PROTOCOALE DE COMUNICAȚIE : LABORATOR 1

Notiuni pregatitoare pentru laboratorul de protocoale de comunicatie

Responsabil: Alecsandru PĂTRAȘCU

Cuprins

Comenzi shell

Gestiunea utilizatorilor

UNIX este un sistem multiutilizator. Un utilizator poate avea mai multe sesiuni pe acelasi calculator sau pe calculatoare diferite. In sistem exista urmatoarea ierarhie:

- un superuser(root) - utilizator privilegiat(admnistrator de sistem)
- utilizatori - un utilizator intra in sistem cu comanda 'login'
- au asociat un nume, o parola, un numar de utilizator

Exista si grupuri de utilizatori.

Informatiile despre utilizatori se gasesc in fisierul /etc/passwd pe care doar superuser-ul il poate modifica. Adaugarea/stergerea de utilizatori in/din sistem o poate face doar superuser-ul fie cu comenzile:

```
# mkuser
# rmuser
```

fie cu ajutorul programului sysadmsh.

Alte comenzi legate de gestiunea utilizatorilor:

```
$ cat /etc/passwd
```

permite vizualizarea fisierului de parole

```
$ logname
```

afiseaza numele utilizatorului curent (numele de login).

```
$ id
```

afiseaza numarul si numele de utilizator si de grup al utilizatorului curent

```
$ who
```

afiseaza numele tuturor utilizatorilor activi la un moment dat (+ informatii despre terminalul pe care lucreaza si momentul deschiderii sesiunii curente)

```
$ who am i
```

afiseaza datele de mai sus (+numele hostului) numai pentru utilizatorul curent

```
$ finger
```

la fel ca who dar cu informatii suplimentare (nume complet, adresa, telefon)

Fisiere

Sistemul de fisiere are o organizare ierarhica, fisierele fiind grupate in cataloage ce alcatuiesc o structura arborescenta. Arborele de fisiere este unic, radacina sa fiind simbolizata prin . Pentru specificarea caii in UNIX se foloseste (in DOS se folseste
).

Clasificarea fisierelor (6 tipuri):

1. fisiere obisnuite: siruri de octeti terminate cu D̂
2. fisiere de tip catalog:tabela cu o intrare pentru fiecare fisier din catalog. Fiecare catalog are doua intrari suplimentare:
   - . - referinta catre repectivul catalog
   - .. - referinta spre catalogul parinte
3. fisiere speciale: asociate dispozitivelor de I/O (in catalogul dev)
   Exemple:
   - /dev/fd0 - prima unit. de disc flexibil existenta
   - /dev/lp0 - prima imprimanta
   - /dev/tty0 - primul din cele 12 terminale virtuale

   Pot fi de tip bloc sau caracter in functie de tipul de terminal asociat. In UNIX, unui dispozitiv de I/O i se asociaza doua nume:
   - major - indica tipul de dispozitiv (ex.: imprimanta).
   - minor - al catelea dispozitiv periferic este din clasa de dispozitive de acelasi tip (ex.: a doua imprimanta).
4. fisierele FIFO ("named pipe") - sunt fisiere speciale utilizate in comunicatia dintre procese prin mecanismul pipe. Difera de celelalte doar prin modul de acces la informatie: Exemplu:
   ```
   proces 1                              proces 2
   ---------                           ---------
   |   W   |--------fisier pipe----|   R   |
   ---------                           ---------
   ```
   Accesul la informatie se poate face o singura data (fara posibilitatea de revenire) iar politica ce gestioneaza acest acces este FIFO.
5. Socketi - tip de fisiere folosit in comunicatia dintre procese prin retea. Un socket poate fi utilizat si pentru comunicatia intre procesele de pe acelasi calculator gazda.
6. Legatura simbolica - tip de fisier ce contine un pointer spre un alt fisier.

Permisiuni (drepturi) de acces

Drepturile de acces sunt asociate fiecarui fisier in parte si sunt in numar de trei :

1. r - drept de citire in acel fisier
2. w - drept de scriere in acel fisier
3. x - drept de a executa acel fisier

Utilizatorii se clasifica in trei categorii, in functie de relatia fata de un fisier :

1. user (proprietarul fisierului - este unic)
2. group (utilizatorii din acelasi grup cu proprietarul fisierului)
3. others (ceilalti utilizatori)

Deci vor trebui sa existe 9 pozitii pentru precizarea completa a drepturilor. Pentru fisierele de tip catalog aceste drepturi au alte semnificatii:

- r - drept de listare a continutului catalogului
- w - se pot crea/sterge fisiere in/din catalog
- x - se poate parcurge catalogul pentru cautarea unui fisier si exista drept de acces la acel fisier

Modificarea permisiunilor de acces

```
$ chmod [pentru_cine] tip_modif nume_fis
```

pentru_cine:

- u = schimb drepturile pentru user
- g = schimb drepturile pentru group
- o = schimb drepturile pentru others
- a = schimb drepturile pentru toti (all)

tip_modificare:

are minim doua caractere:

1. primul caracter:
   - + se adauga niste drepturi
   - - se retrag niste drepturi
   - = se seteaza drepturile
2. al doilea caracter: r, w sau x

Exemple:

```
$ chmod a=x fis # se seteaza drept de executie pentru toata lumea asupra
                # fisierului fis

$ chmod go+wx fis # se atribuie utilizatorilor din acelasi grup cu proprietarul
                  # si celorlati utilizatori drept de scriere si executie a
                  # fisierului fis

$ chmod 754 fis # se seteaza asupra fisierului fis drepturile:
                # 7 = r+w+x  pentru user
                # 5 = r+x  pentru group
                # 4 = r  pentru others
```

Schimbarea proprietarului sau grupului unui fisier

```
$ chown proprietar_nou fisier
$ chgrp grup_nou fisier
```

Comenzi de shell

Un prim exemplu de comanda shell pentru lucrul cu fisiere o reprezinta **ls**. Aceasta fara nici o optiune permite listarea continutului directorului in care va aflati curent. Exista insa fisiere ascunse (cele al caror nume incepe cu ".") care se pot afisa doar prin "ls -a" - in acest caz ls este exemplu de instructiune, pe cand -a este exemplu de optiune care functioneaza pentru o anumita comanda. Aproape toate comenzile shell accepta optiuni.

Alta comanda de shell este **mkdir** (make directory). Comanda permite crearea de noi directoare.

O alta comanda utila o reprezinta **cd** (change directory). Ca efect al acestei comenzi directorul curent (directorul in care va aflati la un anumit moment) se poate schimba. Exista si doua directoare mai speciale, si anume "." - care reprezinta directorul in care va aflati curent si ".." - care reprezinta directorul aflat cu

un nivel mai sus in ierarhia de directoare. De exemplu daca incercati "cd ." veti constata ca ramaneti in cadrul aceluiasi director in care erati si inainte de emiterea comenzii. Pentru a afla in care director din cadrul ierarhiei va aflati la momentul de timp curent puteti folosi comanda pwd. Ca rezultat al executiei acestei comenzi vi se afisa pe ecran calea completa pana la directorul in care va aflati.

Ca exercitiu puteti folosi comenzile de pana acum pentru a explora sistemul de fisiere.

O alta comanda o reprezinta comanda **cp**(copy). Aceasta va permite copierea unuia sau mai multor fisiere. Comanda **mv** va permite mutarea locului unuia sau a mai multor fisiere in cadrul sistemului de fisiere. Pentru stergerea unui fisier puteti folosi una dintre comenzile **rm** sau **rmdir**.

Pentru vizualizarea continutului unui fisier se poate folosi comanda **cat**. Ca rezultat al acestei comenzi continutul fisierului primit ca argument se va afisa daca nu se folosesc alte optiuni pe ecran. Comanda **less** va scrie de asemenea continutul fisierului pe ecran, insa pagina cu pagina. Folositi tasta spatiu pentru a trece de la pagina la pagina si tasta q pentru a determina terminarea afisarii.

Comanda **head** fara argumente va afisa primele 10 linii din fisier pe ecran. Comanda **tail** fara argumente va afisa ultimele 10 linii din fisier pe ecran. Pentru a cauta un anumit cuvant in cadrul fisierului se poate folosi **less**, dupa care folosi secventa "/[nume_de_cautat]" pentru a va pozitiona pe prima aparitie in text a cuvantului respectiv, daca exista. Apoi pentru a trece la urmatoarea aparitie a cuvantului in text folositi tasta n.

Un utilitar destul de folositor este si **wc**, care va permite diverse contorizari. De exemplu pentru a afla numarul total de cuvinte dintr-un fisier folositi *wc -w fisier*, sau pentru a afla numarul de linii din cadrul unui fisier folositi comanda *wc -l fisier*.

Toate aceste comenzi si multe altele incluse in sistemul de operare Unix/Linux au si un help disponibil, care se poate vizualiza folosind comanda man. De exemplu pentru a afla care sunt optiunile comenzii *wc* folositi **man wc**. Ca rezultat pe ecran vi se va afisa lista de parametrii ai comenzii cu explicatii legate de folosirea acestora.

Ca exercitiu de laborator folositi comanda man pentru a studia comenzile de pana acum mai in profunzime.

Compilatorul GCC

In cadrul laboratorului de Protocoale de Comunicatii veti avea de facut o serie de teme in limbajul C/C++.

Sistemul de operare Unix/Linux dispune de un utilitar pentru compilarea fisierelor scrise in aceste limbaje de programare, numit **gcc**. Ca alternative la gcc mai exista si g++, cc si CC. In continuare se va descrie modul in care se foloseste utilitarul gcc.

Sa presupunem ca avem un fisier sursa numit *hello.c*. Compilarea standard a acestui fisier este

```
gcc hello.c
```

Ca rezultat se obtine un fisier a.out care se poate executa. Daca insa compilam folosind

```
gcc hello.c -o hello
```

se va obtine fisierul executabil hello. Puteti insa folosi si o alta secventa de compilare, si anume daca introduceti

```
gcc -c hello.c
gcc hello.o -o hello
```

Rezultatul obtinut va fi acelasi ce cel obtinut prin *gcc hello.c -o hello*. Ceea ce difera este faptul ca prima instructiune din cadrul secventei va fisierul va compila fisierul *hello*.c intr-un fisier cod masina numit *hello.o*, iar cea de-a doua instructiune va lega *hello.o* cu unele dintre librariile sistemului pentru a produce rezultatul final, si anume fisierul executabil *hello*.

De asemenea compilatorul va permite compilarea simultana a mai multor fisiere sursa pentru obtinerea ulterioara a unui fisier executabil. De exemplu:

```
gcc fisier1.c fisier2.c –o program
```

va compila cele doua fisiere sursa si va furniza un fisier executabil numit program.

Compilatorul gcc suporta o serie de optiuni de compilare (pentru studierea acestora mai in amanunt se va folosi man gcc).

- Optiunea *-g* va permite obtinerea unui executabil care va contine informatii de debug. Aceasta optiune este utila in combinatie cu un program de debugging cum ar fi gdb, care de asemenea se va studia in cadrul laboratorului.
- Optiunea *-Wall* specifica compilatorului sa afiseze in timpul compilarii toate avertismentele (warning-urile) despre bucati de cod corecte din punct de vedere sintactic, dar care ar putea contine totusi erori.
- Optiunea *-l* permite legarea de biblioteci ale sistemului. De exemplu, daca programul contine functii matematice cum ar fi sqrt atunci programul se compileaza folosind *gcc program.c -o program -lm*.

Depanarea programelor

Un program de debugging (depanare) este un program folosit pentru rularea altor programe in scopul permiterii ca utilizatorul sa poata exercita un control asupra executiei programului. Utilizatorul poate examina variabilele din program la aparitia unor erori de exemplu.

In Linux un bun program de debugging il constituie **gdb**. Pentru folosirea acestuia programul trebuie sa fi fost compilat cu optiune **-g** (dupa cum am spus mai sus). Dupa compilare se porneste programul gdb cu argumentul numele programului de executat

```
gdb program
```

Ca efect gdb va introduce intr-o interfata text de unde puteti alege o serie de optiuni. In continuare sunt descrise o serie de comenzi pe care le puteti tasta in cadrul acestei interfete.

O prima comanda este **run**. Acesta da in executie programul. Comanda primeste si argumente, pe care le transfera programului care ruleaza.

O alta comanda este **break**. Aceasta primeste ca argument locul in care se creaza un punct de oprire in cadrul executiei. Programul se va opri automat atunci cand ajunge la un astfel de punct de control. De exemplu unele dintre cele mai frecvente locuri de oprire sunt cele de la inceputul functiilor. De exemplu *break Functie* va forma un punct de control la inceputul functiei avand numele *Functie*. Puteti folosi ca argument si un numar, ceea ce inseamna linia la care se va forma punctul de control. In conjunctie cu aceasta comanda se poate folosi comanda *delete*, care are ca efect stergerea unui punct de control generat anterior.

Comanda **step** are ca efect executia urmatoarei instructiuni din cadrul programului. Dupa executia acesteia programul se va opri la urmatoarea instructiune de executat. Similar comanda **next** are acelasi efect, insa ca diferenta daca urmatoarea instructiune este reprezentata de apelul unei functii aceasta se trateaza unitar ca o singura instructiune. Comanda **finish** are ca efect executia unor comenzi step repetate pana cand se ajunge la sfarsitul functiei curente.

Comanda **continue** are ca efect continuarea executiei programului pana la intalnirea unui punct de control sau pana la terminarea programului.

Comanda **where** afiseaza sirul de apeluri de functii prin care s-a ajuns la instructiunea curenta.

Comanda **print** primeste ca argument o expresie C (de obicei numele unei variabile este suficient) si va afisa valoarea acestei expresii in cadrul curent al programului.

Atunci cand nu sunteti siguri de formatul unei comenzi gdb mai exista si comanda **help** care afiseaza informatii despre comenzi gdb.

Pentru terminarea executei programului gdb puteti folosi comanda **quit**.

Fisiere Makefile

Un utilitar util in Linux este **make**. Acesta citeste instructiuni din cadrul unui fisier text pe care apoi le executa. Implicit make citeste instructiunile din fisierul text cu numele *Makefile*, insa se poate specifica numele oricarui fisier text.

In cadrul fisierului Makefile instructiunile sunt scrise pe fiecare linie. In cazul in care instructiunile sunt prea lungi se poate folosi caracterul "
" urmat de Enter, efectul fiind acela ca urmatoarea linie se considera a fi o continuare a liniei curente. Comentariile in cadrul fisierului Makefile incep cu caracterul "#". Dupa acest caracter tot ce urmeaza pana la sfarsitul liniei se considera comentariu si se ignora, exceptie facand caracterul "
".

De exemplu:

```
linia_unu \
linia_doi # comentariu \
linia_trei
```

este similar cu

```
linia_unu linia_doi linia_trei
```

Conceptul de regula in cadrul fisierelor Makefile specifica cum si cand se executa o secventa de instructiuni. De exemplu sa presupunem ca avem un proiect care implica compilarea fisierelor sursa main.c si io.c si producerea fisierului executabil program. Un exemplu de fisier Makefile care ne ajuta in cadrul proiectului nostru atunci ar fi:

```
project : main.o io.o
  gcc main.o io.o -o project

main.o : main.c
  gcc -c main.c

io.o : io.c
  gcc -c io.c
```

Exemplul poate parea un pic fortat, dar este folositor pentru a intelege mai bine conceptul de regula. In exemplu acesta avem de a face cu trei reguli, una pentru generarea fisierului executabil, o alta pentru compilarea fisierului sursa *main.c* si o a treia pentru compilarea fisierului sursa *io.c*.

Liniile care au in componenta ":" se numesc linii de dependenta. Ceea ce se afla la stanga de ":" reprezinta numele dependentei (sau a regulii), pe cand ceea ce se afla la dreapta reprezinta regulile de care depinde regula curenta. De exemplu linia *project : main.o io.o* specifica faptul ca regula de obinere a fisierului executabil depinde de regulile de formare a fisierelor *main.o* si *io.o*. La rulare, make compara momentul de timp cand fisierul project a fost ultima oara modificat cu momentul de timp la care fisierele *main.o* si/sau *io.o* au fost modificate pentru a determina daca este necesara o recompilare sau nu a vreunuia dintre fisierele sursa.

Sa presupunem in exemplul nostru mai departe ca atat *main.c*, cat si *io.c*, depind ambele mai departe de *def.h*. In cazul acesta se pot include dependente aditionale, ca de exemplu putem adauga la exemplul nostru linia:

```
main.obj io.obj : incl.h
```

In fisierul Makefile pot fi incluse si linii de macro-definitii. O linie de macro-definitie este o linie care are un nume de macro-definitie, semnul "=" si o valoare a macro-definitiei. In cadrul fisierului expresiile de forma *$nume* sau *$(nume)* se inlocuiesc cu valorile asociate acelor nume. Daca numele este compus dintr-un singur caracter atunci parantezele pot fi omise. Ca exemplu:

```
CC = gcc
project : main.o io.o
  $(CC) main.o io.o -o project

main.o : main.c
  ${CC} -c main.c

io.o : io.c
  gcc -c io.c
```

Apeluri de sistem pentru operatii cu fisiere

Orice sistem de operare pune la dispozitia programatorilor o serie de servicii prin intermediul carora acestora li se ofera acces la resursele hardware si software gestionate de sistem: lucrul cu tastatura, cu discurile, cu dispozitivul de afisare, gestionarea fisierelor si directoarelor etc. Aceste servicii se numesc apeluri sistem.

De cele mai multe ori, operatiile pe care ele le pot face asupra resurselor gestionate sunt operatii simple, cu destul de putine facilitati. De aceea, frecvent, se pot intalni in bibliotecile specifice limbajelor de programare colectii de functii mai complicate care gestioneaza resursele respective, dar oferind programatorului niveluri suplimentare de abstractizare a operatiilor efectuate, precum si importante facilitati in plus. Acestea sunt functiile de biblioteca.

Trebuie subliniat faptul ca functiile de biblioteca cu ajutorul carora se poate gestiona o anumita resursa sunt implementate folosind chiar functiile sistem corespunzatoare, specifice sistemului de operare.

In continuare se vor prezenta unele apeluri de sistem ale sistemului de operare care permit efectuarea unor operatii asupra fisierelor, cum ar fi deschiderea fisierelor, scrierea si citirea de blocuri de date in si din fisiere, mutarea si copierea fisierelor etc.

Descriptori de fisiere

Pentru a putea actiona asupra unui fisier, este nevoie inainte de toate de o metoda de a identifica in mod unic fisierul. In cazul functiilor discutate, identificarea fisierului se face printr-un asa-numit descriptor de fisier

(file descriptor). Acesta este un numar intreg care este asociat fisierului in momentul deschiderii acestuia.

open

Deschiderea unui fisier este operatia prin care fisierul este pregatit pentru a putea fi prelucrat in continuare. Aceasta operatie se realizeaza prin intermediul functiei **open**:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *pathname, int oflag, [, mode_t mode]);
```

Functia returneaza -1 in caz de eroare. In caz contrar, ea returneaza descriptorul de fisier asociat fisierului deschis.

Parametri:

- pathname - contine numele fisierului
- oflag - optiunile de deschidere a fisierului. Este, in realitate un sir de biti, in care fiecare bit sau grupa de biti are o anumita semnificatie. Pentru fiecare astfel de semnificatie exista definite in fisierul header C fcntl.h cate o constanta. Constantele se pot combina folosind operatorul '—' (sau logic pe biti) din C, pentru a seta mai multi biti (deci a alege mai multe optiuni) in parametrul intreg oflag. Iata cateva din aceste constante:
  - O_RDONLY - deschidere numai pentru citire
  - O_WRONLY - deschidere numai pentru scriere
  - O_RDWR - deschidere pentru citire si scriere
  - O_APPEND - deschidere pentru adaugare la sfarsitul fisierului
  - O_CREAT - crearea fisierului, daca el nu exista deja; daca e folosita cu aceasta optiune, functia open trebuie sa primeasca si parametrul mode.
  - O_EXCL - creare "exclusiva" a fisierului: daca s-a folosit O_CREAT si fisierul exista deja, functia open va returna eroare
  - O_TRUNC - daca fisierul exista, continutul lui este sters
- mode - se foloseste numai in cazul in care fisierul este creat si specifica drepturile de acces asociate fisierului. Acestea se obtin prin combinarea unor constante folosind operatorul sau ('—'), la fel ca si la optiunea precedenta. Constantele pot fi:
  - S_IRUSR - drept de citire pentru proprietarul fisierului (user)
  - S_IWUSR - drept de scriere pentru proprietarul fisierului (user)
  - S_IXUSR - drept de executie pentru proprietarul fisierului (user)
  - S_IRGRP - drept de citire pentru grupul proprietar al fisierului
  - S_IWGRP - drept de scriere pentru grupul proprietar al fisierului
  - S_IXGRP - drept de executie pentru grupul proprietar al fisierului
  - S_IROTH - drept de citire pentru ceilalti utilizatori
  - S_IWOTH - drept de scriere pentru ceilalti utilizatori
  - S_IROTH - drept de executie pentru ceilalti utilizatori

Pentru crearea fisierelor poate fi folosita si functia

```
creat (const char *pathname, mode_t mode)
```

echivalenta cu specificarea optiunilor *O_WRONLY — O_CREAT — O_TRUNC* la functia *open*.

close

Dupa utilizarea fisierului, acesta trebuie inchis, folosind functia

```
int close (int filedes)
```

in care *filedes* este descriptorul de fisier obtinut la open.

read

Citirea datelor dintr-un fisier deschis se face cu functia

```
#include <unistd.h>

ssize_t read(int fd, void *buff, size_t nbytes)
```

Functia citeste un numar de exact *nbytes* octeti de la pozitia curenta in fisierul al carui descriptor este *fd* si ii pune in zona de memorie indicata de pointerul *buff*.

Este posibil ca in fisier sa fie de citit la un moment dat mai putin de nbytes octeti (de exemplu daca s-a ajuns spre sfarsitul fisierului), astfel ca functia read va pune in buffer doar atatia octeti cati poate citi. In orice caz, *functia returneaza numarul de octeti cititi din fisier*, deci acest lucru poate fi usor observat.

Daca s-a ajuns exact la sfarsitul fisierului, functia returneaza 0, iar in caz de eroare, -1.

write

Scrierea datelor se face cu

```
ssize_t write(int fd, void *buff, size_t nbytes)
```

Functia scrie in fisier primii *nbytes* octeti din bufferul indicat de *buff*. Returneaza -1 in caz de eroare si numarul de octeti scrisi in caz de succes.

lseek

Operatiile de scriere si citire in si din fisier se fac la o anumita pozitie in fisier, considerata pozitia curenta. Fiecare operatie de citire, de exemplu, va actualiza indicatorul pozitiei curente incrementand-o cu numarul de octeti cititi. Indicatorul pozitiei curente poate fi setat si in mod explicit, cu ajutorul functiei **lseek**:

```
off_t lseek(int fd, off_t offset, int pos)
```

Functia pozitioneaza indicatorul la deplasamentul offset in fisier, astfel:

- daca parametrul pos ia valoarea SEEK_SET, pozitionarea se face relativ la inceputul fisierului
- daca parametrul pos ia valoarea SEEK_CUR, pozitionarea se face relativ la pozitia curenta
- daca parametrul pos ia valoarea SEEK_END, pozitionarea se face relativ la sfarsitul fisierului

Parametrul offset poate lua si valori negative si reprezinta deplasamentul, calculat in octeti.

In caz de eroare, functia returneaza -1.

Exemplu

Avem un exemplu de folosire a funcţiilor de acces la fişiere în programul de mai jos, care copiază fişierul "sursa" în fişierul "destinatie".

```
#include <unistd.h>     /* pentru open(), exit() */
#include <fcntl.h>      /*        O_RDWR */
#include <errno.h>      /*        perror() */

void fatal(char * mesaj_eroare)
{
      perror(mesaj_eroare);
        exit(1);
}

int main(void)
{
        int miner_sursa, miner_destinatie;
        int copiat;
        char buf[1024];

        miner_sursa = open("sursa", O_RDONLY);
        miner_destinatie = open("destinatie", O_WRONLY | O_CREAT, 0644);
        if (miner_sursa < 0 ||
            miner_destinatie < 0)
                fatal("Nu pot deschide un fisier");
        lseek(miner_sursa, 0, SEEK_SET);
        lseek(miner_destinatie, 0, SEEK_SET);
        while ((copiat = read(miner_sursa, buf, sizeof(buf)))) {
                if (copiat < 0)
                        fatal("Eroare la citire");
                copiat = write(miner_destinatie, buf, copiat);
                if (copiat < 0)
                        fatal("Eroare la scriere");
        }
        close(miner_sursa);
        close(miner_destinatie);
        return 0;
}
```

Standardizare si organizatii implicate

La ora actuala, numarul de producatori de echipamente hardware si de software este, la nivel mondial, foarte ridicat. Pentru ca retelele de calculatoare sa fie functionale, este necesar ca toate aceste produse sa poata interopera - iar aceasta se poate obtine prin stabilirea unor standarde ce trebuie respectate de catre producatori. Prezentam in cele ce urmeaza cateva dintre cele mai importante organizatii care se ocupa de stabilirea standardelor.

ISO - International Standards Organization

ISO este cea mai mare organizatie pentru dezvoltarea standardelor din lume, avand membri din 157 de tari si secretariatul central la Geneva. Standardele ISO fac parte din domenii dintre cele mai diverse si sunt elaborate in cadrul unor comitete tehnice, fiecare comitet ocupandu-se de un anumit subiect. ISO include

organizatii nationale de standardizare din tarile membre - de exemplu ANSI (American National Standards Institute) in S.U.A., BSI in Marea Britanie, AFNOR (Association Francaise de Normalisation) in Franta.

Scurta prezentare ISO: http://www.iso.org/iso/en/aboutiso/introduction/index.html

Catalog de standarde ISO: http://www.iso.org/iso/en/CatalogueListPage.CatalogueList

IEEE - Institute of Electrical and Electronics Engineers

Printre activitatile desfasurate in cadrul IEEE se afla si un grup de standardizare; unul dintre cele mai importante comitete din acest grup este 802, care a standardizat diverse tipuri de retele locale. Cateva exemple de standarde sunt: 802.3 (Ethernet), 802.11 (retele locale fara fir), 802.15 (Bluetooth).

IEEE Standards Association: http://standards.ieee.org/

Standardizarea in Internet

Pe parcursul evolutiei Internet-ului au fost create mai multe organizatii care sa supravegheze dezvoltarea standardelor si tehnologiilor. Prima dintre ele a fost IAB (Internet Activities Board), aparuta inca de la crearea retelei ARPANET, si al carei rol era de a discuta noile protocoale si standarde propuse de catre cercetatori. In 1989 au fost infiintate IRTF (Internet Research Task Force), care se ocupa de probleme de cercetare pe termen lung, si IETF (Internet Engineering Task Force), care rezolva probleme de inginerie pe termen mai scurt. In 1992 a aparut ISOC (Internet Society), care coordoneaza atat standarde, cat si diverse activitati legate de Internet in intreaga lume.

ISOC: http://www.isoc.org/

Documente RFC

Documentele RFC (Request For Comments) sunt rapoarte tehnice care descriu standarde, protocoale sau metodologii legate de Internet. Primele documente RFC au fost scrise la sfarsitul anilor '60, in cadrul IAB, iar la ora actuala exista mai multe mii de astfel de documente. Fiecare document RFC are un numar serial propriu, numerele fiind asignate in ordinea cronologica a aparitiei documentelor; RFC-urile nu sunt niciodata modificate, iar in cazul in care apare, de exemplu, o noua versiune a unui protocol descris de un RFC, se va crea un nou document RFC cu alt numar serial.

Documentele RFC sunt de mai multe tipuri, printre care:

- Standard (STD) - documente care descriu standarde; exemplu: RFC 793 - protocolul TCP Informationale (FYI - For Your Information) - de exemplu: RFC 1983 - Internet Users' Glossary, RFC 1855 - Netiquette Guidelines
- Best Current Practices (BCP) - contin recomandari pentru utilizarea standardelor Internet; de exemplu: RFC 2360 - Guide for Internet Standards Writers, RFC 4107 - Guidelines for Cryptographic Key Management
- Umoristice - publicate de obicei pe 1 aprilie; de exemplu: RFC 2324 - Hyper Text Coffee Pot Control Protocol

RFC Overview: http://www.rfc-editor.org/overview.html

Pagina pentru cautare de documente RFC: http://www.faqs.org/rfcs/index.html

O alta prezentare a documentelor RFC: http://www.cs.tut.fi/ jkorpela/rfcs.html