivi.

De exemplu, la adresa de clasă C "`193.231.18.XXX`", sunt administrate patru reţele LAN, având fiecare în jur de 36-40 staţii fiecare. În această situaţie, administratorul decide definirea a patru subreţele. În consecinţă, el decide ca primii doi biţi să desemneze una dintre cele patru subreţele, iar restul de şase biţi desemnează hosturile din cadrul subreţelei. Lăsăm pe seama cititorului să (descompună în reprezentare internă pe biţi şi să) constate că următoarele patru adrese sunt situate în patru subreţele diferite: "`193.231.18.34`", "`193.231.18.66`", "`193.231.18.129`", "`193.231.18.193`".

Pentru a putea separa cele două zone se defineşte conceptul de mască de subreţea. Aceasta se construieşte punând biţii corespunzători câmpului `clasa`, `reţea` şi `subreţea` la valoarea 1, iar cei ai câmpului `host` la valoarea 0. Pentru exemplul prezentat mai sus, masca de subreţea este, în scriere externă, "`255.255.255.192`".

Aceste măşti sunt utilizate pentru a manevra adresele IP folosind operaţiile booleene cunoscute: **şi** bit cu bit, **sau** bit cu bit, **sau-exclusiv** bit cu bit, pentru a izola subreţelele sau hosturile în cadrul subreţelelor.

Adrese speciale

Intre adresele IP, câteva sunt rezervate pentru destinaţii speciale. Prezentăm câteva dintre acestea.

Adresa "127.0.0.1" este rezervată pentru a desemna, pe orice host, hostul însuşi. Fiecare host foloseşte această adresă pentru aşi trimite sieşi pachete de date, testându-se astfel corecta funcţionare a ierarhiei de protocoale de pe maşina locală.

Adrese de broadcast Dacă într-un câmp de adresă toţi biţii au valoarea 1, atunci adresa desemnează calculatoarele host din toate subreţelele conectate în mod direct la acea adresă. Deci o emisie la această adresă va fi o emisie spre toate aceste hosturi. Spre exemplu, o astfel de emisie este folosită de către protocolul ARP.
Dacă toţi biţii din câmpul de adresă host sunt 0, atunci datele sunt transmise către toate hosturile din subreţeaua specificată. Astfel, de exemplu o cerere lansată de către adresa "193.226.62.0" este transmisă la toate adresele "193.226.62.n" cu n între 1 şi 254, chiar dacă acestea nu sunt adrese reale.

Corespondenţa adrese IP - hosturi

Ultima chestiune care trebuie lămurită în legătură cu adresele IP este legată de modul în care se pun în corespondenţă adresele IP cu hosturile. În marea majoritate a cazurilor, unui host conectat la Internet îi corespunde o singură adresă IP. Există însă posibilitatea ca unui host să-i corespundă mai multe adrese IP. Aceste hosturi sunt deci accesibile prin mai multe căi.

Regula după care se face această corespondenţă este următoarea: unui host i se atribuie atâtea adrese IP câte interfeţe de reţea (plăci de reţea) are înglobate în el. Mai mult chiar, unei plăci de reţea (adaptor de reţea) i se pot atribui mai multe interfeţe logice (o interfaţă de reţea poate fi configurată cu mai multe adrese IP).

Deci hosturile care sunt noduri de reţea, hosturile din care se ramifică mai multe tronsoane de reţea etc. au atribuite mai multe adrese IP.

Legătura dintre adresa IP şi host este realizată de către protocoalele ARP - RARP, prezentate în 1.4.2. Prin acestea se pune în corespondenţă adresa fizică a interfeţei de reţea (adresă Ethernet, Token-Ring etc.) cu adresa IP care va fi folosită pentru comunicaţii prin protocolul TCP/IP.

### 3.1.2 Adrese Internet; servere de domenii

Sistemul de adresare Internet - sau cum mai este el numit specificarea de subdomenii, este conceput încât să permită utilizatorului o scriere mai comodă, mai sugestivă şi mai elastică a adresei hosturilor decât cea cu adrese IP. Corespondenţa între specificarea de subdomenii şi adresele IP revine protocolului de aplicaţie DNS (Domain Name Service).

Sistemul de adresare în Internet

Analog sistemului de adresare poştal, o adresa Internet are o structură relativ fixă: Astfel, spre exemplu, adresa autorului (una dintre ele) este:

florin@cs.ubbcluj.ro

În “traducere”, aceasta înseamnă că este vorba de utilizatorul cu numele de user `florin`, care este membru al domeniului `cs` (Computer Science) din cadrul subdomeniului `ubbcluj` (Universitatea “Babes-Bolyai”, Cluj), din cadrul subdomeniului `ro` indicând țara, România.

Un alt exemplu: una dintre maşinile care a fost folosită la testarea programelor din această carte are adresa IP “`193.226.40.133`”, iar adresa ei internet este:

```
rave.scs.ubbcluj.ro
```

care înseamnă maşina cu numele `rave`, conectată la rețeaua subdomeniului `scs` (Students of Computer Science) din subdomeniul `ubbcluj` al domeniului `ro`.

Se poate constata că există o analogie cu sistemul poştal de adresare. Si în acesta se specifică, cu scopul localizării: numele şi prenumele, strada, numărul, oraşul sau satul, eventual codul poştal, județul, țara.

La adresarea poştală fiecare entitate de adresă este prefixată de tipul ei: cuvântul “strada” sau “str.” urmată de numele efectiv al străzii, “Jud.” urmat de numele județului etc. În consecință, ordinea entităților nu este impusă. În schimb, la adresarea Internet, ordinea cuvintelor în specificare este esențială, tipul entității fiind impus şi de poziția cuvântului în specificare.

O adresă Internet are una dintre următoarele trei forme:

1. `numeuser@domeniu1.domeniu2. … .domeniun`

2. `numeuser@numehost.domeniu1.domeniu2. … .domeniun`

3. `numehost.domeniu1.domeniu2. … .domeniun`

Între cuvintele şi separatorii care compun adresa nu trebuie să apară spații. Principalul separator între cuvinte este caracterul “`.`” (punct).

`numeuser` indică numele utilizatorului de pe maşina `numehost` (tipul 2 de adresă) sau din `domeniu1` (domeniu poştal la tipul 1 de adresă). `numeuser` se scrie înaintea caracterului “`@`”.Primele două tipuri de adrese sunt echivalente, în sensul că `numehost` poate înlocui domeniile pe care le gestionează el. Aceste două tipuri de adrese sunt folosite în principal la comunicațiile prin poşta electronică sau în discuțiile interactive (talk) [20,32]. Mai multe despre poşta electronică, într-un capitol următor al acestei lucrări, sau în lucrările [35,20,32]. Adresele de forma a treia sunt folosite pentru a indica hosturi, şi acestea vor fi cele mai utilizate adrese în lucrarea de față.

Succesiunea `domeniu1.domeniu2. … .domeniun` indică nivele de organizare, care sunt din ce în ce mai generale, de la stânga spre dreapta. Astfel, spre exemplu, adresa de host:

rave.scs.ubbcluj.ro

reprezintă includerea de nivele din fig. 1.22

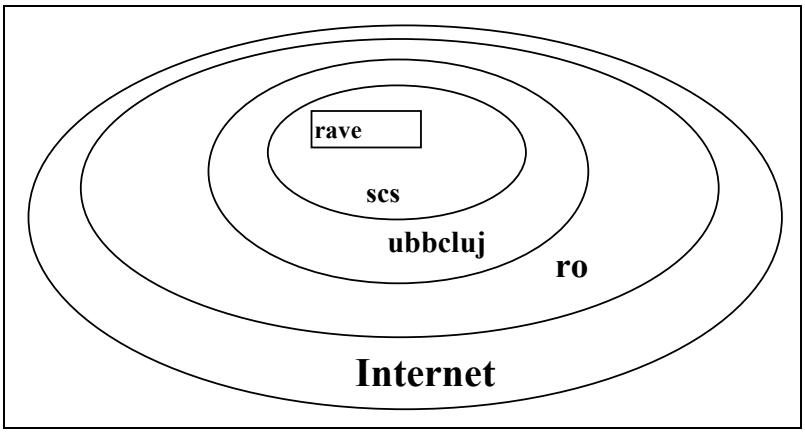



Figura 3.5 Un exemplu de includere de domenii

Fiecare nivel de organizare este indicat printr-un nume de domeniu (de fapt nume de subdomeniu), care este cuprins (inclus) în domeniul scris în dreapta lui.

De aici se deduce prima regulă de organizare: fiecare domeniu îşi defineşte propriile subdomenii, le administrează şi le face publice. Deci stabilirea subdomeniilor este strict de competența administratorului de domeniu.

De exemplu, rețeaua de calculatoare a Universității "Babeş-Bolyai" Cluj, denumită pe scurt UBBNET, este subrețea în cadrul rețelei ROEDUNET (Romanian Educational Network). Administratorii ei, de comun acord cu administratorii domeniului `ro`, au decis ca numele de subdomeniu să fie `ubbcluj`.
La rândul lor, administratorii `ubbcluj` au decis ca fiecare dintre subdomeniile lui să desemneze o facultate, un serviciu sau după caz un departament, în funcție de dimensiunea subrețelei parte din UBBNET pe care o dețin. Tot ca un criteriu de fixare de subdomeniu s-a luat în considerare şi amplasarea topografică în municipiul Cluj a diverselor clădiri ale Universității. La rândul lor, acolo unde este cazul, subdomeniile de facultăți se pot subdivide ş.a.m.d. În tabelul de mai jos este dată lista completă a subdomeniilor domeniului ubbcluj la această oră (septembrie 2002). Numele lor au fost atribuite, pe cât posibil, în urma unor prescurtări din limba engleză.

| | |
|---|---|
| Bioge | Facultatea de Biologie – Geologie |
| Biolog | Departamentul de Biologie animală |
| Chem | Facultatea de Chimie |
| Cs | Departamentul de Informatică al Facultății de Matematică şi Informatică |
| Econ | Facultatea de Stiințe Economice |
| Euro | Facultatea de Studii Europene |
| Gct | Facultatea de Teologie Greco - Catolică |
| Geografie | Facultatea de Geografie |
| Hiphi | Facultatea de Istorie - Filosofie |
| Law | Facultatea de Drept |
| Lett | Facultatea de Litere |
| Math | Facultatea de Matematică şi Informatică |
| Stud.math | Studenții matematicieni ai Facultății de Matematică şi Informatică |
| Ot | Facultatea de Teologie Ortodoxă |

| | |
|---|---|
| Phys | Facultatea de Fizică |
| Polito | Facultatea de Stiințe Politice |
| Psiedu | Facultatea de Psihologie şi Stiința Educației |
| Rt | Facultatea de Teologie Reformată |
| Rct | Facultatea de Teologie Romano – Catolică |
| Scs | Studenții informaticieni ai Facultății de Matematică şi Informatică |
| Sport | Facultatea de Educație Fizică şi Sport |
| Staff | Rectoratul Universității "Babes-Bolyai" |
| Tbs | Scoala de Bussines "Transilvania" |

Din modul de atribuire a numelor de subdomenii rezultă o a doua regulă: numărul total de domenii (n) nu este fixat apriori ci depinde numai de sistemul de organizare adoptat. (Analog cu sistemul poştal clasic, unde o adresă într-un oraş are mai multe elemente decât o adresă dintr-un sat mic).

Cel mai general domeniu, cel care se scrie cel mai în dreapta, este un cod care depinde de țară. Pentru SUA, acesta este codul care indică un domeniu organizațional, unul dintre cele şapte din tabelul de mai jos.

| | |
|---|---|
| com | Organizații comerciale |
| edu | Instituții academice şi educaționale |
| gov | Instituții guvernamentale |
| int | Instituții internaționale (NATO, ONU etc) |
| mil | Instituții militare SUA |
| net | Resurse din rețeaua Internet |
| org | Organizații non – profit |

Dacă domeniul este înafara SUA, atunci el este un cod care indică țara de apartenență. De obicei, acesta este din două litere şi coincide cu codul internațional de marcare a autoturismelor. În tabelul de mai jos indicăm 20 dintre aceste coduri.

| | | | |
|---|---|---|---|
| at | Austria | it | Italia |
| au | Australia | jp | Japonia |
| ca | Canada | mx | Mexic |
| ch | Elveția | nz | Noua Zeelandă |
| de | Germania | pl | Polonia |
| dk | Danemarca | ro | România |
| fi | Finlanda | ru | Federația Rusă |
| fr | Franța | tr | Turcia |
| hu | Ungaria | uk | Marea Britanie |
| il | Israel | va | Vatican |

Specificările de domenii pot fi privite ca şi structuri arborescente. Da fapt, avem de-a face cu mai multe structuri arborescente, fiecare dintre ele având ca şi rădăcină numele de domeniu al țării sau, în cazul SUA, numele domeniului organizațional. În figura 1.23 ilustrăm un fragment din domeniul `ro` şi unul din domeniul `fi`. La acestea ne vom mai referi şi în secțiunile următoare.

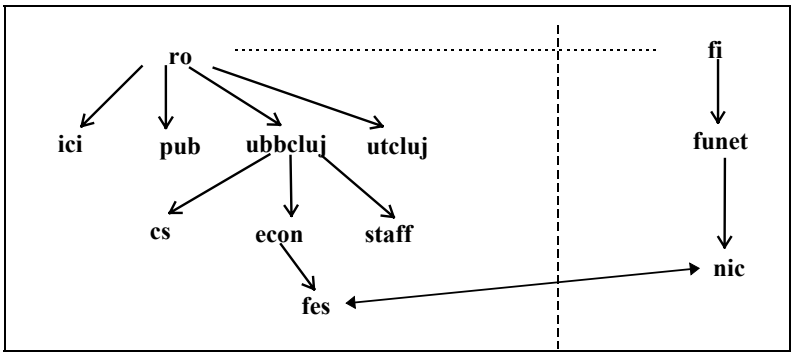



Figura 3.6 Porţiuni din arborii domeniilor ro şi fi

Acum este momentul să enunţăm o a treia regulă de specificare. Ea se enunţă simplu: într-o comunicaţie sursă - destinaţie, sursa este obligată să specifice subdomeniile, începând de la cel mai interior şi până la primul subdomeniu care are ca şi subordonat destinaţia. Deci, spre exemplu, pentru o comuncaţie între maşinile `nessie.cs.ubbcluj.ro` şi `hercule.utcluj.ro`, trebuie să se specifice doar `nessie.cs.ubbcluj` şi `hercule.utcluj`. deoarece domeniul `ro` le conţine pe amândouă. (Aşa cum, într-o scrisoare loco, nu se specifică decât strada şi numărul, nu şi oraşul, judeţul etc.).

Mecanismul serverelor de nume (DNS)

Mecanismul serverelor de nume, mai cunoscut sub numele de mecanismul DNS (Domain Name Service), realizează într-o manieră distribuită, corespondenţa dintre adresele IP şi adresele Internet.

Regula de atribuire stabileşte că fiecărei adrese Internet a unui host (tipul 3 din 1.5.3) îi corespunde o unică adresă IP. Este însă posibil ca mai multe adrese Internet să aibă ca şi corespondent o aceeaşi adresă IP. În acest caz spunem că avem de-a face cu sinonime ale aceleiaşi maşini. De exemplu, adresele `zeus.ubbcluj.ro`, `ftp.ubbcluj.ro` şi `www.ubbcluj.ro` corespund la aceeaşi adresă IP: "`193.231.20.1`". Primul nume este numele propriu-zis al maşinii, al doilea este numele serverului FTP, iar al treilea este numele serverului de Web, ultimele două servere fiind găzduite tot pe maşina `zeus`.

Mecanismul DNS presupune că reţeaua Internet este împânzită de calculatoare speciale, numite servere de nume, prescurtat NS (Name Server). Fiecare NS conţine două tipuri de informaţii:

1) tabele de corespondenţă între adresele Internet şi adresele IP ale unui grup de hosturi aflate în vecinătatea lui;

2) adresele IP şi Internet ale câtorva NS vecine lui.

Fiecare domeniu trebuie să aibă desemnat cel puţin un NS care să-i asigure corespondenţa adresă IP - adresă Internet pentru subdomeniile proprii. Este posibil, dacă domeniul este mare, ca aceste corespondenţe să fie distribuite pe mai multe NS ale domeniului respectiv.

Cum acţionează un NS, atunci când primeşte o adresă Internet şi i se cere adresa IP corespunzătoare? Conform următoarelor reguli, aplicate succesiv:

***a***) Dacă adresa Internet se află în tabelele NS respectiv, atunci întoarce adresa IP celui care a solicitat-o. Dacă nu, atunci:
***b***) Dacă un domeniu superior din adresa Internet se află în tabelele NS respectiv, atunci transmite cererea la NS-ul domeniului superior respectiv. Dacă nu, atunci:
***c***) Dacă adresa Internet primită indică un host aflat în acelaşi domeniu cu NS respectiv, atunci trimite cererea la NS-ul subdomeniului în care se află hostul. Dacă nu, atunci:
***d***) Trimite cererea primului NS pe care-l are înregistrat ca NS vecin. Dacă acesta nu rezolvă cererea, atunci trimite la următorulNS vecin ş.a.m.d.
***e***) Dacă nici unul dintre NS-urile la care s-a trimis cererea nu întoarce o adresă IP, atunci adresa Internet este incorectă.

Credem că este momentul să dăm un exemplu. În fig. 1.23 sunt indicate două hosturi. Să presupunem că un utilizator de pe maşina `fes.econ.ubbcluj.ro` doreşte să solicite un transfer de fişiere prin FTP de la maşina `nic.funet.fi`. Pentru aceasta, el va da comanda:

```
...>ftp   nic.funet.fi
```

După cum am arătat în 1.4.2, în pachetele IP trebuie să apară adresele IP ale corespondenţilor. Evident, adresa IP a maşinii `fes.econ.ubbcluj.ro` este cunoscută. Pentru a afla adresa IP a maşinii nic.funet.fi se vor parcurge următoarele 9 etape:

1. `fes.econ.ubbcluj.ro` nu găseşte în tabelele sale adresa `nic.funet.fi`, deci nu poate aplica regula ***a***. Nu găseşte nici adresa `funet.fi`, nici `fi`, deci nu poate aplica regula ***b***. De asemenea, nu poate aplica nici regula ***c***, deoarece cele două hosturi se află în domenii diferite. În consecinţă, aplică regula ***d*** şi cere NS-ului `econ.ubbcluj.ro`, care-i este NS vecin, să-i rezolve cererea.

2. `econ.ubbcluj.ro`, conform unui raţionament analog punctului 1, aplică regula ***d*** şi cere lui `zeus.ubbcluj.ro`, care-i este NS vecin, să-i rezolve cererea.

3. `zeus.ubbcluj.ro`, conform unui raţionament analog punctului 1, aplică regula ***d*** şi cere lui `hercule.utcluj.ro`, care-i este NS vecin, să-i rezolve cererea.

4. `hercule.utcluj.ro`, conform unui raţionament analog punctului 1, aplică regula ***d*** şi cere lui `pub.pub.ro`, care-i este NS vecin, să-i rezolve cererea.

5. `pub.pub.ro`, conform unui raţionament analog punctului 1, aplică regula ***d*** şi cere lui `ns.ici.ro`, care-i este NS vecin, să-i rezolve cererea.

6. `ns.ici.ro`, conform unui raţionament analog punctului 1, aplică regula ***d*** şi cere lui `ns-a.rnc.ro`, care-i este NS vecin, să-i rezolve cererea.

7. `ns-a.rnc.ro` are în tabelele sale domeniul `fi`. Aplică deci regula ***b*** şi cere la NS-ul domeniului `fi` să-i rezolve cererea. Această operaţiune este ilustrată în fig. 1.21 printr-o linie punctată care leagă `ro` cu `fi`.

8. NS-ul domeniului `fi`, în conformitate cu regula ***c***, cere la NS-ul domeniului `funet.fi` să-i rezolve cererea.

9. NS-ul domeniului funet.fi are în tabelele sale adresa IP a maşinii nic.funet.fi. În consecinţă, conform regulii a, găseşte şi returnează adresa IP a maşinii. Urmează:

10. Adresa IP astfel determinată urmează traseul invers răspunzând pozitiv pe rând NS-urilor indicate la punctele 9, 8, ..., 3, 2 şi în final acesta din urmă dă răspunsul maşinii solicitante. În acest moment maşina `fes.econ.ubbcluj.ro` dispune de adresa IP a maşinii `nic.funet.fi`.

Cu cele două adrese, se pot pregăti pachetele IP care vor circula între cele două maşini (săgeata dublă între `fes` şi `nic` din fig. 1.23).

Trebuie reţinut faptul că rutele de obţinere a adreselor IP au ca intermediari numai servere de nume. În schimb, pachetele IP obişnuite au alte trasee. Iată două astfel de rute între aceleaşi două hosturi. Urmele rutelor au fost obţinute cu ajutorul comenzii:

```
...> traceroute   nic.funet.fi
```

lansată de pe o maşină din domeniul `ubbcluj.ro`. Iată primul traseu, realizat prin furnizorul de servicii Internet RoEduNet, în care se indică adresele Internet şi adresele IP ale celor 18 noduri de pe traseu.

```
 1  UBB-GW.Cluj.Roedu.Net (193.231.20.3)
 2  r-bb1-s9-0.Bucharest.roedu.net (192.129.4.201)
 3  bucharest-br1-fe6-0-0.rdsnet.ro (193.231.191.33)
 4  193.231.184.158 (193.231.184.158)
 5  atvie102-tc-s5-2.ebone.net (195.158.245.85)
 6  atvie203-tc-r3-0.ebone.net (213.174.68.98)
 7  r1-Se0-3-0.0.Wie-VIX.AT.KPNQwest.net (134.222.249.37)
 8  r2-PO1-0-0.wie-KQE.AT.kpnqwest.net (134.222.224.53)
 9  r1-Se1-1-3.ffm-KQ1.DE.kpnqwest.net (134.222.231.57)
10  r1-Se0-3-0.0.hmbg-KQ1.DE.kpnqwest.net (134.222.230.110)
11  r2-Se1-1-0-0.Sthm-KQ1.SE.KPNQwest.net (134.222.230.150)
12  r4-Se7-0-0-0.Sthm-KQ1.SE.KPNQwest.net (134.222.119.246)
13  s-gw.nordu.net (134.222.119.241)
14  fi-gw.nordu.net (193.10.68.42)
15  funet1-rtr.nordu.net (193.10.252.50)
16  funet6-p13-csc0.funet.fi (193.166.255.209)
17  info1-p40-funet6.funet.fi (193.166.255.110)
nic.funet.fi (193.166.3.1)
```

A două rută am obţinut-o pornind tot de la o maşină din domeniul `ubbcluj.ro`, dar traseul este realizat print-un furnizor de servicii Internet privat, care oferă o linie de rezervă reţelei UBBNet. Iată cum arată cel de-al doilea traseu.

```
 1  fiber-gw.cluj.astralnet.ro (193.230.247.193)
 2  atv.codec.ro (193.230.240.22)
 3  r0-fe01.cluj.astralnet.ro (193.230.240.98)
 4  atvie103-ta-s6-0-9c0.ebone.net (195.158.245.81)
```

```
 5  atvie203-tc-r3-0.ebone.net (213.174.68.98)
 6  r1-Se0-3-0.0.Wie-VIX.AT.KPNQwest.net (134.222.249.37)
 7  r2-PO1-0-0.wie-KQE.AT.kpnqwest.net (134.222.224.53)
 8  r1-Se1-1-3.ffm-KQ1.DE.kpnqwest.net (134.222.231.57)
 9  r1-Se0-3-0.0.hmbg-KQ1.DE.kpnqwest.net (134.222.230.110)
10  r2-Se1-1-0-0.Sthm-KQ1.SE.KPNQwest.net (134.222.230.150)
11  r4-Se7-0-0-0.Sthm-KQ1.SE.KPNQwest.net (134.222.119.246)
12  s-gw.nordu.net (134.222.119.241)
13  fi-gw.nordu.net (193.10.68.42)
14  funet1-rtr.nordu.net (193.10.252.50)
15  funet6-p13-csc0.funet.fi (193.166.255.209)
16  info1-p40-funet6.funet.fi (193.166.255.110)
17  nic.funet.fi (193.166.3.1)
```

În încheierea acestei secțiuni, trebuie să arătăm că un mecanism eficient DNS depinde de mai mulți factori. Astfel, pentru a se evita buclarea, algoritmii trebuie să aibă în vedere ca regula d să nu fie aplicată la NS-uri care au mai fost solicitate. O topologie stea [16] este cea mai simplă soluție de evitare a acestui fenomen.

Eficiența mecanismului DNS creşte dacă ordinea de specificare a NS-urilor vecine este bine aleasă, punând pe primele locuri NS-urile mai frecvent folosite.

Mecanismul DNS nu este static, ci dinamic. Astfel, fiecare NS "învață", ținând minte solicitările din ultimele zile. Astfel, repetarea unor cereri face ca traseul de obține a unei adrese IP să fie mult redus.

Specificarea adreselor Web (URL)

Un tip de adrese care extind adresele Internet sunt adresele de specificare a paginilor de Web [51]. Această specificare poartă numele de **URL** (Uniform Resource Locator). O astfel de adresă este de forma:

protocol://server/cale/numedocument

Unele dintre aceste patru zone pot eventual să lipsească.

`protocol` face parte dintre protocoalele comunicațiilor Internet, deci la nivelul de aplicație TCP/IP sau mai sus. Cel mai frecvent protocol este protocolul `http` (Hyper Text Transfer Protocol) care este folosit pentru localizarea paginilor de Web.

`server` indică serverul care conține pagina de Web şi se specifică printr-o adresă Internet aşa cum am prezentat în secțiunea precedentă.

`cale/document` este de fapt o specificare de fişier în stil UNIX [26,37]. Prin el se indică locul documentului în structura de directori a serverului.

Iată, spre exemplu, câteva URL-uri:

http://www.ubbcluj.ro

http://www.ubbcluj.ro/cluj.html
http://home.netscape.com/index.html
ftp://ftp.netscape.com/pub/ceva
news:news.announce.newusers
wais://cnidr.org/directory-of-server
gopher://gopher.micro.umn.edu/11/

Pentru detalii, se pot consulta lucrările[32,35,20,21].

# 4 Elemente de limbaj HTML

## *4.1 Introducere*

Limbajul HTML (HyperText Markup Language) este destinat scrierii documentelor de tip hipertext care se pot vizualiza cu ajutorul navigatoarelor WWW (World Wide Web). Un document HTML este un fişier text care foloseşte numai caractere ASCII, şi poate fi realizat cu orice editor de texte (de ex.: vi, pico, joe, emacs în sistemul de operare UNIX sau variantele acestuia, Notepad sub Windows, BBEDit pe calculatoare Macintosh, editorul programului Norton Commander sau editorul Edit sub DOS). Există programe cu ajutorul cărora textele HTML pot fi vizualizate, de ex. HotMetal care poate fi utilizat pe mai multe platforme.

**HTML** este un limbaj bazat pe **SGML** (Standard Generalized Markup Language), un sistem complex de descriere a documentelor. SGML este utilizat pentru descrierea structurii generale a diferitelor tipuri de documente, fără să fie un limbaj de descriere a paginii (cum este PostScript). Principala preocupare a SGML, deci şi a HTML se răsfrânge asupra conţinutului documentului, nu asupra aspectului lui. Deci, în virtutea trăsăturilor moştenite de la SGML, HTML este un limbaj pentru descrierea documentelor structurate. Teoria din spatele acestui limbaj se bazează pe faptul că majoritatea documentelor au elemente comune (titluri, paragrafe sau liste) şi că dacă defineşti un set de elemente, poți marca elementele documentului cu ***tag***-urile (etichetele) corespunzătoare.

Există şi programe cu ajutorul cărora se pot realiza pagini web fără a cunoaşte limbajul HTML. Acestea funcţionează asemănător editoarelor de texte. Amintim aici doar editorul navigatorului Netscape şi programul FrontPage.

Recomandăm scriitorilor de documente HTML să utilizeze programul `HTMLTidy` care este un program gratuit aflat la: `http://www.w3.org/MarkUp/Guide` (dar şi în pagina Web a cursului), pentru a-şi verifica, automat, corectitudinea textului scris.

Un document HTML poate conţine text şi marcaje. Marcajele HTML sunt de forma

<MARCAJ> ... </MARCAJ>

Foarte rar partea finală, </MARCAJ>, poate să lipsească. Aceste marcaje nu sunt sensibile la tipul literelor (majuscule sau minuscule). Aceasta înseamnă că, de exemplu, următoarele marcaje reprezintă acelaşi lucru:

<HTML> ... </html>
<html> ... </HTML>
<Html> ... </hTmL>

Instrucţiunile HTML pot avea atribute cu sau fără valori. Atributele se scriu în felul următor:

```
<MARCAJ atribut_1[=valoare_1] atribut_2[=valoare_2]...>
```

unde elementele puse în paranteze drepte sunt opţionale.

Nu intenţionăm să descriem toate marcajele şi toate atributele lor, ci doar pe cele mai importante dintre acestea. Cele două tabele din anexă prezintă lista tuturor marcajelor permise în HTML 4.01, lista tuturor atributelor, precum şi marcajele în care acestea pot să apară. Considerăm un exerciţiu util pentru învăţarea HTML, ca cititorul să-şi întocmească un unic tabel cu sintaxele complete ale HTML din combinarea celor două tabele. De asemenea, cititorul are la dispoziţie o serie de site-uri Internet în care exte descris complet limbajul HTML. Adresa de referinţă în acest sens este: http://www.w3.org/MarkUp/

Cu atât mai mult, nu vom descrie acele marcaje sau atribute care sunt acceptate doar de un singur navigator.

## *4.2 Un exemplu simplu de fişier HTML*

Inainte de a prezenta limbajul HTML, preferăm să prezentăm un exemplu simplu de text HTML. Iată mai jos un astfel de program.

```
<HTML>
<HEAD>
<TITLE> Document demonstrativ</TITLE>
</HEAD>
<BODY>
<H1> Salut! </H1>
Acesta este un document minimal HTML.
El arata structura de baza si conceptul de ancora.
<P>
Pentru informatii suplimentare despre HTML se pot consulta:
<A Href="http://www.december.com/html/">
http://www.december.com/htm/ </A>
 sau se poate consulta <BR>E-cartea:
<A Href = " http://www.codec.ro/books/HTML/index.html/">
HTML 3.2 and CGI
Professional Reference Edition
UNLEASHED</A>
by John December and Mark Ginsburg <HR>
Daca doriti sa trimiteti un E-mail persoanei care a scris acest fisier,
va rog sa o faceti. Adresa este:
<ADDRESS> <A Href = "http://cs.ubbcluj.ro/~florin">Home -
pagina personala </A> si
```

```
<A Href = "mailto:ubbadmin@ubbcluj.ro"> Adresa de E - mail </A> <BR> <HR>
</ADDRESS>
<B>Remarcati diferentele intre modul de scrierea in HTML si ceea ce apare pe
ecran.
</B>
</BODY>
</HTML>
```

Fişierul `FirstHTML.html`

Acest fişier, deschis cu navigatorul Netscape, apare ca în fig. 5.1

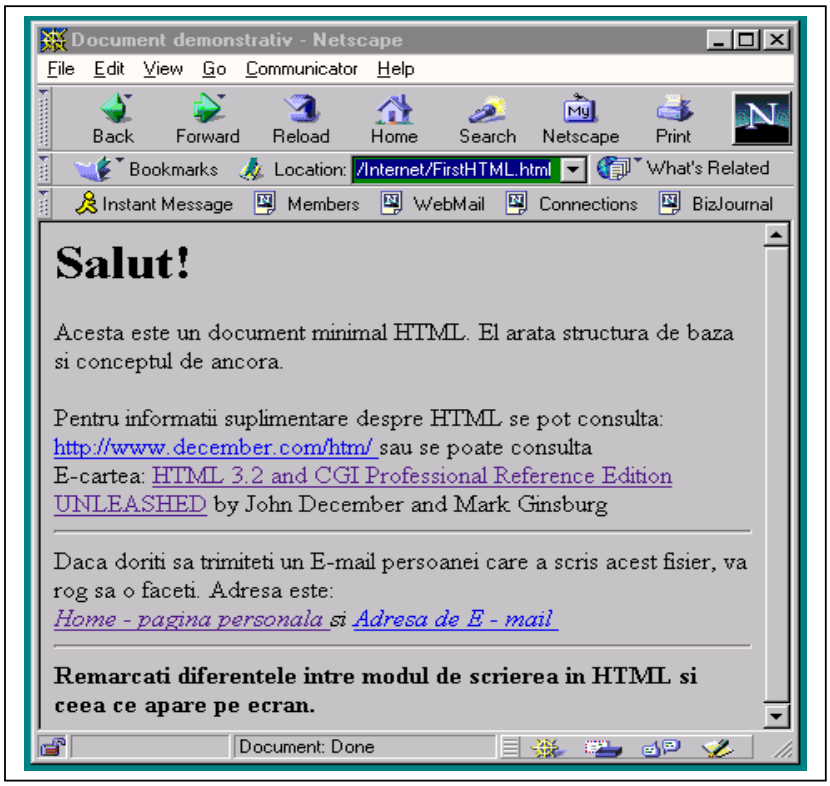



Figura 5.1 Vizualizarea fişierului `FirstHTML.html`

După cum se vede, un fişier HTML apare ca un text ce conţine o serie de construcţii delimitate de simbolurile "<" şi ">". Aceste construcţii specifică indicaţii speciale adresate navigatorului (browserului) Web. Orice construcţie care apare între simbolurile "<" şi ">" o vom numi *marcaj* sau *tag*. În speranţa că nu vor exista confuzii, şi pentru că este mai scurt, vom folosi de multe ori termenul de *tag* în loc de marcaj (natural, marcaj şi tag vor însemna acelaşi lucru).

Tagurile sunt de două feluri:

Taguri *simple*, indicate prin construcţii de forma:

```
<NumeTag AtributeTag>
```

Taguri *compuse*, indicate prin construcţii de forma:

```
<NumeTag AtributeTag> TextAfectatDeTag </NumeTag>
```

În textul HTML de mai sus tagurile <P>, <BR> şi <HR> sunt taguri simple. Efectul lor este respectiv: începutul unui nou paragraf, trecerea la un rând nou (break row) şi trasarea unei linii

orizontale (horizontal row). În acelaşi text, apar tagurile compuse: HTML, HEAD, TITLE, BODY, H1 etc.

## *4.3 Structura unui document HTML*

### 4.3.1 Antet, fond şi culori

Există câteva marcaje care sunt obligatorii într-un document HTML. Acestea sunt următoarele:

<HTML> ... </HTML>
<HEAD> ... </HEAD>
<BODY> ... </BODY>

Un document HTML poate fi de forma:

```
<HTML>
  <HEAD>
        aici apare textul antetului
  </HEAD>
  <BODY>
       aici apare textul corpului documentului
  </BODY>
</HTML>
```

În textul unui document HTML spaţiile nu au importanţă, deci printr-o indentare potrivită textul poate fi mai lizibil. Este de remarcat faptul, că diferite navigatoare pot vizualiza textele în moduri diferite.

**<HTML> ... </HTML>** Acest marcaj are rolul de a delimita textul dintr-un document HTML. Singurele marcaje care pot să apară direct în interiorul lui `<HTML>` sunt `<HEAD>` si `<BODY>`.

**<HEAD> ... </HEAD>** Reprezintă antetul unui document HTML şi conţine o colecţie de informaţii despre document. In interiorul acestuia pot să apară mai multe marcaje. Aici le menţionăm doar pe următoarele:

**<BASE HREF="adresa">** Un document HTML poate să conţină legături înspre alte documente HTML. Aceste documente pot să fie specificate printr-o adresă completă, de exemplu

```
http://www.domeniu.ro/director/document.html
```

sau intr-o formă parţială, ca de exemplu:

```
director/document.html                    (*)
```

Dacă este declarat un marcaj <BASE>, atunci adresele parțiale specificate în document sunt completate cu adresa menționată în <BASE>. În caz contrar acestea sunt completate cu adresa documentului curent. De exemplu, dacă adresa documentului curent este

```
http://www.altdomeniu.ro/dir/doc.html
```

şi nu există <BASE>, atunci adresa specificată mai sus (*) se va completa la

```
http://www.altdomeniu.ro/dir/director/document.html
```

Dacă însă există:

```
<BASE HREF="http://www.domeniu.ro/">
```

atunci adresa specificată mai sus (*) se va completa la

```
http://www.domeniu.ro/director/document.html
```

**<TITLE>...</TITLE>** Orice document HTML trebuie să aibă un titlu. Titlul este un şir de caractere arbitrar care identifică conținutul documentului. Titlul trebuie să apară în interiorul marcajei <HEAD>.

**<META...>** Instrucțiunea <META> este folosită în interiorul lui <HEAD> pentru a introduce informații care nu sunt definite de alte marcaje HTML. Aceste informații pot fi extrase de servere/clienți pentru a fi utilizate în identificarea, indexarea şi catalogarea documentelor. Dintre atributele care pot să apară în interiorul unei marcaje <META> menționăm aici doar NAME şi CONTENT. Iată câteva exemple:

```
<META NAME="description" content="Ion I. Ionescu este cercetator la Institutul X. Această pagină conține detalii despre activitatea sa">

 <META NAME="keywords" CONTENT="cuvânt-cheie-1, cuvânt-cheie-2, cuvânt-cheie-3, şi-alte-cuvinte-cheie">
```

**<BODY>...</BODY>** Conține întregul text şi imaginile care formează pagina de Web, împreună cu toate marcajele HTML care controlează formatarea paginii. In interiorul marcajului <BODY> pot să apară anumite atribute care controlează schema de culori a documentului. în continuare prezentăm cele mai importante atribute.

```
BACKGROUND="URL-al-unui-fişier-ce-conține-o-imagine"
```

Imaginea din fişierul specificat este pusă cap la cap şi formează în acest fel fundalul documentului. în lipsa acestui atribut fundalul va avea culoarea specificată în BGCOLOR.

BGCOLOR="#rrvvaa"

Specifică culoarea fundalului documentului, unde `#rrvvaa` este un triplet scris in hexazecimal care reprezintă ponderea culorilor roşu-verde-albastru în culoarea dorită (`rr=00` înseamna absența completă a culorii roşii, iar `rr=FF` înseamnă prezentă totală a culorii roşii). Valoarea implicită este cea albă.

TEXT="#rrvvaa"

Specifică culoarea textului normal din document.

LINK, VLINK, ALINK
Un document HTML poate să conțină legături înspre alte docuemte HTML sau imagini. Aceste atribute controlează culorile textului care reprezintă legătura, astfel:

`LINK` reprezintă culoarea textului corespunzător unei legături nevizitate,

`VLINK` reprezintă culoarea textului corespunzător unei legături vizitate şi

`ALINK` reprezintă culoarea textului unei legături vizitate şi active (documentul este deja în memoria Cache a navigatorului). Culorile implicite pentru aceste texte sunt, în ordinea menționată, albastru (#0000FF), roz (#400040) şi roşu (#FF0000). Sintaxa acestor atribute este acceaşi cu cea a atributului `TEXT`.

Dăm în continuare o listă de culori mai frecvente. De menționat că se pot folosi şi denumirile englezeşti ale culorilor, de exemplu, în loc de "#FF0000" se poate scrie pur şi simplu *red.*

```
negru            000000     (black)
roşu             FF0000     (red)
alb              FFFFFF     (white)
verde            00FF00     (green)
albastru         0000FF     (blue)
albastru violet  9F5F9F     (blue violet)
galben           FFFF00     (yellow)
maro             A62A2A     (brown)
verde închis          2F4F2F     (dark green)
albastru deschis C0D9D9     (light blue)
maro închis           5C4033     (dark brown)
portocaliu       FF7F00     (orange)
auriu            CD7F32     (gold)
gri              C0C0C0     (grey)
violet                4F2F4F     (violet)
```

Modul cel mai simplu de a vizualiza un document HTML este folosind Netscape. Lansăm programul Netscape şi încărcăm fişierul care conține documentul prin intermediul meniului **File | Open File**.

### 4.3.2 Marcaje de formatare a documentului

Din punctul de vedere al formatării unui document HTML, lucrurile stau foarte asemănător cu preparatorul de texte matematice LaTeX. Aceasta înseamnă că textul dintr-un document HTML se formatează exclusiv conform marcajelor de formatare incluse în text. Formatul fizic pe care îl dăm noi documentului (caractere TAB, caractere spațiu multiple, caractere NEWLINE) nu este luat în seamă de navigatoarele Web. în această secțiune vom prezenta cele mai importante marcaje de

formatare HTML. Dacă sunt prezente, acestea trebuie să apară exclusiv în corpul documentului (între <BODY> şi </BODY>).

**<ADDRESS>...</ADDRESS>** Specifică informaţii cum ar fi adresa sau autorul unui document şi se plasează de obicei la începutul sau sfârşitul documentului. De obicei, textul corespunzător se formatează în italic şi poate să fie indentat. Navigatoarele Web vor presupune că acesta formează un paragraf separat.
**<BLOCKQUOTE>...</BLOCKQUOTE>** Se foloseşte pentru introducerea de informaţii citate din alte surse. Textul va apare cu indentare la stânga şi la dreapta şi, de obicei, în italic. Este un paragraf separat.

**<BR>** Forţează ruperea liniei. Textul care urmează se va scrie de la începutul liniei următoare.

**<CENTER>...</CENTER>** Toate liniile de text dintre <CENTER> şi </CENTER> sunt centrate între marginile din stânga şi dreapta, în loc să fie aliniate la stânga şi la dreapta, cum este normal.

**<FONT ...> ... </FONT>** Permite setarea sau schimbarea dimensiunii fontului, culorii şi tipului textului. Aceste operaţii se realizează prin intermediul câtorva atribute, pe care le vom prezenta în continuare:

SIZE=[+|-]valoare

Permite modificarea dimensiunii fontului (valoare este un număr între 1 şi 7). Dacă se foloseşte + sau − fontul se modifică relativ faţă de dimensiunea de bază.

COLOR=#rrvvaa Permite specificarea culorii textului.

FACE=nume [,nume ,nume ...]

Permite specificarea unui tip de caractere pentru textul respectiv şi, deasemenea, câteva tipuri alternative în caz că acesta nu este disponibil.
Exemplu:

<FONT SIZE=+1 COLOR=red> text </FONT>

**<Hx> ... </Hx>** unde x este un număr între 1 şi 6, defineşte unul din cele şase nivele de titluri (de capitol, secţiune, subsecţiune ş.a.m.d.). Instrucţiunea <Hx> presupune schimbarea fontului, spaţii înainte şi după titlu şi declararea textului respectiv ca paragraf.
Unele navigatoare acceptă folosirea atributului ALIGN pentru alinierea textului în mod corespunzător. Acest atribut are forma

<Hx ALIGN=left> Titlu de nivelul x aliniat la stanga </Hx> sau
<Hx ALIGN=center> Titlu de nivelul x centrat </Hx> sau
<Hx ALIGN=right> Titlu de nivelul x aliniat la dreapta </Hx>

**<HR>** Desenează o linie orizontală ca separator între diferite secţiuni de text. Poate să aibă mai multe atribute, dintre care amintim:

<HR SIZE=numar> Grosimea liniei, în pixeli.
<HR WIDTH=numar[%]>

Lungimea liniei, fie în pixeli (dacă nu există caracterul %), fie în procente din lăţimea totală a paginii (dacă există caracterul %). (Semnele [ ] delimitează un text opţional)

<HR ALIGN=left|right|center>

În cazul în care linia orizontală este mai mică decât lățimea paginii, acest atribut permite specificarea modului de aliniere a liniei. (Semnul | delimitează cuvintele care pot fi alease ca valoarea atributului).

**<NOBR>...</NOBR>** Textul dintre <NOBR> şi </NOBR> nu poate fi rupt pe mai multe linii. Prin urmare, într-o astfel de situație ar fi posibil ca utilizatorul să fie nevoit să ajusteze dimensiunea paginii navigatorului.

**<P>...</P>** Această instrucțiune indică un paragraf. Deoarece începutul unui paragraf este, în acelaşi timp, şi sfârşitul paragrafului precedent, </P> se poate omite. De-asemenea, are un atribut, ALIGN, care indică modul de aliniere a textului din paragraful respectiv:

<P ALIGN=left|center|right> ... </P>

**<PRE>...</PRE>** Textul dintre <PRE> şi </PRE> este afişat aşa cum este el redactat, fără a mai fi formatat de navigator (deci aici şi spațiile sunt luate în considerare). De exemplu, această instrucțiune se poate folosi pentru introducerea de texte sursă într-un document HTML. Are un singur atribut, WIDTH, care indică numărul de caractere dintr-o linie, cu valoarea implicită de 80:

<PRE WIDTH=numar> ... </PRE>

### 4.3.3 Specificare tip şi format caractere

În secțiunea de față descriem două tipuri de marcaje de formatare:
(a) **Marcaje de formatare logică**. Acestea sugerează navigatorului că decizia de formatare a textul respectiv trebuie luată de către acesta, şi nu de către autorul documentului HTML.
(b) **Marcaje de formatare fizică.** Aceste marcaje impun navigatorului un anumit tip de formatare.

In cele ce urmează vom trece în revistă cele mai importante marcaje de formatare. Vom începe cu marcajele de formatare logică:

**<CITE>...</CITE>** Specifică o citare. In mod normal se scrie în italic. Textul respectiv nu se separă ca paragraf separat, şi nici nu se ia vre-o decizie alta decât schimbarea stilului caracterelor textului.

**<EM>...</EM>** Specifică text redactat cu emfază. In mod normal se scrie în italic.

**<KBD>...</KBD>** Indică text tipărit de utilizator. De obicei se foloseşte într-un manual de marcaje. În mod normal se scrie monospațiat. Aceasta înseamnă că fiecare caracter se scrie într-un pătrat de dimensiuni egale. De exemplu, literele i şi m vor ocupa acelaşi spațiu.
**<STRONG>.. </STRONG>** Specifică text redactat cu emfază puternică, şi în mod normal se scrie în bold (caractere îngroşate).

**<VAR>...</VAR>** Indică nume de variabilă. De obicei se scrie în italic.

**<!-- Comentarii -->** Textul dintre şirul <!-- şi următoarea apariţie a şirului --> nu este procesat navigator. Se acceptă spaţii între -- şi > la delimitatorul de sfârşit al comentariului, dar nu între <! şi -- la delimitatorul de început.

În continuare prezentăm marcajele de formatare fizică:

**<B> ... </B>** Textul se scrie în bold.
**<I> ... </I>** Textul se scrie în italic.
**<TT> ... </TT>** Textul se scrie în fontul typewriter.
**<U> .. </U>** Textul se scrie subliniat.
**<STRIKE>…</STRIKE>** Textul se scrie cu o linie orizontală suprapusă.
**<SUB> ... </SUB>** Textul se scrie ca indice inferior şi cu dimensiune mai mică.
**<SUP> ... </SUP>** Textul se scrie ca indice superior şi cu dimensiune mai mică.
**<BIG> ... </BIG>** Textul se scrie utilizând un font de o dimensiune mai mare.
**<SMALL>...</SMALL>** Textul se scrie utilizând un font de o dimensiune mai mică.

### 4.3.4 Scrierea caracterelor ne-standard

Scrierea anumitor caractere într-un document HTML necesită utilizarea unor secvenţe speciale, deoarece acestea fie sunt caractere cu semnificaţie specială în sintaxa comenzilor HTML, fie sunt caractere ne-latine, fie sunt caractere latine cu semne diacritice.
Aceste caractere se specifică printr-o secvenţă tip, de forma &şir; (caracterul ampersand, urmat de un şir de caractere, urmat de caracterul punct-virgulă). In continuare vom descrie modul în care pot fi specificate aceste seturi de caractere.

***Caractere speciale***, sunt în număr de patru:

| | | |
|---|---|---|
| < | se scrie ca | < |
| > | | > |
| & | | & |
| " | | " |

***Caractere specifice unor limbi***. Sunt disponibile foarte multe caractere. Marea lor majoritate se specifică după modelul următor:

&lsemn;

unde **l** este litera de bază (mare sau mică), iar **semn** este o descriere a semnului diacritic care se pune deasupra sau dedesubtul caracterului. Iată descrierile semnelor care pot să apară:

| | |
|---|---|
| &#atilde; | caracterul ă românesc; |
| â | caracterul â românesc; |
| ® | caracterul î românesc; |
| ş | caracterul ş românesc; |
| ţ | caracterul ţ românesc; |
| acute | pentru accent ascuţit; |

| | |
|---|---|
| `cedil` | sedilă; |
| `circ` | accent circumflex; |
| `grave` | accent grav; |
| `ring` | inel; |
| `tilde` | tilda; |
| `uml` | umlaut (două puncte orizontale). |

De exemplu, cuvântul **maşină** se scrie într-un document HTML în acest fel: **ma&351;inã**.

Următoarele patru caractere merită, de-asemenea, menționate:

`&reg;` este echivalent cu semnul de marcă înregistrată;
`&copy;` semnul de copyright;
`&trade;` emnul de marcă comercială;
` ` caracterul spațiu, care însă nu este interpretat de navigator ca şi separator de cuvinte!

***Caractere*** specificate prin ***codul ASCII***

Caracterele ASCII se pot specifica şi prin codul lor numeric, în felul următor:

&#nn;

unde nn este codul ASCII al caracterului respectiv, scris în baza 10, cuprins între 0 şi 255. Facem mențiunea că acest număr trebuie să aibă cel puțin două cifre; dacă este nevoie, numărul se completează cu un zero nesemnificativ.
Iată câteva exemple,

| | | |
|---|---|---|
| `&#38;` | este echivalent cu | `&` |
| `&#162;` | | centul american; |
| `&#163;` | | lira engleză; |
| `&#164;` | | dolarul american; |
| `&#165;` | | yenul japonez; |
| `&#167;` | | semnul de secțiune; |
| `&#169;` | | semnul de copyright; |
| `&#174;` | | semnul de marcă înregistrată. |

## *4.4 Legături hipertext*

**`<A...>...</A>`** Instrucțiunea ancoră este esențială într-un document HTML. Textul dintre `<A...>` şi `</A>` este textul de activare a legăturii respective. Acest text se numeşte hipertext. Utilizatorul va da click cu mouse-ul pe hipertextul respectiv, iar în urma acestei operații navigatorul va executa una din acțiunile specificate de atributele prezente în `<A...>`. Dintre toate atributele, fie `NAME` fie `HREF` sunt obligatorii. Iată în continuare cele mai importante atribute:

<A HREF="url"> ... </A>

La selectarea acestui hipertext navigatorul va afişa documentul specificat prin URL-ul respectiv. În loc să ne deplasăm la un alt document, putem să ne deplasăm la un anumit punct al acelui document sau al documentului curent. Aceste puncte se numesc ancore şi se pot defini folosind atributul NAME:

<A NAME="ancoră"> ... </A>

Admițând că această ancoră a fost creată în documentul specificat prin URL-ul "url", referirea la această ancoră se va face astfel:

<A HREF="url#ancoră"> ... </A>

Dacă însă ancora a fost creată în documentul curent, referirea se va face astfel:

<A HREF="#ancoră> ... </A>

Mai amintim aici că formele pe care le poate lua URL-ul "url" în cadrul atributului HREF sunt cele definite de sintaxa URL. Le menționăm doar pe cele mai interesante:

<A HREF="http://..."> Salt la un alt document HTML
<A HREF="ftp://..."> Realizează o conexiune prin ftp anonim la un server ftp.
<A HREF="ftp://login:pass@..."> Conexiunea la serverul ftp se face utilizând contul "login" şi parola "pass"; dacă parola lipseşte, se va cere explicit.
<A HREF="mailto:user@domeniu.ro"> Activarea acestei legături deschide aplicația de redactare şi expediere a unui mesaj electronic către user@domeniu.ro.
<A HREF="news:..."> Realizează o legătură la un newsgrup.
<A HREF="telnet:..."> Activarea acestei legături inițiază o sesiune **telnet** cu maşina specificată.

Exemplu: Dacă în text apare legătura hipertext:

<A HREF="http://www.ubbcluj.ro">Universitatea Babes-Bolyai din Cluj </A>

navigatorul va vizualiza textul *Universitatea Babes-Bolyai din Cluj* cu culoare specificată în LINK, şi în urma unui clic cu mouse-ul pe acest text se va vizualiza hipertextul de la adresa menționată.

## *4.5 Liste şi tabele*

### 4.5.1 Marcaje de tip listă

HTML suportă câteva tipuri de liste. Toate acestea pot fi imbricate, şi trebuie să apară în corpul unui document HTML.

**<DL>...</DL>** Reprezintă o listă de definiţii. Fiecare termen de definit se precede de <DT>, iar definiţia respectivă se precede de <DD>. Se foloseşte după modelul următor:

```
<DL>
<DT> Termen 1 <DD> Definiţia termenului 1.
<DT> Termen 2 <DD> Definiţia termenului 2.
</DL>
```

Rezultatul va fi:

```
Termen 1
          Definiţia termenului 1.
Termen 2
          Definiţia termenului 2.
```

**<OL>...</OL>** Reprezintă o listă numerotată. Fiecare element al listei se precede de <LI>. Se foloseşte după modelul următor:

```
<OL>
<LI> Primul termen;
<LI> Al doilea termen.
</OL>
```

Rezultatul va fi:

```
1. Primul termen;
2. Al doilea termen.
```

În mod normal numerotarea începe de la 1 şi numărul de ordine se afişează folosind numerele obişnuite: 1, 2, 3, .... Dacă dorim modificarea acestor lucruri, putem folosi atributele START şi TYPE:

<OL START=numar> Specifică numărul de început al numerotării.
<OL TYPE=caracter> Specifică modul de numerotare. Caracterul poate fi A (la numerotare se folosesc litere mari: A, B, C,...), a (litere mici: a, b, c,...), I (numere romane mari: I, II, III,...), i (numere romane mici: i, ii, iii,...), 1 (numere obişnuite: 1, 2, 3,...).

**<UL>...</UL>** Reprezintă o listă nenumarotată. Fiecare element al listei se precede de <LI>. Se foloseşte după modelul următor:

```
<UL>
<LI> Primul termen;
<LI> Al doilea termen.
</UL>
```

Rezultatul va fi:

```
* Primul termen;
* Al doilea tremen.
```

In mod normal fiecare element al listei se precede de un caracter special care variază în funcție de nivelul listei: disc solid, cerc, pătrat etc. Pentru modificarea acestui caracter special se poate folosi atributul `TYPE`:

```
<UL TYPE=disc|circle|square>
```

**`<MENU>...</MENU>`** Reprezintă o listă de elemente, câte una pe linie. Este foarte asemănător cu `<UL>...</UL>`, deosebirea constând în faptul că lista `<MENU>` este mai compactă decât lista `<UL>`. Se foloseşte după modelul următor:

```
<MENU>
<LI> Primul termen.
<LI> Al doilea termen.
</MENU>
```

Listele pot fi imbricate. De exemplu:

```
<UL>
<LI> Primul nivel 1
   <UL>
      <LI> Al doilea nivel 1
      <LI> Al doilea nivel 2
   </UL>
   <UL>
<LI> Primul nivel 2
   <UL>
      <LI> Al doilea nivel 3
      <LI> Al doilea nivel 4
   </UL>
</UL>
```

Reazultatul va fi:

```
* Primul nivel 1
   + Al doilea nivel 1
   + Al doilea nivel 2
```

```
 * Primul nivel 2
   + Al doilea nivel 3
   + Al doilea nivel 4
```

### 4.5.2 Tabele

Într-un document HTML se pot introduce şi tabele. Vom prezenta marcajele necesare şi apoi vom vedea şi un exemplu.

**<TABLE...>...</TABLE>** Este marcajul principal care defineşte un tabel. Toate celelalte marcaje trebuie să apară în interiorul acesteia. Are următoarele atribute mai importante:

`BORDER=nr` grosimea bordurii în pixeli; valoarea implicită este 0, adică fără bordură;

`CELLSPACING=nr` controlează spațiul dintre celulele tabelului; valoarea este dată în pixeli;

`CELLPADDING=nr` controlează spațiul dintre datele din celulă şi pereții celulei; valoarea este dată în pixeli;

`WIDTH=nr[%]` specifică lățimea tabelului, fie în pixeli, fie în procente de lățime a ferestrei;

`HEIGHT=nr[%]` specifică înălțimea tabelului, fie în pixeli, fie în procente de înălțime a ferestrei;

**<TR...>...</TR>** Specifică un rând de celule al tabelului. Numărul de rânduri este dat de numărul de perechi `<TR>...</TR>`. Are următoarele atribute mai importante:

ALIGN=left|center|right
controlează modul de aliniere pe orizontală a textului în interiorul celulelor;

VALIGN=top|middle|bottom|baseline
controlează modul de aliniere pe verticală a textului în celule;

**<TD...>...</TD>** Specifică o celulă de date a tabelului. Trebuie să apară doar în interiorul unui rând (între `<TR>` şi `</TR>`). Dintre atributele mai importante menționăm:

ALIGN=left|center|right
controlează modul de aliniere pe orizontală a textului în interiorul celulei; valoarea implicită este `left`;

VALIGN=top|middle|bottom|baseline
ontrolează modul de aliniere pe verticală a textului în interiorul celulei; valoarea implicită este `middle`;

`WIDTH=nr[%]` specifică lățimea tabelului, fie în pixeli, fie în procente de lățime a tabelului;

`HEIGHT=nr[%]` specifică înălțimea tabelului, fie în pixeli, fie în procente de înălțime a tabelului;

`NOWRAP` rândurile lungi nu se pot rupe;

`COLSPAN=nr` specifică numărul de coloane peste care se va întinde celula aceasta;

ROWSPAN=nr specifică numărul de rânduri peste care se va întinde celula aceasta.

**<TH...>...</TH>** Specifică o celulă antet a tabelului. Este absolut identică cu o celulă <TD> cu singurele diferenţe că textul din interiorul celulei <TH> este bold şi este implicit aliniat cu ALIGN=center.

**<CAPTION...>...</CAPTION>** Este titlul tabelului. Trebuie să apară în interiorul lui <TABLE>, dar nu în interiorul rândurilor sau celulelor. Acceptă atributul

ALIGN=top|bottom controlează locul unde se va scrie titlul respectiv (deasupra sau sub tabel).

Iată mai jos descrierea a două tabele:

```
<html>
<head>
<title>exemple de tabele</title>
</head>
<body>
<TABLE BORDER>
<TR>
  <TD>Celula 1</TD>
  <TD COLSPAN=2>Celulă pe două coloane</TD>
</TR>
<TR>
  <TD>Celula 2</TD>
  <TD>Celula 3</TD>
  <TD>Celula 4</TD>
</TR>
<CAPTION ALIGN=bottom>Titlul tabelului</CAPTION>
</TABLE>

<TABLE BORDER>
<TR>
  <TH ROWSPAN=2></TH>
  <TH COLSPAN=3>Browser</TH>
</TR>
<TR>
  <TH>Netscape</TH>
  <TH>Internet Explorer</TH>
  <TH>Mosaic</TH>
</TR>
<TR>
  <TH ROWSPAN=2>Element</TH>
  <TD>-</TD>
  <TD>X</TD>
  <TD>-</TD>
</TR>
<TR>
  <TD>X</TD>
  <TD>X</TD>
  <TD>X</TD>
</TR>
</TABLE>
</body>
</html>
```

Figura 5.2 prezintă imaginea celor două tabele vizualizată de Internet Explorer.

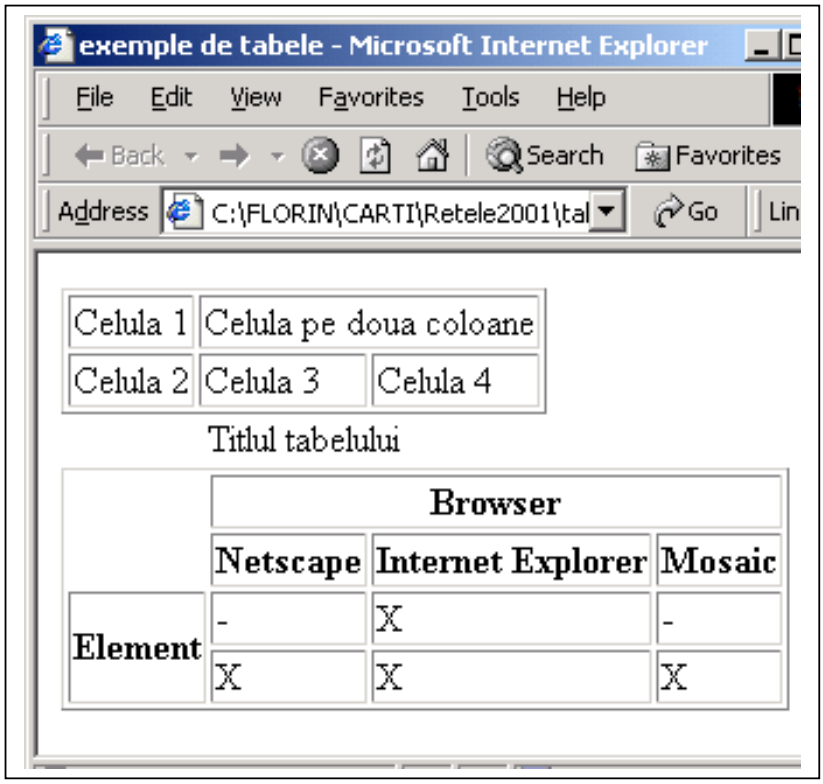



Figura 5.2. Imaginea a două tabele

### *4.6 Introducerea de imagini*

Navigatoarele actuale permit utilizarea formatelor grafice .GIF şi .JPG pentru încărcarea de imagini. Microsoft Internet Explorer permite în plus utilizarea formatelor grafice .PNG (portable network graphics) şi .BMP. Netscape şi Internet Explorer permit utilizarea fişierelor grafice multi-imagine .GIF, ceea ce permite crearea de secvențe animate.

Imaginile posibile într-un document HTML sunt de două tipuri:

Imagini *inline*, care apar direct în pagină odată cu încărcarea paginii. O astfel de imagine se specifică fie printr-un tag <IMG … >, fie printr-un alt mecanism, de exemplu într-un tag <INPUT ...> dintr-un formular. De exemplu, o specificare de forma:

```
<IMG Src = "romap.gif">
```

provoacă inserarea în textul paginii a hărții cu județe a Romaniei (fişierul se află în pagina cursului).

Imagini *externe*, care vor fi încărcate la cererea utilizatorului, printr-o ancoră corespunzătoare. De exemplu, o specificare de forma:

```
<A Href = "romap.gif">harta</A>
```

provoacă afişarea aceleiaşi hărți, într-o pagină separată, atunci când utilizatorul dă clic pe cuvântul `harta`.

**`<IMG ...>`** Se utilizează la introducerea de imagini. De remarcat că instrucțiunea nu are un element de genul `</IMG>`. Iată, în continuare, atributele mai importante:

| | |
|---|---|
| `SRC="url"` | Specifică URL-ul fişierului ce conține imaginea de afişat. |
| `WIDTH=număr` | Specifică lățimea imaginii în pixeli. |
| `HEIGHT=număr` | Specifică înalțimea imaginii în pixeli. |

De remarcat că, deşi nu este necesar, este bine să se folosească aceste ultime două atribute, deoarece astfel utilizatorul nu va aştepta preluarea completă a imaginii înainte de afişarea textului. La aceste ultimele două atribute se poate folosi şi % după număr, în acest caz imaginea se măreşte (număr peste 100%) sau se micşorează (număr sub 100%) corespunzător.

`BORDER=număr` Specifică grosimea bordurii imaginii, în pixeli. Valoarea implicită este 0 (fără bordură).

`ALT="text"` Specifică un text alternativ care se afişează în navigatoare ne-grafice, sau când navigatorul grafic are anulată facilitatea de încărcare a imaginilor.

`ALIGN=valoare` Specifică diferite moduri de aliniere a imaginii. Dintre toate valorile posibile, prezentăm aici următoarele:

`left` Deplasează imaginea la marginea stângă.

`right` Deplasează imaginea la marginea dreaptă.

`top` Se aliniază cu marginea de sus a elementului cel mai înalt din linia curentă.

`absmiddle` Aliniază mijlocul liniei curente cu mijlocul imaginii.

`absbottom` Aliniază partea de jos a liniei curente cu partea de jos a imaginii.

Exemplu:

```
<IMG SRC="univers.gif" ALT="Imaginea universitatii">
```

La vizualizare încărcarea unei imagini poate dura mai mult. Dacă dorim a vizualizare mai rapidă putem decupla în navigator încărcarea automată a imaginilor. În acest caz vizualizarea se face numai dacă facem un clic pe iconul care ascunde imaginea.
Se poate alege o soluție mai elegantă, şi anume includerea unei imagini mici, care se poate mări făcând un clic pe ea. De exemplu:

```
<A HREF="mare.gif">  <IMG SRC="mic.gif"> </A>
```

Aici "`mic.gif`" reprezintă o imagine de mărime mică care se vizualizează automat (dacă nu este decuplată opțiunea respectivă), şi se încarcă imaginea "`mare.gif`", care este aceeaşi imagine dar de mărime mai mare, şi care se încarcă în urma unui clic pe imaginea mică.

# 5 Interacțiuni HTML

Din cele descrise mai sus privind limbajul HTML, se poate descrie orice document care să fie găzduit pe un server Web şi să se navigheze prin el de către orice client Web folosind un navigator oarecare.

Se poate însă observa uşor că de la client nu se poate trimite nimic spre server, cu excepția comenzilor de încărcare de noi documente prin intermediul ancorelor.

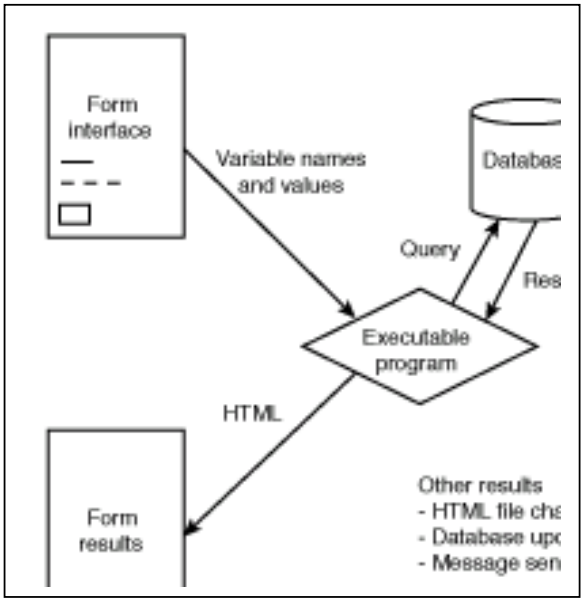
Există în HTML două posibilități prin care clientul poate transmite informații către server: formularele HTML şi hărțile senzitive asociate imaginilor. Le vom prezenta pe rând.

## *5.1 Formulare HTML*

### 5.1.1 Principiul de funcționare

Formularele sunt instrumentul principal HTML de interacțiune cu utilizatorul. El furnizează o serie de facilități grafice de preluare de informații de la utilizator, cum ar fi: căsuțe de dialog, butoane radio, liste de opțiuni etc. După exprimarea opțiunilor de către utilizator, acestea sunt transmise unui program specializat, numit CGI (Common Gateway Interchange). Acest CGI la rândul lui poate interacționa cu fişiere, baze de date etc. aflate pe serverul de Web. Ca rezultat al execuției, CGI dă la ieşire, spre browser, un nou text HTML. În figura 6.1 este prezentată această relație.

Figura 6.1 Relația CGI - HTML




Limbajele folosite pentru CGI pot fi: C, C++, Java (aplicații stand-alone), scripturi Shell, Scripturi Perl etc. Indiferent de limbaj, trebuie precizată modalitatea de primire a informațiilor de către CGI de la formular şi modalitatea de răspuns.

Datele de intrare sunt primite de CGI sub forma unei linii de text. Programul trebuie să privească această linie fie ca primul parametru al liniei de comandă (metoda *Get*) fie ca prima linie din fişierul de intrare standard (metoda *Post*).

Datele de ieşire din CGI sunt invariabil furnizate de către browser ca şi un text HTML sau orice alt tip de document inteligibil în browser, furnizat de CGI prin ieşirea standard.

### 5.1.2 Un exemplu simplu de formular

Exemplul pe care-l vom prezenta este următorul. Vom cere printr-un formular ca userul să-şi dea numele de user şi parola. Serverul va întoarce şirul de caractere pe care l-a primit de la formular.

Pentru aceasta, sunt necesare mai multe fişiere. Astfel, în subdirectorul `public_html` al directorului gazdă există fişierul `index.html`. Conţinutul acestuia este:

```
<HTML>
<HEAD>
<TITLE>Verificare user si parola</TITLE>
</HEAD>
<BODY>
<FORM METHOD="post" ACTION=/cgi-bin/TestCgi.cgi>
<P>Login:<INPUT NAME="Login"></INPUT>
Password:<INPUT TYPE="password" NAME="Password"></INPUT>
<INPUT TYPE="submit" VALUE="Login">
<INPUT TYPE="reset" VALUE="Cancel">
</FORM>
</BODY>
</HTML>
```

Fişierul `index.html`

Lansarea lui în execuţie cu Internet Explorer are ca efect imaginea din fig. 6.2.

După completarea câmpurilor `Login` şi `Password`, se va da clic pe butonul `Login`. Ca efect, va fi lansat în execuţie programul cu numele `TestCgi.cgi` situat în subdirectorul standard `cgi-bin` al directorului standard al serverului Web (de regulă, sub Linux, acesta este `/home/httpd/`). Acesta este de fapt un program CGI.

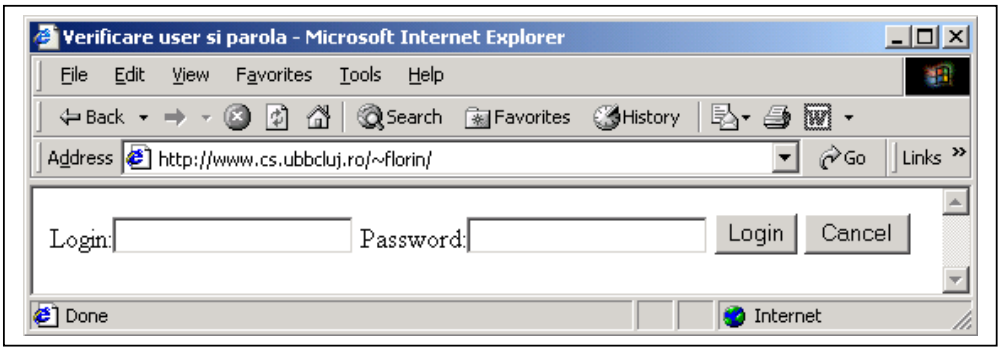



Figura 6.2 Imaginea iniţială a formularului

În acest exemplu, am folosit ca limbaj suport pentru CGI limbajul C sub Unix. Conținutul acestui program este uşor de înțeles, el tipărind la ieşirea standard, pur şi simplu textul HTML de răspuns:

```
#include <stdio.h>
main () {
   char DeLaFormular[1000];  // Aici se citeste mesajul formularului
     fgets(DeLaFormular, 1000, stdin); //Citeste din intrarea standard
                                       // linia primita de la formular
   printf("Content-type: text/html");// Trei linii standard
   printf("\n");                     // cerute de
   printf("\n");                     // protocolul HTTP

   printf("<HTML>");         // Urmeaza tiparirea la iesirea standard
   printf("<HEAD>");         // a documentului HTML de raspuns
   printf("<TITLE>");
   printf("Raspunsul de la CGI");
   printf("</TITLE>");
   printf("</HEAD>");
   printf("<BODY>");
   printf("<H3>");
   printf("CGI a primit stringul:");
   printf("</h3>");
     printf("%s",DeLaFormular);
   printf("</BODY>");
   printf("</HTML>");
  }
```

Fişierul `TestCgi.c`

Dacă transmiterea cererii se făcea folosind metoda GET, atunci în locul lui fgets s-ar fi pus:

```
    strcpy(DeLaFormular, getenv("QUERY_STRING"));
```

După compilare, fişierului executabil i se va atribui numele TestCgi.cgi şi va fi depus în directorul `~florin/public_html/cgi_bin`

După completarea numelui de user, aici `florin`, şi a unei parole, aici am dat drept parolă textul "`o parola oarecare`", se dă click pe butonul `Login`. Imaginea răspuns dată de CGI pe navigator va fi cea din fig. 6.3.

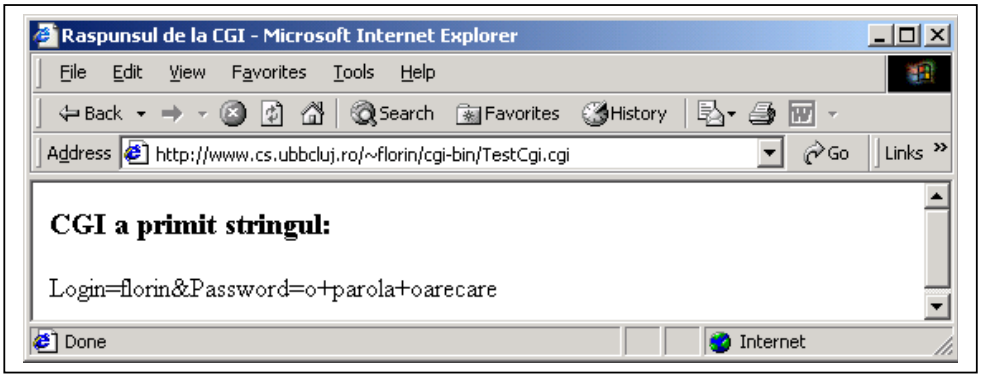



Figura 6.3 Rezultatul întors de CGI

In cazul apelului CGI prin metoda GET, URL-ul de apel ar fi fost:

```
http://www.cs.ubbcluj.ro/~florin/cgi-bin/TestCgi.cgi?Login=florin&Password=o+parola+oarecare
```

### 5.1.3 Principalele taguri folosite la formulare HTML

Tagul <FORM are sintaxa:

```
<FORM Action="URL" Method="get|post" [ Enctype="string" ] >
     Continutul formularului
</FORM>
```

`Action` indică URL-ul programului CGI care va fi lansat la execuţia formularului. Majoritatea serverelor Web pretind ca acest CGI să fie plasat într-un loc anume în structura de directori, iar drepturile asupra lui să fie acordate de către administrator. Evident, această cerinţă apare din motive de securitate. În exemplul de mai sus CGI este plasat în mod standard de către serverul Web Apache.

`Method` indică metoda care este folosită de formular spre a transmite informaţiile spre CGI. Aşa cum am arătat mai sus, prin `get` se transmite ca primul argument al liniei de comandă, iar prin `post` ca şi primă linie a intrării standard.

`Enctype` indică modalitatea de reprezentare a informaţiilor culese prin formular. De regulă acest atribut lipseşte, iar informaţiile sunt prezentate într-o formă standard. În cele ce urmează vom prezenta acest standard de reprezentare. CGI primeşte de la formular o singură linie de text ASCII, cu următoarele caracteristici:

Fiecare câmp al formularului furnizează o pereche `nume=valoare` (semnul = separă numele de valoare).
Două perechi nume=valoare sunt separate între ele prin semnul &.
Caracterele nonalfanumerice sunt înlocuite prin `%xx`, unde xx sunt două cifre hexazecimale reprezentând codul caracterului ASCII. In acest context, pentru a se indica separarea a două linii de informaţie, între ele apare %0D%0A.
Toate caracterele spaţiu din nume sau valoare sunt înlocuite prin caracterul +.

În conţinutul formularului se specifică, printre altele, tagurile <INPUT, <SELECT, <OPTION şi <TEXTAREA, pe care le vom prezenta în continuare. În corpul unui formular nu poate să fie definit un alt formular!.

Tagul <INPUT este instrumentul fundamental de specificare şi obţinere a intrărilor de la utilizator. Prin facilităţile lui, oferă practic toate modalităţile de intrare oferite de GUI actuale (de sub Windows, X Window etc.). Sintaxa tagului este:

```
<INPUT
 Type="text|password|radio|checkbox|image|hidden|submit|reset"
 Name="string"  Value="string"  Checked  Src="URL"  Text
 Size="n"  Maxlength="n"  Align="top|middle|bottom" >
```

Semnificaţia atributelor lui <INPUT este:

`Type` defineşte tipul câmpului de intrare prezentat utilizatorului. Fiecare dintre cele opt cuvinte posibile indică un mod de preluare a informațiilor, astfel:

`text` indică colosirea pentru intrare a unei singure linii de text. Se folosesc la dimensionare şi atributele `Size` şi `Maxlength` (dacă sunt prezente).
`password` indică preluarea unei parole, afişând în ecou caracterul *.
`radio` prezintă un set de butoane radio, din care se poate selecta doar o variantă. Dacă este prezent atributul `Checked`, atunci alternativa va fi selectată implicit.
`checkbox` permite selectarea multiplă la mai multe opțiuni.
`image` permite selectarea unui punct dintr-o imagine. La selectare se trimit automat (ca şi la tipul `submit`) perechile nume/valoare spre CGI. In ce priveşte poziția selectată din imagine, se furnizează două perechi nume.x=x nume.y=y, indicându-se coordonatele, în pixeli, ale punctului selectat din imagine.
`hidden` specifică o valoare setată de formular fără ca utilizatorul să dea vre-un input.
`submit` trimite perechile de informații către CGI.
`reset` resetează valorile formularului la valorile specificate implicit.

`Name` indică numele câmpului, nume care va fi transmis spre CGI ca primă componentă în perechile nume/valoare.

`Value` fixează o valoare implicită pentru câmp. `Checked` fixează o alegere implicită pentru o alternativă radio.

`Src` indică URL-ul atunci când avem `Type=image`.

`Text` indică faptul că este vorba de o singură linie text de intrare. La ENTER, se face automat submit.

`Size` indică dimensiunea afişată pentru intrarea de tip text. `Maxlength` fixează limita maximă a acestei lungimi.

Tagul <TEXTAREA are sintaxa:

```
<TEXTAREA  Name="string"  Rows="n"  Cols="n" >
```

Specifică o intrare text pe mai multe linii. Prin `Rows` şi `Cols` se specifică numărul de linii şi de coloane care dimensionează această zonă de text.

Ca şi <INPUT şi <TEXTAREA, tagul <SELECT este utilizat pentru a prelua informații de la utilizator. În inteiorul unui tag <SELECT pot să apară mai multe taguri <OPTIONS. Dintre opțiunile selectate se poate alege una şi numai una dintre ele. Sintaxa celor două taguri este:

```
<SELECT  Name="string"  Size="n"  Multiple >
     <OPTION  Selected  Value="n" >
     <OPTION  Selected  Value="n" >
     . . .
</SELECT>
```

`Name` este numele atribuit câmpului select. `Size` indică numărul de elemente vizibile la un moment dat dintre toate opțiunile posibile. `Multiple` permite selectarea mai multor valori dintre opțiunile oferite.

`Selected` indică o opțiune selectată în mod implicit. `Value` indică ce valoare va fi trimisă spre CGI în cazul selectării.

## 5.2 CGI şi comunicații prin URLConnection

### 5.2.1 Tehnologia CGI

Tehnologia CGI, acronim de la *Common Gateway Interface* este una din cele mai vechi interfațe standard de comunicare dintre o aplicație program şi un client Web, prin intermediul unui server Web. Astfel, serverul Web transmite cererea clientului, aplicației respective. Aceasta procesează cererea şi transmite rezultatul înapoi la serverul Web, care la rândul său, forwardează răspunsul la clientul care a lansat cererea.

Când utilizatorul solicită o pagină Web, serverul, ca răspuns, îi trimite înapoi pagina respectivă. Dar dacă utilizatorul completează un formular html, serverul transmite informațiile primite unui program CGI, care le procesează, după care trimite înapoi la server un mesaj de confirmare. Un document HTML este un fişier text static, care îşi păstrează o stare constantă. Pe de altă parte, un program CGI este executat în timp real şi generează în mod dinamic, un document HTML.

Un program CGI poate fi scris într-o diversitate de limbaje de programare, cele mai populare fiind: C, C++, Java, Perl, Unix shell,Visual Basic, AppleScript.

Aplicația distribuită este formată din trei nivele (aşa numită arhitectură three-tier):

1. Nivelul *client*, care furnizează, rând pe rând, către nivelul următor – serverul Web -, una sau mai multe linii de text care urmează a fi transformate.
2. Nivelul *server Web*, aflat în general pe o altă maşină decât clientul. Acesta preia liniile furnizate de client, le transmite nivelului CGI, acesta le va transforma, le va întoarce spre serverul Web, care la rândul lui le va comunica transformate clientului.
3. Nivelul *CGI* care execută efectiv transformarea liniilor de text.

Pentru descrierea plicației, programatorul trebuie să implementeze numai clientul şi CGI. Drept server Web îl vom folosi pe cel de pe maşina `linux.scs.ubbcluj.ro`.

## 5.3 Construcția unui contor de pagini Web

Pentru construirea unui counter, folosim un program CGI şi un applet care să comunice cu acesta. CGI-ul îşi întreține un fişier binar de 4 octeți care numărul ultimei consulări a unei anumite pagini. **Numele acestui fişier** este obținut din URL-ul CGI-ului, din care se păstrează doar literele şi cifrele numelui de CGI, iar toate literele mari sunt transformate în litere mici; la acest nume se mai adaugă sufixul `".count"`. De exemplu, dacă URL-ul este:

```
http://www.cs.ubbcluj.ro/~florin/cgi-bin/counterCGI.cgi
```

atunci fişierul întreținut de CGI are numele:

```
countercgicgi.count
```

### 5.3.1 Inițializarea fişierului contor

Programul 7.15 inițializează astfel de fişiere, preluând de la intrarea standard numele fişierului şi valoarea inițială:

```
#include <stdio.h>
#include <fcntl.h>
   main (int c, char *a[]) {
      int i, f;
      if (c != 3) {
        printf("Apel: counterInit numeFisier valoareInitialaContor");
        exit(1);
      }
      f = open(a[1], O_CREAT|O_WRONLY|O_TRUNC, 0666);
      i = atoi(a[2]);
      write(f, &i, sizeof(int));
      close(f);
   }
```

Programul 5.1 **Sursa de inițializare contor**

Pentru a putea fi apelat de către CGI, acest fişier trebuie să aibă cel puțin drepturile 0666!

### 5.3.2 Programul CGI

Acesta primeşte de la intrarea standard numele fişierului. Apoi îl deschide, citeşte valoarea curentă, o incrementează, o rescrie la loc şi închide fişierul. După aceea trimite înapoi răspunsul către apelatorul CGI-ului, în conformitate cu protocolul HTTP pentru transmiterea de text simplu. Răspunsul constă dintr-o linie (de maximum 10 caractere) care conține numai cifrele zecimale ce reprezintă valoarea curentă a contorului.

Textul sursă al CGI este prezentat în programul 7.16.

```
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
   extern int errno;
   main (int c, char *a[]) {
      int i=-1, f;
      char s[256];
      fgets(s, 256, stdin);   // Citeste numele fisierului
      f = open(s, O_RDWR);          // Deschide fisierul
      if (f > 0) {
         lockf(f, F_LOCK, 0);       // Blocheaza fisierul pentru incrementare
         read(f, &i, sizeof(int));  // Citeste numarul
         i++;                       // Incrementeaza
         lseek(f, 0, 0);            // Repozitioneaza la inceput
         write(f, &i, sizeof(int)); // Rescrie noul numar
         lockf(f, F_ULOCK, 0);      // Deblocheaza fisierul
         close(f);                  // Inchide fisierul
      }
      else
         perror("");
      printf("Content-type: text/plain\n\n%d\n", i);  // Trimite numarul la client
   }
```

Programul 5.2 **Sursa `counterCGI.c`**

Pentru cazul nostru, acest fişier executabil va fi fişierul:

```
~/public_html/cgi-bin/counterCGI.cgi
```

# 6 Socket: utilizare din C sub Unix şi din C++ sub Windows

## 6.1 Noțiunea de socket

Un *socket* este un "*capăt de comunicație*" sau "*punct final de comunicație*", care poate fi numit şi adresat într-o rețea. Este locul unde *programul de aplicație se întâlneşte cu furnizorul mediului de transport*. Din perspectiva unui program de aplicație, un socket este *o resursă alocată de sistemul de operare*. Din punctul de vedere al unui sistem de operare, este *o structură de date întreținută de către nucleul sistemului de operare*. Această structură este referită în programele de aplicație printr-un întreg *numit descriptor de socket*.

Două aplicații pot comunica între ele prin socket doar după ce fiecăreia dintre ele sistemul de operare i-a alocat un socket. Există un număr de descriptori de socket rezervați pentru aplicațiile de sistem, restul descriptorilor fiind disponibili pentru utilizatori.

Înterfața socket ascunde utilizatorului detaliile rețelei fizice, ea este caracterizată numai prin serviciile pe care le asigură. Comunicația prin socket este parte integrantă a nivelului *sesiune* din cadrul modelului OSI.

In prezent, mecanismul socket este operațional pe toate sistemele de operare de tip Unix: SunOS, Solaris, SCO, LINUX etc. Incepând cu 1990, au apărut pachete care permit utilizarea acestui mecanism şi sub toate variantele Microsoft WINDOWS: 3.1, 3.11, 95, NT. Mai mult, după cum se va vedea în cele ce urmează, între cele două clase de sisteme de operare compatibilitatea este perfectă.

## 6.2 Tipuri de socket

Din punctul de vedere al transportului efectuat, se disting două tipuri de socket: *socket stream* şi *socket datagram*. Comunicarea este posibilă numai dacă la ambele capete este utilizat acelaşi tip de socket!

### *Socket stream*

Înterfața *socket stream* defineşte un serviciu orientat conexiune. Datele sunt transmise fără erori şi fără duplicări. Ele sunt recepționate în aceeaşi ordine în care au fost transmise. Fluxul de date prin rețea este realizat astfel încât să se evite depăşirea unei valori maxime de octeți la un moment dat. Nu se impun nici un fel de limitări asupra datelor, acestea considerându-se simple şiruri de octeți.

Prin analogie cu comunicația cotidiană, un socket stream este analog cu un aparat telefonic, la care a) proprietarul formează un număr şi după stabilirea legăturii începe să discute cu persoana de la celălalt capăt al firului; b) aparatul plus întreaga rețea telefonică asigură stabilirea legăturii şi asigură stabilitatea acesteia până la închiderea unuia dintre cele două aparate.

Pentru socket stream transportul este realizat folosind protocolul TCP. Fiind vorba de serviciu orientat conexiune, este potrivită analogia cu aparatul telefonic. Unul din protocoalele nivelului aplicație, şi anume File Transfer Protocol (FTP), foloseşte serviciile oferite de socket stream.

### *Socket datagram*

Interfața *socket datagram* definește un serviciu fără conexiune în care datagramele sunt transmise ca şi pachete separate. Acest serviciul nu asigură garanții ale recepționării datelor, care pot fi pierdute ori duplicate, iar datagramele pot ajunge în altă ordine decât cea în care au fost transmise. Mărimea unei datagrame este limitată de numărul de octeți care pot fi transmişi într-o singură tranzacție. Nu se realizează nici o dezasamblare şi reasamblare a pachetelor.

Prin analogie cu comunicația cotidiană, un socket datagram poate fi asemănat cu o cutie poştală în care: a) proprietarul ei (programul de aplicație) depune corespondența de trimis şi de unde îşi ridică corespondența trimisă; b) factorul poştal (furnizorul mediului de transport) deschide cutia de câteva ori pe zi, ridică corespondența de expediat şi depune corespondența sosită la adresa respectivă.

Pentru socket datagram transportul este realizat folosind protocolul UDP. Analogia cu cutia poştală este potrivită acestui tip de socket. Dintre protocoalele nivelului aplicație, TFTP şi NFS se bazează pe un astfel de socket.

### *Repere pentru alegerea tipului de socket*

În vederea alegerii unui anumit tip de socket trebuie luate în considerare mai multe aspecte. Primul dintre acestea este că realizarea unei comunicații cu o aplicație existentă impune folosirea aceluiaşi protocol ca şi aplicația respectivă. De exemplu, dacă aplicația în cauză foloseşte protocolul TCP, atunci se va utiliza socket stream.

Un alt aspect va fi cel legat de siguranța şi eficiența comunicației. Socket stream asigură o conexiune sigură, acest lucru realizându-se însă cu o minimă reducere a performanței, deoarece sunt necesare eforturi de calcul suplimentare pentru menținerea conexiunii şi verificările de corectitudine. Din fericire, la nivel utilizator acest lucru nu se observă. Socket datagram este relativ nesigur deoarece pachetele pot fi ignorate, alterate sau duplicate în timpul transmisiei. Se poate utiliza astfel de socket dacă aplicația implementează ea însăşi un mecanism propriu de corectitudine, sau dacă aplicația nu este prea sensibilă la transmisii eronate. Câştigul constă în viteza de prelucrare, sporită în raport cu cea de la socket stream.

Cantitatea de date care se transferă este încă un indiciu de alegere: pentru cantități mari de date se vor utiliza socket stream.

In fine, dacă comunicația se desfăşoară într-o rețea LAN, atunci de cele mai multe ori socket datagram este suficient. Pentru rețele WAN este mai recomandabilă folosirea socket stream.

## *6.3 Mecanismul de socket în limbajul C*

Intregul mecanism socket este disponibil, indiferent de platformă, prin intermediul unor tipuri de date şi a unor funcții apelabile din limbajul C, eventual din limbajul C++. Evident, această soluție asigură un grad foarte înalt de portabilitate, deoarece aceste limbaje sunt astăzi cele mai răspândite, fie că este vorba de platforme de tip Unix, fie că este vorba de platforme Microsoft WINDOWS. Evident, celelalte platforme existente astăzi în lume sunt aliniate la una dintre aceste două platforme.

Din rațiuni de portabilitate, constantele folosite drept parametri pentru socket sunt numere întregi. Structurile de date folosite sunt definite prin typedef în aceste fişiere header şi respectă sintaxa limbajului C. Din păcate, există unele mici deosebiri, dar din fericire ele se referă numai la numele fişierelor header folosite, care pot să difere de la o platformă la alta.

Astfel, succesiunea citărilor de headere de sub Unix:

```
#include <netinet/in.h>
#include <sys/socket.h>
#include <netdb.h>
```

au fost înlocuite în pachetul WINSOCK printr-o singură citare:

```
#include <winsock.h>
```

In continuare vom cita headerele folosite sub Unix, urmând ca acolo unde este cazul (mai ales la aplicații) să indicăm alternativele.

Tipurile de date folosite în sistemul Unix se găsesc în headerul `<sys/types.h>`, motiv pentru care practic acest header trebuie citat în fiecare program.

### 6.3.1 Adrese socket

Inainte de a detalia utilizarea socket, trebuie să indicăm modalitatea prin care două procese depărtate pot să ia legătura. Pentru realizarea unei astfel de comunicații este necesară existența unui cvintuplu:

```
{Protocol,AdresaLocala,ProcesLocal,AdresaDepartata,ProcesDepartat}
```

Pentru a-şi păstra independența de platformă, socket permite utilizarea mai multor *familii de adrese*. O familie de adrese defineşte un stil de adresare. Toate hosturile care sunt în aceeaşi familie de adrese înțeleg şi folosesc aceeaşi schemă pentru adresarea capetelor socketului. Familiile de adrese sunt identificate prin numere întregi. Numele acestora începe cu AF (Adress Family).

Cea mai importantă familie de adrese este `AF_INET`, familie ce defineşte adresarea în domeniul Internet. Pe lângă acest tip de familie, sistemul Unix recunoaşte şi alte tipuri, cum ar fi AF_UNIX care reprezintă sistemul local (protocoale interne Unix), AF_NS pentru protocoale XEROX NS, AF_IMPLINK pentru IMP (Interface Message Procesor).

Generic, structura unei adrese socket este următoarea:

```
struct sockaddr {
    u_short     sa_family;        /* valoare AF_xxx */
    char        sa_data[14];      /* dependent de familie */
}
```

şi ea este definită în `<sys/socket.h>`.

Câmpul `sa_family` indică familia de adrese folosită.

Conținutul celor 14 octeți ai câmpului `sa_data` este interpretat în funcție de familia de adrese folosită. De fapt, după cum vom vedea, unele familii folosesc chiar mai mult de 14 octeți în acest scop.

Fiecare familie de adrese are definită structura proprie de adresă, structură care se suprapune peste cea generică în momentul transmiterii ei către nucleu. Să ne oprim pentru moment la două familii de adrese: `AF_INET` şi `AF_UNIX`.

Adresele din familia `AF_INET` sunt de tip `struct sockaddr_in` şi sunt definite în `<netinet/in.h>` astfel:

```
struct in_addr {
    u_long      s_addr;     /* pe 32 biti o adresa IP */
}
```

```
struct sockaddr_in {
    short             sin_family;       * valoarea AF_INET */
    u_short           sin_port;         /* numarul de port */
    struct in_addr    sin_addr;         /* adresa IP */
    char  s           in_zero[8];       /* nefolosit */
}
```

De remarcat faptul că structura `sockaddr_in` include în interiorul ei structura `in_addr`.

Adresele din familia AF_UNIX sunt de tip `struct sockaddr_un` şi sunt definite `în <sys/un.h>` astfel:

```
struct sockaddr_un {
    short       sun_family;       /* AF_UNIX */
    char        sun_path[108];    /* cale spre fisier Unix*/
}
```

Trebuie remarcat că ambele structuri de adrese (ca şi structurile celorlalte familii) încep cu numărul indicator de familie. Deci, schimbând indicatorul de familie aceeaşi zonă de memorie poate fi interpretată ca un alt tip de adresă, aşa cum se vede în fig. 1.1

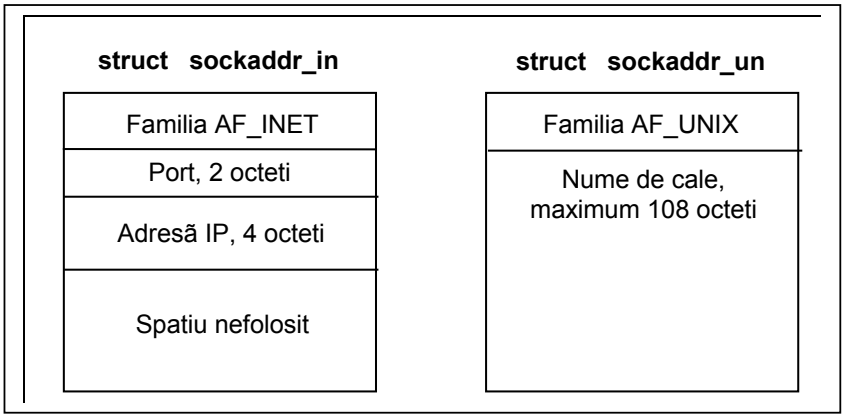



**Figura 6.1 Suprapunerea a două adrese socket**

## *6.4 Scheme cadru de implementări client / server*

Programarea aplicațiilor care folosesc socket urmează nişte scenarii bine stabilite. Este vorba de ordinea de apelare a funcțiilor sistem în funcție de tipul de socket şi de faptul că un program este pe post de server sau pe post de client. (De multe ori, mai ales atunci când se foloseşte socket datagram, dispare înțelesul clasic de client şi de server).

### 6.4.1 Scenariul aplicațiilor socket stream

Pentru aplicațiile care folosesc protocol cu servicii orientate conexiune, deci socket stream, se folosesc câteva apeluri sistem  Unele dintre ele sunt apelate de către server, altele de către client. Succesiunea acestora este dată în fig. 1.2

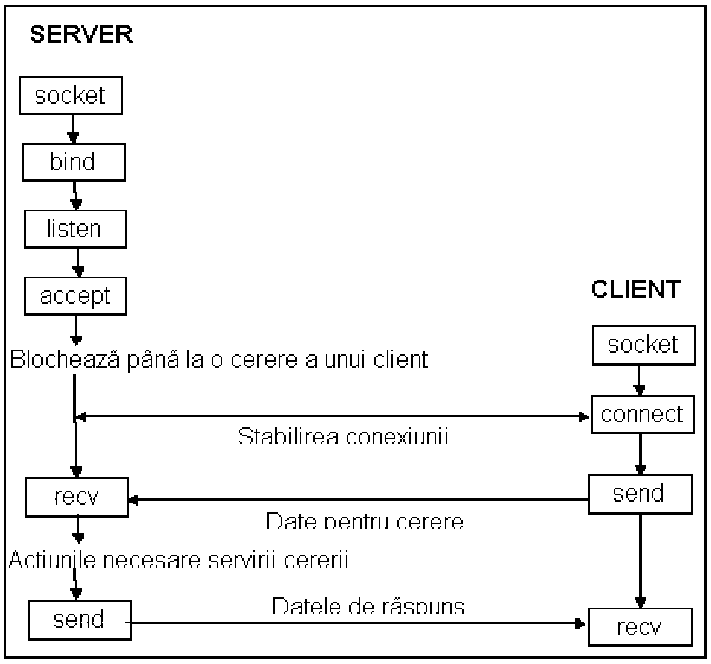



**Figura 6.2 Scenariul aplicațiilor socket stream**

Vom prezenta acum pe scurt rolul acestora, urmând să descriem complet fiecare dintre apelurile sistem într-o secțiune următoare.

Atât serverul, cât şi clientul au câte o parte de inițializare după care urmează descrierile acțiunilor curente.

***Partea de inițializare a serverului***
`socket` cere nucleului sistemului să creeze un socket prin care să poată fi contactat de către clienți. Aici se fixează familia de adrese folosită, precum şi tipul de socket folosit: stream sau datagram.
`bind` transmite nucleului numărul de port la care serverul aşteaptă să fie contactat. precum şi faptul că poate fi contactat prin orice interfață de rețea de pe mașina server.
`listen` cere nucleului să dimensioneze lungimea cozii de aşteptare la socket.

***Partea de inițializare client***
`socket` cere nucleului sistemului să creeze un socket prin care să poată contacta de către clienți. Aici se fixează familia de adrese folosită, precum şi tipul de socket folosit: stream sau datagram.

***Relația curentă între un client şi un server***

La un moment dat ***clientul*** execută un apel sistem `connect`, prin care îi transmite nucleului lui adresa IP şi numărul de port ale serverului căruia doreşte să-i ceară un serviciu. Nucleul, prin mecanismul socket contactează serverul solicitat.

***Serverul***, odată cu lansarea apelului sistem `accept`, rămâne în aşteptare până când un client îl contactează. Odată contactat, el primeşte adresa IP şi numărul de port ale clientului care l-a contactat.

***Clientul*** execută un apel sistem `send` (sau mai multe) prin care transmite serverului un mesaj prin care îi cere un serviciu. ***Serverul***, prin apelul (apelurile) sistem `recv` recepționează acest mesaj.

Apoi, ***Serverul*** execută acțiunile necesare satisfacerii cererii clientului.

Trimiterea rezultatului de către ***server*** către ***client*** se face similar: serverul trimite prin apeluri `send`, iar clientul recepționează prin apeluri `recv`.

In secțiunile care urmează vom descrie pe larg apelurile sistem şi vom da exemple de folosire a lor în aplicații.

Trebuie precizat (detaliile mai târziu) că apelurile sistem send şi recv se comportă ca şi funcții Unix care execută operații de intrare/ieşire obişnuite; printre argumente nu mai apar adrese IP, nici numere de porturi. Ele sunt fixate în momentul contactării `connect - accept`.

## 6.4.2 Scenariul aplicațiilor socket datagram

La fel ca în secțiunea precedentă, vom prezenta acum pe scurt rolul acestora, urmând să descriem complet fiecare dintre apelurile sistem într-o secțiune următoare. Succesiunea acestora este ilustrată în fig. 3.

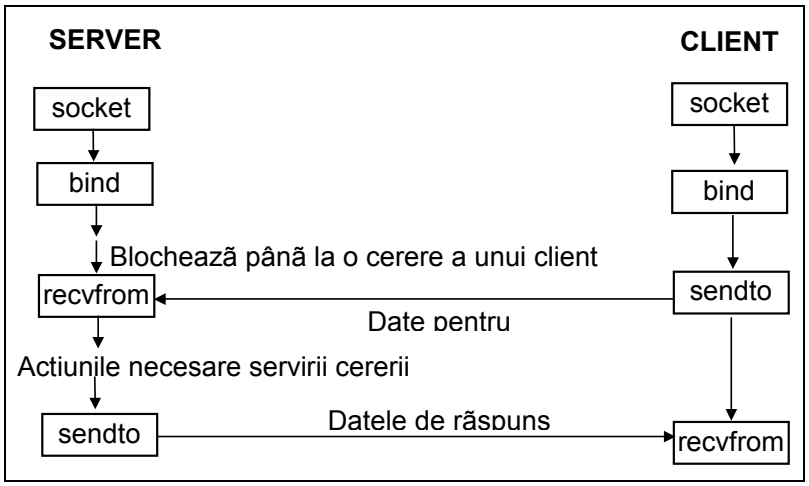



**Figura 6.3 Scenariul aplicațiilor socket datagram**

***Părțile de inițializare ale serverului şi ale clientului***
`socket` cere nucleului sistemului să creeze un socket prin care să poată comunica cu partenerii. Aici se fixează familia de adrese folosită, precum şi tipul de socket folosit: stream sau datagram.
`bind` transmite nucleului numărul de port prin care se va comunica.

***Relația curentă între un client şi un server***
La un moment dat ***clientul*** execută un apel sistem `sendto`, prin care îi transmite nucleului lui adresa IP şi numărul de port ale serverului căruia doreşte să-i ceară un serviciu, precum şi mesajul prin care îi cere serverului un serviciu. Nucleul, prin mecanismul socket contactează serverul solicitat şi-i transmite mesajul. In cazul în care cererea este compusă din mai multe mesaje, ele sunt trimise succesiv prin apeluri independente.

***Serverul***, odată cu lansarea apelului sistem `recvfrom`, rămâne în aşteptare până când un client îl contactează. Odată contactat, el primeşte adresa IP, numărul de port ale clientului care l-a contactat precum şi mesajul prin care i se cere un serviciu.

Apoi, ***Serverul*** execută acțiunile necesare satisfacerii cererii clientului.

Trimiterea rezultatului de către ***server*** către ***client*** se face similar: serverul trimite prin apeluri `sendto`, iar clientul recepționează prin apeluri `recvfrom`, eventual prin mesaje succesive, fiecare dintre ele purtând adresele IP de contact şi numerele de port.

In secţiunile care urmează vom descrie pe larg apelurile sistem şi vom da exemple de folosire a lor în aplicaţii.

## 6.5 Biblioteca de apeluri sistem socket

### 6.5.1 Apeluri socket de conexiune

*Sintaxele apelurilor: socket, bind, listen, connect, accept*

Prototipurile acestor apeluri sistem sunt date mai jos. Pentru simplificarea prezentării rolurilor acestora, acelaşi parametru a fost notat cu acelaşi nume în toate prototipurile de apeluri.

Aceste apeluri sistem trebuie precedate de specificarea fişierelor header:

```
# include <sys/types.h>
# include <sys/socket.h>
```

Prototipurile de apel sunt:

```
int  socket ( int familie, int tip, int protocol );

int  bind   ( int sd, struct sockaddr *local, int lunglocal );
                      /*intoarce  0 la succes, -1 la eroare */

int  listen ( int sd, int lungcoada );
                      /*intoarce  0 la succes, -1 la eroare */

int  connect( int sd, struct sockaddr *departe, int *lungdeparte);
                      /*intoarce  0 la succes, -1 la eroare */

int  accept ( int sd, struct sockaddr *departe, int *lungdeparte);
```

In caz de eşec (parametri eronaţi, conectare imposibilă etc.) sunt semnalate prin întoarcerea unei valori negative. Pentru detalierea tipului de eroare, trebuie să fie citat headerul:

```
# include <errno.h>
```

şi variabila:

```
extern  int  errno;
```

Parametrul `sd` care apare la ultimele patru apeluri este descriptorul de socket, valoare care este întoarsă de către apelul sistem `socket`.

Parametrii `local` şi `departe` reprezintă pointeri către adrese de socket, care sunt structuri de tipul celor descrise în anterior. Asociaţi acestora, parametrii `lunglocal` şi `lungdeparte` sunt întregi sau pointeri la întregi care indică lungimea de reprezentare a acestor adrese.

In continuare, prezentăm pe rând fiecare dintre cele cinci apeluri sistem.

Exemplele lor de utilizare vor fi în concordanţă cu schemele din fig. 1, 2 şi vor fi date în aplicaţiile prezentate în acest capitol.

### *Apelul sistem socket*

Acest apel sistem crează un socket care va asigura conectivitatea programului local, fie el server sau client. Apelul întoarce un întreg, notat în cele ce urmează cu sd, care reprezintă descriptorul de socket. Apelurile sistem care urmează folosesc acest descriptor ca şi parametru.

Parametrul `familie` indică familia de adresă socket utilizată. Printre altele, valoarea ei poate fi `AF_INET`, `AF_UNIX` etc. Se pot folosi şi constantele echivalente `PF_INET`, `PT_UNIX` etc. Pentru restul acestei lucrări vom folosi numai constanta `AF_INET`.

Parametrul `tip` indică tipul de socket folosit. Pentru socket stream acesta are ca valoare constanta `SOCK_STREAM`, iar pentru socket datagram constanta `SOCK_DGRAM`. Mai există şi alte valori posibile, dar pentru acestea nu sunt admise toate combinațiile familie - tip.

Parametrul `protocol` este un argument care are valoarea 0, cu semnificația "lăsăm sistemului alegerea protocolului". Există însă, aplicații specializate unde se cere specificarea. De exemplu, pentru `familie == AF_INET` sunt posibile combinațiile:

| tip | Protocol | Protocol implicit |
|---|---|---|
| SOCK_STREAM | IPPROTO_TCP | TCP |
| SOCK_DGRAM | IPPROTO_UDP | UDP |

Constantele de protocol sunt definite în `<netinet/in.h>`.

### *Apelul sistem bind*

Acest apel sistem se utilizează în mai multe contexte:

1. ***Serverul*** îşi înregistrează adresa socket proprie `local`, pentru socketul `sd`; lungimea adresei `local` este `lunglocal`. In `local` trebuie trecut, spre a fi transmis nucleului, numărul de port la care poate fi contactat. Acesta poate fi un număr de port rezervat sau un număr la dispoziția utilizatorului). Tot în `local`, în câmpul `sin_addr.s_addr` se poate trece fie adresa IP a maşinii server, fie, de cele mai multe ori, constanta `INADDR_ANY` cu semnificația: "pot fi contactat pe orice interfață de rețea de pe maşina pe care lucrez".

2. Un ***client*** legat prin socket datagram `sd` indică nucleului, prin `local` de lungime `lunglocal`, numărul de port şi adresa IP la care poate fi contactat. La această adresă unică îi vor răspunde toate serverele pe care le va contacta pe socket `sd`.

### *Apelul sistem listen*

Precizează numărul maxim `lungcoada` de conexiuni care sunt acceptate pe socket `sd` să aştepte dacă serverul este ocupat. Dacă sosesc mai multe, cele ce depăşesc `lungcoada` vor fi refuzate.

Majoritatea sistemelor de operare nu admit mai mult de cinci clienți simultan.

### *Apelul sistem connect*

Un proces ***client*** care foloseşte socket stream `sd`, solicită nucleul propriu să conecteze, prin `sd`, serverul având adresa socket `departe` reprezentată pe `lungdeparte` octeți.

Acest apel sistem stabileşte pentru client patru din cele cinci elemente ale conexiunii, protocolul fiind stabilit anterior. La cele mai multe protocoale, `connect` întoarce rezultatul abia după stabilirea conexiunii.

Apelul este folosit la legătura de tip conectare. Se poate folosi şi pentru legături fără conectare, pentru a obține adresa serverului.

Adresa IP a clientului este cunoscută şi pusă de către nucleul lui, care atribuie, în mod temporar pe durata conexiunii, a unui număr de port.

*Apelul sistem accept*

Acest apel sistem pune ***serverul*** care aşteaptă o conexiune socket stream `sd`. Serverul rămâne în aşteptare până când un client reuşeşte un connect la el. In momentul realizării conexiunii, în argumentele `departe` şi `lungdeparte` sunt întoarse adresa socket a clientului şi lungimea pe care se reprezintă această adresă. Apelul sistem întoarce trei valori:

1. Un întreg pe care-l vom nota `fd`. Acesta reprezintă un nou descriptor, pe care serverul îl foloseşte pe post de descriptor de fişier pentru schimbul de mesaje cu clientul.
2. Adresa socket a procesului client în `departe`.
3. Lungimea de reprezentare a acestei adrese în `lungdeparte`.

Descriptorul fd întors de accept se comportă ca un descriptor de fişier şi numai prin el serverul manevrează mesajele în ambele sensuri. Pentru o distincție mai clară, se folosesc termenii de socket (descriptor) de întâlnire pentru `sd` şi socket (descriptor) de schimb pentru `fd`.

## 6.5.2 Particularități socket Windows

In continuare, precizăm câteva particularități de utilizare a bibliotecii WINSOCK sub Windows.

Pentru portarea Unix -> WINDOWS a unui program care comunică prin socket sunt necesari şi obligatorii următorii doi paşi:

1. Se vor înlocui liniile cu citarea fişierelor header citate mai sus. Deci:

```
# include <netinet/in.h>
# include <fcntl.h>
# include < sys/socket.h>
# include <netb.h>
```

se va înlocui cu:

```
# include <winsock.h>
```

2. Orice aplicație WINDOWS care foloseşte WINSOCK trebuie să inițializeze biblioteca socket. Aceasta se realizează declarând trei variabile şi apelând două funcții. Apelurile trebuie să fie primele instrucțiuni în cadrul funcției main a programului. Secvența de inițializare este:

```
WORD  wVersiune;
WSADATA     wDate;
int         wEroare;

wVersiune = MAKEWORD(1,1);     /* Se cere versiunea 1.1 */
wEroare = WSAStartup( wVersiune, &wDate); /* Initbiblioteca */
```

Incepând cu WINDOWS 95 se pot genera aplicații consolă la care să fie legată biblioteca pe 32 de biți wsock32.lib. Sub Visual C++ 4.2, opțiunea de includere se află în ierarhia de meniuri `Build/Settings/Link`.

## 6.5.3 Operații de I/O prin socket

*Apelurile sistem send, recv, sendto, recvfrom*

Acestea sistem sunt cele mai simple apeluri sistem prin intermediul cărora se pot face schimburi de date între client şi server prin socket. Utilizarea lor presupune citarea fişierelor header `<sys/types.h>` şi `<sys/socket.h>`. Sintaxele acestor apeluri sunt:

```
int send    (int sd,char *T,int n,int f );
int sendto  (int sd,char *T,int n,int f,struct sockaddr *D,int lD);
int recv    (int sd,char *T,int n,int f);
int recvfrom(int sd,char *T,int n,int f,struct sockaddr *E,int *lE);
```

Parametrul `sd` este descriptorul de socket prin care se realizează comunicația cu partenerul depărtat. Acest întreg provine, aşa cum am văzut, dintr-un apel `socket`, cu excepția cazului `send` şi `recv` dintr-un server socket stream, când acesta provine din apelul unui `accept`.

Apelurile `send` şi `sendto` expediază prin `sd` un număr de maximum `n` octeți pe care-i ia din memoria internă începând cu adresa `T`. Programatorul trebuie să depună în zona tampon de adresă `T` aceşti `n` octeți. In cazul `send`, fiind vorba de socket stream, adresa IP şi portul destinatarului se stabileşte la conexiune. In cazul `sendto`, programatorul trebuie să depună în prealabil adresa IP şi numărul de port al partenerului destinatar în zona punctată de `D` pe o lungime de `lD` octeți. De multe ori, adresa IP şi portul destinatarului se găsesc deja în zona `D` dacă aceeaşi zonă este folosită şi la apelul pereche precedent.
Intregul întors de cele două apeluri reprezintă numărul de octeți care s-a putut transmite. In caz de eroare se întoarce o valoare negativă.

Apelurile `recv` şi `recvfrom` recepționează prin `sd` un număr de maximum `n` octeți pe care-i depune în memoria internă începând cu adresa `T`. Programatorul trebuie să se asigure că zona tampon are rezervați la adresa `T` cel puțin `n` octeți. In cazul `recv`, fiind vorba de socket stream, adresa IP şi portul destinatarului se stabileşte la conexiune. In cazul `recvfrom`, programatorul primeşte adresa IP şi numărul de port al partenerului destinatar în zona punctată de `D` şi lungimea ei de reprezentare la adresa `lD`.
Intregul întors de cele două apeluri reprezintă numărul de octeți care au fost recepționați. Faptul că s-a închis conexiunea este semnalat prin întoarcerea valorii zero. In caz de eroare se întoarce o valoare negativă.

Argumentul `f` este un flag care are de regulă valoarea 0, lăsând astfel pe seama sistemului fixarea unor flaguri. Este posibil totuşi ca el să fie compus din reuniunea (OR) a unora din constantele:

MSG_OOB pentru a emite sau recepționa cu bufferizare;
MSG_PEEK pentru a bloca mesajele până la recepția completă a celui curent;
MSG_DONTROUTE nu se rutează mesajul între rețele.

Observație: tratarea situațiilor în care nu s-au transmis sau recepționat exact cei n octeți solicitați cade exclusiv în sarcina progrramatorului! Dacă prin socket se poate transmite fie şi numai un octet, acesta se va transmite şi operația este considerată încheiată! Un remediu posibil este, de exemplu, funcțiile Send şi Recv.

*Comparații cu apelurile read şi write*

Putem compara operațiile de I/O clasice cu fişiere şi operațiile I/O prin socket prezentate mai sus. Pentru fişiere există apelurile sistem *open, create, close, read, write*. Versiunea inițială a *TCP/IP* din 1981 folosea aceste funcții. S-a constatat însă că nu sunt suficient de bune pentru comunicația în rețele. Iată câteva dintre motive.

1. Relația *client-server* nu este simetrică. O conectare în rețea cere programului să ştie ce rol *(client sau server)* joacă.
2. Conectarea în rețea poate fi bazată pe conexiune sau fără conexiune. Relația *open-close* este orientată spre conexiune, în timp ce relația fără conexiune nu are echivalent.
3. Spre deosebire de fişiere, în rețele *numele* este esențial. Pentru fişiere este suficientă transmiterea unui număr - *descriptor de fişiere*. Pentru comunicații este esențial numele perechii pentru a verifica autoritățile de acces.
4. Parametrii fişierelor sunt mult mai puțini decât cei ce se transmit prin rețea. Astfel, pentru rețea este necesară asocierea celor cinci componente cunoscute:

```
(protocol,adresalocala,portlocal,adresadepartata,portdepartat)
```

asociere care nu-şi găseşte echivalent la operațiile I/O cu fişiere.
5. Pentru unele protocoale de comunicație este semnificativă *dimensiunea limitată a înregistrărilor*. Pentru I/O, aspectul este acela al fluxului de octeți.
6. Interfețele de rețea trebuie să suporte mai multe *protocoale de comunicație.*

### *Apelurile sistem sendmsg şi recvmsg*

Apelurile sistem I/O mai sus amintite nu asigură faptul că toți octeții solicitați vor fi transferați. Pentru a se asigura această cerință prin operații atomice, sunt folosite aceste două apeluri sistem. Ele sunt descrise în headerele <sys/types.h> şi <sys/socket.h> şi au prototipurile:

```
int sendmsg ( int sd, struct msghdr msg, int f );
int recvmsg ( int sd, struct msghdr msg, int f );
```

Semnificația parametrilor sd şi f coincide cu cea a apelurilor sistem precedente. Structura msghdr este:

```
struct    msghdr {
    caddr_t     msg_name;         /*Adresa opțională*/
    int         msg_namelen;      /* lungime adresă*/
    struct      iove   *msg_iov;  /* bucățile de memorie */
    int msg_iovlen;               /* câte bucati sunt */
    caddr_t     msg_accrights;    /* drepturi de acces schimb */
    int   msg_accrightslen;       /* lungine zona de drepturi */
}
```

Primele două câmpuri sunt folosite de către recvfrom şi sendto, pentru lucrul în mod neconexiune. Pentru conexiune se pune *NULL*. Ultimele două elemente sunt folosite pentru *drepturile de acces între procese*.

### *Rutine de conversii ale octeților şi întregilor*

Aceste rutine asigură ordonarea octeților din reprezentarea unui întreg în concordanță cu arhitecturile şi protocoalele cu care se lucrează. De regulă, întregii scurți sunt reprezentați pe 16 biți, iar cei lungi pe 32 biți. Implementările depind de maşină şi sunt de una dintre categoriile: *big endian* sau *little endian*.. In capitolul următor, dedicat RPC, vom detalia această problemă.

La nivelul comunicațiilor prin socket, întregii sunt tratați ca fiind fără semn. Este definită o reprezentare "universală" a lor, numită *reprezentare de rețea*. Folosind fişierele header <sys/types.h> şi <netinet/in.h>, conversia întregilor între reprezentarea de rețea se face cu ajutorul următoarelor patru funcții.

```
u_long  htonl( u_long  I );       /* host to network long */
u_short htons( u_short I );       /* host to network short */
```

```
u_long  ntohl( u_long  I );       /* network to host long */
u_short ntohs( u_short I );       /* network to host short */
```

Funcțiile transformă argumentul `I` şi întorc întregul convertit, astfel:

Funcția `htonl` converteşte un întregul lung de rețea `I` într-un întreg lung local.
Funcția `htons` converteşte un scurt lung de rețea `I` într-un întreg scurt local.
Funcția `ntohl` converteşte un întreg lung local `I` într-un întreg lung de rețea.
Funcția `ntohs` converteşte un întreg lung local `I` într-un întreg lung de rețea.

*Rutine de manevrare a şirurilor de octeți*

Frecvent sunt necesare o serie de operații asupra octeților, privite ca şiruri de octeți ce nu se supun întotdeauna regulilor limbajului C. Aceste funcții fac parte din pachetul de rutine standard C. Deoarece patru dintre ele sunt destul de des folosite în exemplele care urmează, le prezentăm pe scurt.

```
memcpy(char *s, char *d, int n);

memset(char *d, int v, int n);

strcpy( char *d, char*s);

int strcmp(char *a, char *b);
```

`memcpy` copiază `n` octeți de la adresa `s` la adresa `d`. `memset` depune începând cu adresa `d`, `n` octeți cu conținut identic, valoarea `v`. `strcpy` copiază un şir de octeți reprezentat ca în C, de la adresa `s` la adresa `d`. strcmp compară şirurile `a` şi `b`, reprezentate ca în C şi întoarce o valoare negativă dacă `a < b`, o valoare pozitivă dacă `a > b` şi valoarea 0 dacă `a = b`.

## 6.5.4 Gestiunea adreselor IP şi Internet

Pentru conversia acestora sunt necesare headerele `<sys/socket.h>`, `<netinet/in.h>` şi `<arpa/inet.h>`.

*Rutinele inet_addr şi inet_ntoa*

Prototipurile acestor două rutine sunt:

```
unsigned long inet_addr ( char *S );
char    *inet_ntoa      ( struct in_addr I );
```

Rutina `inet_addr` transformă şirul de caractere `S` scris în convenție C şi care conține o adresă IP scrisă în notația punctuală, într-o adresă pe 32 biți a cărei valoare o întoarce.

Rutina `inet_ntoa` face operația inversă, adică transformă întregul lung `I` într-un şir de caractere care conține adresa IP în notația punctuală.

Să considerăm, spre exemplu, secvența de mai jos, unde d este un pointer:

```
strcpy( d, inet_toa( (struct in_addr) inet_addr("193.226.40.34")))
```

Prin aceasta, la adresa `d` se va depune şirul "`193.226.40.34`".

*Rutinele gethostname şi gethostbyname*

```
#include <unistd.h>
int gethostname(char *name, size_t len);
```

Funcția `gethostname` returnează în parametrul de ieşire `name`, de lungime maximă `len`, numele maşinii curente, pe care este conectat utilizatorul.

Adresele IP sunt mai dificil de manipulat de către utilizator şi este de preferat să se folosească în locul lor adresele Internet. După cum se poate însă vedea, rutinele socket lucrează cu adrese IP.

Rutina `gethostbyname` oferă posibilitatea de a obține adresa IP echivalentă cu o adresă Internet. Evident, pentru aceasta ea va provoca apelurile în lanț ale DNS-urilor necesare obținerii acestei adrese. Prototipul acestei rutine este:

```
struct hostent *gethostbyname(char *S);
```

S este şirul, în convenție C care conține adresa Internet.
Structura `hostent`, către care rutina întoarce un pointer, se defineşte astfel:

```
struct hostent {
    char  *h_name;          /* Numele oficial al hostului */
    char  **h_aliases;      /* Lista de aliasuri */
    int   h_addrtype;       /* Tipul adresei hostului */
    int   h_length;         /* Lungimea adresei */
    char  **h_addr_list     /* lista de adrese IP date de DNS */
}
```

Această structură este destul de flexibilă pentru a reține hosturi cu nume multiple şi cu mai multe adrese IP. Câmpul `h_addr_list` este un pointer la aceste adrese IP, scrise în notația punctuală.

In cazul cel mai simplu, şi cel mai des folosit, se va prelua prima adresă IP. Pentru a simplifica accesul la aceasta, odată cu structura se defineşte şi macroul:

```
#define  h_addr  h_addr_list[0]
```

In aproape toate exemplele care urmează determinarea adreselor IP se va face cu această rutină pornind de la adrese Internet.

## 6.6 Exemple de aplicații client / server cu socket

In această secțiune vom prezentaun exemplu simplu de socket de tip stream, sub Unix.

### 6.6.1 rdir: Rezumat de director la distanță

*Prezentarea problemei*

Se cere să se elaboreze o pereche de programe: client şi server, ale căror roluri sunt următoarele.

Clientul primeşte ca parametri în linia de comandă adresa Internet a serverului şi numele unui director despre care se presupune că se află în sistemul de directori al serverului. Clientul contactează serverul, îi transmite acestuia numele de director şi îi cere ca răspuns rezumatul directorului.

In ipoteza că directorul primit există, serverul face rezumatul acestuia şi trimite acest rezumat clientului solicitant. Dacă directorul nu există, sau dacă serverul nu are drepturi de acces la el, va întoarce clientului şirul vid.

Clientul va afişa pe ieşirea standard rezumatul primit de la server.

In continuare prezentăm implementarea acestei probleme, folosind socket-uri stream.

### *O variantă rdir prin TCP*

Implementăm comunicația prin socket de tip stream (orientată conexiune, utilizează nivelul de transport TCP) şi în consecință respectăm întocmai scenariul din fig. 2.

Varianta `rdir`, care ține cont de observațiile de mai sus, este compusă din patru fişiere.

Programul 1.1 prezintă o funcție, care primeste la intrare numele unui director şi intoarce, prin acelasi parametru, lista fisierelor din el.

```
#include <dirent.h>
/* Pentru Windows
#include <windows.h>
*/

int
read_dir (char *dir)
/* Primeste la intrare numele unui director si intoarce,
 * prin acelasi parametru, lista fisierelor din el
 */

{
  DIR *dirp;
  struct dirent *d;
  dirp = opendir (dir);
  dir[0] = '\0';
  if (dirp == NULL)
    return (0);
  while (d = readdir (dirp))
    sprintf (dir, "%s%s\n", dir, d->d_name);
  closedir (dirp);

  /* Pentru Windows
  WIN32_FIND_DATA d;
  HANDLE hf;
  BOOL b;

  strcpy(director, dir);
  strcat(director, "\\*.*");
  dir[0] = '\0';
  hf = FindFirstFile(director, &d);
  b = hf != INVALID_HANDLE_VALUE;
  while (b) {
    sprintf (dir, "%s%s\n", dir, d.cFileName);
    b = FindNextFile(hf, &d);
  }
 */

  return (sizeof (dir));
}
```

**Programul 6.1 Sursa funcției read_dir.c**

Al doilea fişier, programul 1.2, este un antet în care sunt definite constantele număr de port, lungimea bufferului pentru rezumat şi macroul de tratare a erorilor.

```
  #include <stdio.h>
  #include <string.h>
  #include <sys/types.h>
  #include <sys/socket.h>
  #include <netinet/in.h>
  #include <netdb.h>
  #include <signal.h>
  #include <errno.h>

  extern int errno; /*Indicatorul de erori ale apelurilor sistem
*/

  #define ERR(S,N) {fprintf(stderr,"\n%d ",N);\
  perror(S); exit(N);}

  #define PORT_SERVER 12347
  #define DIRSIZE 8192
```

**Programul 6.2 Sursa fişierului header rd.h**

In continuare, prin programele 1.3 şi 1.4 sunt prezentate clientul şi serverul. In acest exemplu, numele maşinii şi directorul solicitat serverului sunt furnizate, de fiecare dată, la lansarea în execuţie a fiecărui client.

In al doilea rând, adresa maşinii server este furnizată printr-o adresă IP. Aşa cum am mai spus, sunt de preferat adresele Internet. În consecință programul client trebuie să convertească adresa Internet în adresă IP.

Al treilea aspect este cel al controlului corectitudinii apelurilor sistem. Se ştie că toate funcțiile C întorc câte o valoare. Semnificaţia valorilor întoarse de apelurile sistem socket a fost deja prezentată. Dacă în apelurile sistem obişnuite se mai permite citarea unui apel `F` pe post de instrucțiune:

```
F ( … ) ;
```

în apelurile sistem de comunicaţii, datorită probabilităţii mai mari de apariţie a unor situaţii neobişnuite / erori, este indicată folosirea acestor apeluri în contextul:

```
if ( F ( … ) … ) …
```

În acest mod se va detecta mai uşor apariţia unei siuaţii excepţionale. În programele noastre, tratarea acestora se face simplu: un macrou `ERR` dă un mesaj pe ieşirea standard, după care programul se termină cu un anumit cod de eroare.

Sursa aplicaţiei client este dată în programul 1.3 `(rdct.c)`.

```
  #include <rd.h>
  /* Pentru Windows
  #include <winsock.h>
  */

  main (int argc, char *argv)
  {
    char dir[DIRSIZE];
    int sd;
    struct sockaddr_in serv_addr;
    struct hostent *hp;

  /* Pentru Windows
    WORD wVR;
```

```
  WSADATA wD;
  int we;

  wVR = MAKEWORD(1,1);
  we = WSAStartup(wVR, &wD);
*/

/* Deschide un socket */
  if ((sd = socket (AF_INET, SOCK_STREAM, 0)) < 0)
    ERR ("socket", 1);

/* Obtine adresa IP a serverului */
  if (!(hp = gethostbyname (argv[1])))
    ERR ("gethostbyname", 2);

/* Pregateste adresa pentru conexiune */
  memset ((char *) &serv_addr, 0, sizeof (serv_addr));
  serv_addr.sin_family = AF_INET;
  memcpy (hp->h_addr, (char *) &serv_addr.sin_addr.s_addr,
      hp->h_length);
  serv_addr.sin_port = htons (PORT_SERVER);

/* Solicita conexiunea la server */
  if (connect (sd, (struct sockaddr *) &serv_addr,
           sizeof (serv_addr)) < 0)
    ERR ("connect", 3);

/* Emite o cerere catre server */
  if (send (sd, argv[2], strlen (argv[2]), 0) != 0)
    ERR ("send", 4);

/* Asteapta raspunsul de la server */
  if (recv (sd, dir, DIRSIZE, 0) < 0)
    ERR ("recv", 5);

/* Tipareste rezultatul */
  printf ("%s\n", dir);
  close (sd);

/* Pentru Windows
    closesocket (sd);
*/
}
```

**Programul 6.3 Sursa clientului rdct.c**

In programul client, datele de intrare se preiau din linia de comandă: `argv[1]` conține adresa Internet a serverului şi `argv[2]` conține numele directorului. A doua adăugire esențială este obținerea adresei IP din adresa Internet a serverului (folosind apelul `gethostbyname`) şi depunerea acesteia în adresa socket.

De remarcat faptul că în programul 1.3 sunt câteva linii de comentariu care prezintă adăugirile necesare pentru rularea clientului sub Windows.

Programul 1.4 este sursa serverului (`rdst.c`). Acest server aşteaptă într-o buclă conexiunile clienților, iar pentru fiecare nou client conectat, creează un proces fiu pentru tratarea interacțiunii respective.

```
#include "read_dir.c"
#include <rd.h>

main ()
{
  char dir[DIRSIZE];
  int sd, fd, len_addr;
```

```
    struct sockaddr_in serv_addr, clie_addr;

    signal (SIGCHLD, SIG_IGN);/* Pentru evitare zombie */

  /* Creeaza un socket */
    if ((sd = socket (AF_INET, SOCK_STREAM, 0)) < 0)
      ERR ("socket", 1);

  /* Pregateste adresa server */
    memset ((char *) &serv_addr, 0, sizeof (serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl (INADDR_ANY);
    serv_addr.sin_port = htons (PORT_SERVER);

  /* Leaga sd de adresa server */
    if (bind (sd, (struct sockaddr *) &serv_addr,
          sizeof (serv_addr)) != 0)
      ERR ("bind", 2);

  /* Listen */
    if (listen (sd, 5) != 0)
      ERR ("listen", 3);

  /* Bucla principala server */
    for (;;) {

  /* Accepta o conexiune de la un client */
      if ((fd = accept (sd, (struct sockaddr *) &clie_addr,
                  &len_addr)) < 0)
        ERR ("accept", 4);

  /* Primeste cererea de la client */
      if (recv (fd, dir, DIRSIZE, 0) < 0)
        ERR ("recv", 5);

  /* Deserveste clientul in proces separat */
      if (fork () == 0) {
        close (sd);
        read_dir (dir);

    /* Trimite raspunsul la cerere */
        if (send (fd, dir, strlen (dir), 0) < 0)
      ERR ("send", 6);

        close (fd);
        exit (0);
      }
    }
  }
```

**Programul 6.4 Sursa serverului rdst.c**

In continuare, se compilează sursa programului server şi lansează în execuție, în background:

```
$  cc  -o  rds  rdst.c
$  rds &
```

## 6.6.2 Client FTP noninteractiv

După cum se ştie de la cursul de rețele de calculatoare, FTP este un protocol de comunicații situal la cel mai înalt nivel în modelul OSI. Prin urmare, FTP este un protocol la nivel de aplicație. Acest protocol este conceput pentru a fi folosit:

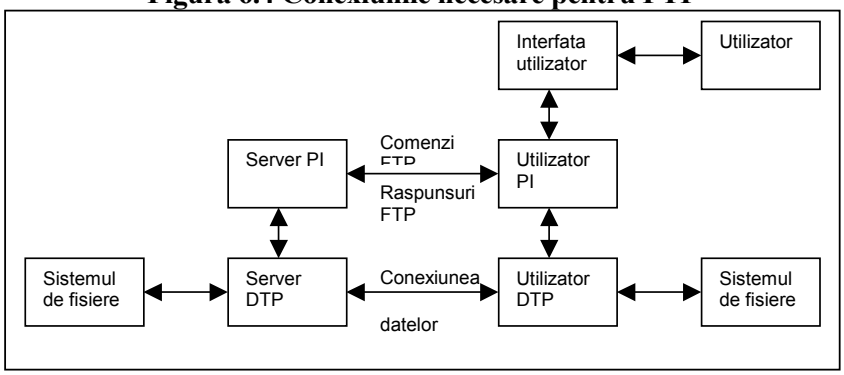
- prin intermediul liniei de comandă;
- prin intermediul interfețelor grafice;
- din cadrul aplicațiilor proprii.

Protocolul FTP utilizează o terminologie proprie, definită prin specificațiile RFC959, aflate la adresa `http://www.faqs.org/rfcs/rfc959.html`:

- `access controls` – defineste drepturile de acces ale utilizatorului la sistem
- `control connection` –calea de comunicație între cele două interpretoare de protocoale: USER-PI şi SERVER-PI, pentru schimbul de comenzi.
- `data connection` – conexiune ful duplex prin care de transferă datele
- `DTP` – Data Transfer Process
- `End-of-Line` – separatorul de sfârşit de linie
- `EOF` – condiția de sfârşit de fişier
- `PI` – interpretorul de protocol implementat atât de partea serverului cât şi de partea clientului

Modelul conexiunii FTP este ilustrat în fig. 1.4.

**Figura 6.4 Conexiunile necesare pentru FTP**




După cum se vede protocolul FTP are nevoie de două conexiuni pentru a putea transfera date, una pentru schimbul de comenzi între PI-urile client şi server şi una pentru transferul de date.

### *Aplicatia "Client FTP.cpp"*

Aplicația următoare este un client FTP noninteractiv folosit pentru trimiterea fişierelor de pe maşina locală pe un server la distantă. Noninteractivitatea este dată de posibilitatea de a configura parametri de lucru (adrese, parole, directoare şi fişiere) cu ajutorul unui fişier de configurare.

In momentul de față acest client de FTP este o parte din nucleul aplicațiilor de căutare de persoane în rețeaua UBBNET, este utilizat în aplicația de consultare a notelor pentru cursanții învățământului la distanță, respectiv face parte din nucleul aplicațiilor de backup al serverelor de comunicații din rețeaua UBBNET.

Sursa **`ClientFTP.cpp`**

```
#include <condefs.h>
#pragma hdrstop
#pragma argsused
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <stdlib.h>
#include <winsock.h>
#include <fcntl.h>
```

```
#include <io.h>

   int configReadFtp(char *configfile, int port,
                     char *ip, char *lip,
                     char *user, char *pass,
                     char *dir, char *ldir,
                     char *files, int *nrfiles) {
   // Functia de citire a configuratiei clientului ftp.
   // Citirea se face din fisierul specificat de configfile
   // Intoarce 0 la configurare reusita sau -1 la esec
      int i;
      FILE *fConf;
      char temp[255], *p, *q, *r;
      if ((fConf=fopen(configfile, "r"))==NULL)
         return (-1); // Codul -1 = eroare la citirea fisierului de configurare
      for ( (*nrfiles)=0, r=files; ; ) {
         p = fgets(temp, 255, fConf); // Citeste o linie din fisierul de configurare
         if (p == NULL)
            break; // terminat citirea fisierului
         for( q=p+strlen(p)-1; ((*q)<=' ')&&(q-p>0); q--);
         *(q+1) = '\0'; // Ignora sfarsit de linie
         if (strlen(p) <=2)
            continue; // Ignorarea liniilor scurte
         if (p[0] == '#')
            continue; // Ignorarea liniilor comentariu
         if ((q = strstr(p, "=")) == NULL) {
            strcpy(r, temp); // Copiaza nume fisier de transferat
            r += strlen(r) + 1; // la fisierul urmator
            (*nrfiles)++;
            continue;
         } //Linie nume de fisier
         q++;
         for (i=0; i<q-p; i++)
            p[i] = tolower(p[i]); // transformat nume in litere mici
         if (strstr(p, "localip=") != NULL) {
            for (i=0; i<strlen(q); i++)
               if (q[i] == '.')
                  q[i] = ','; //IP local +port
            strcpy(lip, q);    // au octetii separati
            strcat(lip, ",");  // prin virgula
            itoa(port/256, temp, 10);
            strcat(lip, temp);
            strcat(lip, ",");
            itoa(port%256, temp, 10);
            strcat(lip, temp);
            continue;
         } //IPlocal
         if (strstr(p, "serverip=") != NULL) {
            strcpy(ip, q); // Copiaza IP server
            continue;
         } //IPserver
         if (strstr(p, "user=") != NULL) {
            strcpy(user, q); // Copiaza user
            continue;
         } //user
         if (strstr(p, "password=") != NULL) {
            strcpy(pass, q); // Copiaza parola
            continue;
         } //password
         if (strstr(p, "remotedir=") != NULL) {
            strcpy(dir, q); // Copiaza director la distanta
            continue;
         } //remotedir
         if (strstr(p, "localdir=") != NULL) {
            strcpy(ldir, q); // Copiaza director local
```

```
            continue;
         } //localdir
      }//for
      fclose(fConf);
      return(0); // terminarea normala a fixarii configuratiei
   }//configReadFtp

   int authFtp(struct sockaddr_in *saddr, char *ip, char *user, char *pass) {
   // Functia de autenticicare a unui client FTP.
   // Deschide conexiunea de control si autentifica ip, user si parola.
   // Intoarce >0 (descriptorul socket) la succes sau <0 la esec/
      int sock;
      char mesaj[1024];

   // creare socket
      sock = socket(AF_INET,SOCK_STREAM,0);
      if (sock < 0)
         return (-1); // Eroare la creare socket
      memset((char *)saddr, 0, sizeof(struct sockaddr));
      saddr->sin_family=AF_INET;
      saddr->sin_addr.s_addr=inet_addr(ip);
      saddr->sin_port=htons(21);

   // conectarea la server
      if(connect (sock, (struct sockaddr*)saddr, sizeof(struct sockaddr_in)) < 0)
         return (-2); // Eroare la connect

   //preluare mesaj initial server
      if (recv(sock, mesaj, 1024, 0) <= 3)
         return (-3); // eroare rcv mesaj initial
      if (strstr(mesaj, "220") != mesaj)
         return(-4); // Trebuie sa raspunda 220 - awaiting input

   //Trimitere nume de user la server si asteptare raspuns
      strcpy(mesaj,"user ");
      strcat(mesaj, user);
      strcat (mesaj,"\n");
      if (send(sock, mesaj, strlen(mesaj), 0) != strlen(mesaj))
         return (-5); // transmitere eronata user
      if (recv(sock, mesaj, 1024, 0) <= 3)
         return (-6); // eroare raspuns user
      if (strstr(mesaj, "331") != mesaj)
         return(-7); // Trebuie sa raspunda 331 - User OK

   //Trimitere parola la server si asteptare raspuns
      strcpy(mesaj,"pass ");
      strcat(mesaj, pass);
      strcat (mesaj,"\n");
      if (send(sock, mesaj, strlen(mesaj), 0) != strlen(mesaj))
         return (-8); // transmitere eronata parola
      if (recv(sock, mesaj, 1024, 0) <= 3)
         return (-9); // eroare raspuns parola
      if (strstr(mesaj, "230") != mesaj)
         return(-10); // Trebuie sa raspunda 230 - User login

   // Intoarce socketul de control
      return (sock);
   } //authFtp

   int setContextFtp(int sock, char *type, char *dir) {
   //Functia de fixare a contextului FTP: tipul transferului:
   // ascii sau binar (type)
   // directorul curent de pe serverul ftp (dir)
   // Intoarce 0 la context fixat sau <0 la esec.
      char mesaj[1024];
```

```
   //Trimite comanda TYPE si asteapta confirmarea
      strcpy(mesaj,"TYPE ");
      strcat(mesaj, type);
      strcat(mesaj, "\n");
      if (send(sock, mesaj, strlen(mesaj), 0) != strlen(mesaj))
         return (-11); // transmitere eronata type
      if (recv(sock, mesaj, 1024, 0) <= 3)
         return (-12); // eroare raspuns type
      if (strstr(mesaj, "200") != mesaj)
         return(-13); // Trebuie sa raspunda 200 - Command Ok

   //Trimite comanda CWD si asteapta confirmarea
      strcpy(mesaj,"CWD ");
      strcat(mesaj, dir);
      strcat(mesaj,"\n");
      if (send(sock, mesaj, strlen(mesaj), 0) != strlen(mesaj))
         return (-14); // transmitere eronata director curent
      if (recv(sock, mesaj, 1024, 0) <= 3)
         return (-15); // eroare raspuns cwd
      if (strstr(mesaj, "250") != mesaj)
         return(-16); // Trebuie sa raspunda 250 - Requested file action Ok

      return (0); // context fixat
   }//setContextFTP

   int putAsciiFtp(int sockc, char *ldir, int port, char *files, int nrfiles) {
   //Deschidere conexiunea de date. Socketul asculta pe masina locala pe
   // portul port. La el se va conecta serverul de FTP pentru a realiza
   // transferul efectiv al fisierelor.
   // Intoarce 0 la derulare normala,
   // <0 la erori de conexiune sau
   // >0 (al catelea fisier este in transfer) la erori de transfer fisiere .

   // Aceasta functie este conceputa pentru a face put de fisiere text (ASCII),
   // deci fisierele locale sunt tratate cu: fopen, fgets, fputs, fclose.
   // Pentru a face operatii get in loc de put, se va transmite RETR in loc de STOR.
   // Pentru schimburi (get, put) in mod binar, se vor folosi functiile:
   // open, read, write, close.

      char mesaj[1024], *r;
      struct sockaddr_in sockd; // Socket de asteptare a transferului de date
      struct sockaddr_in socke; // Socket pe care ftp trimite date
      int socketd, sockete, fd, lung, i;
      FILE *ft;

      socketd = socket(AF_INET, SOCK_STREAM, 0);
      if (socketd < 0)
         return(-20); // eroare la deschidere socket de date
      memset((char *)&sockd, 0, sizeof(sockd));
      sockd.sin_family = AF_INET;
      sockd.sin_addr.s_addr = htonl(INADDR_ANY);
      sockd.sin_port = htons(port);

   // rezervarea portului in kernel
      if( bind(socketd,(struct sockaddr *) &sockd, sizeof(sockd)) <0)
         return (-21); //Eroare la  bind
      if (listen(socketd, 2) < 0)
         return (-22); // Eroare la listen

   //Se trimit continuturile fisierelor
      for (i=0, r=files; i<nrfiles; i++, r+=strlen(r)+1) {
         // Pentru fiecare fisier specificat in fisierul de configurare
         // se creeaza o conexiune inversa pentru transmiterea fisierelor
```

```
         // Trimite comanda PORT si asteapta raspunsul
         strcpy(mesaj,"port ");
         strcat(mesaj, ldir);
         strcat(mesaj,"\n");
         if (send(sockc, mesaj, strlen(mesaj), 0) != strlen(mesaj))
            return (-23); // transmitere eronata port
         if (recv(sockc, mesaj, 1024, 0) <= 3)
            return (-24); // eroare raspuns port
         if (strstr(mesaj, "200") != mesaj)
            return(-25); // Trebuie sa raspunda 200 - Command Ok

        // Trimite comanda STOR si asteapta raspunsul
         strcpy(mesaj,"stor ");
         strcat(mesaj, r);
         strcat(mesaj,"\n");
         if (send(sockc, mesaj, strlen(mesaj), 0) != strlen(mesaj))
            return (-26); // transmitere eronata stor
         if (recv(sockc, mesaj, 1024, 0) <= 3)
            return (-27); // eroare raspuns stor
         if (strstr(mesaj, "150") != mesaj)
            return(-28); // Trebuie sa raspunda 150 - Open ascii connection

         // Exista conexiune, trimite un fisier
         lung = sizeof(socke);
         fd=accept(socketd,(struct sockaddr *) &socke, &lung);
         if (fd < 0)
            return (-30); // eroare la accept
         if ((ft = fopen(r, "r")) == NULL)
            return (-31); // eroare la deschiderea fisierului

            // Bucla de trimitere a continutului fisierului
         for ( ;fgets(mesaj, 1024, ft) != NULL; )
            if (send(fd, mesaj, strlen(mesaj), 0) != strlen(mesaj))
               return (i+1); // eroare la transmiterea celui de-al i-lea fisier
         fclose(ft);
         close(fd);
         shutdown(fd,2);

            // Raspunsul de la transmiterea unui fisier
         if (recv(sockc, mesaj, 1024, 0) <= 3)
            return (-33); // eroare raspuns transmitere fisier
      }//end for

      close(socketd);
      return (0); // terminare normala
   }//putFTP

   int main(int argc, char* argv[]) {
   // Plecand de la fisierul de configurare CONFIG_FILE se transfera o
   // serie de fisiere ASCII de la client la un server de FTP

   // Macrou pentru erori
   #define ERR(S,N) {fprintf(stderr,"\n%d ",N);perror(S);exit(abs(N));}
   // Constante
   #define PORT 1234
   #define CONFIG_FILE ".\\config.ftp"

      int socketc, nrFile;
      struct sockaddr_in sockc; // Socket de conexiune (la port 21)
      char IP[1024], LIP[1024], USER[1024], PASS[1024],
         DIR[1024], LDIR[1024], argFile[50*255];

      WORD wVersiune;
      WSADATA wDate;
      int wEroare, i;
```

```
      char *r;

      if ((i = configReadFtp(CONFIG_FILE, PORT, IP, LIP, USER, PASS, DIR, LDIR,
                           argFile, &nrFile)) < 0)
         ERR("Eroare la citirea fisierului de configurare", i);

      wVersiune = MAKEWORD((1),(1));
      wEroare = WSAStartup(wVersiune, &wDate);
      if (wEroare!=0)
         ERR("Winsock not supported..\n", wEroare);

      if ((socketc = authFtp(&sockc, IP, USER, PASS)) < 0)
         ERR("Eroare de autentificare FTP", socketc);

      if ((i = setContextFtp(socketc, "A", DIR)) < 0)
         ERR("Eroare la stabilire context FTP", i);

      if ((i = putAsciiFtp(socketc, LIP, PORT, argFile, nrFile)) != 0)
         ERR("Eroare la transferurile de fisiere", i);

      close(socketc);
      printf("S-au transferat pe server fisierele:\n");
      for (i=0, r=argFile; i<nrFile; i++, r+=strlen(r)+1 )
         printf("%s\n", r);
      printf("Pentru continuare tastati <ENTER>. Pentru intrerupere <CTRL/C>");
      gets(IP);
      return 0;
   }//main()
```

Programul CGI **`GetNote.c`** este:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

   main(int c, char *a[]) {
   // CGI: Primeste la intrare un numar de serie, un numar de semestru
   // si un cod de acces al unui student: prima litera din nume,
   // prima din prenume si numarul matricol.
   // Intoarce pagina html care cuprinde notele, sau un mesaj de eroare
      char antet[1024], student[1024], *p, *q,
         fisier[50], ser[50], sem[50], cod[50], wcod[50];
      int i;
      FILE *fe;
      strcpy(antet, getenv("QUERY_STRING"));
      strcat(antet, "&");
      p = strstr(antet, "ser=");
      for (p+=4, i=0; *p != '&'; p++, i++)
         ser[i] = *p;
      ser[i] = '\0';
      p = strstr(antet, "sem=");
      for (p+=4, i=0; *p != '&'; p++, i++)
         sem[i] = *p;
      sem[i] = '\0';
      p = strstr(antet, "cod=");
      for (p+=4, i=0; *p != '&'; p++, i++)
         cod[i] = *p;
      cod[i] = '\0';
      for (i=0; i<strlen(cod); i++)
         cod[i] = tolower(cod[i]);
      strcpy(fisier, ser);
      strcat(fisier, ".");
      strcat(fisier, sem);
      fe = NULL;
```

```
      if ((strlen(ser)<1) || (strlen(sem)<1) || (strlen(cod)<3))
         goto ERR; // serie, semestru sau cod eronate
      fe = fopen(fisier, "r");
      if (fe == NULL)
         goto ERR; // serie/semestru eronate
      fgets(antet, sizeof(antet), fe);
      antet[strlen(antet)-1] = '\0'; // sters \n
      for ( ; ; ) { // cautarea studentului
         p = fgets(student, sizeof(student), fe);
         if (p == NULL)
            goto ERR; // nu s-a gasit studentul
         student[strlen(student)-1] = '\0'; // sters \n
         wcod[0] = student[0];
         p = strchr(student, '\t') + 1;
         wcod[1] = p[0];
         p = strchr(p, '\t') + 1;
         for (i=0; p[i] != '\t'; i++)
            wcod[i+2] = p[i];
         wcod[i+2] = '\0';
         for (i=0; i< strlen(wcod); i++)
            wcod[i] = tolower(wcod[i]);
         if (strcmp(cod, wcod) == 0)
            break; // l-a gasit
      } // sfarsit ciclul de cautare a studentului
      fclose(fe);

   // Scrierea textului de raspuns pozitiv
      printf("Content-type: text/html\n\n");
      printf("<html><head><title>");
      printf("Note de la seria %s semestrul %s", ser, sem);
      printf("</title></head><body><h3>");
      printf("Note de la seria %s semestrul %s", ser, sem);
      printf("</h3><table>");
      for ( p=antet, q=student; ; ) {
         for ( i=0; p[i] != '\t'; i++)
            ser[i] = p[i];
         ser[i] = '\0';
         p += i+1;
         for ( i=0; q[i] != '\t'; i++)
            sem[i] = q[i];
         sem[i] = '\0';
         q += i+1;
         printf("<tr><td align=\"right\">%s: </td><td><b>%s</b></td></tr>",
               ser, sem);
         if (strchr(p, '\t') == NULL)
            break;
      } // terminat scriere tabel
      printf("</table>");
      goto STOP; // spre terminarea CGI

   ERR:
   // Scrierea textului de raspuns negativ
      if (fe != NULL)
         fclose(fe);
      printf("Content-type: text/html\n\n");
      printf("<html><head><title>");
      printf("Note de la seria %s semestrul %s", ser, sem);
      printf("</title></head><body><h3>Date eronate!</h3><p>");
      printf("<i>Ati solicitat:</i> ");
      printf("Seria: <b>%s</b> ", ser);
      printf("Semestrul: <b>%s</b> ", sem);
      printf("Codul: <b>%s</b>", cod);
      printf("<p><h4>Aceste date nu sunt toate corecte, ");
      printf("sau nu s-a transmis inca fisierul cu note!</h4>");
   STOP:
```

```
      printf("</body></html>");

   } // main
```

Textul client **`CerNote.html`** este:

```
<html><head><title>Solicitare note</title></head>
<body>
<form method="get" action="./cgi-bin/GetNote.cgi">
<h3>Pentru obtinerea unor note, completati formularul:</h3>
<table>
<tr><td>Numarul seriei de curs (cifre arabe):</td>
<td><input type="text" name="ser"></td></tr>
<tr><td>Numarul semestrului (cifre arabe):</td>
<td><input type="text" name="sem"></td></tr>
<tr><td>Codul de acces:</td>
<td><input type="text" name="cod"></td></tr>
<tr><td><input type="reset" value="Anuleaza completarea"></td>
<td><input type="submit" value="Solicita notele"></td></tr>
</table>
<p><em>Precizarea 1: </em>Desi se obisnuieste scrierea lor si cu
cifre romane, numarul seriei si numarul semestrului se cer
cu cifre arabe.
<p><em>Precizarea 2: </em>Codul de acces este format din doua
litere (indiferent ca sunt mari sau mici), urmate de numarul matricol.
Prima litera este prima litera a primului nume de familie, in
conformitate cu buletinul, <em>pentru doamne se va lua in considerare
numele de fata</em>. A doua litera este prima litera a primului prenume,
tot in conformitate cu buletinul.
<p><em>Exemplu: </em>Pentru a obtine notele din seria IV, sem III, Doamna
avand numele de familie
<i><b>I</b>onescu - Galati (cas Vasilescu)</i>,
iar prenumele <i><b>E</b>lena Adina</i>, cu numarul
matricol <b><i>731</i></b>, va trebui sa completeze:
<center><table><tr>
<td> Seria: </td><td><b> 4 </b></td><td> Semestrul: </td><td><b> 3 </b></td>
<td> Codul: </td><td><td><b> ie731 </b></td>
</tr></table></center>
</form></body></html>
```

Programul de filtrare **`Filtru.cpp`** este:

```
#include <condefs.h>
#pragma hdrstop
#pragma argsused
#include <stdio.h>
#include <dir.h>
#include <string.h>
#include <ctype.h>

   int main(int argc, char *argv[]) {
      struct ffblk sfisier;
      char fisier[50], fisiere[50][50], fisieretxt[50][50],
         antet[1024], student[1024], ser[1024], sem[1024], tab[1024],
         *p, *q, *r;
      int fd, n, i, j, k, t, nf=0;
      FILE *fe, *fr;

      fd = findfirst("*.txt", &sfisier, 0);
      for ( ; !fd; fd = findnext(&sfisier) ) { // ciclul fisier dupa fisier
         strcpy(fisier, sfisier.ff_name);
         printf("\nSe va filtra \"%s\"\n", fisier);
         strlwr(fisier); // fa totul in litere mici
         p = strstr(fisier, "ser"); // p la inceputul lui ser
         if (p == NULL)
```

```
            continue;  // lipseste seria
         for ( p += 3 ; (!isdigit(*p) && strlen(p)); p++); // la cifra
         if (*p == 0)
            continue; // lipseste numarul seriei
         for ( i=0; isdigit(*p); ser[i]=*p, i++, p++); // cifre serie
         ser[i] = '\0'; // zeroul terminal
         p = strstr(p, "sem");
         if (*p == 0)
            continue; // lipseste semestrul
         for ( p += 3 ; (!isdigit(*p) && strlen(p)); p++); // la cifra
         if (*p == 0)
            continue; // lipseste numarul seriei
         for ( i=0; isdigit(*p); sem[i]=*p, i++, p++); // cifrele semestru
         sem[i] = '\0'; // zeroul terminal
         strcat(ser, ".");
         strcat(ser, sem);
         printf("\tFsierul \"%s\" (serie.semestru) va contine informatiile
filtrate\n", ser);

      // Incepe filtrarea fisierului
         fe = fopen(fisier, "r");
         for ( ; ; ) { // ignorarea pana la antet
            p = fgets(antet, sizeof(antet), fe);
            if (p == NULL)
               goto NEXTFILE; // fisier eronat
            p = strstr(antet, "Nume\t");
            if (p == antet)
               break;
         } // preluat prima linie a antetului
         antet[strlen(antet)-1] = '\0';
         strcat(antet, "\t");
         for ( i=0, n=0; i < strlen(antet); i++)
            if (antet[i] == '\t') {
               tab[n] = '\t'; // taburi suplimentare
               n++;
            }
         tab[n] = '\0';
         if ((n < 4) || (strlen(antet) < 2*n))
            goto NEXTFILE; // campuri scurte in antet
         printf("\tAntetul are %d campuri, adica:\n", n);

         for ( ; ; ) { // preluat completarile la antet
            p = fgets(student, sizeof(student), fe);
            if (p == NULL)
               goto NEXTFILE; // fisier eronat
            student[strlen(student)-1] = '\0';
            strcat(student, tab); // completat cu taburi
            for (p=student, i=0; i<n; p=strstr(p,"\t")+1, i++);
            *p = '\0'; // retinut numarul exact de taburi
            if (student[0] != '\t')
               break; // terminat completarile
            p = student + 1; // de aici ia completarea
            q = strstr(p, "\t"); // pana aici tine completarea
            r = strstr(antet+1, "\t");
            r = strstr(r+1, "\t"); // de aici se pune
            for (i=1; i<n; i++, r=strstr(q-p+r+1,"\t"), p=q+1, q=strstr(p,"\t")) {
               if (p == q)
                  continue;
               for ( j=strlen(r), k=j+q-p; j >= 0; j--, k--)
                  r[k] = r[j]; // fa loc completarii
               for ( j=0; j<q-p; j++)
                  r[j] = p[j]; // muta completarea
            } // terminat adaugirea curenta
         } // terminat adaugirile la antet
```

```
      fr = fopen(ser, "w");
      strcat(antet, "\n");
      fputs(antet, fr); // scrie antetul
      strcat(student,"\n");
      t = 0; // numarul de studenti filtrati
      if (strlen(student) >= n + 6) {
         fputs(student, fr);
         t++;
      } // ignora liniile prea scurte
      for (i=0; i<n-1; i++)
         *strchr(antet, '\t') = '|'; // pentru afisare
      antet[strlen(antet)-2] = '\0';
      printf("\t\"%s\"\n", antet);
      for ( ; ; ) { // preluat liniile de dupa antet
         p = fgets(student, sizeof(student), fe);
         if (p == NULL)
            break; // terminat copierea
         student[strlen(student)-1] = '\0';
         strcat(student, tab); // completat cu taburi
         for (p=student, i=0; i<n; p=strstr(p,"\t")+1, i++);
         *p = '\0'; // retinut numarul exact de taburi
         strcat(student, "\n");
         if (strlen(student) >= n + 6) {
            fputs(student, fr);
            t++;
         } // ignora liniile prea scurte
      } // depus liniile in fisierul filtrat
      fclose(fr); // inchis fisierul filtrat
      printf("\t\"%s\" contine %d studenti\n", ser, t);
      strcpy(fisiere[nf], ser);
      strcat(fisiere[nf], "\n"); //numele fisierului filtrat
      strcpy(fisieretxt[nf], fisier);
      strcat(fisieretxt[nf], "\n"); // numele fisierului din care s-a filtrat
      nf++;
      printf("S-a trimis \"%s\" spre depunere server\n\n", ser);
   NEXTFILE:
      fclose(fe);
   } // Terminat filtrarea fisierelor *.txt

   // Urmeaza pregatirea fisierului de configurare si a celui de stergeri
   if ((fe = fopen("Gconfig.ftp", "r")) == NULL)
      return (1);
   fr = fopen("config.ftp", "w");
   for ( ; ; ) {
      p = fgets(antet, sizeof(antet), fe);
      if (p == NULL)
         break;
      if (strlen(p) <= 2)
         continue; // ignora liniile scurte
      if (p[0] == '#')
         continue; // ignora liniile comentariu
      if ((q=strstr(p, "=")) == NULL)
         continue; // ignora liniile ce nu contin parametri
      q++;
      for (i=0; i < q-p; i++)
         p[i] = tolower(p[i]); // numele de camp cu litere mici
      if (strstr(p, "password=") != NULL)
         for(i=0; i<strlen(q); i++) // DeCripteaza parola citita
            q[i]=(isdigit(q[i]))?('z'+'0'-q[i])
               :((isupper(q[i]))&&(q[i]<='P'))?('p'+'A'-q[i])
               :((isupper(q[i]))&&(q[i]>='Q'))?('Z'+'Q'-q[i])
               :((islower(q[i]))&&(q[i]<='p'))?('P'+'a'-q[i])
               :((islower(q[i]))&&(q[i]>='q'))?('9'+'q'-q[i])
               :(q[i])
               ;
```

```
         fputs(antet, fr);
      }//for
      fclose(fe);
      fe = fopen("stergeri.bat", "w"); // creaza scriptul de stergeri
      for (i=0; i<nf; i++) {
         fputs(fisiere[i], fr); // scrie numele fisierelor de transmis
         fprintf(fe, "del %s", fisiere[i]);
         fprintf(fe, "del %s", fisieretxt[i]);
      }//for
      fclose(fr);
      fclose(fe);
      printf("Au fost filtrate %d fisiere.\n", nf);
      printf("Pentru continuare tastati <ENTER>. Pentru intrerupere <CTRL/C>");
      gets(antet); // asteapta un enter
      return (0); // Terminare normala
   } // main
```