

# LUCRAREA NR. 1

## Conversia numerelor întregi și reale în diferite baze de numerație

---

Lucrarea prezintă metodele de conversie a numerelor întregi și reale din baza de numerație 10 (zecimal) în bazele 2, 8 și 16 (binar, octal, hexazecimal) și invers.

### 1.1. Generalități

Fiecare sistem de numerație are un număr de cifre și/sau litere corespunzător bazei acestuia. Astfel, sistemul zecimal conține cifre de la 0 la 9 (baza 10), sistemul hexazecimal conține cifre de la 0 la 9 și litere de A la F (baza 16), sistemul octal conține cifre de 0 la 7 (baza 8) iar sistemul binar conține cifrele 0 și 1 (baza 2). Pentru a face distincție între numerele din diferite baze de numerație există mai multe metode de notare:

- a) La sfârșitul numărului se adaugă o literă corespunzătoare bazei de numerație:
  - B - binar (ex. 10011101B)
  - Q - octal (ex. 23701Q)
  - D - zecimal (ex. 5429D)
  - H - hexazecimal (ex. FD37BH)
- b) La sfârșitul numărului se adaugă, în paranteze, baza căreia îi aparține numărul:
  - (2) - binar (ex. 10011011(2))
  - (8) - octal (ex. 24673(8))
  - (10) - zecimal (ex. 9546(10))
  - (16) - hexazecimal (ex. 34A4D(16))
- c) La sfârșitul numărului se adaugă ca și indice, în paranteze, baza căreia îi aparține numărul:
  - Număr<sub>(2)</sub> - binar (ex. 101101<sub>(2)</sub>)

- Număr<sub>(8)</sub> - octal (ex. 5572<sub>(8)</sub>)
- Număr<sub>(10)</sub> - zecimal (ex. 9334<sub>(10)</sub>)
- Număr<sub>(16)</sub> - hexazecimal (ex. 53FD1<sub>(16)</sub>)

*Observație:*

Deoarece baza 10 este considerată o bază implicită, numerele din această bază nu trebuie să fie urmate de simbolul corespunzător bazei.

## 1.2. Conversia numerelor din baza 10 în alte baze de numerație

În cazul unui număr real, conversia din baza 10 în altă bază se face separat pentru partea întreagă și pentru partea zecimală. În *tabelul 1.1* sunt prezentate numerele de la 0 la 15 în baza 10 și corespondentul lor în binar, octal și hexazecimal.

**Tabelul 1.1.** Reprezentarea numerelor în diferite baze de numerație.

Zecimal	Binar	Octal	Hexazecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

### 1.2.1. Conversia părții întregi

Cea mai simplă metodă de conversie a unui număr din baza 10 în altă bază de numerație este împărțirea succesivă a numărului respectiv la baza în care se dorește conversia: se împarte numărul la bază iar în continuare câtul obținut se împarte la bază până când acesta devine zero. Rezultatul final se obține prin scrierea resturilor fiecărei împărțiri, în ordine inversă.

#### Exemplul 1:

Să se convertească numărul 173 din baza 10 în bazele 2, 8 și 16.

Rezolvare ( tabelul 1.2):

$$\text{număr} : \text{bază} = \text{cât} + \text{rest}$$

Notă:

Dacă în urma împărțirii numărului la bază rezultă un cât mai mic decât baza, nu mai este necesară împărțirea câtului la bază, acesta reprezentând prima cifră din numărul rezultat în noua bază.

Tabelul 1.2. Conversia părții întregi (exemplu)

Binar	Octal	Hexazecimal
$173 : 2 = 86+1$	$173 : 8 = 21+5$	$173 : 16 = 10+13$
$86 : 2 = 43+0$	$21 : 8 = 2+5$	$10 : 16 = 0+10$
$43 : 2 = 21+1$	$2 : 8 = 0+2$	
$21 : 2 = 10+1$		
$10 : 2 = 5+0$		
$5 : 2 = 2+1$		
$2 : 2 = 1+0$		
$1 : 2 = 0+1$		
<b>10101101<sub>(2)</sub></b>	<b>255<sub>(8)</sub></b>	<b>AD<sub>(16)</sub></b>

### 1.2.2. Conversia părții zecimale

Conversia părții zecimale a unui număr din baza 10 în altă bază de numerație se realizează înmulțind partea zecimală (fracționară) cu baza în care dorim să facem conversia. În continuare se înmulțește succesiv partea fracționară a rezultatului înmulțirii precedente cu baza. Rezultatul în noua bază este reprezentat de partea întreagă a fiecărei înmulțiri. În cazul ideal, rezultatul final se obține în momentul în care partea fracționară a rezultatului înmulțirii cu baza este zero. De

## Conversia numerelor întregi și reale în diferite baze de numerație

cele mai multe ori, însă, partea fracționară nu devine zero niciodată (sau devine zero după un număr foarte mare de înmulțiri). De aceea este necesară stabilirea preciziei de reprezentare a părții fracționare rezultate (numărul de cifre a părții fracționare rezultate).

### Exemplul 2:

Să se convertească numărul 0.136 din baza 10 în bazele 2, 8 și 16.

Precizia de reprezentare: 8 (parte fracționară din 8 cifre).

Rezolvare ( **tabelul 1.3**):

$$\text{număr} \times \text{bază} = \text{parte fracționară} + \text{parte întreagă}$$

**Tabelul 1.3.** Conversia părții zecimale (exemplu)

Binar	Octal	Hexazecimal
$0.136 \times 2 = 0.272+0$	$0.136 \times 8 = 0.088+1$	$0.136 \times 16 = 0.176+2$
$0.272 \times 2 = 0.544+0$	$0.088 \times 8 = 0.704+0$	$0.176 \times 16 = 0.816+2$
$0.544 \times 2 = 0.088+1$	$0.704 \times 8 = 0.632+5$	$0.816 \times 16 = 0.056+13$
$0.088 \times 2 = 0.176+0$	$0.632 \times 8 = 0.056+5$	$0.056 \times 16 = 0.896+0$
$0.176 \times 2 = 0.352+0$	$0.056 \times 8 = 0.448+0$	$0.896 \times 16 = 0.336+14$
$0.352 \times 2 = 0.704+0$	$0.448 \times 8 = 0.584+3$	$0.336 \times 16 = 0.376+5$
$0.704 \times 2 = 0.408+1$	$0.584 \times 8 = 0.672+4$	$0.376 \times 16 = 0.016+6$
$0.408 \times 2 = 0.816+0$	$0.672 \times 8 = 0.376+5$	$0.016 \times 16 = 0.256+0$
<b>0.00100010<sub>(2)</sub></b>	<b>0.10550345<sub>(8)</sub></b>	<b>0.22D0E560<sub>(16)</sub></b>

### 1.3. Conversia numerelor dintr-o bază oarecare în baza 10

Pentru conversia unui număr dintr-o bază oarecare în baza 10 se va folosi următoarea notație:

$$x_n x_{n-1} x_{n-2} \dots x_1 x_0, z_1 z_2 \dots z_{m-1} z_m \text{ (b)}$$

Unde:

- $x_n x_{n-1} x_{n-2} \dots x_1 x_0$  - reprezintă partea întreagă a numărului (Exemplu:  $nr = 1101.011_{(2)}$ , partea întreagă:  $x_3 x_2 x_1 x_0 = 1101_{(2)}$ );
- $z_1 z_2 \dots z_{m-1} z_m$  - reprezintă partea fracționară a numărului (Exemplu:  $nr = 1101.011_{(2)}$ , partea fracționară:  $0.z_1 z_2 z_3 = 0.011_{(2)}$ ).

Conversia numărului în baza 10 se realizează cu următoarea relație:

$$N_{(10)} = x_n \cdot b^n + x_{n-1} \cdot b^{n-1} + \dots + x_1 \cdot b^1 + x_0 \cdot b^0 + z_1 \cdot b^{-1} + z_2 \cdot b^{-2} + \dots + z_{m-1} \cdot b^{-(m-1)} + z_m \cdot b^{-m}$$

unde:  $b$  reprezintă baza din care se face conversia.

**Exemplul 3:**

*binar*  $\rightarrow$  *zecimal* :

$$1101.011_{(2)} = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = 8 + 4 + 0 + 1 + 0 + 0.25 + 0.125 = 13.375_{(10)}$$

*octal*  $\rightarrow$  *zecimal* :

$$127.03_{(8)} = 1 \cdot 8^2 + 2 \cdot 8^1 + 7 \cdot 8^0 + 0 \cdot 8^{-1} + 3 \cdot 8^{-2} = 64 + 16 + 7 + 0 + 0.0468 = 87.0468_{(10)}$$

*hexazecimal*  $\rightarrow$  *zecimal* :

$$2A.01_{(16)} = 2 \cdot 16^1 + 10 \cdot 16^0 + 0 \cdot 16^{-1} + 1 \cdot 16^{-2} = 32 + 10 + 0 + 0.0039 = 42.0039_{(10)}$$

#### 1.4. Exerciții propuse

1) Să se convertească următoarele numere din baza 10 în bazele 2, 8 și 16:

- |        |            |
|--------|------------|
| a) 218 | f) 119.345 |
| b) 402 | g) 200.08  |
| c) 109 | h) 108.932 |
| d) 351 | i) 245.115 |
| e) 254 | j) 406.422 |

2) Să se convertească următoarele numere din bazele 2, 8 și 16 în baza 10:

- |                             |                               |
|-----------------------------|-------------------------------|
| a) 110100110 <sub>(2)</sub> | g) 101101.1011 <sub>(2)</sub> |
| b) 100101101 <sub>(2)</sub> | h) 111001.1001 <sub>(2)</sub> |
| c) 167 <sub>(8)</sub>       | i) 7322.115 <sub>(8)</sub>    |
| d) 314 <sub>(8)</sub>       | j) 1121.631 <sub>(8)</sub>    |
| e) E45B <sub>(16)</sub>     | k) 2CB1.AE <sub>(16)</sub>    |
| f) 95D2 <sub>(16)</sub>     | l) 55CC.9D <sub>(16)</sub>    |

3) Să se convertească următoarele numere din binar în octal și hexazecimal:

*Conversia numerelor întregi și reale în diferite baze de numerație*

---

a)  $100000110_{(2)}$

b)  $110111101_{(2)}$

c)  $100111101_{(2)}$

d)  $101000001_{(2)}$

e)  $111011_{(2)}$

f)  $101001_{(2)}$

g)  $111101_{(2)}$

h)  $100100_{(2)}$

## LUCRAREA NR. 2

### Operații cu numere în diferite baze de numerație

---

Lucrarea prezintă modul în care se efectuează operațiile simple cu numere din diferite baze de numerație: adunarea, scăderea, înmulțirea și împărțirea. De asemenea, lucrarea prezintă metodele de reprezentare a numerelor negative în binar.

#### 2.1. Adunarea numerelor în diferite baze de numerație

În orice bază de numerație, adunarea se face după aceleași reguli ca și în zecimal, cu observația că cifra cea mai mare dintr-o bază " $b$ " va fi  $b - 1$ , adică 9 în zecimal, 7 în octal, 1 în binar și F în hexazecimal. Dacă prin adunarea a două cifre de rang " $i$ " se va obține un rezultat mai mare decât  $b - 1$ , va fi necesar "*transportul*" unei unități spre cifra de rang următor " $i + 1$ ", iar pe poziția de rang " $i$ " va rămâne restul împărțirii rezultatului adunării cifrelor de rang " $i + 1$ " la bază. *Transportul* spre cifra de rang " $i + 1$ " va deveni o nouă unitate la suma cifrelor de rang " $i + 1$ ". În tabelul 2.1 este prezentat un exemplu de adunare a numerelor în binar, octal și hexazecimal.

*Tabelul 2.1. Adunarea numerelor în diferite baze de numerație.*

Binar	Octal	Hexazecimal
$\begin{array}{r} 11101+ \\ 11010 \\ \hline 110111 \end{array}$	$\begin{array}{r} 1702+ \\ 2131 \\ \hline 4033 \end{array}$	$\begin{array}{r} 1C6F+ \\ 1411 \\ \hline 3080 \end{array}$

#### 2.2. Scăderea numerelor în diferite baze de numerație

Ca și în cazul adunării, pentru scăderea numerelor în binar, octal și hexazecimal se aplică regulile de la scăderea în zecimal: dacă nu se pot scădea două cifre de rang " $i$ " (adică cifra descăzutului este mai mică decât a scăzătorului) se face "*împrumut*" o unitate din cifra de rang următor  $i + 1$ . În cazul în care cifra din care se face "*împrumutul*" este 0, se face împrumutul mai departe din cifra de rang următor. În tabelul 2.2 este prezentat un exemplu de scădere a numerelor în binar,

octal și hexazecimal.

Tabelul 2.2. Scăderea numerelor în diferite baze de numerație.

Binar	Octal	Hexazecimal
$\begin{array}{r} 10101- \\ \underline{110} \\ 1111 \end{array}$	$\begin{array}{r} 322- \\ \underline{131} \\ 171 \end{array}$	$\begin{array}{r} AF9- \\ \underline{13F} \\ 9BA \end{array}$

### 2.3. Înmulțirea și împărțirea numerelor în binar

Înmulțirea și împărțirea numerelor în sistemul de numerație binar se efectuează ca și în cazul numerelor din sistemul zecimal. În cazul înmulțirii în binar vom avea, ca și în zecimal, " $1 \times 1 = 1$ ", " $0 \times 0 = 0$ ", iar " $0 \times 1 = 0$ ". Împărțirea numerelor în sistemul de numerație binar are același rezultat ca și în zecimal, adică un *cât* și un *rest*. În tabelul 2.3 este prezentat câte un exemplu de înmulțire și împărțire a numerelor în binar.

Tabelul 2.3. Înmulțirea și împărțirea numerelor în binar.

Înmulțire	Împărțire
$\begin{array}{r} 1011 \times \\ \underline{1101} \\ 1011 \\ 0000 \\ 1011 \\ \underline{1011} \\ 10001111 \end{array}$	$\begin{array}{r} 11101 \overline{)101} \\ \underline{101} \quad 101 \\ 0100 \\ \underline{000} \\ 1001 \\ \underline{101} \\ 100 \end{array}$

Notă:

Dacă în urma împărțirii a două numere în binar rezultă un rest diferit de zero și mai mic decât împărțitorul, pentru obținerea părții fracționare se poate continua împărțirea astfel: se adaugă cifra 0 la rest și virgula la cât și se continuă prin împărțirea restului la împărțitor, rezultatele fiind adăugate la cât după virgulă.

### 2.4. Reprezentarea binară a numerelor negative

Pentru reprezentarea numerelor negative în binar, bitul din stânga reprezentării numărului este folosit ca bit de semn.



Astfel avem bitul de semn:

0 - pentru numere pozitive (+)

1 - pentru numere negative (-)

Ceilalți biți din componența numărului sunt folosiți pentru reprezentarea valorii.

#### 2.4.1. Codul direct

Numerele întregi se reprezintă prin valoare absolută și semn. În cazul codului direct, pentru numerele negative bitul de semn este 1, iar ceilalți " $n - 1$ " biți servesc pentru reprezentarea valorii absolute a numărului. De exemplu, numărul  $N = -5$  se poate reprezenta pe 8 biți astfel:  $10000101_{(2)}$ , unde valoarea absolută este  $0000101_{(2)}$  iar primul bit este bitul de semn.

Domeniul de reprezentare în cazul codului direct va fi:

- $2^{n-1}$  valori pozitive de la 0 la  $2^{n-1} - 1$
- $2^{n-1}$  valori negative de la  $-(2^{n-1} - 1)$  la 0

Se poate observa că există două reprezentări ale lui zero, respectiv 00000000 și 10000000, iar numărul maxim și numărul minim dintr-un domeniu au aceeași valoare absolută, respectiv 01111111 și 11111111.

#### 2.4.2. Codul invers

Pentru numerele negative reprezentate în codul invers (complement față de 1) bitul de semn este 1 iar ceilalți " $n - 1$ " biți servesc la reprezentarea valorii absolute **negate** a numărului de reprezentat. Negarea se realizează la nivel de bit: biții "0" devin "1" iar biții "1" devin "0". De exemplu, numărul  $N = -5$  se va reprezenta în codul invers astfel:  $1111010_{(2)}$ , unde  $1111010_{(2)}$  reprezintă valoarea absolută negată a numărului  $0000101_{(2)}$ .

Matematic, complementul față de 1 al unui număr negativ  $N$  care se reprezintă pe  $n$  biți se definește astfel:

$$C_1(N) = 2^n - 1 - V \quad (2.1)$$

unde:

$n$  - numărul de biți al reprezentării;

$V$  - valoarea absolută a numărului de reprezentat.

### 2.4.3. Codul complementar

Pentru reprezentarea numerelor negative în codul complementar (complement față de 2) se aplică următoarea regulă de complementare: se reprezintă numărul în valoare absolută, apoi se inversează bit cu bit (inclusiv bitul de semn care devine 1), iar la rezultatul obținut se adună "1". În consecință, complementul față de 2 se obține din complementul față de 1 la care se adună "1". De exemplu, numărul  $N = -5$  în codul complementar va avea valoarea:

$$\begin{array}{r} 00000101 \text{ (inversare)} \\ 11111010 + \\ \hline 1 \\ \hline 11111011 \end{array}$$

Din punct de vedere matematic, complementul față de 2 al unui număr negativ  $N$  este:

$$C_2(N) = 2^n - V \quad (2.2)$$

unde:

$n$  - numărul de biți al reprezentării;

$V$  - valoarea absolută a numărului de reprezentat.

În cazul codului complementar bitul din stânga rămâne tot timpul bit de semn. Codul complementar este cel mai utilizat în reprezentarea numerelor algebrice în calculator.

### 2.5. Exerciții propuse

1) Să se efectueze următoarele operații:

- |                                    |                                   |
|------------------------------------|-----------------------------------|
| a) $1101001_{(2)} + 1010111_{(2)}$ | g) $10110_{(2)} - 1101_{(2)}$     |
| b) $1000100_{(2)} + 1001111_{(2)}$ | h) $11101011_{(2)} - 11101_{(2)}$ |
| c) $1733_{(8)} + 234_{(8)}$        | i) $7100_{(8)} - 324_{(8)}$       |
| d) $1022_{(8)} + 7721_{(8)}$       | j) $1021_{(8)} - 261_{(8)}$       |
| e) $AC97_{(16)} + 33ED_{(16)}$     | k) $AA31_{(16)} - 2FC_{(16)}$     |
| f) $922A_{(16)} + 4522_{(16)}$     | l) $FD124_{(16)} - AF3C_{(16)}$   |

2) Să se efectueze următoarele operații::

a)  $110100110_{(2)} \times 11001_{(2)}$

b)  $100101101_{(2)} \times 10011_{(2)}$

c)  $111010001_{(2)} \times 1110_{(2)}$

d)  $110111101_{(2)} \times 101_{(2)}$

e)  $10111_{(2)} : 110_{(2)}$

f)  $10101_{(2)} : 100_{(2)}$

g)  $110011_{(2)} : 1101_{(2)}$

h)  $100010_{(2)} : 101_{(2)}$

3) Să se reprezinte următoarele numere în sistemul binar, utilizând codul complementar:

a)  $-167$

b)  $-622$

c)  $-1125$

d)  $-96$

e)  $-101$

f)  $-127$

g)  $-23$

h)  $-114$

## LUCRAREA NR. 3

### Sistemul de operare *Linux*. Prezentare și comenzi de bază

---

Această lucrare prezintă o scurtă introducere în sistemul de operare *Linux*, modul de configurare și utilizare al distribuției *Slax 6* precum și o serie de comenzi de bază folosite pentru afișarea informațiilor în sistemul de operare *Linux*.

#### 3.1. Structura sistemului de operare *Linux*

Sistemul de operare *Linux* stochează datele sub formă de fișiere și directoare. În *Linux*, fișierele și directoarele pot fi:

- *reale* - în cazul în care conțin date înregistrate pe mediile de stocare ale calculatorului (ex. *hard disk*);
- *virtuale* - reprezintă legături simbolice către fișiere sau directoare și sunt stocate în memoria RAM.

Din punct de vedere software, în *Linux* orice aplicație lansată în execuție se numește *proces*. Atât comenzile de bază cât și celelalte aplicații *Linux* sunt lansate în execuție cu ajutorul interpretorului de comenzi, denumit *shell*. *Shell*-ul are rolul de a prelua comenzile adresate sistemului de operare de către utilizator sau de către alte procese.

“Nucleul” unui sistem de operare, inclusiv al sistemului de operare *Linux*, se numește *kernel*. *Kernel*-ul are rolul de mediator între programe și componentele *hardware*. De asemenea, *kernel*-ul filtrează pachetele de date care trec prin rețea și gestionează procesele care rulează în memorie. Tot în cadrul *kernel*-ului sunt stocate *driver*ele componentelor *hardware* existente în sistem. În anumite cazuri *driver*ele sunt compilate sub formă de *module*. Principalul avantaj al folosirii *driver*elor sub formă de module este faptul că acestea pot fi încărcate în memorie doar când este necesar. În figura 3.1 este prezentat fluxul normal de date într-un sistem de operare *Linux*.

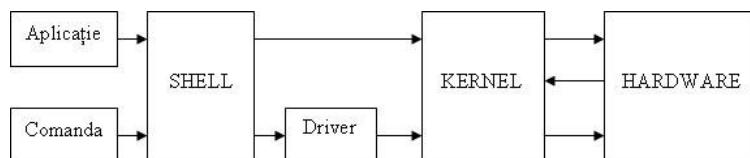


Figura 3.1. Fluxul de date în Linux.

### 3.2. Structura directoarelor în sistemul de operare Linux

În sistemul de operare Linux structura directoarelor este standard, fiecare director având o anumită semnificație. Deoarece există peste 350 de versiuni oficiale de Linux denumite *distribuții*, fiecare distribuție poate avea directoare în plus față de structura standard. În figura 3.2 este prezentată structura standard de directoare a sistemului de operare Linux.

```
root@slax:~# tree -L 1 /
/
|-- bin
|-- boot
|-- dev
|-- etc
|-- home
|-- lib
|-- mnt
|-- opt
|-- proc
|-- root
|-- sbin
|-- sys
|-- tmp
|-- usr
`-- var

15 directories, 0 files
root@slax:~# █
```

Figura 3.2. Structura de directoare în Linux.

Directoarele din structura standard sunt prezentate în tabelul 3.1.

**Tabelul 3.1.** Semnificația directorilor în Linux

Director	Semnificație
<b>/</b>	Reprezintă directorul rădăcină, echivalentul directorului C:\ în sistemul de operare <i>Windows</i> .
<b>/bin</b>	Conține programe executabile (comenzi de bază) fără de care sistemul de operare nu ar funcționa. Aceste programe sunt accesibile tuturor utilizatorilor din sistem.
<b>/boot</b>	Conține imaginea <i>kernelului</i> sub formă de arhivă. Imaginea este dezarhivată în timpul procesului de <i>bootare</i> iar conținutul arhivei ( <i>drivere</i> , programe, etc.) este încărcat în memorie.
<b>/dev</b>	Conține legături simbolice (fișiere virtuale) către componentele <i>hardware</i> și componentele logice ale sistemului (dispozitive virtuale).
<b>/etc</b>	Conține fișiere de configurare a serviciilor și a unor componente <i>hardware</i> . Tot în directorul <i>/etc</i> se găsesc <i>scripturile</i> care inițializează funcțiile <i>kernelului</i> și încarcă în memorie <i>driverele</i> .
<b>/home</b>	Conține directoarele personale ale utilizatorilor cu drepturi limitate. Implicit, fiecărui utilizator adăugat în sistem i se creează în directorul <i>/home</i> un director personal cu acces necondiționat, având numele utilizatorului (ex. <i>/home/foo</i> pt. utilizatorul <i>foo</i> ).
<b>/lib</b>	Conține biblioteci cu funcții și rutine folosite în procesul de <i>bootare</i> și la lansarea în execuție a unor aplicații.
<b>/mnt</b>	Conține directoare în care pot fi montate celelalte dispozitive fizice sau logice de stocare a datelor din sistem (ex. alte partiții sau <i>hard disk-uri</i> , <i>CD-ROM</i> , <i>Floppy</i> , memorii <i>USB</i> , imagini de <i>CD</i> , etc). Directoarele din <i>/mnt</i> sunt echivalentul literelor de unitate din <i>Windows</i> (ex. <i>A</i> - <i>Floopy</i> , <i>D</i> - a doua partiție, <i>E</i> - <i>CD-ROM</i> , etc).
<b>/opt</b>	Conține directoare cu aplicații opționale care au un număr mare de fișiere, altele decât cele din pachetul de bază <i>Linux</i> (ex. KDE [ <i>/opt/kde</i> ] - interfața grafică).
<b>/proc</b>	Conține fișiere și directoare virtuale cu informații despre procese și componentele <i>hardware</i> ale sistemului. Acest director este creat în timpul procesului de <i>bootare</i> , iar informațiile pe care le conține sunt actualizate în timp real.
Continuare în pagina următoare	

Tabela 3.1 – continuare

Director	Semnificație
<b>/root</b>	Reprezintă directorul personal al utilizatorului <i>root</i> . Într-un sistem <i>Linux</i> , utilizatorul <i>root</i> este administratorul sau utilizatorul cu drepturi depline.
<b>/sbin</b>	Conține programe executabile (comenzi de bază) pentru configurarea sistemului. Aceste programe pot fi accesate doar de către administrator (utilizatorul <i>root</i> ).
<b>/tmp</b>	Conține fișiere și directoare folosite de aplicații pentru stocarea datelor temporare. Datele din acest director sunt șterse la fiecare repornire a sistemului.
<b>/usr</b>	Conține aplicații instalate în plus față de pachetul de bază și sunt accesibile tuturor utilizatorilor din sistem. Tot aici sunt stocate (opțional) fișierele sursă ale kernelului ( <i>/usr/src/linux</i> ).
<b>/var</b>	Conține fișiere care sunt modificate dinamic (conținut și dimensiune variabilă). În aceste fișiere sunt stocate datele rezultate în urma monitorizării dinamice a sistemului ( <i>log-uri</i> ).

### 3.3. Comenzi de bază în sistemul de operare *Linux*

Pentru accesarea unei comenzi de bază în *Linux*, utilizatorul trebuie să folosească un program de tip consolă.

În modul text există mai multe aplicații de tip consolă (*shell*): **bash** (*Bourne Again SHell*), **csh** (*C SHell*), **ksh** (*Korn SHell*), etc. În modul grafic al versiunii *Slax* această aplicație se numește **Konsole** (figura 3.3).



Figura 3.3. KDE Konsole.

În continuare sunt prezentate o serie de comenzi de bază în *Linux*, utilizate pentru afișarea anumitor date despre sistem și pentru configurarea sistemului (adăugarea altor utilizatori, ștergerea utilizatorilor, etc.)

**pwd** - afișează numele directorului curent (figura 3.4).

```
root@slax:~# pwd
/root
root@slax:~# █
```

Figura 3.4. Comanda pwd.

**whatis** comandă/aplicație - afișează informații despre o comandă/aplicație (figura 3.5).

```
root@slax:~# whatis uname
uname (1) - print system information
root@slax:~# █
```

Figura 3.5. Comanda whatis.

**whereis** comandă/aplicație - afișează locația unei comenzi/aplicații (figura 3.6).

```
root@slax:~# whereis mc
mc: /usr/bin/mc /usr/lib/mc /usr/share/mc /usr/man/man1/mc.1.gz
root@slax:~# █
```

Figura 3.6. Comanda whereis.

**whoami** - afișează numele utilizatorului curent (figura 3.7).

```
root@slax:~# whoami
root
root@slax:~# █
```

Figura 3.7. Comanda whoami.

**w** - afișează informații despre utilizatorii conectați la sistem (figura 3.8).

```
root@slax:~# w
11:30:43 up 51 min, 1 user, load average: 0.04, 0.13, 0.14
USER  TTY      FROM          LOGIN@  IDLE   JCPU   PCPU WHAT
root  tty1    -             10:40   49:57  0.06s  0.00s /bin/sh /usr/X11R6/bin/startx
root@slax:~# █
```

Figura 3.8. Comanda w.

**man** comandă/aplicație - afișează manualul unei comenzi sau al unei aplicații.

**df** - afișează informații despre partiții (figura 3.9).

**uname** opțiuni - afișează informații despre sistem (figura 3.10).

Opțiuni:

-s - numele kernel-ului;

-n - numele calculatorului în rețea;



```
root@slax:~# df
Filesystem            1K-blocks      Used Available Use% Mounted on
tmpfs                  545852        265368    280484   49% /
/dev/hda1              30716248      8149800   22566448   27% /mnt/hda1
/dev/hda5              37656328     34595968   3060360    92% /mnt/hda5
/dev/hda6              8971120       4180716   4327332    50% /mnt/hda6
root@slax:~# █
```

Figura 3.9. Comanda *df*.

- v - versiunea *kernel*-ului;
- p - tipul procesorului;
- o - numele sistemului de operare;
- a - toate informațiile.

```
root@slax:~# uname -a
Linux slax 2.6.16 #95 Wed May 17 10:16:21 GMT 2006 i686 pentium4 i386 GNU/Linux
root@slax:~# █
```

Figura 3.10. Comanda *uname*.

*dmesg* - afișează informații salvate de *kernel* despre componentele *hardware* și starea acestora (figura 3.11).

```
root@slax:~# dmesg
MSFT 0x00000097) @ 0x1fff00c0
ACPI: DSDT (v001 VIA VIA_K7 0x00001000 MSFT 0x0100000d) @ 0x00000000
ACPI: PM-Timer IO Port: 0x808
Allocating PCI resources starting at 30000000 (gap: 20000000:dec00000)
Built 1 zonelists
Kernel command line: vga=769 changes=slaxsave.dat max_loop=255 initrd=boot/in
linuxrc load_ramdisk=1 prompt_ramdisk=0 ramdisk_size=4444 root=/dev/ram0 rw B
t/vmlinuz
Enabling fast FPU save and restore... done.
Enabling unmasked SIMD FPU exception support... done.
Initializing CPU#0
PID hash table entries: 2048 (order: 11, 32768 bytes)
Detected 1466.945 MHz processor.
Using pmtmr for high-res timesource
Console: colour dummy device 80x25
Dentry cache hash table entries: 131072 (order: 7, 524288 bytes)
Inode-cache hash table entries: 65536 (order: 6, 262144 bytes)
```

Figura 3.11. Comanda *dmesg*.

*date* opțiuni - afișează sau setează data și ora sistemului (figura 3.12). (vezi “*man date*” pentru opțiuni și sintaxa completă a comenzii).

*cal* opțiuni - afișează calendarul lunii sau a anului selectat (figura 3.13).

Opțiuni:

*lună* - calendarul lunii selectate (ex. “*cal 2 2008*” - luna februarie 2008);

```
root@slax:~# date +%F\ %T
2006-11-07 11:48:58
root@slax:~# █
```

Figura 3.12. Comanda date.

**an** - calendarul anului selectat (ex. "cal 2008" - anul 2008);

**-3** - afișează luna precedentă și luna următoare celei selectate.

```
root@slax:~# cal
November 2006
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30

root@slax:~# █
```

Figura 3.13. Comanda cal.

**su nume\_utilizator** - schimbă utilizatorul curent în *nume\_utilizator*.

Comanda "su", utilizată fără parametri, schimbă utilizatorul curent în "root". Dacă se adaugă în fața numelui utilizatorului semnul "-", comanda "su" va schimba automat și directorul curent în directorul personal al noului utilizator.

**adduser** - adaugă un utilizator nou în sistem.

Această comandă este accesibilă doar utilizatorului *root*. Parametrii obligatorii la adăugarea unui nou utilizator sunt numele utilizatorului și parola, restul parametrilor fiind opționali. Implicit, sistemul de operare creează un director personal al noului utilizator în directorul */home*, având numele utilizatorului (ex. */home/test* - pt. utilizatorul *test*).

Adăugarea unui utilizator nou se realizează în două etape:

1. Introducerea datelor specifice contului (nume utilizator, ID, grup, director personal, *shell*, etc.) (figura 3.14)

```
root@slax:~# adduser

Login name for new user []: test

User ID ('UID') [ defaults to next available ]:

Initial group [ users ]:

Additional groups (comma separated) []:

Home directory [ /home/test ]

Shell [ /bin/bash ]

Expiry date (YYYY-MM-DD) []:

New account will be created as follows:

-----
Login name.....: test
UID.....: [ Next available ]
Initial group....: users
Additional groups: [ None ]
Home directory...: /home/test
Shell.....: /bin/bash
Expiry date.....: [ Never ]
```

*Figura 3.14. Comanda adduser - date specifice contului.*

2. Introducerea datelor personale ale utilizatorului (opțional) și setarea parolei (figura 3.15)

```
Creating new account...

Changing the user information for test
Enter the new value, or press ENTER for the default
  Full Name []:
  Room Number []:
  Work Phone []:
  Home Phone []:
  Other []:
Changing password for test
Enter the new password (minimum of 5, maximum of 127 characters)
Please use a combination of upper and lower case letters and numbers.
New password: *****
Re-enter new password: *****
Password changed.

Account setup complete.
root@slax:~# █
```

*Figura 3.15. Comanda adduser - date utilizator.*

**userdel** opțiuni nume\_utilizator - șterge un utilizator din sistem.

Opțiuni:

**-r** - șterge și directorul personal al utilizatorului (/home/nume\_utilizator).

**finger** *opțiuni nume\_utilizator* - afișează informații despre utilizatori.

*Opțiuni:*

-**s** - afișează (dacă sunt setate) numele utilizatorului, numele real al utilizatorului, terminalul (tipul consolei), timpul scurs de la ultima autentificare, ora autentificării, locația biroului și numărul de telefon;

-**p** - afișează în plus față de opțiunea "-s" și directorul personal al utilizatorului, numărul de telefon personal, numele *shell*-ului și starea adresei de email.

**chfn** *opțiuni nume\_utilizator* - schimbă informațiile despre utilizatori.

*Opțiuni:*

-**f** *nume\_real* - schimbă numele real al utilizatorului;

-**r** *locație\_birou* - schimbă numele locației biroului;

-**w** *numar\_telefon* - schimbă numărul de telefon al biroului;

-**h** *numar\_telefon* - schimbă numărul de telefon personal al utilizatorului;

-**o** *alte\_informații* - schimbă sau adaugă informații suplimentare despre contul utilizatorului.

La adăugarea noilor informații (mai puțin pentru opțiunea "-o") nu este permisă folosirea caracterelor ",", ".", și "=".

**passwd** *nume\_utilizator* - schimbă parola unui utilizator.

*Opțiuni:*

**nume\_utilizator** - opțiunea poate fi folosită doar de utilizatorul *root* pentru a schimba parola utilizatorului *nume\_utilizator*.

Comanda *passwd*, fără opțiuni, poate fi utilizată de către orice utilizator pentru schimbarea parolei propriului cont. La schimbarea parolei, comanda *passwd* cere vechea parolă a contului. Utilizatorul *root* poate schimba parola altui utilizator fără a avea nevoie de vechea parolă a utilizatorului.

**find** *locație opțiuni* - caută fișiere sau directoare în directorul *locație*.

*Opțiuni:*

-**name** '*expresie*' - caută fișierele sau directoarele de forma *expresie* (ex. "*find* / -name '\*.c'" - caută toate fișierele cu extensia ".c" din directorul "/" și din subdirectoarele acestuia);

-**type** *tip* - caută fișierele sau directoarele de tipul *tip*. Tipurile fișierelor sau directoarelor pot fi: **b** sau **c** - fișiere speciale; **d** - director; **f** - fișier normal; **p** - pipe;

**l** - legătură simbolică (*shortcut*); **s** - soclu. (ex. "*find / -name 'etc' -type d*" - caută directorul "*etc*" în directorul "/" și în subdirectoarele acestuia);

**-maxdepth nivel** - nivelul subdirectoarelor în care să se realizeze căutarea (ex. *find / -name '\*' -type d -maxdepth 1* - afișează primul nivel de subdirectoare din "/");

**-printf 'format'** - afișează rezultatul căutării în formatul specificat (ex. "*find / -name '\*.c' -type f -printf '%p %s\n'*" - afișează numele și mărimea fișierelor cu extensia '.c' din directorul "/" și subdirectoarele acestuia). (Vezi "*man find*" pentru informații suplimentare).

**shutdown opțiuni now** - repornește sau oprește calculatorul.

Opțiuni:

**-r** - repornește calculatorul;

**-h** - oprește calculatorul.

**reboot** - repornește calculatorul (echivalent cu *shutdown -r now*).

**poweroff** - oprește calculatorul (echivalent cu *shutdown -h now*).

### 3.4. Exerciții propuse

1) Să se adauge în sistem un cont pentru utilizatorul *student* știind că:

- nume utilizator: *student*
- shell: */bin/bash*
- parola: *student123*
- nume real: *Student anul 1*
- locație birou: *sala 309*
- număr telefon birou: *0264401200*

2) Să se caute și să se afișeze toate fișierele cu extensia '*.conf*' din directorul '*/etc*' fără a căuta în subdirectoarele acestuia.

3) Să se afișeze data și ora sistemului în următorul format:

*ZZ.LL.AAAA ora:minut:secunda*

## LUCRAREA NR. 4

# Gestionarea fișierelor, directoarelor și proceselor în *Linux*

---

Lucrarea prezintă o serie de comenzi de bază folosite pentru manipularea fișierelor, directoarelor și proceselor în sistemul de operare *Linux*. De asemenea, lucrarea prezintă modul de lucru cu editoarele simple de text și câteva comenzi de bază folosite pentru setarea drepturilor utilizatorilor asupra fișierelor și directoarelor.

### 4.1. Operatori de redirecționare

În *Linux* există trei dispozitive logice de intrare/ieșire:

- Intrarea standard (*stdin*), de la care se citesc datele de intrare. Implicit, intrarea standard are asociată tastatura.
- Ieșirea standard (*stdout*), unde sunt afișate datele de ieșire. Implicit, ieșirea standard are asociat terminalul curent (consola).
- Ieșirea de eroare standard (*stderr*), la care sunt afișate mesajele de eroare. Implicit, aceasta are asociat terminalul curent (consola).

În cadrul *shell*-ului, există posibilitatea redirecționării acestor dispozitive, după cum urmează:

- redirecționarea intrării standard se realizează prin intermediul operatorului de redirecționare "<";
- redirecționarea ieșirii standard se realizează cu ajutorul operatorului ">". De exemplu, comanda "*dmesg > fisier1*" va trimite ieșirea comenzii "*dmesg*" către fișierul "*fisier1*". De asemenea, poate fi utilizat și operatorul ">>" care, spre deosebire de operatorul ">", nu suprascrie fișierul spre care se face redirecționarea, ci adaugă ieșirea la sfârșitul acestuia;
- redirecționarea ieșirii de eroare se realizează cu "*2 >*", cifra 2 reprezentând numărul *descriptorului de fișier* corespunzător ieșirii standard pentru erori (*stderr*).

Pentru readirecționare, în linia de comandă, pot fi utilizați unul sau mai mulți operatori:

*comanda < date\_intrare > rezultate*

Un mecanism frecvent utilizat în linia de comandă este mecanismul “pipe” care constă în înlănțuirea comenzilor: prima comandă trimite ieșirea standard celei de-a doua comenzi, ș.a.m.d. Trimiterea ieșirii se realizează cu operatorul “|”. Sintaxa acestui mecanism este:

*comanda1|comanda2*

O utilizare frecventă a acestui mecanism o constituie comenzile de tip *filtru*. Cele mai uzuale comenzi de acest gen sunt:

**more** - paginează textul primit ca intrare, cu posibilitatea de defilare în jos cu câte o linie (cu tasta *Enter*) sau cu câte o pagină de ecran (cu tasta *Space*) (ex. “*dmesg|more*”).

**less** - paginează textul primit ca intrare (ca și comanda *more*), cu posibilitatea de defilare în jos sau în sus cu câte o linie (cu tastele ↓ și ↑) sau cu câte o pagină de ecran (cu tastele *PageDown* și *PageUp*) (ex. *dmesg | less*).

**grep expresie** - caută șirul de caractere *expresie* în cadrul intrării, transmițând la ieșire doar liniile de text care conțin respectivul șir (ex. “*dmesg|grepCPU*” - afișează doar liniile care conțin șirul de caractere “*CPU*” din ieșirea comenzii “*dmesg*”). Șirul de caractere căutat poate conține unul din următoarele *meta*-caractere:

- “^” - indică începutul unei linii (ex. “*dmesg | grep ^CPU*” - afișează doar liniile care conțin la început șirul de caractere “*CPU*”);
- “\$” - indică sfârșitul unei linii (ex. “*dmesg | grep "IRQ 18"\$*” - afișează doar liniile care conțin la sfârșit șirul de caractere “*IRQ 18*”).

### 4.2. Editoare de text

Unul din cele mai simple editoare de text din Linux se numește *nano*. Acesta se poate lansa în execuție prin apăsarea lui în linia de comandă a consolei, fără nici

un parametru suplimentar sau având ca și parametru numele fișierului de editat (ex. *nano fisier1*).

Editorul *nano* conține două zone principale: zona de editare și meniul cu funcții accesibile în editor. Funcțiile accesibile utilizatorului sunt prezentate în tabelul 4.1.

**Tabelul 4.1.** Funcții ale editorului *nano*

Funcție		Semnificație
Ctrl+G	Get Help	Afișează informații detaliate despre funcțiile editorului
Ctrl+X	Exit	Închide editorul
Ctrl+O	WriteOut	Scrie modificările în fișier
Ctrl+J	Justify	Formatează textul din paragraful curent
Ctrl+R	Read File	Adaugă conținutul unui fișier în pagina curentă
Ctrl+W	Where Is	Caută un cuvânt și poziționează cursorul la începutul acestuia
Ctrl+Y	Prev Pg	Trece în pagina precedentă
Ctrl+V	Next Pg	Trece în pagina următoare
Ctrl+K	Cut Text	Șterge linia curentă și o copiază în memorie
Ctrl+U	UnCut Text	Scrie linia copiată în memorie la poziția cursorului
Ctrl+C	Cur Pos	Afișează poziția curentă a cursorului
Ctrl+T	To Spell	Apelează corectorul de text

În cazul în care editorul este lansat în execuție fără nici un parametru, acesta va cere salvarea datelor într-un fișier în momentul apelării funcțiilor *WriteOut* sau *Exit*.

Pe lângă editorul *nano*, în *Linux* se pot folosi editoare de text mult mai complexe, cum ar fi: *vi*, *vim*, *latex*, *joe* în consolă sau *KEdit*, *KWrite*, *gvim* în modul grafic.

### 4.3. Fișiere și directoare

Ca și în alte sisteme de operare, interpretorul de comenzi din *Linux* permite vizualizarea, copierea, mutarea, ștergerea sau redenumirea fișierelor și directoarelor. Comenzile destinate acestor operații sunt accesibile tuturor utilizatorilor. Aceste operații sunt condiționate de drepturile utilizatorului asupra fișierelor sau directoarelor asupra cărora se efectuează operația.

**ls opțiuni director** - afișează fișierele și subdirectoarele dintr-un director (figura 4.1).



```
root@slax:~# ls -al /
total 2
drwxr-xr-x 60 root root 220 Oct 24 13:46 ./
drwxr-xr-x 60 root root 220 Oct 24 13:46 ../
drwxrwxrwx 2 root root 41 May 11 2006 bin/
drwxr-xr-x 5 root root 2048 Mar 12 2008 boot/
drwxr-xr-x 17 root root 14460 Oct 24 13:47 dev/
drwxrwxrwx 50 root root 240 Oct 24 13:47 etc/
drwxr-xr-x 2 root root 3 Oct 6 1997 home/
drwxr-xr-x 2 root root 32 Oct 22 06:03 laboratoare/
drwxrwxrwx 8 root root 182 Apr 6 2007 lib/
drwxr-xr-x 8 root root 160 Oct 24 13:46 mnt/
drwxr-xr-x 3 root root 20 Nov 12 2007 opt/
dr-xr-xr-x 137 root root 0 Oct 24 13:45 proc/
drwxr-xr-x 9 root root 260 Oct 24 13:52 root/
drwxr-xr-x 2 root root 20 May 8 16:21 sbin/
drwxr-xr-x 2 root root 3 Apr 7 2007 srv/
drwxr-xr-x 13 root root 0 Oct 24 13:45 sys/
drwxrwxrwt 6 root root 140 Oct 24 13:48 tmp/
drwxr-xr-x 82 root root 60 Sep 18 2006 usr/
drwxr-xr-x 35 root root 20 Dec 27 2006 var/
root@slax:~#
```

Figura 4.1. Comanda ls.

#### Opţiuni:

(fără opţiuni) - afişează doar numele fişierelor şi subdirectoarelor;

**-l** - afişează următoarele informaţii suplimentare despre fişiere şi directoare: drepturile de acces, numărul de legături, dimensiunea fişierelor, data ultimei actualizări, numele fişierelor sau directoarelor;

**-h** - afişează dimensiunea fişierelor în *Kb*, *Mb*, *Gb* (implicit, dimensiunea este afişată în bytes);

**-t** - sortează fişierele şi directoarele după data ultimei actualizări;

**-u** - la afişare se ia în considerare data ultimei accesări în loc de data ultimei actualizări (pentru opţiunile **-l** şi **-t**);

**-r** - inversează ordinea de sortare;

**-R** - afişează şi conţinutul subdirectoarelor;

**-a** - afişează şi directoarele **."** şi **.."** (**."** - directorul curent; **.."** directorul părinte);

**-A** - nu afişează directoarele **."** şi **.."**.

**cd locaţie** - schimbă directorul curent în *locaţie* (figura 4.2). Parametrul *locaţie* poate avea ca şi valori numele unui director sau următoarele legături simbolice:

- “.” - schimbă directorul curent în directorul părinte (ex. `cd ..`);
- “~” - schimbă directorul curent în directorul personal al utilizatorului curent (ex. `cd ~` - schimbă directorul curent în `/root` dacă utilizatorul curent este `root`).

```
root@slax:~# pwd
/root
root@slax:~# cd /etc
root@slax:/etc# pwd
/etc
root@slax:/etc# cd ..
root@slax:/# pwd
/
root@slax:/# cd ~
root@slax:~# pwd
/root
root@slax:~# █
```

Figura 4.2. Comanda `cd`.

**touch** *opțiuni nume\_fișier* - creează un fișier sau schimbă data accesării unui fișier (figura 4.3).

*Opțiuni:*

(fără opțiuni) - creează un fișier nou;

**-a** - schimbă data ultimei accesări a fișierului;

**-m** - schimbă data ultimei modificări a fișierului.

```
root@slax:~# touch fisier1
root@slax:~# ls -al fisier1
-rw-r--r-- 1 root root 0 Oct 24 14:03 fisier1
root@slax:~# █
```

Figura 4.3. Comanda `touch`.

**cat** *operator nume\_fișier* - afișează conținutul unui fișier (figura 4.4). Comanda `cat` se poate folosi și cu operatorii de redirecționare a ieșirii pentru scrierea datelor în fișier.

```
root@slax:~/test# cat > fisier
acesta este un fisier text
root@slax:~/test# cat fisier
acesta este un fisier text
root@slax:~/test# cat >> fisier
completare la fisierul text
root@slax:~/test# cat fisier
acesta este un fisier text
completare la fisierul text
root@slax:~/test# █
```

Figura 4.4. Comanda cat.

**mkdir** *opţiuni* nume\_director - creează unul sau mai multe directoare (figura 4.5).

*Opţiuni:*

(fără *opţiuni*) - creează un director nou;

**-v** - afişează un mesaj la crearea fiecărui director;

**-p** - creează un director împreună cu directoarele *părinte*, dacă acestea nu există.

```
root@slax:~# mkdir test
root@slax:~# cd test
root@slax:~/test# ls -al
total 0
drwxr-xr-x  2 root root  40 Oct 24 14:05 ./
drwxr-xr-x 11 root root 440 Oct 24 14:05 ../
root@slax:~/test# mkdir director1
root@slax:~/test# ls -al
total 0
drwxr-xr-x  3 root root  60 Oct 24 14:05 ./
drwxr-xr-x 11 root root 440 Oct 24 14:05 ../
drwxr-xr-x  2 root root  40 Oct 24 14:05 director1/
root@slax:~/test# █
```

Figura 4.5. Comanda mkdir.

**rmdir** *opţiuni* nume\_director - şterge unul sau mai multe directoare.

*Opţiuni:*

(fără *opţiuni*) - şterge un director (dacă acesta este gol);

**-v** - afişează un mesaj la ştergerea fiecărui director;

**-p** - şterge o structură de directoare (ex. *rm -p test/dir1* - şterge directorul *test* şi subdirectorul *dir1*).

**rm** *opţiuni* fişier/director - şterge unul sau mai multe fişiere sau directoare (figura

4.6).

*Opțiuni:*

- i - cere permisiunea de ștergere a fiecărui fișier sau director;
- r - șterge recursiv conținutul subdirectoarelor (fișierele și subdirectoarele acestora);
- v - afișează un mesaj la ștergerea fiecărui fișier sau director.

```
root@slax:~/test# ls -al fisier2
/bin/ls: cannot access fisier2: No such file or directory
root@slax:~/test# touch fisier2
root@slax:~/test# ls -al fisier2
-rw-r--r-- 1 root root 0 Oct 24 14:07 fisier2
root@slax:~/test# rm -i fisier2
rm: remove regular empty file `fisier2'? y
root@slax:~/test# ls -al fisier2
/bin/ls: cannot access fisier2: No such file or directory
root@slax:~/test#
```

Figura 4.6. Comanda rm.

**cp** *opțiuni sursă destinație* - creează o copie a fișierului/directorului *sursă* în directorul *destinație* (dacă *destinație* este numele unui director existent) (figura 4.7). Dacă *sursă* este un fișier iar *destinație* nu există, se creează o copie a fișierului *sursă* cu numele *destinație*.

*Opțiuni:*

(fără opțiuni) - copiază *sursă* în *destinație*;

- v - afișează un mesaj la copierea fiecărui fișier/director;
- R - copiază recursiv directorul *sursă* în directorul *destinație*;
- f - forțează rescrierea fișierului *destinație* dacă acesta există;
- i - cere permisiunea de rescriere a fiecărui fișier/director;
- s - creează o legătură simbolică a sursei în *destinație*.

```
root@slax:~/test# touch fisier3
root@slax:~/test# ls -al
total 0
drwxr-xr-x  2 root root  60 Oct 24 14:09 ./
drwxr-xr-x 11 root root 480 Oct 24 14:08 ../
-rw-r--r--  1 root root   0 Oct 24 14:09 fisier3
root@slax:~/test# cp fisier3 fisier4
root@slax:~/test# ls -al
total 0
drwxr-xr-x  2 root root  80 Oct 24 14:09 ./
drwxr-xr-x 11 root root 480 Oct 24 14:08 ../
-rw-r--r--  1 root root   0 Oct 24 14:09 fisier3
-rw-r--r--  1 root root   0 Oct 24 14:09 fisier4
root@slax:~/test# █
```

Figura 4.7. Comanda cp.

**mv** opţiuni *sursă* *destinaţie* - redenumeste sau mută un fişier sau un director (figura 4.8). Dacă *sursă* este fişier iar *destinaţie* director, comanda *mv* mută fişierul *sursă* în directorul *destinaţie*.

Opţiuni:

- v - afişează un mesaj la redenumirea/mutarea fiecărui fişier/director;
- u - mută fişierele sau directoarele *sursă* în directorul *destinaţie* doar dacă sursa este mai nouă sau nu există în directorul *destinaţie*;
- f - forţează rescrierea fişierului *destinaţie* dacă acesta există;
- i - cere permisiunea de rescriere a fiecărui fişier/director.

```
root@slax:~/test# ls -al fisier4
-rw-r--r-- 1 root root 0 Oct 24 14:09 fisier4
root@slax:~/test# mv -v fisier4 fisier5
'fisier4' -> 'fisier5'
root@slax:~/test# ls -al fisier4
/bin/ls: cannot access fisier4: No such file or directory
root@slax:~/test# ls -al fisier5
-rw-r--r-- 1 root root 0 Oct 24 14:09 fisier5
root@slax:~/test# █
```

Figura 4.8. Comanda mv.

#### 4.4. Procese

*Procesul* stă la baza oricărei activităţi în sistemul de operare *Linux*. La lansarea în execuţie a unei comenzi sau a unei aplicaţii se creează un nou proces. Controlul proceselor este un element important în programarea pe sisteme *multitasking* care include operaţii de creare, manipulare şi sincronizare a proceselor. Controlul

proceselor este realizat prin câteva apeluri (semnale) sistem care sunt funcții înglobate în nucleu, accesibile utilizatorilor. Utilizarea corespunzătoare a apelurilor este esențială pentru controlul corect al proceselor.

Un proces reprezintă instanța execuției unei aplicații și nu trebuie confundat cu programul, care este, în fond, fișierul executat de proces. Într-un sistem *multitasking*, mai multe procese pot executa concurrent același program și fiecare proces poate fi transformat să execute un program anume.

În *Linux*, fiecare proces are asociat un număr de identificare unic denumit *identificator de proces*, prescurtat *PID*. *PID*-ul este un număr întreg pozitiv care poate fi reatribuit după terminarea procesului care-l deține.

În continuare sunt prezentate câteva comenzi utilizate pentru afișarea și controlul proceselor în *Linux*.

**ps opțiuni** - afișează procesele care rulează în sistem.

*Opțiuni:*

**-a** - afișează toate procesele care rulează în sistem (nu doar cele lansate de utilizatorul curent);

**-u** - afișează numele utilizatorului care controlează procesul;

**-U nume\_utilizator** - afișează procesele pornite de un anumit utilizator ;

**-x** - afișează toate procesele pornite de utilizatorul curent.

În figura 4.9 este prezentat un exemplu al comenzii *ps*.

```
root@slax:~# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.2  0.0   772   308 ?        Ss   13:45   0:01 init [4]
root         2  0.0  0.0     0     0 ?        S<   13:45   0:00 [kthreadd]
root         3  0.0  0.0     0     0 ?        S<   13:45   0:00 [migration/0]
root         4  0.0  0.0     0     0 ?        S<   13:45   0:00 [ksoftirqd/0]
root         5  0.0  0.0     0     0 ?        S<   13:45   0:00 [migration/1]
root         6  0.0  0.0     0     0 ?        S<   13:45   0:00 [ksoftirqd/1]
root         7  0.0  0.0     0     0 ?        S<   13:45   0:00 [events/0]
root         8  0.0  0.0     0     0 ?        S<   13:45   0:00 [events/1]
root         9  0.0  0.0     0     0 ?        S<   13:45   0:00 [khelper]
root        108  0.0  0.0     0     0 ?        S<   13:45   0:00 [kblockd/0]
```

Figura 4.9. Comanda *ps*.

Datele afișate în figura 4.9 au următoarele semnificații:

- *USER* - numele utilizatorului care a lansat procesul;
- *PID* - *ID*-ul procesului;

- **%CPU** - procentul de utilizare a procesorului;
- **%MEM** - procentul de utilizare a memoriei;
- **VSZ** - dimensiunea procesului în *Kbytes*;
- **RSS** - memoria RAM utilizată de proces;
- **TTY** - terminalul utilizat de proces (dacă există);
- **STAT** - starea procesului (vezi tabelul 4.2);
- **START** - data/ora la care a fost pornit procesul;
- **TIME** - timpul de utilizare a procesorului (cumulat);
- **COMMAND** - numele comenzii/aplicaţiei pe care o execută procesul.

**Tabelul 4.2.** Starea proceselor în Linux

Sîmbol	Semnificaţie
<b>D</b>	Proces continuu (neînterupt)
<b>R</b>	Proces activ
<b>S</b>	Proces în aşteptare
<b>T</b>	Proces oprit
<b>W</b>	Proces care realizează o paginare
<b>X</b>	Proces terminat
<b>Z</b>	Proces terminat vizibil ( <i>Zombie</i> )
<b>&lt;</b>	Proces cu prioritate mare
<b>N</b>	Proces cu prioritate mică
<b>L</b>	Proces în timp real (cu zonă de memorie blocată)
<b>s</b>	Proces care controlează o sesiune
<b>l</b>	Proces multifilar
<b>+</b>	Proces care rulează în prim plan ( <i>foreground</i> )

**pstree** *opţiuni* - afişează arborescent procesele care rulează în sistem.

*Opţiuni:*

- p** - afişează numele şi PID-ul procesului;
- n** - ordonează după *PID* procesele care au acelaşi proces părinte. Implicit, ordonarea se face după numele procesului.

**kill** *opțiuni PID* - trimite un semnal procesului cu ID-ul *PID*.

*Opțiuni:*

-**l** - afișează lista semnalelor care pot fi trimise către un proces;

-**N** - trimite semnalul cu ID-ul *N* către procesul cu ID-ul *PID*. (vezi *kill -l*).

Pentru oprirea unui proces se utilizează semnalul cu ID-ul 9 (*SIGKILL*) (ex. *kill -9 1234* )

**killall** *opțiuni nume\_proces* - trimite un semnal procesului cu numele *nume\_proces*. Opțiunile comenzii *killall* sunt aceleași ca și ale comenzii *kill*. Avantajul comenzii *killall* este posibilitatea opririi mai multor procese cu același nume.

#### 4.5. Drepturi asupra fișierelor și directoarelor

În *Linux*, conturile utilizatorilor se împart în două categorii: conturi create de către administrator (de către *root*) și conturi create de către sistemul de operare. Utilizatorii creați de către sistemul de operare se numesc *demoni* (*daemons*). *Demonii* nu oferă posibilitatea de autentificare deoarece au destinații speciale cum ar fi: lansare în execuție a unor servicii (*http, ftp, baze de date, etc*), încărcarea în memorie a *driverelor*, comunicația cu anumite periferice sau componente *hardware*, etc. Un exemplu de *daemon* este contul utilizator denumit *gpm*, care gestionează comunicația sistemului de operare cu *mouse-ul*.

Utilizatorii au drepturi limitate în sistem, excepție făcând utilizatorul *root*. Fiecare utilizator face parte dintr-unul sau mai multe grupuri de utilizatori. Implicit, grupul utilizatorilor cu drepturi limitate este grupul *users*. Grupurile de utilizatori sunt utile când se dorește acordarea unor drepturi suplimentare asupra fișierelor sau directoarelor.

Drepturile asupra unui fișier sau director pot fi vizualizate cu ajutorul comenzii "*ls -l nume\_fișier/nume\_director*". Prima coloană reprezintă drepturile asupra fișierului (sau directorului). A treia și a patra coloană reprezintă numele proprietarului, respectiv a grupului de utilizatori din care face parte proprietarul (utilizatorul care a creat fișierul sau directorul). Drepturile sunt reprezentate de un șir format din 10 caractere din care:

- primul caracter - reprezintă tipul fișierului ( "-" - fișier normal; "d" - director; "l" - legătură simbolică; "c" - fișier special; "s" - soclu (socket); "p" - conducte (pipes).);



- caracterele 2, 3 şi 4 - drepturile proprietarului;
- caracterele 5, 6 şi 7 - drepturile utilizatorilor din acelaşi grup cu proprietarul;
- caracterele 8, 9 şi 10 - drepturile tuturor utilizatorilor (alţii decât proprietarul şi cei din acelaşi grup cu proprietarul).

Drepturile utilizatorilor pot fi:

- **r** - (read) drept de citire;
- **w** - (write) drept de scriere;
- **x** - (execute) drept de execuţie.

Schimbarea drepturilor asupra unui fişier sau director poate fi făcută de către proprietarul fişierului sau directorului (sau de către *root*) cu ajutorul comenzii *chmod*.

**chmod** *drepturi fişier/director* - schimbă drepturile asupra unui fişier sau director.

*drepturi* - reprezintă noile drepturi sub forma unui şir de caractere, format din trei grupuri de caractere, după cum urmează:

*grupul 1:*

- u** - drepturile proprietarului;
- g** - drepturile grupului;
- o** - drepturile celorlalţi utilizatori;
- a** - drepturile tuturor utilizatorilor.

*grupul 2:*

- +** adaugă drepturi;
- şterge drepturi;
- =** şterge toate drepturile si setează noile drepturi.

*grupul 3:*

- r** - drept de citire;
- w** - drept de scriere;
- x** - drept de execuţie.

De exemplu, comanda "*chmod ug+rw fisier1*" va adăuga drepturile *citire* şi *scriere* pentru proprietar şi grup asupra fişierului *fisier1*.

Comanda *chmod* se mai poate folosi şi cu drepturile exprimate în formă numerică. Fiecare drept are atribuit un număr întreg unic după cum urmează:

- 4 - drept de citire;
- 2 - drept de scriere;
- 1 - drept de execuție;
- 0 - nici un drept.

Drepturile în format numeric sunt exprimate sub forma unui număr format din 3 cifre, fiecare reprezentând suma drepturilor pentru proprietar, grup și alți utilizatori. De exemplu, comanda "*chmod 764 fisier*" va seta drepturile "*rw*" pentru proprietar (4+2+1), "*rw*" pentru grup (4+2) și "*r*" pentru alți utilizatori (4).

Numerele atribuite drepturilor nu sunt arbitrar alese. Drepturile pentru fiecare grup de utilizatori (proprietar, grup, alții) sunt reprezentate în binar pe 3 biți. Astfel pentru fiecare drept se setează bitul corespunzător, în ordinea *citire, scriere, execuție* rezultând o combinație de drepturi sub forma unui număr în binar pe 3 biți (de la 000 la 111) reprezentat în zecimal sub forma unei cifre (de la 0 la 7).

În figura 4.10 este prezentat un exemplu pentru comanda *chmod* (drepturi literare și numerice).

```
root@slax:~/test# touch drepturi
root@slax:~/test# ls -al drepturi
-rw-r--r-- 1 root root 0 Oct 24 14:12 drepturi
root@slax:~/test# chmod go+w drepturi
root@slax:~/test# ls -al drepturi
-rw-rw-rw- 1 root root 0 Oct 24 14:12 drepturi
root@slax:~/test# chmod 600 drepturi
root@slax:~/test# ls -al drepturi
-rw----- 1 root root 0 Oct 24 14:12 drepturi
root@slax:~/test# █
```

Figura 4.10. Comanda *chmod*.

**chown** *proprietar\_nou fișier* - schimbă proprietarul al unui fișier sau director (figura 4.11).

```
root@slax:~/test# touch fisier_owner
root@slax:~/test# ls -al fisier_owner
-rw-r--r-- 1 root root 0 Oct 24 14:18 fisier_owner
root@slax:~/test# chown student fisier_owner
root@slax:~/test# ls -al fisier_owner
-rw-r--r-- 1 student root 0 Oct 24 14:18 fisier_owner
root@slax:~/test# █
```

Figura 4.11. Comanda *chown*.

**chgrp** *grup\_nou* *fișier* - schimbă grupul curent al unui fișier sau director (figura 4.12).

```
root@slax:~/test# touch fisier_grup
root@slax:~/test# ls -al fisier_grup
-rw-r--r-- 1 root root 0 Oct 24 14:15 fisier_grup
root@slax:~/test# chgrp users fisier_grup
root@slax:~/test# ls -al fisier_grup
-rw-r--r-- 1 root users 0 Oct 24 14:15 fisier_grup
root@slax:~/test# █
```

Figura 4.12. Comanda *chgrp*.

#### 4.6. Exerciții propuse

- 1) Să se creeze în directorul */tmp* un fișier care să conțină toate liniile care încep cu un spațiu urmat de cuvântul *BIOS* din *output*-ul comenzii *dmesg*. Să se determine mărimea fișierului rezultat cu ajutorul comenzii *find*.
- 2) Să se creeze în directorul */root* un subdirector denumit *dmesg*. În subdirectorul *dmesg* să se creeze un alt subdirector denumit *bios*. Să se mute fișierul creat la *exercițiul 1* în subdirectorul *bios*. Să se creeze o copie a fișierului creat la *exercițiul 1* în subdirectorul *dmesg*. Să se mute subdirectorul *dmesg* în directorul */tmp*.
- 3) Să se seteze următoarele drepturi pentru fișierul creat la *exercițiul 1*:

a) -rwxr-xr-x	d) -r-x- -x- -x
b) -rw-rw-rw-	e) -r- -r- - r- -
c) -rwxrwxr- -	f) -rw-r- -r- -

## LUCRAREA NR. 5

### Fișiere de comenzi - noțiuni de bază

---

Lucrarea prezintă modul de funcționare al fișierelor executabile în *Linux* precum și metodele de realizare și modul de funcționare al fișierelor de comenzi (*shell scripts*).

#### 5.1. Fișiere executabile

În sistemul de operare *Linux* există două tipuri de fișiere executabile:

- fișiere executabile de tip *ELF* (*Executable Linking Format*);
- fișiere executabile de tip *script* (fișier de comenzi sau *script shell*).

Fișierele executabile de tip *ELF* sunt fișiere compilate binar (conținut criptat de compilator) iar fișierele de comenzi sunt fișiere text care conțin seturi de comenzi specifice interpretorului de comenzi utilizat.

Primele caractere ale unui fișier executabil de tip *ELF* sunt chiar caracterele *ELF*, restul caracterelor reprezentând date criptate de compilator. În cazul fișierelor de comenzi, prima linie este întotdeauna numele interpretorului de comenzi, precedat de caracterele *"#!"* (ex. *"#!/bin/bash"*).

Un fișier executabil se poate lansa în execuție doar dacă are setate drepturile de execuție. Odată setate aceste drepturi, fișierul executabil se apelează astfel: *.nume\_fișier* (figura 5.1)

```
root@slax:~$ cat > exec.sh
#!/bin/bash
echo "Exemplu fisier executabil"
root@slax:~$ ls -al exec.sh
-rw-r--r-- 1 root root 45 2008-10-29 13:40 exec.sh
root@slax:~$ chmod +x exec.sh
root@slax:~$ ls -al exec.sh
-rwxr-xr-x 1 root root 45 2008-10-29 13:40 exec.sh
root@slax:~$ ./exec.sh
Exemplu fisier executabil
root@slax:~$ █
```

Figura 5.1. Apelarea unui fișier executabil.

Fișierele executabile care se află în directoarele `"/bin"`, `"/sbin"`, `"/usr/bin"` și `"/usr/sbin"` nu trebuie să fie precedate de `"/"` în momentul apelării. Interpretorul de comenzi va căuta numele executabilului apelat de la tastatură în aceste directoare. Dacă executabilul nu este găsit, interpretorul va afișa mesajul de eroare: `-bash: comanda: command not found`.

## 5.2. Variabile

Variabilele sunt seturi de date care conțin diferite informații utilizabile de către aplicații sau de către alte variabile. În interpretoarele de comenzi din *Linux* există trei tipuri de variabile: variabile sistem, variabile ale scripturilor shell și variabile definite de utilizatori. Orice variabilă în *Linux* se notează cu `"$nume_variabilă"`. În tabelul 5.1 sunt prezentate o parte din variabilele sistem împreună cu semnificația lor.

Tabelul 5.1. Variabile sistem în Linux

Variabilă	Semnificație
<b>\$HOME</b>	conține numele directorului personal al utilizatorului curent
<b>\$HOSTNAME</b>	conține numele sistemului în rețea
<b>\$MACHTYPE</b>	conține numele arhitecturii sistemului
<b>\$OSTYPE</b>	conține numele sistemului de operare
<b>\$PATH</b>	conține lista de directoare în care <i>shell</i> -ul va căuta executabilele
<b>\$PWD</b>	conține numele directorului curent
<b>\$SHELL</b>	conține numele interpretorului de comenzi utilizat
<b>\$TERM</b>	conține numele terminalului (consolei)
<b>\$UID</b>	conține ID-ul utilizatorului curent
<b>\$USER</b>	conține numele contului utilizatorului curent

Variabilele pot fi definite în două moduri: `nume_variabilă="valoare"` sau `export nume_variabilă="valoare"`.

Valoarea unei variabile se poate afișa cu ajutorul comenzii *echo* (figura 5.2).

**echo** *opțiuni* `$nume_variabilă` - afișează valoarea unei variabile.

*Opțiuni:*

**-n** - păstrează cursorul în linia curentă după afișare

```
root@slax:~$ export var1="Continut variabila var1"
root@slax:~$ echo $var1
Continut variabila var1
root@slax:~$ echo -n $var1
Continut variabila var1root@slax:~$
```

Figura 5.2. Comanda echo.

### 5.3. Scripturi Shell

Un *script shell* este un fișier executabil care conține seturi de comenzi de bază și/sau comenzi specifice *shell*-urilor. Orice *script shell* este inițial un fișier text în care se scriu comenzile necesare. După editare, scripturile trebuie transformate în fișiere executabile utilizând comanda "*chmod*". Lansarea în execuție a unui *script shell* se poate face în trei moduri:

- ***./nume\_fișier* argumente** - dacă fișierul executabil se află în directorul curent;
- ***/locuție/./nume\_fișier* argumente** - dacă fișierul nu se află în directorul curent;
- ***nume\_fișier* argumente** - dacă fișierul executabil se află într-unul din directoarele salvate în variabila \$PATH.

Parametrul "*argumente*" reprezintă valorile care se transferă spre *scriptul shell* direct din linia de comandă. Dacă se dorește transferul mai multor valori, acestea vor fi delimitate prin *spațiu* iar dacă valorile conțin *spațiu* acestea vor fi încadrate între ghilimele (ex. *./script 12 "ab cd"* - unde "12" reprezintă valoarea argumentului 1 iar "ab cd" valoarea argumentului 2).

Într-un *script shell* comenzile pot fi scrise pe linii separate sau în aceeași linie delimitate de ";". Prima linie a unui *script* va reprezenta întotdeauna locația și numele interpretorului de comenzi (ex. *#!/bin/bash*).

**Exemplul 1** - Să se scrie un *script shell* care să afișeze numele directorului curent.

```
#!/bin/bash
echo "Directorul curent este: $PWD"
```

În cadrul unui *script shell* variabilele se pot defini/modifica la fel ca și în linia de comandă (tabelul 5.2).

Tabelul 5.2. Variabile interne - atribuirea valorilor

Tip atribuire	Explicații
<b>var1="exemplu text"</b>	atribuie variabilei <i>var1</i> valoarea <i>exemplu text</i>
<b>var1=1116</b>	atribuie variabilei <i>var1</i> valoarea întreagă 1116
<b>var1=\$PWD</b>	atribuie variabilei <i>var1</i> valoarea altei variabile (\$PWD)
<b>read var1</b>	citește de la tastatură valoarea variabilei <i>var1</i>

**Exemplul 2** - Să se scrie un *script shell* care să citească de la tastatură valoarea variabilei *var1*, să atribuie această valoare variabilei *var2* și să afișeze valoarea variabilei *var2*.

```
#!/bin/bash
echo -n "Introduceți valoarea variabilei var1:"
read var1
var2=$var1
echo "Valoarea variabilei var2 este: $var2"
```

Pe lângă variabilele definite de utilizator, *scripturile shell* mai conțin și un set de variabile predefinite. Valorile acestor variabile pot fi utilizate în interiorul *scripturilor* ca și cele definite de utilizator, cu diferența că, pentru o parte din ele, atribuirea se face automat de către *script* în momentul lansării în execuție (tabelul 5.3).

Tabelul 5.3. Variabile predefinite în scripturile shell

Variabilă	Semnificație
<b>\$#</b>	numărul de argumente cu care se lansează în execuție <i>scriptul</i>
<b>\$*</b>	valoarea concatenată a argumentelor
<b>\$@</b>	valorile tuturor argumentelor sub formă de listă
<b>\$0</b>	numele <i>scriptului</i>
<b>\$N</b>	valoarea argumentului <i>N</i> , unde <i>N</i> reprezintă numărul de ordine al argumentului (ex. <i>./script 12</i> - \$1 = 12)
<b>\$\$</b>	ID-ul procesului generat de <i>script</i>
<b>\$( comandă )</b>	valoarea returnată de <i>comandă</i> (ex. <i>\$( ls )</i> - valoarea returnată de comanda <i>ls</i> )
Continuare în pagina următoare	

**Tabela 5.3 – continuare**

Variabilă	Semnificație
<code>\$(( a+b ))</code>	valoarea rezultată în urma unei operații matematice (ex. <code>var1=\$(( a+b ))</code> - unde <code>a</code> și <code>b</code> sunt variabile)
<code>\${variabilă}text</code>	adaugă un text la valoarea unei variabile

**Exemplul 3** - Să se scrie un *script shell* care să afișeze numele *scriptului*, numărul de argumente și valoarea primelor două argumente.

```
#!/bin/bash
echo "Numele scriptului este: $0"
echo "Numarul de argumente este: $# "
echo "Valoarea argumentului 1 este: $1"
echo "Valoarea argumentului 2 este: $2"
```

Într-un script shell operațiile matematice se pot realiza în trei moduri:

- `let "variabilă=expresie"`
- `variabilă='expr expresie'`
- `variabilă=$(( expresie ))`

**Exemplul 4** - Să se scrie un *script shell* care să calculeze suma a două numere citite de la tastatură.

```
#!/bin/bash
echo -n "Introduceti valoarea lui a: "
read a
echo -n "Introduceti valoarea lui b: "
read b
let "c = a + b"
echo "Suma numerelor este: $c"
```

**Exemplul 5** - Să se scrie un *script shell* care să calculeze suma și produsul a două numere introduse ca argumente la numele *scriptului*.



```
#!/bin/bash
a=$1
b=$2
c=`expr $a + $b`
d=$((a * b))
echo "Suma numerelor este: $c"
echo "Produsul numerelor este: $d"
```

#### 5.4. Funcții definite de utilizator

Utilizarea funcțiilor poate simplifica în mare măsură un program. Ca și în limbajele de programare, în *scripturile shell* utilizatorul poate defini propriile funcții. Definirea unor funcții proprii este utilă în cazul în care același set de comenzi trebuie folosit de mai multe ori într-un program.

Sintaxa unei funcții în *scripturile shell* este următoarea:

```
function nume_functie {
    comanda1
    comanda2
    ...
    comandaN
}
```

Aplearea funcțiilor definite de utilizator se realizează prin "*nume\_functie argumente*". În cadrul funcțiilor, variabilele predefinite (tabelul 5.3) preiau valorile argumentelor cu care se apelează funcțiile. De exemplu, dacă se definește funcția "*suma*" și se apelează în *script* cu "*suma 12 14*", variabila "\$1", în corpul funcției, va avea valoarea 12.

**Exemplul 6** - Să se scrie un *script shell* care să calculeze suma a două numere cu ajutorul unei funcții definite în interiorul *scriptului*.

```
#!/bin/bash
function suma {
    a=$1
    b=$2
    c=$(( $a + $b ))
    echo "Suma este: $c"
}
echo -n "a= "
read a
echo -n "b= "
read b
suma $a $b
```

### 5.5. Exerciții propuse

- 1) Să se scrie un *script shell* care să calculeze suma și produsul a 4 numere întregi introduse de la tastatură.
- 2) Să se scrie un *script shell* care să calculeze suma și produsul a 3 numere întregi introduse ca argumente la numele *scriptului*, utilizând funcții definite de utilizator.
- 3) Să se găsească și să se corecteze erorile din următorul *script shell*:

```
#!/bin/bash
function produs {
    nr1=$1
    nr2=$2
    c=$(( $nr1 * $nr2 ))
    echo "Produsul este: c"
}
echo -n "a= "
read $a
echo -n "b= "
read $b
produs a b
```

## LUCRAREA NR. 6

### Fișiere de comenzi - instrucțiuni de control

Scopul acestei lucrări este înțelegerea modului de funcționare instrucțiunilor de control decizionale (*if*, *else*, *case*) și a celor de tip *buclă* (*for*, *while*, *until*) în cadrul fișierelor de comenzi.

#### 6.1. Instrucțiuni decizionale

Scripturile shell funcționează, într-o anumită măsură, ca și limbajele de programare. Acestea permit folosirea instrucțiunilor decizionale (*if*, *else*, *case*) pentru compararea valorilor anumitor variabile, indiferent de tipul de date conținut de acestea.

Spre deosebire de limbajele de programare, în scripturile shell, operatorii folosiți pentru compararea datelor sunt diferiți, în funcție de tipul de date.

În tabelul 6.1 sunt prezentați operatorii care pot fi utilizați pentru compararea numerelor întregi.

*Tabelul 6.1. Operatori de comparare pentru numere întregi*

Operator	Semnificație
<b>\$var1 -eq \$var2</b>	returnează TRUE dacă var1 este egală cu var2
<b>\$var1 -ge \$var2</b>	returnează TRUE dacă var1 este mai mare sau egală ca var2
<b>\$var1 -gt \$var2</b>	returnează TRUE dacă var1 este mai mare ca var2
<b>\$var1 -le \$var2</b>	returnează TRUE dacă var1 este mai mică sau egală ca var2
<b>\$var1 -lt \$var2</b>	returnează TRUE dacă var1 este mai mică decât var2
<b>\$var1 -ne \$var2</b>	returnează TRUE dacă var1 nu este egală cu var2

În tabelul 6.2 sunt prezentați operatorii care pot fi utilizați pentru compararea șirurilor de caractere.

*Tabelul 6.2. Operatori de comparare pentru șiruri de caractere*

Operator	Semnificație
<b>\$var1 = \$var2</b>	returnează TRUE dacă șirurile var1 și var2 sunt identice
Continuare în pagina următoare	

**Tabela 6.2** – continuare

Operator	Semnificație
<b>\$var1 != \$var2</b>	returnează TRUE dacă șirurile var1 și var2 nu sunt identice
<b>-n \$var1</b>	returnează TRUE dacă lungimea șirului var1 nu este zero
<b>-z \$var1</b>	returnează TRUE dacă lungimea șirului var1 este zero

În tabelul 6.3 sunt prezentați operatorii care pot fi utilizați pentru compararea fișierelor și directoarelor.

**Tabelul 6.3.** Operatori de comparare pentru fișiere și directoare

Operator	Semnificație
<b>-f \$var1</b>	returnează TRUE dacă var1 este fișier
<b>-d \$var1</b>	returnează TRUE dacă var1 este director
<b>-r \$var1</b>	returnează TRUE dacă var1 este un fișier sau director cu drept de citire
<b>-w \$var1</b>	returnează TRUE dacă var1 este un fișier sau director cu drept de scriere
<b>-x \$var1</b>	returnează TRUE dacă var1 este un fișier sau director cu drept de execuție

Blocurile logice *if* pot fi folosite în trei moduri:

1) *if-then*

Sintaxă:

```
if [ conditie ]
then
    comanda1
    comanda2
    ...
    comandaN
fi
```

2) *if-then-else*

Sintaxă:

```
if [ conditie ]
then
    comanda1
    comanda2
    ...
    comandaN
else
    comanda1
    comanda2
    ...
    comandaN
fi
```

### 3) *if-then-elif-then-else*

Sintaxă:

```
if [ conditie ]
then
    comanda1
    comanda2
    ...
    comandaN
elif [ conditie ]
then
    comanda1
    comanda2
    ...
    comandaN
else
    comanda1
    comanda2
    ...
    comandaN
fi
```

**Exemplul 1** - Să se scrie un script shell care să citească o valoare întreagă de la tastatură și să verifice dacă valoarea variabilei este egală, mai mică sau mai mare decât zero.

```
#!/bin/bash
echo -n "Introduceți valoarea variabilei var1:"
read var1
if [ $var1 -eq 0 ]
then
    echo "var1 este egala cu 0"
elif [ $var1 -lt 0 ]
then
    echo "var1 este mai mica decat 0"
else
    echo "var1 este mai mare ca 0"
fi
```

Instrucțiunea **case** compară valoarea unei variabile cu valorile existente într-o listă predefinită și execută setul de comenzi corespunzător valorii găsite (dacă există).

Sintaxa instrucțiunii **case** este următoarea:

```
case $nume_variabila in
    valoare1)
        comanda1;
        comanda2;
        ...
        comandaN;;
    valoare2)
        comanda1;
        comanda2;
        ...
        comandaN;;
    *)
        comanda1;
```

```

        comanda2;
        ...
        comandaN;;
    esac

```

**Exemplul 2** - Să se scrie un script shell care să identifice valoarea unei variabile într-o listă de valori.

```

#!/bin/bash
echo -n "Introduceti valoarea variabilei var1:"
read var1
case $var1 in
    1)
        echo "var1=1";;
    2)
        echo "var1=2";;
    3)
        echo "var1=3";;
    *)
        echo "var1 nu este in lista de valori";;
esac

```

Caracterul "\*" reprezintă valoarea *default*, adică setul de comenzi care trebuie executat dacă variabila are orice valoare care nu există în lista de valori. Pentru orice set de comenzi se pot scrie mai multe valori, delimitate prin caracterul "|":

```

case $nume_variabila in
    val1|val2|val3)
        comenzi;
        ...
esac

```

## 6.2. Instrucțiuni de tip buclă

Instrucțiunile de tip buclă pot fi folosite pentru parcurgerea unor liste de la valori (instrucțiunea *for*) sau pentru executarea condiționată a unor seturi de

## Fișiere de comenzi - instrucțiuni de control

---

comenzi (instrucțiunile *while* și *until*).

Instrucțiunea **for** parcurge o listă de valori atribuind, pe rând, fiecare valoare unei variabile.

Sintaxă:

```
for var1 in sir_de_valori
do
    comanda1
    comanda2
    ...
    comandaN
done
```

Parametrul *sir\_de\_valori* poate fi un șir de numere întregi, un șir de caractere și/sau cuvinte sau o variabilă care conține un șir de valori. Condiția ca șirul să fie valid este să conțină valori delimitate prin spațiu.

**Exemplul 3** - Să se scrie un script shell care să afișeze numărul de fișiere și de directoare din directorul curent.

```
#!/bin/bash
lista=$( ls )
fisiere=0
directoare=0
for nume in $lista
do
    if [ -f $nume ]
    then
        let "fisiere=fisiere+1"
    fi
    if [ -d $nume ]
    then
        let "directoare=directoare+1"
    fi
done
echo "Numarul de fisiere: $fisiere"
echo "Numarul de directoare: $directoare"
```



Instrucțiunea ***while*** execută un set de comenzi atâta timp cât condiția de execuție este adevărată.

Sintaxă:

```
while [ conditie ]
do
    comanda1
    comanda2
    ...
    comandaN
done
```

**Exemplul 4** - Să se scrie un script shell care să creeze  $N$  fișiere având numele de forma *fișier.1*, *fișier.2*, ..., *fișier.N*.

```
#!/bin/bash
echo -n "Introduceti numele fisierului"
read nume
echo -m "Introduceti numarul de fisiere"
read numar
i=0
while [ $i -le $numar ]
do
    fisier=${nume}.$i
    touch $fisier
    let "i=i+1"
done
echo "Fisierele au fost create"
ls ${nume}.*
```

Instrucțiunea ***until*** execută un set de comenzi până când condiția de execuție devine adevărată.

Sintaxă:

```
until [ conditie ]
do
```

## Fișiere de comenzi - instrucțiuni de control

---

```
comanda1
comanda2
...
comandaN
done
```

**Exemplul 5** - Să se scrie un script shell care să numere fișierele și subdirectoarele din diferite directoare citite de la tastatură până când numele directorului introdus este *"exit"*. Să se afișeze numărul de directoare parcurse, numărul de fișiere găsite și numărul de subdirectoare găsite.

```
#!/bin/bash
echo -n "Introduceti numele unui director: "
read var
lista=$(ls $var)
fisiere=0
directoare=0
i=1
until [ $var = "exit" ]
do
    for nume in $lista
    do
        locatie=${var}/${nume}
        echo "Nume complet: $locatie"
        if [ -f $locatie ]
        then
            let "fisiere=fisiere+1"
        fi
        if [ -d $locatie ]
        then
            let "directoare=directoare+1"
        fi
    done
    echo -n "Introduceti alt director [$i]: "
    read var
```

```

if [ $var != "exit" ]
then
    let "i=i+1"
    lista=$(ls $var)
fi
done
echo "Numarul de directoare parcurse: $i"
echo "Numarul de fisiere gasite: $fisiere"
echo "Numarul de directoare gasite: $directoare"

```

### 6.3. Exerciții propuse

- 1) Să se scrie un script shell care să calculeze suma a  $N$  numere întregi introduse ca argumente la numele scriptului.
- 2) Să se scrie un script shell care să afișeze numărul minim și maxim dintr-un șir de numere întregi introduse ca argumente la numele scriptului.
- 3) Să se scrie un script shell care să afișeze timpul în care unei variabile  $i$  se atribuie automat valori de la 0 la 200000 (vezi comanda *date +%s*).
- 4) Să se scrie un script shell care să calculeze suma, diferența sau produsul a două numere în funcție de operatorul introdus de la tastatura (+, - sau \*).
- 5) Să se găsească și să se corecteze erorile din următorul script shell:

```

#!/bin/bash
function produs (
    prod=1
    lista=$@
    for $val in lista
    do
        if [ val -ne 0 ]
        then
            let prod=prod*val
        fi
    done
}

```

### *Fişiere de comenzi - instrucţiuni de control*

---

```
done
    echo "Produsul numerelor este: prod"
)
arg=$@
produs $arg
```

## LUCRAREA NR. 8

### Configurarea rețelelor în *Linux*

Scopul acestei lucrări este înțelegerea modului de configurare a rețelelor în sistemul de operare *Linux* precum și înțelegerea funcționării protocoalelor de comunicație folosite pentru transmiterea datelor.

#### 8.1. Protocolul TCP

Protocolul TCP (*Transmission Control Protocol*) este unul din protocoalele care stau la baza comunicațiilor într-o rețea sau pe internet. Cu ajutorul acestui protocol, aplicațiile care rulează pe calculatoare conectate la o rețea, pot crea conexiuni la alte aplicații rulate în rețea și pot efectua transferuri de date. Acest protocol garantează transmiterea datelor de la o aplicație la alta și poate distinge datele de la mai multe aplicații concurente care rulează pe același calculator.

Conexiunile care se realizează prin intermediul acestui protocol sunt de tip *client-server*. Orice conexiune de acest tip are trei etape:

- stabilirea conexiunii;
- transferul de date;
- închiderea conexiunii.

Fiecare din etapele de mai sus poate avea mai multe stări (tabelul 8.1). Cu ajutorul acestor stări, protocolul TCP are un control mult mai bun asupra corectitudinii transferului de date. Astfel, protocolul evită erorile de transfer datorate conexiunilor slabe.

**Tabelul 8.1.** Starea unei conexiuni TCP

Stare	Descriere
LISTEN	reprezintă așteptarea unei cereri de conexiune
SYN-SENT	reprezintă așteptarea unei cereri de conexiune potrivită după ce a trimis o cerere de conexiune
SYN-RECEIVED	reprezintă așteptarea unei cereri de confirmarea a conexiunii după ce a trimis și a primit o cerere de conexiune
Continuare în pagina următoare	

Tabela 8.1 – continuare

Stare	Descriere
ESTABLISHED	reprezintă o conexiune deschisă, adică starea normală pentru efectuarea unui transfer de date
FIN-WAIT-1	reprezintă așteptarea unei cereri de terminare a conexiunii sau o confirmare a unei cereri de terminare transmise anterior
FIN-WAIT-2	reprezintă așteptarea unei cereri de terminare a conexiunii
CLOSE-WAIT	reprezintă așteptarea unei cereri de terminare a conexiunii din partea utilizatorului local
CLOSING	reprezintă așteptarea unei confirmări a cererii de terminare a conexiunii din partea protocolului
LAST-ACK	reprezintă așteptarea unei confirmări a cererii de terminare a conexiunii transmise anterior de către protocol
TIME-WAIT	reprezintă un interval de timp în care închiderea conexiunii este pusă în așteptare pentru a asigura confirmarea cererii de terminare a conexiunii
CLOSED	reprezintă încheierea conexiunii

Conexiunile sau stările unei conexiuni pot fi vizualizate cu ajutorul comenzii *netstat*.

**netstat** *opțiuni* - afișează informații despre conexiunile curente.

*Opțiuni:*

- a - afișează toate serviciile și porturile care pot realiza conexiuni;
- s - afișează statistici pentru fiecare protocol;
- p - afișează PID-ul și numele programului care a deschis conexiunea;
- c - afișează continuu informații despre conexiuni.

## 8.2. Protocolul UDP

Conexiunile *UDP* (*User Datagram Protocol*) sunt conexiuni fără stare deoarece ele nu includ etapele de stabilire a conexiunii sau închidere a acesteia. Primirea a două pachete de date *UDP* într-o ordine specifică nu ne spune nimic despre ordinea în care au fost trimise. Cu toate acestea, este posibilă setarea anumitor stări ale conexiunii în interiorul *kernel*-ului.

Aplicațiile care utilizează acest tip de protocol acceptă erori, pierderi sau duplicări ale pachetelor de informații trimise/primate. Dintre aplicațiile sau serviciile care folosesc acest tip de protocol se pot aminti: jocuri multiplayer real-time, DNS (*Domain Name System*), DHCP (*Dynamic Host Configuration Protocol*), RIP (*Routing Information Potocol*), etc.

### 8.3. Protocolul ICMP

Pachetele ICMP (*Internet Control Message Protocol*) sunt considerate a fi pachete fără stare, deoarece sunt folosite pentru controlul conexiunilor, nu pentru a stabili conexiuni. Cu toate acestea există patru tipuri de pachete ICMP care generează pachete de răspuns de două tipuri: *NEW* și *ESTABLISHED*. Tipurile ICMP sunt:

- *echo request* - cerere ecou;
- *echo reply* - răspuns de tip ecou;
- *timestamp request* - cererea amprente de timp;
- *timestamp reply* - răspuns la cererea amprente de timp;
- *information request* - cererea de informație;
- *information reply* - răspuns la cererea de informație;
- *address mask request* - cererea măștii adresei;
- *address mask reply* - răspuns la cererea măștii adresei.

Cererea de amprentă de timp și de informație fac parte din versiunile mai vechi ale protocolului ICMP și nu se mai utilizează. Mesajele de tip ecou sunt utilizate pentru verificarea conexiunii (comanda *ping*), iar cererile de mască a adresei sunt utilizate de către dispozitivele de conectarea la rețea pentru obținerea măștii rețelei de la *router*.

**ping opțiuni adresă** - trimite pachete *echo request* către *adresă* și afișează timpul de răspuns al *adresei* la aceste pachete.

*Opțiuni:*

-**i interval** - transmite pachete *echo request* la un *interval* de timp prestabilit (în secunde);

-**I interfață** - permite alegerea *interfeței* de rețea care va trimite pachetele de tip *echo request* către *adresă*;

-**c număr\_pachete** - oprește comanda după transmiterea unui număr prestabilit de pachete;

*adresă* - reprezintă IP-ul sau adresa calculatorului spre care se trimit pachetele.

**traceroute destinație** - afișează toate adresele prin care trece un pachet până la adresa *destinație*. Comanda *traceroute* afișează, de asemenea, timpul de răspuns al fiecărei adrese prin care trece pachetul.

### 8.4. Protocolul IP

IP (Internet Protocol) este un protocol care asigură un serviciu de transmitere a datelor, fără conexiune permanentă. Acesta identifică fiecare interfață logică a echipamentelor conectate printr-un număr numit "adresă IP". Standardul folosit în majoritatea cazurilor este IPv4. În IPv4, standardul curent pentru comunicarea în Internet, adresa IP este reprezentată pe 32 de biți (de ex. 192.168.0.1). Alocarea adreselor IP nu este arbitrară; ea se face de către organizații însărcinate cu distribuirea de spații de adrese. De exemplu, RIPE este responsabilă cu gestiunea spațiului de adrese atribuit Europei.

Adresele IP se împart în cinci clase, de la A până la E (tabelul 8.2). Împărțirea se face în funcție de configurația binară a primului octet din adresa IP.

*Tabelul 8.2. Clasele de IP-uri*

Clasa	IP-uri
A	0.0.0.0 - 127.255.255.255
B	127.0.0.0 - 191.255.255.255
C	192.0.0.0 - 223.255.255.255
D	224.0.0.0 - 239.255.255.255
E	240.0.0.0 - 255.255.255.255

În internet se folosesc numai adresele IP de clasă A, B și C, clasele D și E fiind rezervate. De asemenea, există trei intervale rezervate pentru adresare privată



(adrese interne) pentru clasele A, B și C (tabelul 8.3).

**Tabelul 8.3.** IP-uri pentru adresare privată

Clasa	IP-uri
A	10.0.0.0 - 10.255.255.255
B	172.16.0.0 - 172.31.255.255
C	192.168.0.0 - 192.168.255.255

Adresele de clasă A din intervalul 127.0.0.0 - 127.255.255.255 reprezintă o subclasă specială denumită **loopback**. Această subclasă este folosită pentru diagnosticarea nodului local.

În *Linux*, atribuirea unei adrese IP se poate face pentru interfețele fizice de rețea sau pentru interfețele virtuale. Interfețele fizice sunt denumite *ethN*, unde *N* este un număr unic atribuit fiecărei interfețe (ex. *eth0*, *eth1*, etc). Fac excepție de la această notație interfețele de rețea wireless, bluetooth sau USB. Pentru acestea se pot întâlni notații ca: *ra0* - placă de rețea wireless cu chipset Ralink, *wlan0* - placă de rețea wireless cu chipset Intel, *bnep0* - interfață bluetooth, *usb0* - interfață USB, etc.

Interfețele virtuale sunt denumite *ethN:M* unde *N* este un număr unic atribuit interfeței fizice iar *M* un număr unic atribuit interfeței virtuale (ex. *eth0:1*, *eth0:2*, etc). Aceste interfețe sunt utile în cazul în care se dorește atribuirea mai multor adrese IP aceleiași interfețe fizice.

În *Linux*, atribuirea unei adrese IP atât unei interfețe fizice cât și unei interfețe virtuale se face cu ajutorul comenzii *ifconfig*.

**ifconfig** interfață IP opțiuni - atribuie o adresă IP unei interfețe de rețea

Opțiuni:

**netmask** mască - setează masca rețelei pentru interfață

**up** - activează interfața de rețea

**down** - dezactivează interfața de rețea

**Exemplul 1** - Să se seteze adresa IP 192.168.0.2 și masca rețelei 255.255.255.0 pentru interfața de rețea *eth0*.

```
ifconfig eth0 192.168.0.2 netmask 255.255.255.0 up
```

Pentru accesarea calculatoarelor din afara rețelei locale va trebui stabilită *tabela de rutare*. Tabela de rutare conține adrese ale calculatoarelor (serverelor) cu ajutorul cărora adresa IP a calculatorului este făcută publică în altă rețea sau pe internet. Aceste servere se mai numesc *gateway* și au rolul de a redirecționa datele/pachetele în interiorul sau în afara rețelei locale. Tabela de rutare se stabilește cu ajutorul comenzii *route*

*Adăugarea unei rute:*

```
route add [-net|-host] adresă [netmask adresă_mască] [default] [gw adresă_GW]  
[reject] dev interfață
```

*Ștergerea unei rute:*

```
route del [-net|-host] adresă [netmask adresă_mască] [gw adresă_GW] dev interfață
```

*Opțiuni:*

**adresă** - gazda sau rețeaua destinație  
**-net** | **-host** - setează ruta pentru o rețea sau pentru un calculator gazdă  
**netmask** *adresă\_mască* - masca de rețelei destinație  
**gw** *adresă\_GW* - adresa *gateway*  
**reject** - instalează o rută blocată (opțional)  
**default** - instalează o rută primară  
**dev** *interfață* - instalează ruta pentru o interfață

**Exemplul 2** - Să se seteze tabela de rutare pentru interfața *eth0* având ca și *gateway* primar adresa 192.168.0.1.

```
route add default gw 192.168.0.1 dev eth0
```

### 8.5. Comunicarea între utilizatori

Sistemul de operare *Linux* este un sistem multiuser și are încorporate facilitățile de interconectare în rețea, astfel oferind o serie de utilitare pentru comunicarea între utilizatori. Comunicarea poate fi realizată atât între utilizatori ai aceluiași sistem cât și între utilizatori din sisteme diferite.

**mesg opțiuni** - activează sau dezactivează primirea mesajelor de la alți utilizatori

*Opțiuni:*

**y** - activează primirea mesajelor

**n** - dezactivează primirea mesajelor

**write nume\_utilizator** - permite dialogul în timp real între doi utilizatori ai aceluiași sistem

După ce fiecare din cei doi utilizatori au lansat *write* mesajele se vor afișa pe ambele ecrane. Lansarea în execuție a utilitarul *write* este permisă doar după activarea primirii mesajelor cu ajutorul comenzii *mesg*. Încheierea conversației se face cu combinația de taste Ctrl+D.

**talk utilizator** - permite dialogul în timp real între doi utilizatori din același sistem sau din sisteme diferite

*Sintaxă:*

*talk utilizator* - pornește conversația cu un *utilizator* din același sistem (ex. *talk student*)

*talk utilizator@gazdă* - pornește conversația cu un *utilizator* dintr-un alt sistem (ex. *talk student@aut.utcluj.ro*)

Încheierea conversației se face cu combinația de taste Ctrl+C.

## 8.6. Exerciții propuse

- 1) Să se scrie un script shell care să configureze interfața de rețea *eth0* cu valori pentru IP și *netmask* citite de la tastatură.
- 2) Să se seteze tabela de rutare pentru interfața *eth0* știind că adresa *gateway* este 172.27.208.1.
- 3) Să se creeze două conturi utilizator cu drepturi limitate și să se pornească o conversație între acești utilizatori folosind două console diferite și comanda *write*.

## LUCRAREA NR. 9

# Calculul adreselor IP în configurarea rețelelor și subrețelelor de calculatoare

Scopul acestei lucrări este înțelegerea metodelor de calcul a IP-urilor și a modului de configurare a subrețelelor în funcție de clasele de IP-uri.

### 9.1. Adrese IP

O adresă IP este un număr format din 4 grupuri a câte 8 biți atribuit unei interfețe de rețea. Adresele IP se împart în 5 clase (A,B,C,D și E) din care doar 3 se pot folosi pe internet (A, B și C).

Orice adresă IP are două componente:

- componenta rețea (*network*)
- componenta gazdă (*host*)

Aceste componente depind de clasa din care face parte IP-ul.

Fiecărei clase de IP-uri îi corespunde o mască (*subnet mask*). Maska este un filtru care determină cărei subrețele (*subnet*) îi aparține o adresă IP și este formată din adresa de rețea (*network address*) plus biții de identificare ai subrețelei. Prin convenție, biții de rețea ai măștii au valoarea 1. Valorile măștilor pentru fiecare clasă de IP-uri sunt prezentate în tabelul 9.1.

**Tabelul 9.1.** Măștile claselor de IP-uri

Clasă IP	Mască	Mască (binar)
A (0.0.0.0-127.0.0.1)	255.0.0.0	11111111.00000000.00000000.00000000
B (128.0.0.0-191.255.255.255)	255.255.0.0	11111111.11111111.00000000.00000000
C (192.0.0.0-223.255.255.255)	255.255.255.0	11111111.11111111.11111111.00000000

Conform acestei convenții, fiecare clasă de IP-uri conține un număr predefinit de rețele și gazde (tabelul 9.2).

**Tabelul 9.2.** Numărul de rețele și gazde per clasă

Clasă IP	Număr rețele	Număr gazde
A (0.0.0.0-127.0.0.1)	126	16.777.216
B (128.0.0.0-191.255.255.255)	16.384	65534
C (192.0.0.0-223.255.255.255)	2.097.152	254

Masca subrețelei (subnet mask) se poate modifica astfel încât să se obțină un număr dorit de subrețele. Astfel, masca subrețelei conține trei componente: componenta de rețea, componenta de subrețea și componenta de gazdă.

Exemplu: mască IP clasa A cu subrețele:

$$\overbrace{11111111}^{\text{network}}.\overbrace{11111111}^{\text{subnet}}.\overbrace{00000000}^{\text{host}}.\overbrace{00000000}^{\text{host}}$$

Numărul de subrețele a unei clase de IP-uri este egal cu  $2^N$ , unde N reprezintă numărul de biți egali cu "1" din componenta de subrețea a măștii corespunzătoare clasei. Astfel, pentru un IP din clasa B având masca 255.255.0.0 (11111111.11111111.00000000.00000000), numărul de subrețele va fi  $2^0 = 1$ , deoarece componenta de subrețea nu există (nici unul din primii biți de gazdă nu sunt egali cu 1).

$$\overbrace{11111111}^{\text{retea(network)}}.\overbrace{11111111}^{\text{gazda(host)}}.\overbrace{00000000}^{\text{gazda(host)}}.\overbrace{00000000}^{\text{gazda(host)}}$$

Numărul de IP-uri alocabile pe fiecare subrețea este egal cu  $2^M - 2$ , unde M reprezintă numărul de biți egali cu "0" din componenta gazdă (host). În fiecare subrețea, primul și ultimul IP nu sunt alocabile, ele reprezentând adresa rețelei (primul IP) și adresa broadcast (ultimul IP).

Pentru scrierea IP-urilor se mai folosește și următoarea notație: IP / N. (Exemplu: 192.168.100.2/29, unde N reprezintă numărul de biți egali cu "1" din masca subrețelei).

Astfel, IP-ul 192.168.100.2/29 (IP clasa C) va avea masca:

$$11111111.11111111.11111111.11111000$$

sau 255.255.255.248. Deci, acest IP face parte dintr-o rețea cu 32 subrețele (nr. subrețele =  $2^5$ ).

$$\overbrace{11111111}^{\text{network}}.\overbrace{11111111}^{\text{network}}.\overbrace{11111111}^{\text{network}}.\overbrace{11111000}^{\text{subnet host}}$$

Fiecare subrețea din această clasă conține 6 IP-uri alocabile ( $2^3 - 2 = 6$ ) și un număr total de 192 IP-uri alocabile (nr. IP-uri alocabile = nr. subrețele \* nr. IP-uri alocabile per subrețea).

Pentru a stabili din ce subrețea face parte un IP oarecare este necesară cunoașterea măștii subrețelei. În continuare va trebui să se determine adresa rețelei și adresa broadcast, care reprezintă primul și ultimul IP din subrețea. Pentru determinarea adresei rețelei (subrețelei) se va efectua, în binar, operația logică AND între adresa IP și masca subrețelei.

**Exemplu:**

Se consideră următorul IP: 192.168.100.34/29

IP (binar) - 11000000.10101000.01100100.00100010

Masca subrețelei - 255.255.255.248

Masca subrețelei (binar) - 11111111.11111111.11111111.11111000

$$\begin{array}{r} 11000000.10101000.01100100.00100010 \\ 11111111.11111111.11111111.11111000 \\ \hline 11000000.10101000.01100100.00100000 \end{array} \quad \begin{array}{l} AND \\ \\ = 192.168.100.32 \end{array}$$

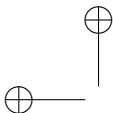
Pentru determinarea adresei broadcast se va efectua, în binar, operația logică XOR (SAU exclusiv) între adresa rețelei (subrețelei) și masca subrețelei inversată.

$$\begin{array}{r} 11000000.10101000.01100100.00100000 \\ 11111111.11111111.11111111.00000111 \\ \hline 11000000.10101000.01100100.00100111 \end{array} \quad \begin{array}{l} XOR \\ \\ = 192.168.100.39 \end{array}$$

**NOTĂ:** În cazul în care componenta "subrețea" a unei măști există (rețeaua este împărțită în subrețele), aceasta va conține întotdeauna numai biți egali cu "1".

## 9.2. Exerciții propuse

- 1) Să se determine numărul de subrețele și numărul de IP-uri alocabile din clasa 172.27.17.0/23.
- 2) Să se găsească adresa subrețelei și adresa broadcast pentru IP-ul 194.68.12.145/27.
- 3) O firmă oarecare este structurată pe 6 departamente, iar fiecare departament are în dotare 25 calculatoare. Firmei i s-a alocat clasa de IP-uri 193.226.18.0/24. Să se împartă clasa de IP-uri în subrețele astfel încât fiecare departament să



### Calculul adreselor IP în configurarea rețelelor și subrețelelor de calculatoare

aibă subrețeaua proprie (25 IP-uri alocate în fiecare subrețea). Să se scrie masca subrețelei și primul IP alocabil al fiecărui departament.

- 4) Să se determine masca subrețelei și adresa subrețelei din care face parte IP-ul 194.124.16.235, știind că numărul total de adrese broadcast este 8.

## LUCRAREA NR. 10

### Compilatoare de C sub Linux

---

Scopul acestei lucrări este familiarizarea cu compilatorul de programe C sub Linux *gcc* precum și a modului de scriere și organizare a aplicațiilor în C sub Linux.

#### 10.1. Compilatorul GCC

Compilatorul GCC sau GNU Compiler Collection este unul din cele mai utilizate și performante compilatoare pentru C/C++ (în special C, pentru C++ folosindu-se des compilatorul G++). Și aceasta aparține colecției de software GNU, adică este un program open source. GCC este un compilator extensibil, deși este dedicat pentru C dar cu extensii și pentru alte limbaje de programare cum ar fi: C++, Pascal, Fortran, Objective-C etc.

GCC este un program open source cu toate acestea executabilele generate de acesta nu trebuie să fie neapărat open source, chiar dacă include librăriile standard C sau C++.

Prima versiune GCC este datată în anul 1986. Pe parcursul anilor au apărut versiuni din ce în ce mai evolute, la început necesitând instalarea separată a compilatorului, mai apoi fiind incluse în distribuțiile Linux.

Sintaxa generală a compilatorului *gcc* este:

***gcc* opțiuni nume\_fișier**

În linux, fișierle standard C au extensia “.c”. Pentru generarea unui fișier executabil dintr-un fișier standard C se va utiliza compilatorul *gcc* astfel:

***gcc -o fișier\_executabil fișier.c***

**Exemplul 1** - Să se compileze următorul program C (ex1.c):



```
#include <stdio.h>

int main()
{
    printf("Hello\n");
    return 0;
}
```

### Rezolvare:

```
root@slax:~# gcc -o ex1 ex1.c
root@slax:~# ./ex1
Hello
```

```
root@slax:~#
```

În tabelul 10.1 sunt prezentate o parte din opțiunile care pot fi utilizate pentru compilarea fișierelor C.

**Tabelul 10.1.** Opțiunile compilatorului gcc

Opțiune	Descriere
-I/ <i>locatie</i>	caută fișierele “.h” în directorul <i>locatie</i> . Implicit, fișierele “.h” sunt căutate în directorul curent și în directoarele care conțin <i>header</i> ele pentru librăriile standard (/usr/include, /usr/src/include).
-L/ <i>locatie</i>	caută librăriile și în directorul <i>locatie</i> pe lângă directoarele standard “/lib”, “/usr/lib”.
-lnume_ <i>librarie</i>	permite utilizarea funcțiilor din librăria libnume_ <i>librarie</i>
-static	include în fișierul executabil toate funcțiile din librăriile utilizate în fișierul “.c” ( <i>avantaj</i> : executabilul poate fi folosit pe alte sisteme fără a fi nevoie de librăriile cu funcții; <i>dezavantaj</i> : dimensiune considerabil mai mare a executabilului).
-c <i>fișier.c</i>	compilează un fișier C și generează un fișier obiect “fișier.o”.
-D <i>macro</i>	definește macro-ul cu numele <i>macro</i> (opțiune pentru preprocesor - identică cu “#define NUME valoare”)
Continuare în pagina următoare	

**Tabela 10.1 – continuare**

Opțiune	Descriere
-D <i>macro=valoare</i>	definește macro-ul cu numele <i>macro</i> și îi atribuie valoarea <i>valoare</i> .
-Wall	afișează toate mesajele de avertizare.

Anumite funcții utilizate în programele C necesită definirea unor fișiere *header* suplimentare iar în anumite cazuri compilarea fișierului necesită utilizarea unor librării suplimentare. De exemplu, pentru utilizarea funcțiilor matematice standard se va defini fișierul *header* "math.h" iar compilarea va necesita utilizarea opțiunii "-lm" care permite apelarea funcțiilor matematice din librăria "libm" asociată fișierului *header* "math.h".

**Exemplul 2** - Să se scrie și să se compileze un program C (ex2.c) care calculează funcțiile *sinus* și *cosinus* ale unui unghi oarecare introdus de la tastatură:

```
#include <stdio.h>
#include <math.h>
void main(void)
{
    double unghi, sin_unghi, cos_unghi;
    printf("Unghi: ");
    scanf("%lf", &unghi);
    sin_unghi=sin((unghi*3.14)/180);
    cos_unghi=cos((unghi*3.14)/180);
    printf("sin(%f)= %f\n",unghi,sin_unghi);
    printf("cos(%f)= %f\n",unghi,cos_unghi);
}
```

**Rezolvare:**

```
root@slax:~# gcc -o ex2 ex2.c -lm
root@slax:~# ./ex2
Unghi: 30
sin(30.000000)= 0.499770
cos(30.000000)= 0.866158
root@slax:~#
```

## 10.2. Compilarea aplicațiilor cu surse multiple

În anumite cazuri, aplicațiile C complexe necesită o organizare mai riguroasă. De obicei, aplicațiile complexe conțin mai multe fișiere sursă ceea ce permite utilizatorului o mai bună înțelegere a modului de organizare a aplicației.

Orice aplicație *multi-sursă* conține un fișier sursă principal (de obicei *main.c*) și unul sau mai multe fișiere sursă secundare care conțin corpul funcțiilor definite de utilizator.

Pe lângă fișierele sursă, aplicațiile multi-sursă mai pot conține și fișiere de tip *header* (.h) care conțin definițiile funcțiilor utilizator.

Compilarea aplicațiilor de acest tip se face în doi pași:

- se compilează toate fișierele sursă utilizând opțiunea "-c" cu ajutorul căreia se generează câte un fișier obiect (cu extensia ".o") pentru fiecare fișier sursă.

```
gcc -c fisier1.c fisier2.c ... fisierN.c main.c
```

- se generează fișierul executabil utilizând opțiunea "-o" și fișierele obiect generate anterior.

```
gcc -o executabil fisier1.o ... fisierN.o main.o
```

**Exemplul 3** - Să se scrie și să se compileze o aplicație multi-sursă care calculează suma și produsul a două numere întregi citite de la tastatură. Funcțiile *suma* și *produs* se vor scrie într-un fișier separat denumit *functii.c*.

**main.c:**

```
#include <stdio.h>
int main()
{
    int a,b;
    printf("a= ");
    scanf("%d",&a);
    printf("b= ");
    scanf("%d",&b);
```

```

        printf("Suma= %d\n", suma(a,b));
        printf("Produs= %d\n", prod(a,b));
        return 0;
    }

```

#### **functii.c:**

```

int suma(int i, int j)
{
    int s;
    s = i+j;
    return s;
}

int prod(int i, int j)
{
    int p;
    p = i*j;
    return p;
}

```

#### **Rezolvare:**

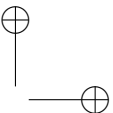
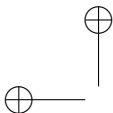
```

root@slax:~# gcc -c functii.c main.c
root@slax:~# gcc -o ex3 functii.o main.o
root@slax:~# ./ex3
a= 12
b= 13
Suma= 25
Produs= 156
root@slax:~#

```

### **10.3. Exerciții propuse**

- 1) Să se scrie și să se compileze un program C care calculează media geometrică a  $N$  numere întregi introduse de la tastatură.



## Compilatoare de C sub Linux

---

- 2) Să se scrie și să se compileze un program C care calculează funcțiile *tangentă* și *cotangentă* ale unui unghi oarecare.
- 3) Să se scrie și să se compileze un program C care calculează media aritmetică și media geometrică a  $N$  numere întregi introduse de la tastatură. Funcțiile de calcul ale celor două medii se vor scrie într-un fișier separat, denumit *medii.c*.

## LUCRAREA NR. 11

### Makefile-uri

---

Scopul acestei lucrări este înțelegerea modului de funcționare al utilitarului *make* și cunoașterea modului organizare a fișierelor *Makefile*.

#### 11.1. Structura fișierelor *Makefile*

Aplicațiile complexe în C sub Linux conțin, în majoritatea cazurilor, un număr mare de fișiere sursă. De asemenea, în multe cazuri, compilarea fișierelor sursă necesită mai multe comenzi, fiecare având un număr mare de opțiuni. Dacă aplicația se află în stadiul de dezvoltare, orice testare va necesita compilarea tuturor surselor. Astfel, în anumite cazuri, procesul de compilare poate dura mai mult decât cel de programare.

Pentru a simplifica procesul de compilare, în special în cazul aplicațiilor cu surse multiple, se recomandă utilizarea fișierelor *Makefile* respectiv a utilitarului *make*.

Fișierele *Makefile* sunt fișiere text asociate utilitarului *make* care conțin macro-uri și comenzi pentru compilarea aplicațiilor (C, C++, etc).

Macro-urile dintr-un fișier *Makefile* reprezintă variabile pe care utilizatorul le poate folosi pentru compilarea aplicațiilor. Acestea li se pot atribui valori ca și în cazul scripturilor Shell, adică "variabilă=valoare". Macro-urile predefinite sunt:

- **CC** - numele compilatorului
- **OBJS** - numele fișierelor obiect (.o) care se vor genera
- **CFLAGS** - opțiunile utilizate pentru compilarea fișierelor sursă (ex: *-c -Wall*)
- **LFLAGS** - opțiunile utilizate pentru generarea executabilului (ex: *-lm -Wall*)

Pe lângă macro-uri, în fișierele *Makefile* se definesc și fișierele *target*, astfel:

```
target: dependinte  
    <TAB> comanda
```

## Makefile-uri

---

Target reprezintă numele fișierului care trebuie generat iar “dependinte” reprezintă numele fișierului sau fișierelor de care depinde fișierul *target*.

Fișierele Makefile se vor scrie după următoarele reguli: macro-urile se vor scrie întotdeauna cu litere mari; comenzile pentru generarea fișierelor target vor fi precedate de *TAB*; comenzile vor fi scrise pe baza macro-urilor, după următoarea sintaxă:

```
<TAB> $(MACROx) $(MACROy) optiuni $(MACROz) ... $(MACROxx)
```

exemplu:

```
$(CC) -o ex1 $(LFLAGS) $(OBJS)
```

Astfel, structura unui fișier Makefile este următoarea:

```
MACRO1=valoare
```

```
...
```

```
MACROn=valoare
```

```
target1: dependinte
```

```
<TAB> comenzi
```

```
...
```

```
targetN: dependinte
```

```
<TAB> comenzi
```

### 11.2. Compilarea aplicațiilor C cu utilitarul *make*

Compilarea aplicațiilor după scrierea fișierului Makefile se va face cu comanda *make target*.

**Notă:** *target* poate reprezenta și un cuvânt cheie folosit pentru operații suplimentare asupra aplicației. De exemplu, pentru ștergerea fișierelor obiect după generarea fișierului executabil se poate folosi:

```
clean:
    rm -rf *.o
```

Prin apelarea comenzii “*make clean*”, se vor sterge toate fişierele cu extensia “.o” din directorul curent.

**Exemplul 1** - Să se creeze un fişier Makefile pentru compilarea următorului program C (ex1.c):

```
#include <stdio.h>
#include <math.h>
int main()
{
    int u=45;
    double res;
    res = sin((u*3.14)/180);
    printf("sin(%d) = %lf\n", u, res);
    return 0;
}
```

**Makefile:**

```
OBJS= ex1.o
CC= gcc
CFLAGS= -c -Wall
LFLAGS = -lm -Wall

ex1.o: ex1.c
    $(CC) $(CFLAGS) ex1.c
ex1: $(OBJS)
    $(CC) $(LFLAGS) $(OBJS) -o ex1
clean:
    rm -rf *.o ex1
```

**compilare:**



## Makefile-uri

---

```
root@slax:~# make ex1
gcc -c -Wall ex1.c
gcc -lm -Wall ex1.o -o ex1
root@slax:~# ./ex1
sin(45) = 0.706825
root@slax:~#
```

**Exemplul 2** - Să se creeze un fișier Makefile pentru compilarea următoarei aplicații cu surse multiple:

**main.c:**

```
#include <stdio.h>
double ma(int nr[20], int n);
double mg(int nr[20], int n);
int main() {
    double med_a, med_g;
    int i, n, nr[20];
    printf("n= ");
    scanf("%d", &n);
    for (i=0; i<n; i++) {
        printf("nr[%d]= ", i+1);
        scanf("%d", &nr[i]);
    }
    med_a = ma(nr, n);
    med_g = mg(nr, n);
    printf("Media aritmetica= %lf\n", med_a);
    printf("Media geometrica= %lf\n", med_g);
    return 0;
}
```

**medii.c:**

```
#include <math.h>
double ma(int nr[20], int n)
{
    int i;
```

```
        double res=0;
        for (i=0;i<n;i++)
        {
            res=res+nr[i];
        }
        return (res/n);
    }

double mg(int nr[20], int n)
{
    int i;
    double res=1;
    for (i=0;i<n;i++)
    {
        res=res*nr[i];
    }
    return sqrt(res);
}
```

**Makefile:**

```
OBJS= main.o medii.o
CC= gcc
CFLAGS= -c -Wall
LFLAGS = -lm -Wall

main.o: main.c
    $(CC) $(CFLAGS) main.c
medii.o: medii.c
    $(CC) $(CFLAGS) medii.c
ex2: $(OBJS)
    $(CC) $(LFLAGS) $(OBJS) -o ex2
clean:
    rm -rf *.o ex2
```

### *compilare:*

```
root@slax:~# make ex2
gcc -c -Wall main.c
gcc -c -Wall medii.c
gcc -lm -Wall main.o medii.o -o ex2
root@slax:~# ./ex2
n= 3
nr[1]= 1
nr[2]= 2
nr[3]= 3
Media aritmetica= 2.000000
Media geometrica= 2.449490
root@slax:~#
```

### 11.3. Exerciții propuse

- 1) Să se scrie un program C care calculează rădăcinile reale sau complexe ale unei ecuații de gradul 2 de forma  $ax^2 + bx + c = 0$ , unde  $a$ ,  $b$  și  $c$  sunt numere întregi introduse de la tastatură. Să se scrie un fișier *Makefile* pentru acest program.
- 2) Să se scrie un program C pentru calculul combinărilor de “n” luate câte “k” știind că:  $C_n^k = \frac{n!}{(n-k)! * k!}$ . Funcțiile *factorial* și *combinari* se vor scrie într-un fișier denumit *funcții.c* iar apelarea lor și citirea valorilor lui “n” și “k” se va face în fișierul *main.c*. Să se scrie un fișier *Makefile* pentru acest program.
- 3) Să se scrie un program C pentru calculul determinantului unei matrice 4x4 cu elemente numere reale introduse de la tastatură. Să se scrie un fișier *Makefile* pentru acest program.