

Sisteme de operare, mecanisme interne și principii de proiectare : Îndrumar de laborator / Universitatea Tehnică a Moldovei, Facultatea CIM, Departamentul Ingineria Software și Automatică ; alcătuitori: Beșliu Victor, Ciorbă Dumitru, Colesnic Victor ; redactor responsabil: Victor Colesnic. – Chișinău : Tehnica-UTM, 2021. – 88 p. : fig., tab.

Resurse bibliogr.: p. 88 (9 tit.). – 10 ex.

ISBN 978-9975-45-672-2.

004.45(076.5)

S 61

Cuprins

Introducere.....	5
1 Instalarea SO GNU/Linux.....	7
2 Utilizarea SO GNU/Linux.....	19
3 Bazele utilizării consolei a SO GNU/Linux.....	30
4 Procesarea fluxurilor textuale în SO GNU/Linux.....	48
5 Monitorizarea proceselor.....	62
6 Gestionarea proceselor în SO GNU/Linux	74
7 Manipularea fișierelor în SO GNU/Linux	91
8 Utilizarea consolei în SO Microsoft Windows Server.....	101
Bibliografia.....	109

Introducere

Prin noțiunea **sistem de operare** înțelegem modulele program ale sistemului de calcul care administrează resursele fizice (procesoarele, memoria, dispozitivele de intrare/ieșire etc.) și logice (datele, informația, aplicațiile etc.). Modulele în cauză soluționează situațiile de conflict, optimizează productivitatea sistemului, sporesc eficiența utilizării lui, asigură protecția componentelor și securitatea informației. Concomitent, ele joacă rolul de intermediar (interfață) între utilizator și calculator.

Sistemele de operare moderne reprezintă un software complex în care sunt implementate o varietate de soluții tehnologice. Sistemele de operare utilizate pe stațiile de lucru și pe calculatoarele personale implementează, de obicei, principiile transparenței managementului și neimplicarea utilizatorului în alocarea resurselor și alte sarcini ale sistemului de operare, punând la dispoziția utilizatorului interfețe comode și prietenoști, textuale sau grafice pentru accesul la resursele sistemului de calcul – date sau informații, aplicații sau echipamente.

Însă, în cazul utilizării sistemelor de operare pentru gestionarea aplicațiilor server, pentru rezolvarea sarcinilor cu cerințe specifice de performanță, fiabilitate și securitate, este necesară administrarea regulată a sistemului de operare, monitorizarea software-ului de aplicație și a sistemului de operare însuși, selectarea parametrilor optimi de control și controlul manual, la necesitate.

Pentru realizarea acestor cerințe este necesară înțelegerea principiilor de funcționare a mecanismelor sistemului de operare.

Îndrumarul de laborator propus constă din opt lucrări de laborator. Primele șapte laboratoare sunt dedicate SO **GNU/Linux**.

De ce **GNU/Linux**? În primul rând, mai mult de două treimi din serverele din întreaga lume rulează SO **GNU/Linux** și, în al doilea rând, SO **GNU/Linux** are o abordare specifică pentru

gestionarea resurselor și o administrare, fundamental diferită de alte familii de sisteme de operare, de exemplu, **Windows**.

Pentru SO **GNU/Linux**, managementul sistemului include utilizarea interfeței textuale (de comandă, consolă) și script-urile de control, precum și afișarea informațiilor despre starea și configurația sistemului de operare sub formă de fișiere text, extrem de importante și necesare unui viitor inginer IT.

1 Instalarea SO GNU/Linux

1.1 Scopul lucrării: metodele de instalare a SO **GNU/Linux**, crearea partițiilor.

1.2 Indicații metodice

Sunt mai multe metode de instalare a SO **GNU/Linux**. Instalarea nativă este utilă dacă planificați să utilizați în continuare **Linux-ul**. Sistemul **Linux** are nevoie de cel puțin două partiții separate pentru a putea funcționa. Dacă pe disc există deja **Windows**, este necesară redimensionarea partițiilor existente pe diskul rigid, pentru a putea crea partițiile destinate **Linux-ului**.


Pentru studierea **Linux-ului** se recomandă instalarea lui pe o mașină virtuală (mai simplă de instalat decât sistemul). Vom utiliza mașina virtuală - **VirtualBox** și distribuția de **Linux – Mint**. Dacă aveți instalat sistemul **Windows10**, el permite instalarea sistemului **Linux**, distribuția **Ubuntu**, ca un subsistem.

Executați următorii pași:

- Descărcați și instalați **VirtualBox**.
- Descărcați distribuția **Linux-Mint Desktop**.

Executați clic pe **New** și introduceți denumirea sistemului. Se recomandă memorie RAM de **2 GB** (figura 1.1). Bifați - **Create a virtual hard disk now**. Executați clic - **Create**.

Executați clic pe butonul **VHD (Virtual Hard Disk)** (se recomandă 14 GB) și **Dynamically allocated** > Click **Create** (figura 1.2).

Click **Storage>Controller: IDE- Empty** > clic  > **Choose Virtual Optical Disk File**, și indicați calea spre distribuția **Mint** descărcată > **OK** (fig. 1.3).

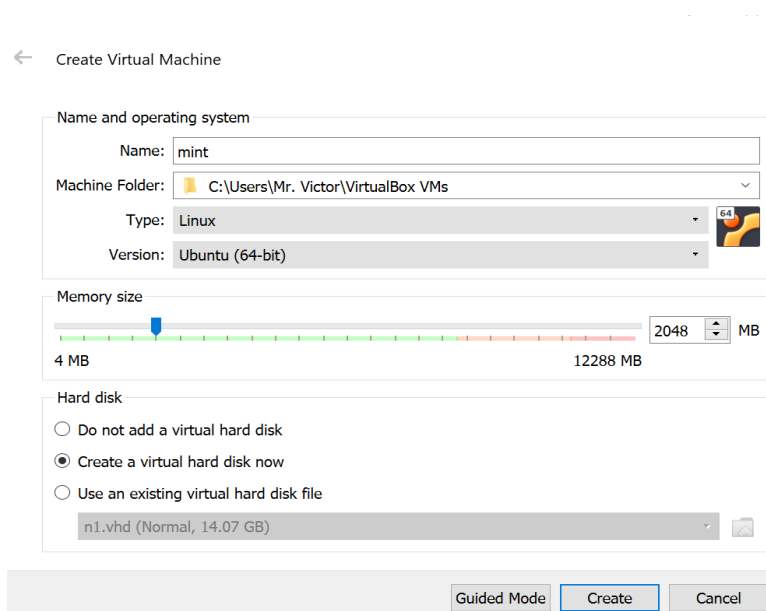


Figura 1.1 – Alocarea memoriei RAM

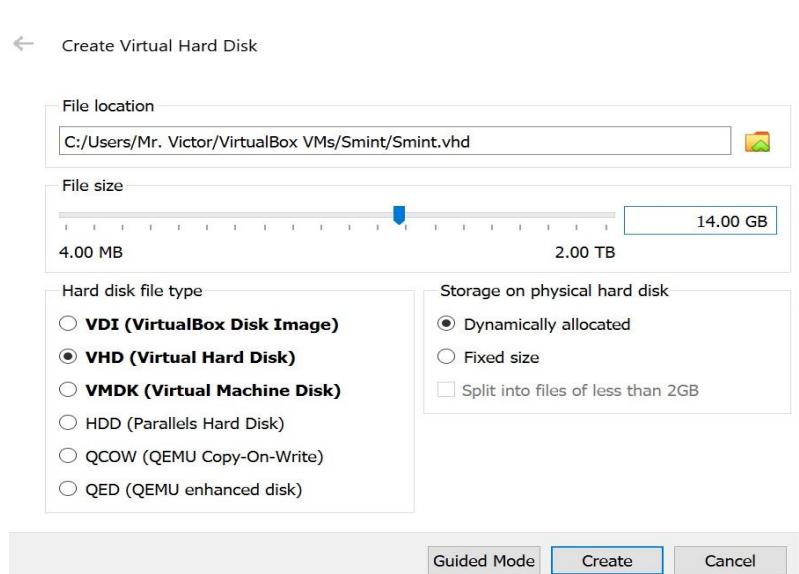


Figura 1.2 – Setarea capacității discului virtual

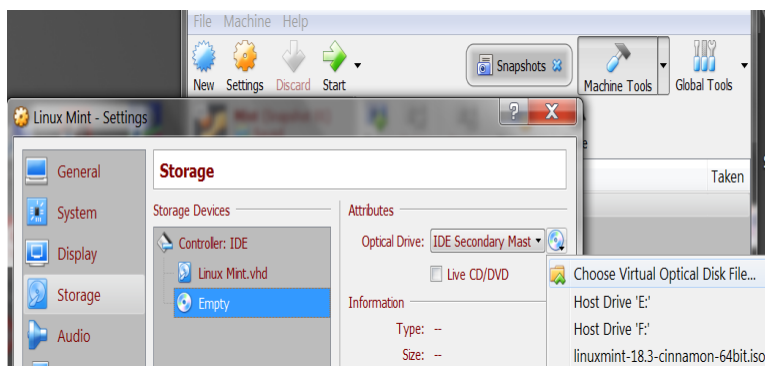


Figura 1.3 - Calea spre distribuția **Linux Mint**

Executați clic - Click **Display**> **Scale Factor**> Min 100% (figura 1.4).

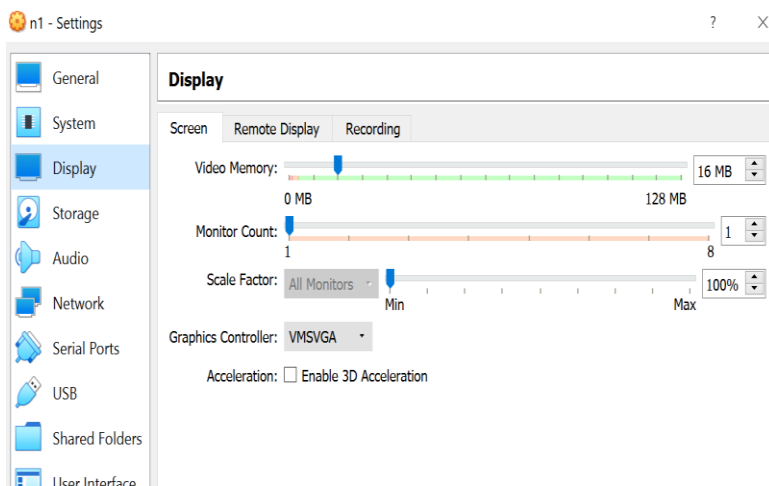
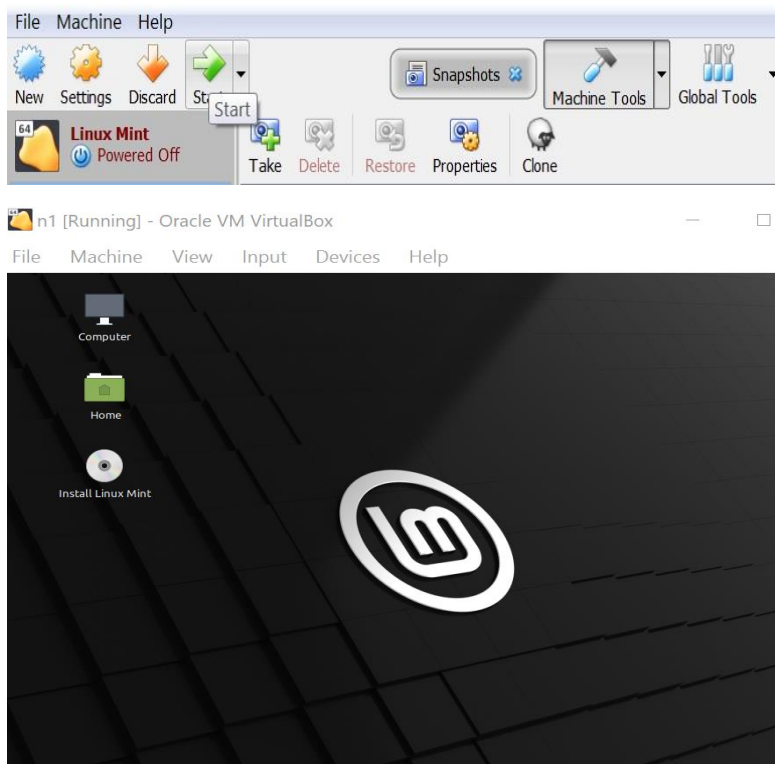


Figura 1.4 – Selectarea **Scale Factor**

Continuăm instalarea **Linux-ului**

Start – Start Linux Mint Install Linux Mint >



Install Linux Mint selectați limba de instalare >
Continue > Something else (figura 1.5).

Executați clic pe butonul **New Partition Table - Continue > /dev/sda > + .** (figura 1.6).

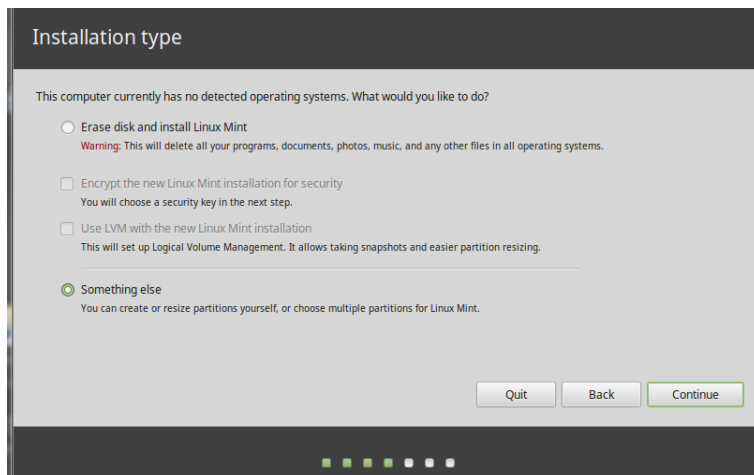


Figura 1.5 – Selectarea modului de instalare

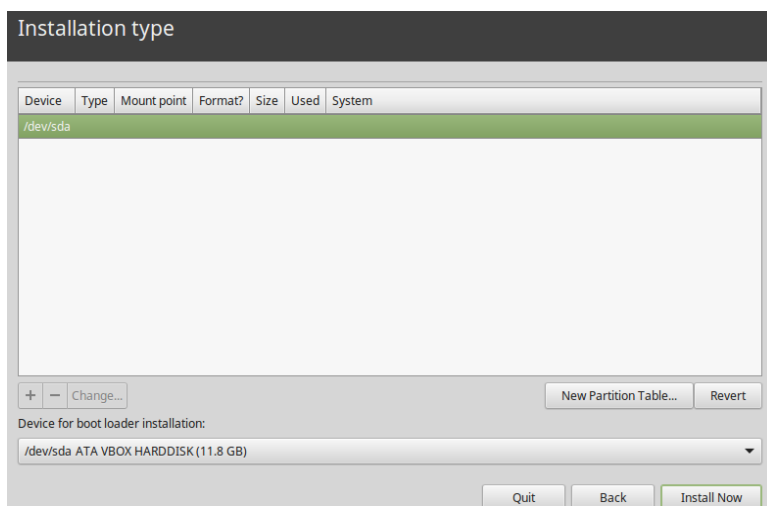


Figura 1.6 –Pași de instalare. Crearea partițiilor

Se recomandă crearea a următoarelor partiții:

- o partiție **swap** (este tratată ca o continuare a memoriei RAM). Partiția **swap** se recomandă de 1 GB;

- partiția **root (/)**, unde se va afla directorul-rădăcină al sistemului, și care va conține toate fișierele din sistem;
- o partiție **/home**, care va conține fișierele utilizatorului (1 GB).

Selectați (figura 1.7):

- Size 1 GB.
- Type for the new partition – Logical.
- Location for the new partition – beginning of this space.
- Use as: swap area.
- OK.

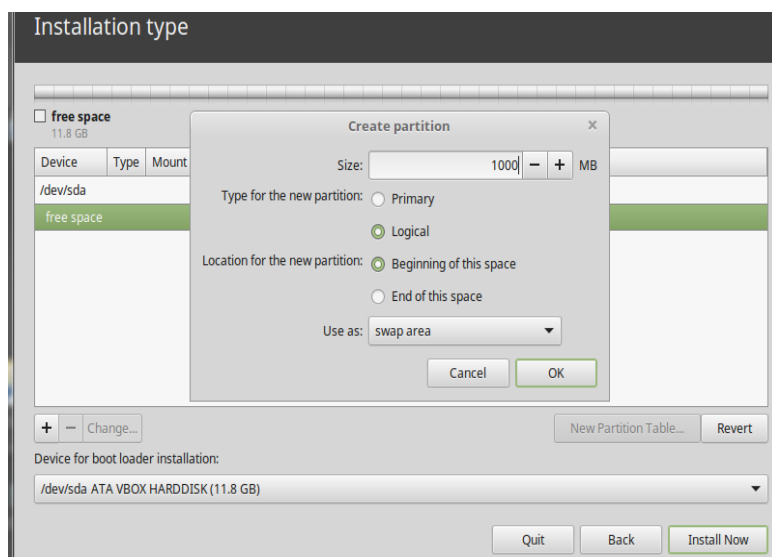


Figura 1.7 –Pași de instalare. Crearea partiției **swop**

Creați partiția **/home** – pentru aplicațiile utilizatorului (figura 1.8). Se recomandă un spațiu de 1GB. Această partiție este asemănătoare partiției **D:** în **Windows**.

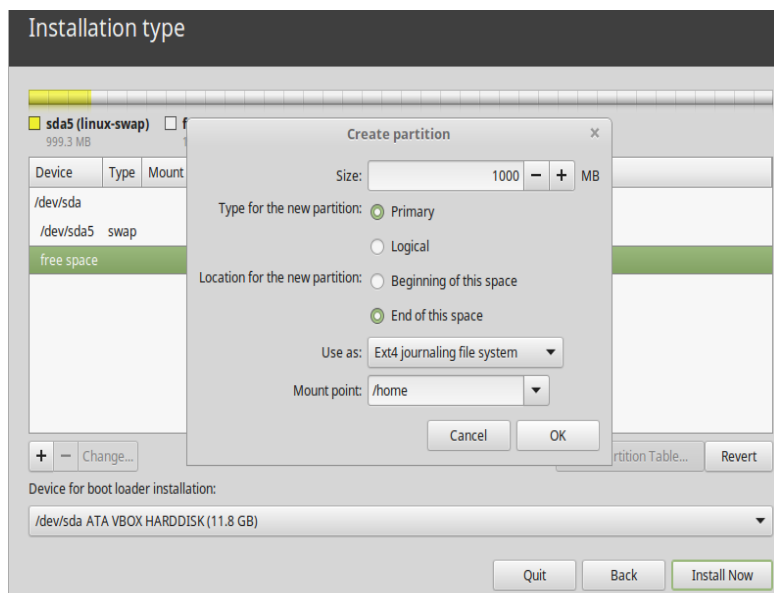


Figura 1.8 –Pașii de instalare. Crearea partiției *home*

Creați partiția **root** (figura 1.9). Sistemul de fișiere - **Ext4 journaling file system** – un sistem de fișiere, cu suport pentru jurnalizare. Punctul de montare indicați **/** - directorul-rădăcină al sistemului. Partiția **/** este corespondenta partiției **C:** din **Windows**. În aceasta partiție vor fi instalate sistemul de operare și aplicațiile.

Executați clic **Install Now** (figura 1.10).

La pasul următor vi se va cere să alegeți locația în care vă aflați.

Creați un utilizator, introduceți o parolă și alegeți tipul de autentificare.

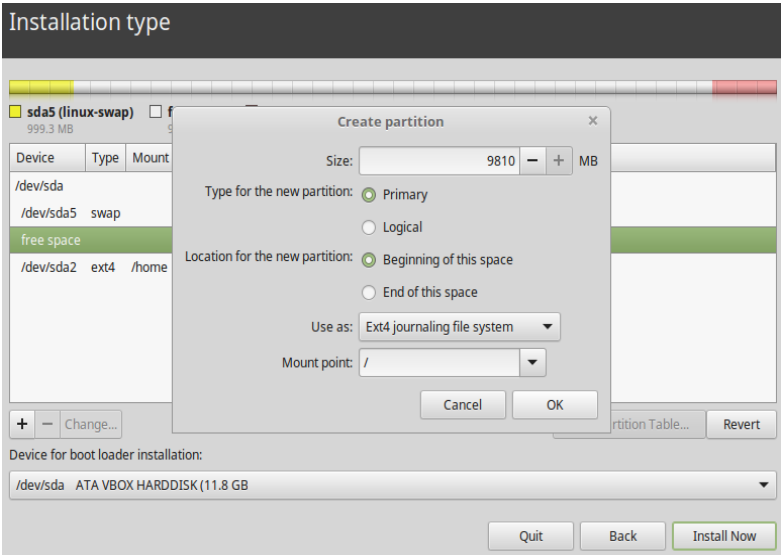


Figura 1.9 –Pași de instalare. Crearea partiției **root**

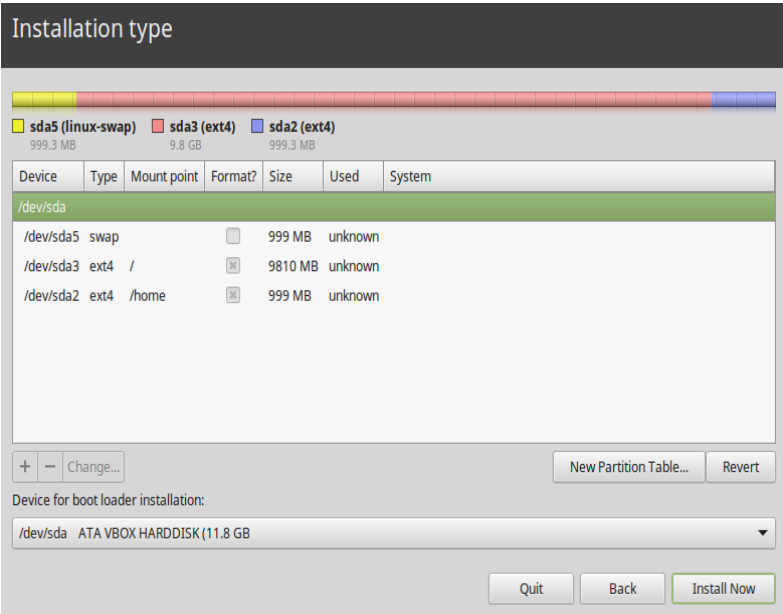


Figura 1.10 –Partițiile create

Dacă aveți instalat sistemul de operare Windows10 puteți instala sistemul Linux în Windows10 ca un subsistem.

Activați modul *Developer Mode* (figura 1.11).

- Deschideți *Settings* -> *Update and Security* -> *For developers*
- Selectați butonul *Developer mode*.

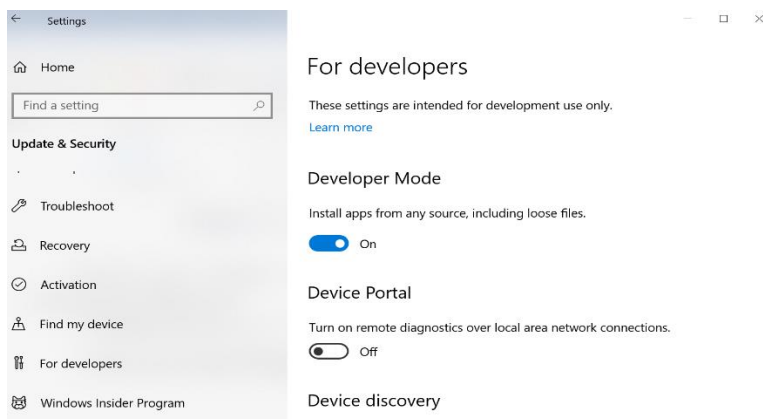


Figura 1.11 – Fereastra “For developers”

Activați apoi funcția *Windows Subsystem for Linux* (din interfața de utilizator):

- În **Start** căutați *Turn Windows features on or off* (figura 1.12).
- Selectați (bifați) **Windows Subsystem for Linux** > OK.

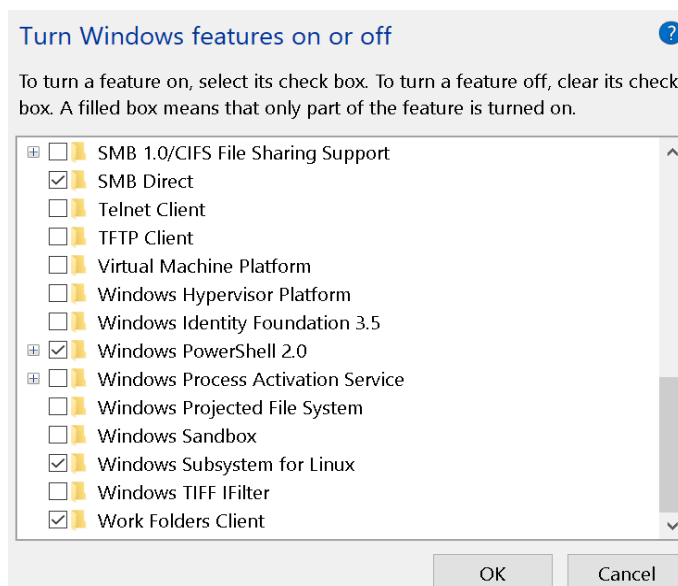


Figura 1.12 - Windows features

După activarea funcției **Windows Subsystem for Linux**, resetați calculatorul.

Windows completed the requested changes.

Windows needs to reboot your PC to finish installing the requested changes.

După resetarea calculatorului, porniți *Microsoft Store* din meniul *Start* și căutați „Linux” în magazin. Executați clic pe „Get the apps” sub banner „Linux on Windows?”.

Mai detaliat link-ul -

<https://www.howtogeek.com/249966/how-to-install-and-use-the-linux-bash-shell-on-windows-10/>

1.3 Sarcină pentru lucrarea de laborator

Instalați **Linux-ul**, în **mod manual**, pe calculatorul dumneavoastră.

Condiții conform baremului:

- 1) descrierea etapelor realizate (maximal 3 screenshot-uri) pentru instalarea SO **LINUX**;
- 2) descrierea metodelor de instalare a **Linux-ului** (virtual, alături de alt sistem ...);
- 3) clasificarea distribuțiilor **Linux** după platformele ce le utilizează (server, desktop ...); de asemenea se vor prezenta 3 cele mai populare versiuni **LINUX** existente la momentul actual și **principalele caracteristici tehnice** ale acestora.
- 4) Creați directoarele necesare partajării de fișiere dintre SO instalat pe calculator și SO **GNU/Linux** instalat în **VirtualBox**.

Baremul – realizarea p.1, 2 – nota 6; 1-3 –nota 9; 1-4 - nota 10.

Perfectați și prezentați profesorului raportul și primiți întrebări sau sarcină pentru susținere.

1.4 Sarcină pentru lucrul neauditorial

Pregătiți și executați punctele a), b), c), și d) din 2.3 din lucrarea de laborator 2. Rezultatele execuției vor sta la baza pregătirii către lucrarea de laborator 2.

Задания к лабораторной работе 1

Установите Linux, в ручном режиме, на Вашем компьютере.

Содержание отчёта

Отчёт по лабораторной работе должен содержать:

- 1) Описание шагов (не более 3-х screenshot) при установке операционной системы LINUX.
- 2) Описание методов установки LINUX (рядом с другой ОС, виртуально ...).
- 3) Классификацию дистрибутивов LINUX по платформам применения (сервер, desktop ...), а также приведите 3 наиболее популярных дистрибутива LINUX и **их основные технические характеристики.**
- 4) Создайте каталоги для совместного использования файлов между ОС, установленной на Вашем компьютере, и Linux, установленной в VirtualBox.

Выполненная лабораторная работа будет оценена следующим образом: выполнение п.1,2 – 6 баллов, 1-3 – 9, 1-4 – 10.

Задачи для неаудиторной работы

Подготовить и выполнить пункты а), б), с) и d) раздела 2.3 лабораторной работы 2. Результаты выполнения будут основой для оценки подготовки студентов к лабораторной работе 2.

2 Utilizarea SO GNU/Linux

2.1 Scopul lucrării: studierea tiparului sistemului de fișiere navigarea, căutarea fișierelor, proprietățile fișierelor și schimbarea permisiunilor acestora, căutarea de ajutor, câteva comenzi ale interpretorului.

2.2 Indicații metodice

Tiparul sistemului de fișiere. Fișierele din **Linux** sunt prezentate, prin convenție, ca o structură arborescentă. Pe un sistem **Linux** standard veți găsi că tiparul sistemului de fișiere se încadrează în schema prezentată mai jos (figura 2.1).

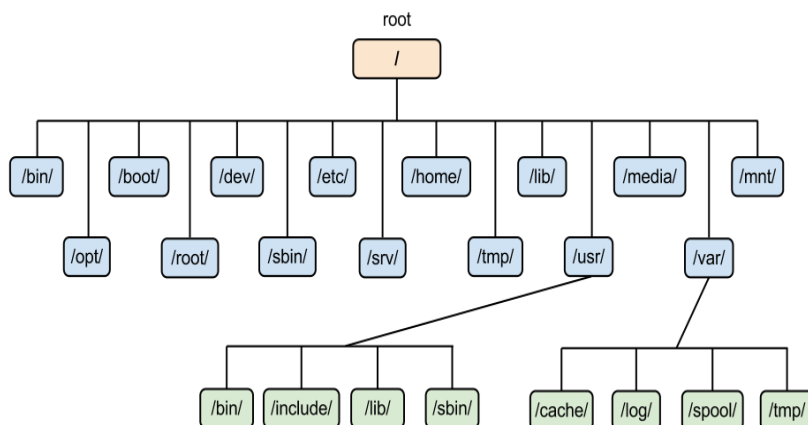


Figura 2.1 – Structura generală a sistemului de fișiere **Linux**

În această structură, de obicei, în directorul **bin** se află fișierele speciale pentru dispozitive periferice: compilatoare, asamblatoare, instrumente pentru dezvoltarea de programe; în directorul **etc** – date de sistem cu acces limitat și controlat, utilitare de sistem destinate în special administratorului sistemului, fișiere cu parole, fișiere cu comenzi **SHELL** de inițializare ș. a.

În funcție de distribuție și destinația calculatorului, dar și de preferințele administratorului sistemului, structura poate varia pentru că directoarele pot fi adăugate sau eliminate după dorință. Nici numele directoarelor nu sunt neapărat necesare în această formă; ele sunt doar o convenție. Structura arborescentă a sistemului de fișiere începe cu trunchiul, indicat de caracterul (/). Acest director, care conține toate directoarele și fișierele sistemului, este numit directorul **root** sau „rădăcina” sistemului de fișiere.

Tipuri de fișiere. Componenta sistemului de operare care “are grijă” de fișiere se numește Sistem de Gestiune a Fișierelor (SGF). O descriere simplă a unui sistem **UNIX**, care se aplică și **Linux-ului**, este următoarea: „Într-un sistem **UNIX**, orice este un fișier; dacă ceva nu este fișier, atunci este proces.”

Această propoziție este adevărată deoarece există fișiere speciale care sunt mai mult decât niște simple fișiere (cele numite **pipes** – conexiuni sau **sockets**, de exemplu), dar pentru simplitate, a spune că orice este un fișier constituie o generalizare acceptată. Un sistem **Linux**, la fel ca și **UNIX**, nu face nici o diferență între un director și un fișier, deoarece un director este doar un fișier care conține numele altor fișiere. Programele, serviciile, textele, imaginile și așa mai departe, sunt toate fișiere. Dispozitivele, în general, sunt considerate și ele fișiere, din punctul de vedere al sistemului.

Exemplu: Comanda **ls -l** afișează tipul fișierului, utilizând primul caracter al fiecărei linii:

```
ubuntu@ubuntu:~/Desktop$ ls -l
total 24
-rwxr-xr-x 1 ubuntu ubuntu 9004 Oct  6 11:28 examples.desktop
-rwxr-xr-x 1 ubuntu ubuntu 8195 Oct  6 11:28 ubiquity.desktop
ubuntu@ubuntu:~/Desktop$
```

Schema de acces la fișiere prevede trei drepturi de acces: *read* (r) – citire, *write* (w) – scriere și *execute* (x) – execuție și trei

categorii de utilizatori: *user (u)* - proprietar, *group (g)* - grup și *others (o)* - ceilalți utilizatori.

```
ubuntu@ubuntu:~$ ls -l
total 0
drwxr-xr-x 2 ubuntu ubuntu 80 Oct 6 11:28 Desktop
drwxr-xr-x 2 ubuntu ubuntu 40 Oct 6 11:28 Documents
drwxr-xr-x 2 ubuntu ubuntu 40 Oct 6 11:28 Downloads
drwxr-xr-x 2 ubuntu ubuntu 40 Oct 6 11:28 Music
drwxr-xr-x 2 ubuntu ubuntu 40 Oct 6 11:28 Pictures
drwxr-xr-x 2 ubuntu ubuntu 40 Oct 6 11:28 Public
drwxr-xr-x 2 ubuntu ubuntu 40 Oct 6 11:28 Templates
drwxr-xr-x 2 ubuntu ubuntu 40 Oct 6 11:28 Videos
ubuntu@ubuntu:~$
```

Rezultă că vor trebui să existe 9 poziții pentru precizarea completă a drepturilor (3 drepturi de acces x 3 categorii de utilizatori).

Pentru fișiere, semnificația drepturilor de acces reiese din numele acestor drepturi, *write* incluzând și posibilitatea de ștergere.

Pentru fișierele director, drepturile de acces au alte semnificații:

- *read* – există posibilitatea de listare a directorului cu comanda **ls**;
- *write* – se pot crea/șterge fișiere director;
- *execute* – se poate parcurge directorul pentru accesul la fișierele conținute.

Drepturile de acces pot fi vizualizate cu comanda **ls**, cu opțiunile:

- **l** (forma lungă);
- **a** (toate intrările);
- **t** (sortează după tipul ultimei modificări);
- **r** (ordine inversă).

Primul caracter dintr-o linie indică tipul fișierului: director (d), fișier special (b sau c) sau fișier ordinar (-) (tabelul 2.1).

Următoarele nouă caractere descriu drepturile de acces ale *proprietarului fișierului* (primele trei caractere), *membrilor grupului* (următoarele trei) și *celorlalți utilizatori* (ultimele trei caractere).

Tabelul 2.1 – Tip fișier

Simbol	Tipul fișierului
–	Fișier obișnuit
d	Director
l	Legătură
c	Fișier special
s	Socket
p	Numite conexiuni (pipe)
b	Dispozitiv

Literele *r*, *w*, *x* sunt întotdeauna listate în această ordine; prezența unei litere indică prezența dreptului, iar semnul minus indică absența dreptului respectiv. Următoarele coloane indică, în ordine, numărul de legături, numele proprietarului, numele grupului, numărul de caractere din fișier și data la care fișierul a fost modificat ultima oară.

Schimbarea drepturilor de acces se face cu comanda **chmod**. În această comandă este necesar să fie specificate următoarele informații:

- pentru ce persoane se stabilesc drepturile de acces;
- care sunt drepturile care se modifică;
- care este fișierul ale cărui drepturi de acces se modifică.

Pentru persoanele ale căror drepturi de acces se stabilesc pot fi folosite caracterele **u** (utilizator), **g** (grup), **o** (alții) sau **a** (toți).

Pentru drepturile care se modifică elementele se dau sub forma unui grup de două caractere. Primul caracter este **+** (pentru acordarea

dreptului) sau - (pentru retragerea dreptului) urmat de unul din caracterele **r**, **w** sau **x** care se referă la dreptul de acces în discuție.

De exemplu, creăm un nou fișier **gicu** cu comanda **touch**. Cu comanda **ls -l** afișăm drepturile de acces la acest fișier.

```
File Edit View Search Terminal Help
stud@victor-V / $ cd
stud@victor-V ~ $ touch gicu
stud@victor-V ~ $ ls -l
total 0
-rw-rw-r-- 1 stud stud 0 Dec 18 13:42 gicu
stud@victor-V ~ $
```

Observăm că fișierul nou creat nu poate fi executat (lipsa x). Pentru a acorda un drept de acces la acest fișier este utilizată comanda **chmod** cu opțiuni - caractere sau cu cifre conform tabelului 2.2.

De exemplu, comanda **chmod u+x gicu** acordă grupului utilizatorilor (*user*) dreptul de execuție a fișierului **gicu**, iar comanda **chmod 777 gicu** acordă toate drepturile (r,w,x - 4+2+1) pentru oricine (user, group, others).

Tabelul 2.2 - Coduri pentru drepturi de acces și tipuri de utilizatori

	Cod	Semnificație
Drepturi de acces	0 sau -	Drepturile de acces asociate fișierului nu sunt acordate
	4 sau r	Categoria de utilizatori definită are drepturi de citire
	2 sau w	Categoria de utilizatori definită are drepturi de scriere
	1 sau x	Categoria de utilizatori definită poate rula fișierul

Tipuri de utilizatori	u	Permisele acordate utilizatorilor (user)
	g	Permisele acordate grupurilor (group)
	o	Permisele acordate celorlalți (others)

Comenzi pentru începerea lucrului. În tabelul 2.3 sunt prezentate câteva comenzi, de care avem nevoie pentru început.

Tabelul 2.3 - Comenzi ale interpretorului limbajului de comandă

Comanda	Ce execută
ls	Afișează o listă cu fișierele din directorul de lucru
cd <i>director</i>	schimbă directorul
touch <i>filename</i>	crează un fișier vid (0 octeți)
comandă>fișier	redirecționarea ieșirii standard în fișier, conținutul vechi al fișierului este șters.
comandă>>fișier	redirecționarea ieșirii standard în fișier, conținutul vechi al fișierului nu se șterge.
passwd	schimbă parola pentru utilizatorul curent
file <i>filename</i>	afișează tipul fișierului al cărui nume este filename
cat <i>textfile</i>	afișează conținutul unui fișier text pe ecran
pwd	afișează directorul curent (print working directory)
sudo adduser <i>user</i>	adăugarea unui nou utilizator, user – numele utilizatorului adăugat

su user	schimbă utilizatorul (dacă cunoașteți parola)
Exit sau logout	terminare sesiune
man comandă	afișarea paginilor din manualul comenzii comandă
info comandă	afișarea de informații despre comandă
apropos string	comandă de căutare a fișierelor cu pagini de manual
df	returnează informații legate de partițiile montate
which comandă	arată traseul complet al comenzii în cauză
echo șir	afișează șirul de caractere introdus
wc fișier	numără liniile, cuvintele și caracterele dintr-un fișier
umask [valoare]	Arată sau schimbă modul în care sunt create fișiere noi

Dacă doriți să vă fie prezentate mai multe informații despre comenzi utilizați **man comandă**.

În cazul comenzilor, ca o completare a paginilor **man**, puteți citi paginile **Info**, folosind comanda **info**. Aceste pagini conțin, de regulă, informații mai recente și, într-un fel, mai ușor de utilizat.

Un index cu explicații scurte despre comenzi este disponibil prin comanda **whatis**, precum în exemplul de mai jos.

```
whatis ls
```

Modificarea apartenenței fișierului la un utilizator sau un grup. Comenzile **chown** (schimbă utilizatorul) sau **chgrp** (schimbă

grupul) modifică apartenența fișierului. Schimbarea deținătorilor unui fișier este o sarcină des întâlnită de administratorii de sistem în mediile în care este nevoie ca fișierele să fie partajate de către grupuri de utilizatori. Ambele comenzi sunt flexibile, după cum puteți afla prin opțiunea **-help**. Comanda **chown** poate fi aplicată pentru a schimba deținătorii unui fișier atât la nivel de utilizator, cât și la nivel de grup, pe când **chgrp** schimbă doar grupul care deține acel fișier. Desigur, sistemul va verifica dacă utilizatorul care introduce aceste schimbări are suficiente permisiuni asupra fișierului sau fișierelor asupra cărora dorește să folosească aceste comenzi. Pentru a modifica doar utilizatorul deținător al unui fișier, folosiți sintaxa de mai jos:

chown utilizator_nou fișier

Dacă folosiți două puncte după numele de utilizator, va fi modificat și grupul deținător (consultați paginile **Info**).

2.3 Sarcină pentru lucrarea de laborator

Executați următorii pași:

- a) **Autentificați-vă** ca utilizator obișnuit. Executați operațiile enumerate mai jos și găsiți răspunsuri la întrebări.
- b) **Introduceți** comanda „cd ..”. Repetați comanda. Comentați. Folosiți comanda pwd. Comentați. Listați conținutul directorului cu ajutorul comenzii ls. Comentați. Încercați comenzile ls -a; ls -al în această ordine. Comentați.
- c) **Căutarea de ajutor**. Citiți **man ls**. Citiți info passwd. Introduceți comanda apropos pwd. Comentați.
- d) **Turul sistemului**. Treceți în directorul /proc. Găsiți și afișați răspunsurile la următoarele întrebări: Pe ce procesor rulează sistemul dumneavoastră? Ce volume de memorie RAM, swap sunt instalate? Ce drivere sunt instalate? De câte ore rulează sistemul? Ce sisteme de fișiere sunt recunoscute de sistem? Mutați-vă în /etc. Ce versiune a

sistemului de operare folosiți? Ce versiune a *shell bash* este instalată în sistemul dumneavoastră?

- e) Executați acțiunile sau răspundeți la întrebările de mai jos:
 - 1) Care este modul implicit pentru crearea fișierelor în cazul dumneavoastră? (*umask*).
 - 2) Schimbați deținătorii directorului **/etc** în utilizatorul și grupul asociate dumneavoastră.
 - 3) Schimbați permisiunile asociate fișierului **~/ .bashrc** în așa fel încât numai dumneavoastră și grupul dumneavoastră să îl poată citi.
 - 4) Introduceți comanda **locate root**. Comentați.
- f) Toate acțiunile executate în punctele b – e să fie incluse în raport.
- g) Perfectați și prezentați profesorului raportul și primiți întrebări sau sarcină pentru susținere.

2.4 Sarcină pentru lucrul neauditorial

Subiecte de pregătire către lucrarea de laborator 3: Ce este **Shell**? Când se folosește *citarea*? *Citarea* și **backslash-ul**. Destinația următorilor operatori - **;**, **&&**, **||**, **|**. Utilizarea ghilimelelor, redirecționarea, operațiile executate:

\$ sort < lista > lista_sortata

Instrucțiunile: **let**, **test**, **read**, **break**, **continue**, **exit**. Structuri de control – **if**, **select**, **case**, **while**, **until** și **for**.

Задания к лабораторной работе №2

Выполните следующее:

- a) Войдите в систему под именем и паролем установленными во время установки **LINUX**. Выполните перечисленные ниже операции и найдите ответы на вопросы.
- b) Введите команду **cd ..**. Повторите команду. Что происходит? Введите команду **pwd**. Прокомментируйте. Просмотрите содержание каталога командой **ls**. Что выводится? Введите команды **ls -a**; **ls -al** в этом порядке. Что происходит?
- c) **Помощь.** Прочтите **man ls**. Прочтите **info passwd**. Примените команду **apropos pwd**. Прокомментируйте.
- d) **Обзор системы.** Перейдите в каталог **/proc**. На каком процессоре работает Ваша система? Какой объем памяти используется на данный момент? Какой объем памяти отведен под **swap**? Какие драйверы загружены? Сколько часов активна система? Какие файловые системы поддерживаются операционной системой? Перейдите в **/etc**. Какую версию ОС используете? Какая версия shell-a **bash** установлена в Вашей системе?

е) Выполните действия и ответьте на нижеприведенные вопросы:

- 1) Какой способ, по умолчанию, принят для создания файлов? (**umask**).
- 2) Измените права каталога **/etc** на Ваши права как пользователя и Вашей группы.
- 3) Измените права файла **~/.bashrc** так, чтобы только вы и ваша группа могли прочитать его.
- 4) Введите команду **locate root**. Что заметили?

Все действия выполненные в пунктах b) –e) сохраните в Отчете.

Подготовьте и представьте отчет преподавателю и получите вопросы или задание для защиты лабораторной работы.

Задачи для неаудиторной работы

Вопросы для подготовки студентов к лабораторной работе 3: Объясните понятие ***Shell***. Когда применяется экранирование? Экранирование и ***backslash***. Назначение следующих операторов - ***;***, ***&&***, ***||***, ***|***. Применение кавычек, перенаправления.

Следующие выполняемые команды:

```
$ sort < list > sorted_list
```

Команды: ***let***, ***test***, ***read***, ***break***, ***continue***, ***exit***. Операторы циклов – ***if***, ***select***, ***case***, ***while***, ***until*** и ***for***.

3 Bazele utilizării consolei a SO GNU/Linux

3.1 Scopul lucrării: utilizarea consolei, structurilor de control, crearea script-urilor în bash

3.2 Indicații metodice

Shell scripting. Interpretorul de comenzi **Shell** permite utilizatorului realizarea unor programe, numite **Shell Scripts**, facilitând construirea unor secvențe de comenzi ce sunt plasate în niște fișiere de tip text și sunt executate în mod identic cu programele compilate. **Shell-ul** dispune de un limbaj de programare și este identificat cu o interfață în linie de comandă. Recunoaște toate noțiunile de bază specifice oricărui limbaj de programare, cum ar fi: variabile, instrucțiuni de atribuire, structuri de control (**if**, **while**, **for**, **case**), proceduri/funcții, parametri.

În continuare ne vom referi la **Bash** (**Bourne Again SHell**). Există și alte **shell-uri** pe sisteme **GNU/Linux** precum **tcsh**, **zsh**, **ash**, etc. **Microsoft** oferă **PowerShell** pe sistemele **Windows**.

Un simplu script shell. Un script care afișează mesajul "Hello, World!" este următorul:

```
#!/bin/bash
# afiseaza mesaj
clear
echo "Hello, World!"
exit 0
```

Execuția acestuia este următoarea:

```
~$chmod +x hello.sh
~$./hello.sh
```

```
Hello, World!
```

Se observă că este necesar ca fișierul să fie executabil pentru a putea fi interpretat. Șirul **#!** de la începutul fișierului poartă denumirea de **shebang**. Acesta indică sistemului ce program va fi invocat pentru a interpreta script-ul. Exemple pot fi:

```
#!/bin/sh
#!/bin/bash
#!/usr/bin/perl
#!/usr/bin/python
#!/usr/bin/ruby
```

Un script poate fi rulat prin precizarea explicită a interpretorului în linia de comandă:

```
~$ bash hello.sh
Hello, World!
```

În această situație nu este nevoie ca script-ul să fie executabil și nici nu este nevoie de prezența liniei **#!/bin/bash**. Caracterul **#** semnifică începutul unui comentariu care durează până la sfârșitul liniei. Comanda **exit** este folosită pentru a indica valoarea de retur a script-ului. Este implicit 0. Pentru a scrie script-uri în **Bash** este necesar un editor de text. Exemple: **nano**, **gedit**.

Shell-ul prezintă o serie de operatori folosiți pentru execuția comenzilor.

Concatenarea comenzilor. Următorii operatori sunt folosiți pentru concatenarea diverselor comenzi:

- **comanda1 ; comanda2** - comenzile sunt executate secvențial (una după alta);

- **comanda1 && comanda2** - *comanda2* este executată numai dacă *comanda1* are valoarea de retur 0;
- **comanda1 || comanda2** - *comanda2* este executată numai dacă *comanda1* are valoarea de retur diferită de 0.

Înlănțuirea comenzilor. Înlănțuirea comenzilor se realizează folosind operatorul | (**pipe**) . În această situație ieșirea unei comenzi devine intrare pentru următoarea. Exemplu:

```
cat /var/log/*.log |wc -l
```

Utilizarea ghilimelelor. În programarea de tip **shell bash** există trei tipuri de ghilimele:

- "*ghilimele duble*" - tot ce se află între ghilimelele duble se consideră o înșiruire de caractere, cu excepția caracterelor \ și \$.

Exemplu:

- \$ echo -e "4 + \n5" - afișează 4 + și 5 pe următoarea linie.
- '*ghilimele simple*' – nu va fi prelucrat caracterul \$.

Exemplu:

```
n=3
```

\$ echo -e '4 + \$n\n5' - va afișa 4+\$n pe o linie și 5 pe linia următoare.

- '*ghilimele înclinate*' - sunt folosite pentru a executa comanda aflată între acestea. Trebuie menționat că folosirea ghilimelelor înclinate are ca efect execuția comenzii **shell** aflate între acestea și folosirea în script a rezultatului execuției.

Exemplu:

```
$ echo "Astazi este `date`"
```

Se observă că este afișată data de astăzi.

Citarea. Fiecare dintre caracterele \$, `, ", \, |, &, ;, (), < >, spațiu, **tab** are o semnificație specifică pentru **shell** și trebuie citate ca să fie luate în sensul lor literal. **Citarea (în unele surse - escaparea) este folosită pentru a dezactiva tratarea specială a acestor caractere.**

Există trei metode de citare: caracterul **Backslash** (\), ghilimele simple (apostrofe, ') și ghilimele duble (ghilimele propriu-zise, ").

Backslash-ul (\) păstrează valoarea literală a următorului caracter.

Includerea caracterelor între apostrofe păstrează valoarea literală a fiecărui caracter dintre cele incluse. Între apostrofe nu poate să apară un apostrof, nici măcar precedat de **backslash**.

Includerea caracterelor între ghilimele duble păstrează valoarea literală a tuturor caracterelor dintre ghilimele, cu excepțiile: \$, `, și \. Caracterele \$ și ` își păstrează semnificația specială și între ghilimele duble. **Backslash-ul** (\) își păstrează semnificația specială numai atunci când e urmat de unul din următoarele caractere: \$, `, ", \, sau punct (.). Ghilimele duble (") pot fi citate între ghilimele duble numai dacă sunt precedate de \.

Redirecționarea intrărilor și ieșirilor standard. În cea mai mare parte comenzile au ieșirile spre monitor iar intrările sunt luate de la tastatură (**I/O** standard), dar în **Linux**, ca și în alte sisteme de operare, intrările/ ieșirile pot fi redirecționate către fișiere.

Există trei simboluri de redirecționare:

>, >> și <

Pot fi folosite mai multe simboluri de redirecționare în aceeași linie de comandă, la fel ca în exemplul următor:

```
$ cat > lista
ubuntu
```

```
linux
mint
debian
$ sort < lista > lista_sortata
$ cat lista_sortata
debian
linux
mint
ubuntu
```

În exemplul anterior comanda **sort** a preluat datele de intrare din fișierul *lista*, iar rezultatul a fost redirecționat către fișierul *lista_sortata*. În acest caz, fișierul sursă trebuie să fie diferit de fișierul destinație pentru a evita pierderea conținutului fișierului.

Variabile. Ca orice limbaj, **shell-ul** permite utilizarea de variabile. Spre deosebire de limbajele cunoscute, variabilele **shell** nu au tipuri. O variabilă poate fi evaluată atât ca număr cât și ca șir de caractere.

Exemple de definire de variabile:

```
var1=2
var2=4asa
var3='abcd'
my_var="asdf"
my_other_var="a 1 3 4"
new_var=$var1
new_var2=${var2}var3
```


Sintaxa `shell` nu permite spații între numele variabilei și caracterul `=` sau între caracterul `=` și valoarea variabilei.

Se observă că valoarea unei variabile este referită prin folosirea simbolului `$`.

Exemple de folosire de variabile:

```
$ echo $var1
2
$ echo $var12

$ echo ${var1}2
22
$ echo "$var1"
2
$ echo "$var1"2
22
$ echo $var1$my_other_var
2a 1 3 4
$ echo "$var1 $my_other_var"
2 a 1 3 4
```

Operații cu variabile. Asupra variabilelor se pot efectua operații aritmetice (folosind operatori matematici) și operații logice (folosind operatori logici)

Operatorii matematici uzuali:

- `+` - adunarea;
- `-` - scăderea;

- / - împărțirea;
- % - restul împărțirii (modulo) ;
- * - înmulțirea;
- ** - ridicarea la putere.

Pentru a evalua o expresie se folosesc parantezele pătrate:

[expresie]

Returnează starea 0 ("true") sau 1 ("false") în funcție de rezultatul evaluării expresiei condiționale "expr". Expresiile pot fi unare sau binare.

Evaluarea aritmetică. Expandarea aritmetica permite evaluarea unei expresii aritmetice și substituirea ei cu rezultatul.

Există două formate pentru expandarea aritmetica:

\$(expresie) ;

\$((expresie)).

Expresia e tratată ca și cum ar fi între ghilimele înclinate, dar ghilimelele aflate între paranteze nu vor fi tratate special. Toate elementele din expresie suportă expandarea de parametri, substituirea de comenzi și înlăturarea semnelor de citare. Substituirile aritmetice pot fi cuprinse unele în altele.

Exemplu:

f=2

echo \$[f+2] afișează 4

Un caz particular de evaluare, îl constituie comanda **let** a cărei sintaxă este:

let arg [arg ...]

Fiecare "arg" este o expresie aritmetică, ce trebuie evaluată.

Exemplu:

```
Let a=2\*2
```

Operatori aritmetici folosiți în evaluarea expresiilor:

- `-eq` (este egal cu);
- `-ne` (nu este egal cu);
- `-lt` (mai mic decat);
- `-le` (mai mic sau egal);
- `-gt` (mai mare decat);
- `-ge` (mai mare sau egal).

Comanda `test`. Această comandă este folosită în `shell` pentru a evalua expresii condiționate. Se folosește pentru a evalua o condiție care este folosită la luarea unei decizii sau ca condiție pentru terminarea sau continuarea iterațiilor. Are următoarea formă:

```
test expresie sau [expresie] -> întoarce True sau False
```

Câțiva operatori predefiniți pot fi folosiți cu această comandă. Sunt 4 grupuri de operatori: pentru întregi, pentru șiruri de caractere, pentru fișiere și operatori logici.

Operatori pentru întregi:

- `int1 -eq int2` - întoarce True dacă `int1=int2`;
- `int1 -ge int2` - întoarce True dacă `int1>=int2`;
- `int1 -gt int2` - întoarce True dacă `int1>int2`;

- `int1 -le int2` - întoarce `True` dacă `int1 <= int2`;
- `int1 -lt int2` - întoarce `True` dacă `int1 < int2`;
- `int1 -ne int2` - întoarce `True` dacă `int1` diferit de `int2`.

Operatori pentru șiruri de caractere:

- `str1==str2` - întoarce `True` dacă `str1==str2`;
- `str1!=str2` - întoarce `True` dacă `str1` diferit de `str2`;
- `str` - întoarce `True` dacă `str` nu este vid;
- `-n str` - întoarce `True` dacă lungimea șirului este mai mare ca 0;
- `-z str` - întoarce `True` dacă șirul are lungimea 0.

Exemplu: [`_"$a" _` = `_"$b" _`] - se compară valorile *a* și *b*, simbolul „`_`” semnifică spațiu (blank space).

Operatori pentru fișiere:

- `-d nume_fișier` - întoarce `True` dacă fișierul este director;
- `-f nume_fișier` - întoarce `True` dacă fișierul este fișier obișnuit;
- `-r nume_fișier` - întoarce `True` dacă fișierul poate fi citit de către proces;
- `-s nume_fișier` - întoarce `True` dacă fișierul are o lungime diferită de 0;
- `-w nume_fișier` - întoarce `True` dacă fișierul poate fie scris de către proces;
- `-x nume_fișier` - întoarce `True` dacă fișierul este executabil.

Operatori logici:

- !expresie - întoarce True dacă expresia nu este adevărată;
- expresie1 -a expresie2 - întoarce True dacă cele două expresii sunt ambele adevărate (și);
- expresie1 -o expresie2 - întoarce True dacă expresia 1 sau expresia 2 sunt adevărate (sau).

Introducerea de la tastatură, comanda *read*. Aceasta comandă este folosită pentru preluarea datelor de la un utilizator, prin intermediul tastaturii (delimitarea spațiu sau tab, după introducerea ultimului caracter - enter) și memorarea datelor în variabile.

Sintaxa:

read variabila1 variabila2 ... variabilaN

Exemplu: Următorul script îi cere utilizatorului numele și apoi așteaptă introducerea acestuia de la tastatură. Utilizatorul introduce numele la tastatură (după scrierea numelui, clic ENTER) și numele introdus prin intermediul tastaturii, este memorat în variabila `fuser` (exemplul următor) și apoi afișează calendarul lunii curente.

```
$nano script.sh
#
#script pentru citire de la tastatura
#
echo "Numele dumneavoastra va rog:"
read fuser
echo "Buna $fuser, calendarul lunii `cal`"
```

Parametrii poziționali. Când rulați un program script care suportă sau are nevoie de un număr de opțiuni, fiecare din aceste opțiuni este stocată într-un parametru pozițional. Primul parametru este stocat într-o variabilă cu numărul "1", al doilea - "2" etc. Aceste variabile sunt rezervate de către `shell` și nu pot fi definite de utilizator. Pentru accesarea acestor variabile, se utilizează semnul

"\$" (\$1, \$2, \$3, ...,), la fel ca variabilele definite de utilizator. Argumentele care urmează după \$9 trebuie să fie incluse în paranteze acolade (\${10}, \${11}, \${12}, ...) .

Alte variabile importante:

- \$# - conține numărul de opțiuni transmise programului în linia de comandă;
- \$? - conține valoarea de ieșire a ultimei comenzi executate (orice program executat întoarce o valoare; de obicei valoarea 0 înseamnă terminare normală a programului);
- \$0 - conține numele script-ului (numele programului);
- \$* - conține toate argumentele care au fost transmise;
- "\$@" - conține toate argumentele care au fost transmise programului în linia de comandă, fiecare între "".

Structuri de control pentru script-uri. În cadrul programării shell deosebim 2 tipuri de structuri principale:

- structuri de decizie: `if`, `select` și `case`
- structuri iterative (de buclă): `while`, `until` și `for`.

Structura **if** – execută comenzi și în funcție de rezultatul acestora se execută una dintre ramurile de declarații. Sintaxa:

```
if <lista_comenzi_1>
then
<lista_comenzi_2>
[ else
<lista_comenzi_3> ]
fi
```

Semantica: Se execută comenzile din <lista_comenzi_1> și dacă codul de retur al ultimei comenzi din ea este 0 (i.e. terminare cu succes), atunci se execută comenzile din <lista_comenzi_2>, iar altfel (i.e. codul de retur este diferit de 0) se execută comenzile din <lista_comenzi_3>.

Observații:

- Deseori `<lista_comenzi_1>` poate fi comanda de testare a unei condiții.
- Ramura `else` este opțională.
- Structura `if` are și o formă sintactică imbricată.

```
if <lista_comenzi_1>
then
<lista_comenzi_2>
elif <lista_comenzi_3>
then
<lista_comenzi_4>
elif <lista_comenzi_5>
.....
else
<lista_comenzi_N>
fi
```

Structura **case** are sintaxa:

```
case <expr> in
<sir_valori_1> ) <lista_comenzi_1> ;;
<sir_valori_2> ) <lista_comenzi_2> ;;
.....
<sir_valori_N-1> ) <lista_comenzi_N-1> ;;
<sir_valori_N> ) <lista_comenzi_N>
esac
```

Semantica: Dacă valoarea `<expr>` se găsește în lista de valori `<sir_valori_1>`, atunci se execută `<lista_comenzi_1>` și apoi execuția lui `case` se termină. Altfel: dacă valoarea `<expr>` se găsește în lista de valori `<sir_valori_2>`, atunci se execută `<lista_comenzi_2>` și apoi execuția lui `case` se termină. Altfel: ... s.a.m.d.

Structura iterativă **while** are sintaxa:

```
while <lista_comenzi_1>
do
<lista_comenzi_2>
done
```

Semantica: Se execută comenzile din <lista_comenzi_1> și dacă codul de retur al ultimei comenzi din ea este 0 (i.e. terminarea cu succes), atunci se execută comenzile din <lista_comenzi_2> și se reia bucla. Altfel, se termină execuția buclei *while*.

Observații:

- Deci bucla *while* se execută iterativ atât timp cât codul returnat de <lista_comenzi_1> este 0 (succes).
- Deseori <lista_comenzi_1> poate fi comanda:

test <argumente>

sau, echivalent:

[<argumente>]

care este o comandă ce exprimă testarea unei condiții, după cum vom vedea mai jos.

Exemplu:

```
while true
do
date;
sleep 60;
done
```

Efect: se afișează încontinuu pe ecran, din minut în minut, data și ora curentă.

Structura iterativă **until** are sintaxa:

```
until <lista_comenzi_1>
do
<lista_comenzi_2>
done
```

Semantica: Se execută comenzile din <lista_comenzi_1> și dacă codul de retur al ultimei comenzi din ea este diferit de 0 (i.e. terminare cu eroare), atunci se execută comenzile din <lista_comenzi_2> și se reia bucla. Altfel, se termină execuția buclei *until*.

Structura iterativă **for** are sintaxa:

```
for <var> [ in <text> ]
do
<lista_comenzi>
done
```

Semantica: <text> descrie o listă de valori pe care le ia succesiv variabila <var>; pentru fiecare asemenea valoare a lui <var>, se execută comenzile din <lista_comenzi>.

Observatii:

- Se poate folosi în corpul buclei *for* valoarea variabilei <var>.
- Toate instrucțiunile, exceptând *do* și *done*, trebuie să fie urmate de caracterul ";" sau caracterul *newline*.
- Dacă lipsește partea opțională "in <text>", atunci ca valori pentru <var> se folosesc argumentele din \$* (i.e. parametrii din linia de comandă).

Exemplu:

```
for i in `ls -t`  
do  
echo $i  
done  
sau:  
for i in `ls -t` ; do echo $i ; done
```

Efect: același cu comanda `ls -t` (se afișează conținutul directorului curent, sortat după dată).

Alte comenzi utile ce pot apărea în script-uri:

- comanda **break**, cu sintaxa:

break [*n*]

unde *n* este 1 în cazul când lipsește.

Efect: se iese din *n* bucle *do-done* imbricate, execuția continuând cu următoarea comandă de după *done*.

- comanda **continue**, cu sintaxa:

continue [*n*]

unde *n* este 1 în cazul când lipsește.

Efect: pentru *n*=1 se reîncepe bucla curentă *do-done* (de la pasul de reinițializare), respectiv pentru *n*>1 efectul este ca și cum se execută de *n* ori comanda *continue* 1.

- comanda **exit**, cu sintaxa:

exit *cod*

Efect: se termină (se oprește) execuția script-ului în care apare și se întoarce drept cod de retur valoarea specificată.

3.3 Sarcină pentru lucrarea de laborator

Executați următorii pași:

- a) Creați catalogul propriu în directorul `/home/user/`. *user* în cazul dat este numele utilizatorului. Toate scripturile și fișierele pentru extragerea rezultatelor le veți crea în acest catalog (`mkdir lab3`).
- b) Scrieți script-uri, care vor rezolva următoarele probleme:
 - 1) De la tastatură, ca parametri, script-ului se transmit două șiruri de caractere. Afișați un mesaj despre egalitatea sau inegalitatea șirurilor introduse.
 - 2) De la tastatură, ca parametri, script-ului se transmit trei numere întregi. Afișați cel mai mare număr.
 - 3) De la tastatură introduceți caractere până când este introdus caracterul "q". Imprimați șirul introdus pe o singură linie.
 - 4) Introduceți numere întregi de la tastatură, ultimul caracter introdus - un număr par. Afișați numărul de cifre introduse.
 - 5) Creați un meniu textual din patru elemente. La introducerea numărului elementului din meniu, va fi lansat editorul *nano*, editorul *xed*, browser-ul *Firefox*, sau ieșirea din meniu.
 - 6) Dacă scriptul este lansat din directorul *Home*, afișați calea spre directorul *Home* și ieșiți cu codul 0. În caz contrar, ieșiți cu codul 1 și afișați un mesaj de eroare.
- c) Prezentați profesorului script-urile și primiți întrebări sau sarcină pentru susținerea lucrării de laborator. Baremul – rezolvarea 1, 2 – nota 6; 1-4 – nota 8; 1-6 nota 10.

3.4 Sarcină pentru lucrul neauditorial

Subiecte de pregătire către lucrarea de laborator 4: Explicați noțiunile - *stdin*, *stdout*, *stderr*. Instrucțiunile - *sort*, *uniq*, *cut*, *head*, *tail*, *tr*. Destinația utilităților *grep*, *sed*, *awk*. Clase de caractere POSIX.

3.5 Link-uri utile

<http://www.linuxcommand.org/index.php>.

<http://tille.garrels.be/training/tldp/ITL-Romanian.pdf>

Задания к лабораторной работе №3

Создайте свой собственный каталог в каталоге **/home/user/**. **user** в данном случае является именем пользователя. Все скрипты и файлы для хранения результатов будут созданы в этом каталоге (`mkdir lab3`).

Напишите скрипты, для решения следующих задач:

- 1) В параметрах скрипта передаются две строки. Вывести сообщение о равенстве или неравенстве переданных строк.
- 2) В параметрах при запуске скрипта передаются три целых числа. Вывести максимальное из них.
- 3) Считывать символы с клавиатуры, пока не будет введен символ "v". После этого вывести последовательность считанных символов в виде одной строки.
- 4) Считывать с клавиатуры целые числа, пока не будет введено четное число. После этого вывести количество считанных чисел.
- 5) Создать текстовое меню с четырьмя пунктами. При вводе пользователем номера пункта меню происходит запуск редактора **nano**, редактора **xed**, браузера **Firefox** или выход из меню.

- 6) Если скрипт запущен из домашнего директория, вывести на экран путь к домашнему директорию и выйти с кодом 0. В противном случае вывести сообщение об ошибке и выйти с кодом 1.

Предъявите скрипты преподавателю и получите вопросы или задание для защиты лабораторной работы. Выполненная лабораторная работа будет оценена следующим образом: выполнение п.1,2 – 6 баллов, 1-4 – 8, 1-6 – 10.

Задачи для неаудиторной работы

Вопросы для подготовки студентов к лабораторной работе 4: Объясните понятия: *stdin*, *stdout*, *stderr*. Команды - *sort*, *uniq*, *cut*, *head*, *tail*, *tr*. Назначение команд *grep*, *sed*, *awk*. Классы символов **POSIX**.

4 Procesarea fluxurilor textuale în SO GNU/Linux

4.1 Scopul lucrării: studierea procesului, intrarea și ieșirea standard a procesului, legarea proceselor de I/O, utilizarea ieșirii unui proces ca parametru al unui alt proces, expresii regulate și filtrarea fluxurilor textuale.

4.2 Indicații metodice

Interfața principală în SO **GNU/Linux** este interfața de tip consolă cu introducerea și extragerea textuală a datelor. Aceasta determină abordarea administrării obiectelor sistemului de operare în reprezentarea lor textuală. De exemplu, starea proceselor este reflectată sub forma unui set de fișiere textuale în **/proc**, informația despre evenimentele care au loc în sistem este păstrată în fișierele textuale ale jurnalelor, setările unor pachete sunt păstrate în fișiere textuale de configurare. Din această cauză pentru soluționarea problemelor, legate de administrarea sistemului de operare, este necesar să cunoaștem modalitatea de lucru cu fluxurile textuale.

Controlul (administrarea) intrărilor-ieșirilor proceselor (comenzilor). Implicit, fiecare proces are deschise trei fișiere (trei fluxuri standard) – **stdin** (intrarea standard, tastatura), **stdout** (ieșirea standard, ecranul) și **stderr** (ieșirea standard (la ecran) pentru mesajele de eroare). Aceste trei fluxuri standard (ca și oricare alte fișiere deschise) pot fi redirecționate către alte dispozitive (fișiere), altele decât ecranul sau tastatura. Termenul “redirecționare” semnifică în acest caz: ieșirea fișierului (comenzii, programului) să fie transmisă la intrarea altui fișier (comandă, program). Pentru aceasta este utilizat numărul descriptorului standard.

Descriptorii fișierelor deschise implicit sunt prezentate în figura 4.1.

0 = **stdin**
 1 = **stdout**
 2 = **stderr**

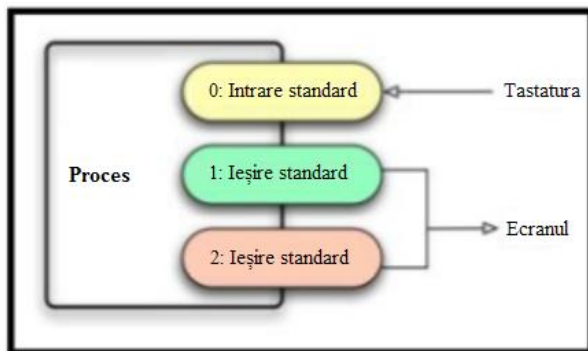


Figura 4.1 - Descriptorii fișierelor

Redirecționarea ieșirii standard. Rezultatul execuției unei comenzi este afișat de **bash** la consola de ieșire (implicit, la ecran). Această ieșire poate fi redirecționată către un fișier folosind semnul >.

Comanda > fișier – redirecționarea ieșirii standard către un fișier, conținutul vechi al fișierului este șters.

Concatenare. Putem adăuga ieșirea unui proces la sfârșitul unui fișier, care va fi creat dacă acesta nu exista anterior, folosind semnul >>.

Comanda >> fișier – redirecționarea ieșirii standard către un fișier, fluxul textual se va adăuga la sfârșitul fișierului.

Alte redirecționări:

- **comanda1|comanda2** – redirecționarea ieșirii standard a primei comenzi către intrarea standard a comenzii a doua, ceea ce permite crearea unei benzi de comenzi.
- **comanda1\$(comanda2)** – transmiterea ieșirii comenzii 2 ca parametru al comenzii 1. În cadrul unui script construcția **\$(comanda2)** poate fi utilizată, de exemplu, pentru

transmiterea rezultatelor obținute de comanda2 ca parametri pentru ciclul for ... în.

Operații cu șiruri de caractere (comenzi interne bash):

- **`${#string}`** – returnează lungimea șirului (**`string`** – nume variabilă);
- **`${string:position:length}`** – extrage **`$length`** simboluri din **`$string`**, începând cu poziția **`$position`**. Caz particular: **`${string:position}`** extrage un subșir din **`$string`**, începând cu poziția **`$position`**.
- **`${string#substring}`** – elimină subșirul **`$substring`** cel mai mic din cele depistate în șirul **`$string`**. Căutarea are loc de la începutul șirului. **`$substring`** este o expresie regulată.
- **`${string##substring}`** – elimină subșirul **`$substring`** cel mai mare din cele depistate în șirul **`$string`**. Căutarea are loc de la începutul șirului. **`$substring`** este o expresie regulată.
- **`${string/substring/replacement}`** – înlocuiește prima intrare **`$substring`** cu șirul **`$replacement`**. **`$substring`** – este o expresie regulată.
- **`${string//substring/replacement}`** – înlocuiește toate intrările **`$substring`** cu șirul **`$replacement`**. **`$substring`** este o expresie regulată.

Operații cu șiruri de caractere (comenzi externe). Anumite comenzi au nevoie neapărat de argumente, aceste argumente fiind opționale în cazul altor comenzi. *Puteți verifica dacă o comandă suportă opțiuni și argumente și care dintre ele sunt valide, consultând paginile **man**, **info**.*

Comenzi utile:

- Comanda **sort** – sortează textul în ordine de creștere sau descreștere, în dependență de opțiuni.
- Comanda **uniq** – elimină șirurile care se repetă dintr-un fișier care a fost sortat.
- Comanda **cut** – extrage unele câmpuri din fișierele textuale (prin câmp înțelegem secvență de simboluri până la spațiu).
- Comanda **head** – extrage liniile de început din fișier la **stdout**.
- Comanda **tail** – extrage liniile de sfârșit din fișier la **stdout**.
- Comanda **wc** – calculează cantitatea de cuvinte/linii/simboluri din fișier sau flux.
- Comanda **tr** – înlocuiește unele simboluri cu altele.

Utilitarele *grep*, *sed*, *awk*. Comanda **grep** – utilitară pentru căutarea și sortarea liniilor introduse și afișarea anumitor tipare (șiruri de caractere) ca rezultat al procesării:

grep pattern [file...] – caută fragmente de text în fișier/fișiere, care corespund șablonului **pattern**, unde **pattern** poate fi linie simplă sau expresie regulată.

Comanda **Sed** – “editor de flux” neinteractiv. Preia textul de la dispozitivul **stdin** sau dintr-un fișier text, execută unele operații asupra liniilor după care afișează rezultatul pe **stdout** sau într-un fișier. **Sed** determină, reieșind din spațiul de adrese dat, liniile asupra cărora vor fi executate operațiile. Spațiul de adresă al liniilor este desemnat de numărul de ordine al acestora sau de șablon. De exemplu, comanda **3d** va obliga **sed** să elimine linia a treia, iar instrucțiunea **/windows/d** va conduce la eliminarea tuturor liniilor,

care conțin cuvântul "**windows**". Cel mai frecvent sunt utilizate comenzile: **p** – imprimare (pe **stdout**), **d** – eliminare/ștergere și **s** – înlocuire.

Comanda **awk** – utilitară pentru căutarea contextuală și transformare a textului, instrument pentru extragerea și/sau procesarea câmpurilor (coloanelor) în fișiere text structurate. **Awk** împarte fiecare linie în câmpuri individuale. În mod implicit, câmpurile sunt secvențe de caractere separate prin spații, dar există posibilitatea de a desemna alt caracter ca separator de câmp. **Awk** analizează și procesează fiecare câmp aparte.

Expresii regulate – set de simboluri și/sau meta simboluri, înzestrate cu proprietăți speciale.

Expresiile regulate au rolul de a specifica mulțimi de șiruri de caractere. Ele stabilesc un șablon pentru potriviri de text. Sunt utilizate în general pentru a identifica dintr-un text anumite șiruri de caractere.

În general orice caracter ce apare într-o expresie regulată identifică un caracter din text. Există însă anumite caractere speciale care permit definirea de șabloane variabile.

În tabelul 4.1 sunt prezentate denumirile unor caractere speciale.

Tabelul 4.1 – Caractere speciale

Caracter	Nume
#	Hash
&	Ampersand
*	Asrerisk
/	Slash
\	Backslash

@	At sign
^	Caret
`	Grave accent (backtick)
~	Tilde
	Vertical bar

În expresiile regulate se utilizează două tipuri de caractere:

- caractere alfanumerice, care desemnează exact caracterele reprezentate;
- caractere speciale sau operatori care desemnează repetiții, alegeri, opțiuni, etc.

Semnificația caracterelor speciale:

- a) \ -caracter de citare - anulează semnificația specială a caracterului care îl urmează;
- b) . -caracter arbitrar (orice caracter);
- c) Operatori de repetiție - *, +, ? :
 - 1) * - de 0 sau mai multe ori;
 - 2) + - de 1 sau mai multe ori;
 - 3) ? - de 0 sau 1 ori.
- d) \$ - indică sfârșitul unei linii;
- e) ^ - indică începutul unei linii.. Expresia "^\$" corespunde liniei vide;
- f) [s] - desemnează o mulțime de caractere. În mulțimea s singurele caractere speciale sunt: -,] și ^. De exemplu:
 - 1) [abc] -unul dintre caracterele a, b sau c;
 - 2) [A-Za-z] -o litera;

- 3) `[]` -unul dintre caracterele '[' sau '[';
- 4) `[...]` -orice caracter din ... ; sunt admise diapazoane de tipul a-z, a-z0-9;
- 5) `[^...]` -orice caracter în afară de ... ; sunt admise diapazoane.

Pentru expresii regulate utilitarele `awk` și `egrep` acceptă construcții de forma:

- `\(expr\)*` - zero sau mai multe apariții ale expresiei regulate;
- `\(expr\)+` - una sau mai multe apariții ale expresiei regulate;
- `\(expr\)?` - zero sau o apariție a expresiei regulate.

Două expresii regulate separate prin '|' semnifică una dintre ele.
De exemplu:

`(ab|cd)?(ef)*`

Pe prima poziție apare sau un **ab** sau **c** urmat de cel puțin un **d**, și urmat de oricâte grupuri de **ef**.

Expresiile regulate între paranteze simple realizează o grupare.
De exemplu:

`(a|b)(c|d)` identifică **ac**, **ad**, **bc** sau **bd**.

"Toate caracterele situate între" își pierd semnificația specială.

Caracterul `/` indică contextul următor. De exemplu:

`ab/cd` - ab doar dacă este urmat de cd.

Caracterele `{}` semnifică repetiții ale unei expresii. De exemplu:

- `a\{1,5\}` - una până la cinci apariții ale lui a;
- `exp\{6,\}` - cel puțin 6 apariții ale expresiei exp;
- `exp\{2\}` - exact două apariții ale expresiei exp;

- `[0-9]\{2,\}` - secvență din două sau mai multe cifre.

Dacă mai multe caractere speciale apar într-o expresie regulată, se aplică regula de precedență. Conform acesteea, precedența cea mai mare o are `[]` după care grupul `*`, `+` și `?` și ultimul este `|`.

La înlocuirea șirurilor de caractere specificate prin expresii regulate caracterul `&` identifică expresia regulată. Pentru a genera caracterul `&` acesta trebuie prefixat de `\`.

Caracterul `\` urmat de `<Enter>` generează o linie nouă.

Clase de caractere **POSIX** :

- `[:class:]` metodă alternativă de a specifica un diapazon de caractere.
- `[:alnum:]` corespunde caracterelor alfanumerice. Este echivalentă expresiei `[A-Za-z0-9]`.
- `[:alpha:]` corespunde caracterelor alfabetului. Este echivalentă expresiei `[A-Za-z]`.
- `[:blank:]` corespunde caracterului spațiu sau tabulare.
- `[:cntrl:]` corespunde caracterelor de control.
- `[:digit:]` corespunde setului de cifre zecimale. Este echivalentă expresiei `[0-9]`.
- `[:lower:]` corespunde caracterelor alfabetului din registrul inferior. Echivalentă expresiei `[a-z]`.
- `[:space:]` corespunde caracterelor de spațiere (spațiu și tabulare orizontală).
- `[:graf:]` corespunde unui set de caractere din gama ASCII 33 - 126.
- `[:print:]` corespunde unui set de caractere din gama ASCII 32 - 126.

- [:**upper**:] corespunde caracterelor alfabetului din registrul superior. Echivalentă expresiei [**A-Z**].
- [:**xdigit**:] corespunde setului de cifre hexazecimale. Este echivalentă expresiei [**0-9A-Fa-f**].

În tabelul 4.1 sunt prezentate câteva exemple de opțiuni *sed*.

Tabelul 4.1 – Exemple de opțiuni **sed**

Opțiunea	Descrierea
8d	Elimină (șterge) linia 8
/^\$/d	Elimină toate liniile vide
1,/^\$/d	Elimină toate liniile până la prima linie vidă, inclusiv
/Jones/p	Afișarea liniilor ce conțin expresia „Jones“ (cu parametrul -n)
s/Windows/Linux/	În fiecare rând, se înlocuiește prima apariție a cuvântului „Windows“ cu cuvântul „Linux“
s/BSOD/stability/g	În fiecare rând, se înlocuiesc toate cuvintele „BSOD“ pe „stability“
s/ *\$//	Elimină toate spațiile la sfârșitul fiecărei linii
s/00*/0/g	Se înlocuiesc toate secvențele de zerouri cu un singur "0"

<code>/GUI/d</code>	Elimină toate liniile care conțin "GUI"
<code>s/GUI//g</code>	Elimină toate expresiile "GUI" găsite, lăsând restul liniei neschimbat

Exemple cu expresii regulate:

```
grep -i '^..j.r$' /usr/share/dict/british-english
```

Expresia regulată `'^..j.r$'` identifică un șir (cuvânt) care:

- constă din 5 litere (între `^` și `$`);
- a treia literă este `j` și ultima literă este `r`.

```
grep '[ -]F[a-z]\+[^\.]' lista.txt
```

Expresia regulată `'[-]F[a-z]\+[^\.]'` identifică un șir care:

- începe cu unul dintre caracterele spațiu (blank) sau minus (-);
- continuă cu litera F (majusculă);
- continuă cu litere mici (folosind setul `[a-z]`);
- literele mici de oricâte ori, cel puțin o dată (folosind caracterul plus (+));
- continuă cu orice caracter diferit de punct (., dot), folosind expresia `[^\.]`. Folosim backslash (\) pentru citarea caracterului punct (., dot), altfel ar fi însemnat orice caracter.

4.3 Sarcină pentru lucrarea de laborator

Executați următorii pași:

- a) Creați un nou subdirector unde veți plasa toate scripturile și fișierele pentru extragerea rezultatelor.
- b) Scrieți script-uri, care vor rezolva următoarele probleme:
 - 1) Creați fișierul **errors.log**, în care veți plasa toate liniile din toate fișierele directorului **/var/log/** accesibile pentru citire, care conțin secvența de caractere **ACPI**. Afișați pe ecran liniile din **errors.log**, care conțin numele fișierelor, excluzând calea acestor fișiere.
 - 2) Calculați numărul total de linii în fișierele directorului **/var/log/** cu extensia **log**. Afișați acest număr pe ecran.
 - 3) Creați fișierul **full.log**, în care veți plasa liniile din fișierul **/var/log/Xorg.0.log**, care conțin avertizări și mesaje informaționale, înlocuind marcherii avertizărilor (**WW**) și mesajelor informaționale (**II**) cu cuvintele **Warning:** și **Information:**, astfel încât în fișierul rezultat mai întâi să apară toate mesajele informaționale, iar apoi toate avertizările. Afișați conținutul fișierului **full.log** pe ecran.
 - 4) Creați fișierul **emails.lst**, în care veți plasa toate adresele de poștă electronică, depistate în fișierele directorului **/etc**. Adresele vor urma consecutiv, separate prin virgulă. Afișați conținutul fișierului **emails.lst** pe ecran.
 - 5) Găsiți în directorul **/bin** toate fișierele, ce reprezintă script-uri și afișați pe ecran numele interpretoarelor cu numărul ce indică frecvența lor de utilizare (în script-uri). Script-urile pot începe cu una din următoarele linii:

```
#!/bin/sh
#!/bin/bash
#!/usr/bin/perl
#!/usr/bin/tcl.
```


- 6) Afișați lista utilizatorilor sistemului cu indicarea **UID** al fiecăruia, cu sortarea după **UID**. Informațiile despre utilizatori sunt păstrate în fișierul **/etc/passwd**. În fiecare linie a acestui fișier primul câmp conține numele utilizatorului, iar câmpul al treilea - **UID**, separatorul – două puncte (:).
- 7) Afișați 5 cel mai frecvent întâlnite cuvinte din **man bash** cu lungimea de cel puțin 3 caractere.
- c) Prezentați profesorului script-urile și primiți întrebări sau sarcină pentru susținerea lucrării de laborator. Baremul – rezolvarea 1, 2 – nota 6; 1-4 –nota 8; 1-7 - nota 10.

4.4 Sarcină pentru lucrul neauditorial

Subiecte de pregătire către lucrarea de laborator 5: Explicați noțiunea de *proces*, *PID*, instrucțiunile – *ps*, *pstree*, *top*.

4.5 Link-uri utile

<http://ac.upg-ploiesti.ro/cursuri/so/laborator/lab04.html>

<http://ac.upg-ploiesti.ro/cursuri/so/laborator/lab05.html#GREP>

Задания к лабораторной работе №4

Все скрипты и файлы для хранения результатов будут созданы в каталоге Lab4.

Напишите скрипты, для решения следующих задач:

- 1) Создайте файл **errors.log**, в котором сохраните все строки из всех файлов каталога **/var/log/** доступные для чтения, которые содержат последовательности символов ACPI. Выведите на монитор строки из

errors.log и удалите путь к файлу, оставив имя файла.

- 2) Рассчитать общее количество строк в файлах каталога **/var/log/** с расширением **log** и выведите это число на экран.
- 3) Создайте файл **full.log**, в котором будут сохранены строки из файла **/var/log/Xorg.0.log**, содержащие предупреждения и информационные сообщения, заменяя маркеры предупреждений и информационных сообщений словами **Warning:** и **Information:**, так что в результирующем файле сначала появлялись бы все информационные сообщения, а затем все предупреждения. Выведите содержимое файла **full.log** на экран.
- 4) Создайте файл **emails.lst**, в котором будут сохранены все адреса электронной почты, разделенные запятыми, найденные в файлах каталога **/etc** и выведите его содержимое на экран..
- 5) Найдите в каталоге **/bin** все файлы, которые содержат сценарии и отобразите на экране количество использований каждого интерпретатора. Сценарии могут начинаться с одной из следующих строк:

```
#!/bin/sh  
#!/bin/bash  
#!/usr/bin/perl  
#!/usr/bin/tcl
```

- 6) Выведите на экран список пользователей системы с их **UID**, отсортированный по **UID**. Информация о пользователях хранится в файле **/etc/passwd**. В каждой строке этого файла, первое поле содержит имя

пользователя, а третье поле - **UID**. Разделитель – двоеточие (:).

- 7) Отобразите 5 наиболее распространенных слова в **man** для команды **bash** длиной не менее 3 символов.

Все скрипты представьте преподавателю и получите вопросы или задание для защиты лабораторной работы. Выполненная лабораторная работа будет оценена следующим образом: выполнение п.1,2 – 6 баллов, 1-4 – 8, 1-7 – 10.

Задачи для неаудиторной работы

Вопросы для подготовки студентов к лабораторной работе 5: Объясните понятия *процесс*, *PID*, команды – *ps*, *pstree*, *top*.

5 Monitorizarea proceselor

5.1 Scopul lucrării: modurile de obținere de informații despre rularea proceselor, despre resursele utilizate de procese, prezentarea rezultatelor în diverse forme.

5.2 Indicații metodice

Un proces reprezintă un program în execuție și are atașate o serie de informații specifice precum instrucțiunile programului, resurse folosite (precum fișiere deschise), unul sau mai multe fire de execuție și alte informații necesare procesului de execuție în paralel.

Fiecărui proces în Linux îi este asociat un identificator PID (**P**rocess **I**dentifier) format dintr-un număr care ia valori între 0 și 65535.

Procesele în Linux sunt ierarhizate sub forma unui arbore, având ca rădăcină procesul *init*. Părintele - A, al unui proces - B, este procesul ce a creat procesul B. ID-ul procesului părinte este referit din perspectiva procesului copil ca PPID (Parent Process ID). PPID al procesului *init* este 0.

Orice proces Linux va avea un set de caracteristici comune, ce oferă informații despre acesta:

- PID – sau Process ID, este un identificator de proces sub forma unui număr întreg unic.
- PPID – similar cu PID, cu excepția că reprezintă identificatorul procesului care a dat naștere procesului curent (cunoscut și ca proces părinte).
- Terminalul atașat – prescurtat TTY, reprezintă terminalul la care procesul curent este atașat.
- RUID - Real User ID, reprezintă identificatorul utilizatorului care a lansat aplicația. Similar există și EUID (sau Effective User ID) pentru identificarea drepturilor reale la resursele sistemului.

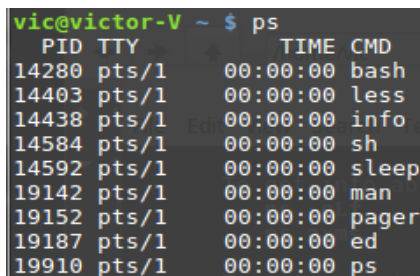
- RGID și EGID – similar cu RUID și EUID, doar că se referă la identificatorul grupului de utilizatori.
- factorul *nice* – folosit pentru a determina, așa cum sugerează și numele, „factorul de prietenie” al procesului cu scopul stabilirii priorității de execuție.

Afișarea informațiilor despre procese. O listare a proceselor poate fi efectuată prin comanda **ps** (ps – process status) :

ps [opțiuni]

Execuția *ps* fără opțiuni (figura 5.1):

\$ ps



```

vic@victor-V ~ $ ps
  PID TTY          TIME CMD
 14280 pts/1    00:00:00 bash
 14403 pts/1    00:00:00 less
 14438 pts/1    00:00:00 info
 14584 pts/1    00:00:00 sh
 14592 pts/1    00:00:00 sleep
 19142 pts/1    00:00:00 man
 19152 pts/1    00:00:00 pager
 19187 pts/1    00:00:00 ed
 19910 pts/1    00:00:00 ps

```

Figura 5.1 – Rularea *ps*

Principalele opțiuni sunt:

- aux - afișează informații despre toate procesele din sistem;
- l - afișează informații despre procese, diferită de –u;
- u - afișează informații despre procese, dar diferită de –l;
- e - afișează informații despre toate procesele din sistem utilizând sintaxa standardă.

Exemplu de lansare a comenzii *ps*:

\$ ps -aux

Aici apare o listă a tuturor proceselor de pe sistemul nostru în momentul de față (figura 5.2).

```

vic@victor-V ~ $ ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.4 185248 4176 ?        Ss   Feb23   0:02 /sbin/init splash
root         2  0.0  0.0      0     0 ?        S    Feb23   0:00 [kthreadd]
root         4  0.0  0.0      0     0 ?        S<   Feb23   0:00 [kworker/0:0H]
root         6  0.0  0.0      0     0 ?        S    Feb23   0:00 [ksoftirqd/0]
root         7  0.0  0.0      0     0 ?        S    Feb23   0:14 [rcu_sched]
root         8  0.0  0.0      0     0 ?        S    Feb23   0:00 [rcu_bh]
root         9  0.0  0.0      0     0 ?        S    Feb23   0:00 [migration/0]
root        10  0.0  0.0      0     0 ?        S<   Feb23   0:00 [lru-add-drain]
root        11  0.0  0.0      0     0 ?        S    Feb23   0:00 [watchdog/0]
root        12  0.0  0.0      0     0 ?        S    Feb23   0:00 [cpuhp/0]
root        13  0.0  0.0      0     0 ?        S    Feb23   0:00 [kdevtmpfs]

```

Figura 5.2

Alt exemplu de apelare a comenzii *ps* (figura 5.3):

```
$ ps -l
```

```

vic@victor-V ~ $ ps -l
F S      UID      PID      PPID      C  PRI  NI  ADDR  SZ  WCHAN      TTY      TIME CMD
0 S      1000    14280    13971    0  80   0   -    5598 wait    pts/1    00:00:00 bash
0 T      1000    14403    14280    0  80   0   -    2414 signal   pts/1    00:00:00 less
0 T      1000    14438    14280    0  80   0   -    2464 signal   pts/1    00:00:00 info
0 T      1000    14584    14280    0  80   0   -    1127 signal   pts/1    00:00:00 sh
0 T      1000    14592    14584    0  80   0   -    1823 signal   pts/1    00:00:00 sleep
0 T      1000    19142    14280    0  80   0   -    4577 signal   pts/1    00:00:00 man
0 T      1000    19152    19142    0  80   0   -    2414 signal   pts/1    00:00:00 pager
0 T      1000    19187    14280    0  80   0   -    1827 signal   pts/1    00:00:00 ed
0 R      1000    20041    14280    0  80   0   -    7229 -        pts/1    00:00:00 ps

```

Figura 5.3

În tabelul 5.1 este prezentată descrierea coloanelor afișate de comanda **ps** cu opțiunile *-aux*, *-l*.

Tabelul 5.1 – Descrierea coloanelor

Antetul coloanei	Descrierea
%CPU	Timpul de procesor, utilizat de proces, %
%MEM	Volumul de memorie utilizat de proces, % de la memoria de sistem
ADDR	Adresa procesului în memorie
C sau CP	Utilizare CPU, %
COMMAND	Denumirea procesului, include argumente

NI	Prioritatea unui proces (nice value), valori între [-20, 19]
F	Flags
PID	Identificatorul procesului
PPID	Identificatorul procesului părinte
PRI	Prioritatea procesului. Un număr mai mare înseamnă o prioritate mai mică
RSS	<u>Resident set size</u> - este o secvență de memorie RAM (non-swapped) folosită de un proces (în kBytes)
S sau STAT	Starea procesului (process status code)
START sau STIME	Data lansării procesului
VSZ	Volumul procesului în memorie (virtuală), în unități de 1024 de octeți, număr întreg zecimal
TIME	Timpul de procesor, cumulat de proces, format „[dd-] hh: mm: ss ”
TT sau TTY	Terminal asociat procesului
UID sau USER	Nume de utilizator, proprietarul procesului
WCHAN	Wait channel - the address of an event on which a particular process is waiting

Unele valori ale parametrilor ce pot apărea în coloana STAT:

- R – procesul se execută la moment;

- D – proces în regim „sleep” (în așteptarea de operațiuni de intrare /ieșire);
- I – procesul este inactiv;
- S - proces întrerupt (așteptare pentru finalizarea unui eveniment);
- s - frecvent utilizat;
- < - prioritate înaltă;
- Z – proces zombi.

Mai multe informații pot fi obținute: **ps --help** sau **man ps**. Comanda **ps** arată doar procesele active la momentul în care ați introdus această comandă, adică este o “fotografie” a proceselor la momentul respectiv.

Comanda **ps tree** afișează toate procesele care rulează sub forma unui arbore (tree). Dintre opțiunile cele mai des folosite:

- n - sortarea proceselor după identificatorul procesului (pid);
- p – afișarea PID, numere zecimale în paranteze, după fiecare nume de proces;
- u - afișarea identificatorului utilizatorului (uid).

Comanda **top** este utilizată pentru monitorizarea proceselor în timp real, actualizând informațiile la fiecare 5 secunde. Informația afișată poate fi controlată de la tastatură. Tastați **h** și primiți un „help”.

Comanda are multe opțiuni, dintre care:

- **d** (delay) - specifică intervalul dintre actualizările ecranului;
- **n** - numărul de iterații;
- **p** (process identifier) - identificatorul procesului, care va fi urmărit.

Cele mai utile comenzi (taste) pentru monitorizarea proceselor sunt:

- Shift + M - sortarea proceselor după volumul de memorie utilizat (câmpul %MEM);
- Shift + P - sortarea proceselor în funcție de timpul de microprocesor utilizat (câmpul %CPU). Este metoda de sortare implicită;
- U - afișează procesele utilizatorului specificat. Comanda vă va cere numele utilizatorului. Fără nume vor fi afișate toate procesele;
- i - afișează doar procesele curente (procesele în care câmpul STAT are valoarea R, Running).

5.3 Informații detaliate despre procesele în rulare

Directorul `/proc` (process) - conține fișiere referitoare la informațiile despre sistem folosite de kernel. Aceste fișiere sunt utilizate ca o interfață pentru structurile de date din kernel. Majoritatea sunt doar pentru citire, dar unele fișiere vă permit să modificați variabilele kernel-ului.

Directorul `/proc` conține următoarele subdirectoare:

a) directoarele de proces au ca nume identificatorul procesului. Fiecare director conține următoarele fișiere:

- 1) `cmdline` - linia de comandă cu care a fost pornit procesul;
- 2) `cwd` - link-ul către directorul procesului curent;
- 3) `cpu` - informații despre utilizarea fiecărui procesor;
- 4) `environ` - conținutul variabilelor de mediu pentru proces;
- 5) `exe` - este un fișier (legătură simbolică), ce conține calea (pathname) comenzii executate;
- 6) `fd` - un subdirector care conține o intrare pentru fiecare fișier deschis de proces în momentul dat. Numele fiecărei intrări este numărul descriptorului de fișier și reprezintă o

legătură simbolică al fișierului real. Așa, 0 - este o intrare standardă, 1- o ieșire standardă, 2 - este o ieșire standardă de eroare, etc.

- 7) `maps` - mapările de memorie pentru executabilele și bibliotecile asociate procesului;

Fișierul `maps` are formatul prezentat în tabelul 5.1.

Tabelul 5.1 – Conținutul fișierului `maps`

address	perms	offset	ev	inode	pathname
00400000-00452000	r-xp	00000000	8:02	173521	/usr/bin/dbus-daemon
00651000-00652000	rw-p	00051000	8:02	173521	/usr/bin/dbus-daemon
00652000-00655000	rw-p	00052000	8:02	173521	/usr/bin/dbus-daemon
00e03000-00e24000	rw-p	00000000	0:00	0	[heap]
00e24000-011f7000	rw-p	00000000	0:00	0	[heap]
...					

Unde,

- `address` - indică spațiul de memorie ocupat de proces;
- `perms` – indică modurile de acces:
 - `r` (read) – citirea;
 - `w` (write) – scrierea;
 - `x` (execute) – executarea;
 - `s` (shared) – partajat;
 - `p` (private) – privat.

- `offset` – indica offset-ul în fișierul care a fost mapat (folosind *mmap*). Dacă fișierul nu a fost mapat, este 0;
 - `device` - indica numărul de dispozitiv, major (major) și minor (minor) (în hex);
 - `inode` - indică un descriptor, 0 - indică că nu există descriptori asociate cu această zonă de memorie;
 - `pathname` - indică calea și numele fișierului;
 - `heap` – indică o mulțime de procese (the process's heap).
- 8) `mem` - memoria alocată procesului;
- 9) `stat` – informații despre starea procesului;
- 10) `statm` – informații despre memoria utilizată, în pagini:
- `size` – volumul total (același ca `VmSize` în `/proc/[pid]/status`);
 - `resident` - resident set size (același ca `VmRSS` în `/proc/[pid]/status`);
 - `share` – paginile partajate;
 - `text` - text (code);
 - `lib` - librăria (library) (neutilizat de la versiunea Linux 2.6; întotdeauna 0) ;
 - `data` - data + stack;
 - `dt` - pagini „murdare” (dirty pages, neutilizat de la versiunea Linux 2.6; întotdeauna 0).

Procesarea datelor. Prelucrarea datelor proceselor se realizează, de regulă, prin scrierea de șiruri de comenzi (`pipe`) pentru procesarea fluxurilor de text și (sau) prin procesarea ciclică a liniilor din fișiere. Se recomandă să aplicați comenzile studiate - *grep*, *sed*, *awk*, *tr*, *sort*, *uniq*, *wc*, *paste*, precum și funcții pentru lucrul cu șiruri de caractere.

Pentru a accesa informații despre memorie folosiți comanda *free*. Comanda *free* - returnează informații despre memoria liberă

și utilizată în sistem, fizică și virtuală (spațiul de swap pe `harddisk`).

5.4 Sarcină pentru lucrarea de laborator

Executați următorii pași:

- a) Toate script-urile și fișierele pentru extragerea rezultatelor le veți crea în subdirectorul `lab5`.
- b) Scrieți script-uri, care vor rezolva următoarele probleme:
 - 1) Aflați numărul de procese inițializate de utilizatorul ***user*** și introduceți într-un fișier perechea „PID: comanda” a proceselor inițializate. Afișați conținutul acestui fișier și numărul de procese.
 - 2) Afișați PID-ul procesului, ultimul lansat (cu timpul de lansare).
 - 3) Introduceți într-un fișier o listă cu PID-urile proceselor, pornite cu comenzile localizate în `/sbin/`. Afișați conținutul acest fișier.
 - 4) Pentru fiecare proces, calculați diferența dintre memoria totală (`statm: size`) și rezidentă (`statm: resident`) ale memoriei de proces (în pagini). Introduceți într-un fișier linii de tip „PID:diferența”, sortate în ordinea descrescătoare a acestor diferențe. Afișați conținutul acestui fișier.
 - 5) Pentru toate procesele înregistrate, la moment, în sistem, introduceți într-un fișier linii de tip

ProcessID=PID:Parent_ProcessID=PPID:

Average_Time=avg_atom

Preluati valorile PPID și PID din fișierele *status*,
valoarea **avg_atom**

(`avg_atom=se.sum_exec_runtime/nr_switches`)

din fișierele ***sched*** care se află în subdirectoarele, cu nume ce corespund **PID** proceselor în directorul **/proc**. Sortați aceste linii conform identificatorilor proceselor părinte. Rezultatul sortării introduceți într-un fișier și afișați-l.

- 6) În fișierul obținut în 5, după fiecare grup de înregistrări cu același identificator al procesului părinte, introduceți o linie de tip

Sum_switches_of_ParentID=N is M, unde **N=PPID**, iar **M** este suma calculată **voluntary_ctxt_switches+nonvoluntary_ctxt_switches** din **status** pentru acest proces. Afișați conținutul acestui fișier.

- c) Prezentați profesorului script-urile și primiți întrebări sau sarcină pentru susținerea lucrării de laborator.

Baremul – rezolvarea 1, 2 – nota 6; 1-4 –nota 8; 1-6 - nota 10.

5.5 Sarcină pentru lucrul neauditorial

Subiecte de pregătire către lucrarea de laborator 6: Explicați comenzile – *kill, killall, pidof, pgrep, pkill, nice, renice, at, tail, sleep, cron, trap*. Explicați notiunea de *handler*.

5.6 Link-uri utile

<https://www.techonthenet.com/linux/commands/>

<https://linux.die.net/man/>

Задания к лабораторной работе №5

Все скрипты и файлы для хранения результатов будут созданы в каталоге Lab5.

Напишите скрипты, для решения следующих задач:

- 1) Посчитать количество процессов, запущенных пользователем **user**, и вывести в файл пары **PID:команда**

- для таких процессов. Вывести на экран содержимое файла и количество процессов.
- 2) Вывести на экран **PID** процесса, запущенного последним (с последним временем запуска).
 - 3) Вывести в файл список **PID** всех процессов, которые были запущены командами, расположенными в **/sbin/**. Вывести на экран содержимое файла.
 - 4) Для каждого процесса посчитать разность между общим размером (statm: size) и резидентной части (statm: resident) памяти процесса (в страницах). Вывести в файл строки вида **PID:разность**, отсортированные по убыванию этой разности. Вывести на экран содержимое файла.
 - 5) Для всех зарегистрированных в данный момент в системе процессов выведите в один файл строки

ProcessID=PID : Parent_ProcessID=PPID :

Average_Time=avg_atom

Значения **PPID** и **PID** возьмите из файлов *status*, значение **avg_atom** (avg_atom=se.sum_exec_runtime / nr_switches) из файлов *sched*, которые находятся в директориях с названиями, соответствующими **PID** процессов в **/proc**. Отсортируйте эти строки по идентификаторам родительских процессов. Содержимое файла выведите на экран.

- 6) В полученном на предыдущем шаге файле после каждой группы записей с одинаковым идентификатором родительского процесса вставить строку вида

Sum_switches_of_ParentID=N is M, где N = PPID,
 а **M** – сума, посчитанная как
voluntary_ctxt_switches +
nonvoluntary_ctxt_switches для данного
 процесса. Содержимое файла выведите на экран.

Предъявите скрипты преподавателю и получите вопрос или задание для защиты лабораторной работы. Выполненная лабораторная работа будет оценена следующим образом: выполнение п.1,2 – 6 баллов, 1-4 – 8, 1-6 – 10.

Задачи для неаудиторной работы

Вопросы для подготовки студентов к лабораторной работе
6: Объясните команды – *kill*, *killall*, *pidof*, *pgrep*, *pkill*, *nice*, *renice*, *at*, *tail*, *sleep*, *cron*, *trap*. Объясните понятие *handler*.

6 Gestionarea proceselor în SO GNU/Linux

6.1 Scopul lucrării: studierea comenzilor destinate gestionării proceselor, programarea timpului de start al proceselor, transferul și gestionarea datelor între procese.

6.2 Indicații metodice

Principalele sarcini ale managementului proceselor în SO GNU/Linux sunt: gestionarea priorităților proceselor; programarea lansării proceselor, organizarea schimbului de date între procese, de exemplu, cu utilizarea semnalelor.

Pentru a automatiza controlul sistemului, administratorii de sistem, creează script-uri de control. Secvențele de comenzi sunt concatenate cu ajutorul operatorilor speciali. Rezultatul execuției unei comenzi este afișat de *bash* pe consola de ieșire (implicit, ecran). Această ieșire poate fi redirecționată către un fișier folosind semnul > ,

comanda > fișier – redirecționarea ieșirii standard în fișier, conținutul vechi al fișierului este șters.

Putem adăuga ieșirea unui proces la sfârșitul unui fișier, care va fi creat dacă acesta nu exista anterior, folosind semnul >> .

Comanda >> fișier – redirecționarea ieșirii standard în fișier, fluxul text se va adăuga la sfârșitul fișierului.

Alte redirecționări:

- **comanda1|comanda2** – redirecționarea ieșirii standard a primei comenzi ca intrare standard la comanda a doua, ceea ce permite crearea unei benzi de comenzi;
- **comanda1\$(comanda2)** – transmiterea ieșirii comenzii 2 ca parametru al comenzii 1. În cadrul unui script construcția \$(comanda2) poate fi utilizată, de exemplu pentru

transmiterea rezultatelor obținute de comanda2 în parametrii ciclului `for ... in`;

- **comanda1; comanda2** - comenzile sunt executate secvențial (una după alta);
- **comanda&** - rulează această comandă în fundal (eliberează terminalul);
- **comanda1&&comanda2** - comanda2 este executată numai dacă comanda1 are valoare de retur 0;
- **comanda1||comanda2** - comanda2 este executată numai dacă comanda1 are valoare de retur diferită de 0;
- {

comanda1

comanda2

} – concatenarea comenzilor după directivele `||`, `&&` sau în corpul buclelor și funcțiilor.

Comenzi pentru controlul proceselor. Prezintă unele comenzi utile (descrierea detaliată a caracteristicilor și a sintaxei comenzilor - **man comanda** (sau **info comanda**)):

- **kill** – transmite un semnal procesului. Semnalul poate fi un număr sau un nume simbolic. În mod implicit (fără a specifica semnalul), se transmite semnalul de terminare a procesului. Identificarea procesului se petrece după PID. Utilizați comanda **kill -l** și primiți o listă a semnalelor de sistem disponibile în GNU/Linux.
- **killall** - funcționează în același mod ca și comanda **kill**, dar pentru identificarea procesului utilizează un nume simbolic în loc de PID;
- **pidof** – afișează ID-ul procesului a unui program în execuție;

- **pgrep** – determină PID-ul proceselor cu caracteristicile specificate (de exemplu, lansate de un anumit utilizator);
- **kill** – trimite un semnal unui grup de procese cu caracteristicile specificate;
- **nice** – lansează procesul cu valoarea de prioritate specificată. Reducerea valorii (creșterea priorității execuției) poate fi inițiată numai de utilizatorul *root*;
- **renice** – modifică prioritate (valoarea) procesului lansat. Reducerea valorii (creșterea priorității execuției) poate fi inițiată numai de utilizatorul *root*;
- **at** – comandă introdusă de la tastatură, care urmează să fie executată ulterior;
- **tail** - vă permite să depistați și afișați liniile la sfârșitul fișierului;
- **sleep** – introduce o pauză în execuția script-ului;
- **cron** – un daemon care permite execuția comenzilor la un moment de timp.

cron rulează în background și verifică în permanență directoarele **/etc/cron.daily**, **/etc/cron.hourly**, **/etc/cron.monthly** și **/etc/cron.weekly** pentru a găsi script-urile adăugate. Comenzile sunt specificate în fișierul **/etc/crontab** (prin adăugarea de linii în fișierul *crontab* sau folosind comanda *crontab*). Comenzile ce vor fi executate doar o singură dată sunt adăugate folosind *at*.

Pentru a deschide fișierul de configurare **cron**, se va executa comanda:

```
$ crontab -e
```

Structura unei linii constă din următoarele câmpuri:

minute	ora	zi-din-lună	luna	zi-din-săptămână	utilizatorul	cale/către/c
m	h	dom	mon	dow	user	omandă
						comand

Unde:

- minute: 0 – 59;
- ora: 0 – 23;
- ziua-din-lună: 0 – 31;
- luna: 1 – 12 (12 == decembrie), sau *jan, feb, mar, ...*;
- ziua-din-săptămână: 0 – 6 (0 == duminică), sau *sun, mon, tue, wed, thu, fri, sat*;
- cale/către/comandă: calea către script-ul (comanda) care trebuie să fie executat.

Exemple de utilizare.

Linia de mai jos execută script-ul *backup.sh* în fiecare zi a săptămânii la ora 2 dimineața:

```
0 2 * * * /root/backup.sh
```

Linia următoare se execută în fiecare zi din două în două ore, începând cu ora 0:23, continuând cu 02:23, 04:23, etc.

```
23 0-23/2 * * * cale/către/comandă
```

Pentru **cron** există 4 operatori:

- **asteriscul (*)** - specifică toate valorile posibile pentru un câmp; de exemplu, un * în câmpul *oră* înseamnă că linia

respectivă va fi executată în fiecare oră, un * în câmpul *zi a lunii* semnifică execuția în fiecare zi;

- **virgula (,)** - acest operator specifică o listă de valori; de exemplu, dacă avem în câmpul *zi a lunii* **1,5,9,17,23,28**, linia respectivă se va executa doar în zilele specificate din fiecare lună;
- **liniuța (-)** - acest operator specifică un interval de valori; de exemplu, putem scrie **1-5** în loc de **1,2,3,4,5**;
- **separatorul slash (/)** - specifică pasul valorilor respective; de exemplu, cum am avut mai sus **0-23/2**, semnifică faptul că linia se va executa din 2 în două ore; tot la fiecare două ore mai poate fi specificat prin ***/2**; la fiecare 20 de minute putem specifica dacă trecem în câmpul minute ***/20**.

Organizarea interacțiunii dintre două procese. Deoarece esența interacțiunii constă în transferul de date și/sau controlul de la un proces la altul, vom considera două metode pentru organizarea unei astfel de interacțiuni: transferul de date prin fișier și transferul controlului prin semnal.

Interacțiunea proceselor printr-un fișier. Pentru a demonstra transferul de informații printr-un fișier, vom considera două script-uri - "Generator" și "Procesor" (tabelul 6.1). Este necesar ca informația să fie introdusă de la consola care utilizează script-ul "Generator" și se fie afișată pe ecran de consola ce utilizează script-ul "Procesor". Citirea liniei "QUIT", de către "Generator", duce la terminarea script-ului "Procesor". Fiecare script rulează în consola sa virtuală. Comutarea între console permite să gestionați script-urile și să urmăriți rezultatele funcționării lor.

Tabelul 6.1 – Script-uri

Generator	Procesor
#!/bin/bash while true; do read LINE echo \$LINE >> data.txt done	#!/bin/bash (tail -n 0 -f data.txt) while true; do read LINE; case \$LINE in QUIT) Echo "exit" killall tail exit ;; *) echo \$LINE ;; esac done

Script-ul "Generator" într-o buclă infinită citește liniile din consolă și le adaugă la sfârșitul fișierului *data.txt*.

Comanda *tail*, utilizată în "Procesor", are o opțiune folositoare prin care arată continuu ultimele *n* linii ale unui fișier care are un conținut în permanentă schimbare. Această opțiune **-f**, permite citirea dintr-un fișier numai dacă a fost adăugată informație în acest fișier, este utilizată de administratorii de sisteme pentru a verifica fișierele jurnal.

Opțiunile **-n 0** împiedică citirea din fișier dacă conținutul lui nu a fost actualizat după lansarea comenzii **tail**. Din moment ce trebuie redirecționată ieșirea comenzii **tail** la intrarea script-ului "Procesor", utilizați operatorul **pipe |**. Parantezele vă permit să rulați un subproces (procesul copil) independent în cadrul procesului părinte, iar operatorul **pipe |** va redirecționa ieșirea din acest subproces ca intrare pentru procesul părinte.

Astfel, comanda **read** din acest script execută ieșirea comenzii **tail**. Restul script-ului se bazează pe construcțiile studiate în laboratoarele anterioare. Singura excepție este comanda **killall tail**. Cu această comandă, se finalizează procesul fiu **tail** înainte de a finaliza procesul părinte. Comanda **killall** în acest caz este folosită pentru a simplifica codul, dar nu este întotdeauna corectă. Este mai corect să finalizați un proces utilizând comanda **kill** și **PID**-ul procesului.

Interacțiunea proceselor prin intermediul semnalelor. Semnalele (tabelul 6.2) reprezintă modul principal de transfer a controlului de la un proces la altul. Există semnale de sistem (de exemplu, **SIGTERM**, **SIGKILL**, etc.), dar este posibil ca procesului săi fie transferat un semnal al utilizatorului.

Tabelul 6.2 - Semnale

Num. semnal	Nume	Descrierea	Poate fi intercentat	Poate fi blocat	Taste
1	HUP	Hangup -semnalul indică procesului închiderea terminalului în care a fost pornit	Da	Da	

2	INT	Interrupt - semnalul este trimis procesului de terminalul său, în cazul când un utilizator dorește să întrerupă procesul	Da	Da	<Ctrl>+ <C> sau
3	QUIT	Întreruperea procesului	Da	Da	<Ctrl>+ <I>
4	ILL	Illegal Instruction - semnalul este trimis programului, când se încearcă să se execute o instrucțiune necunoscută sau privilegiată	Da	Da	
8	FPE	Floating Point Exception - eroare de calcul, de exemplu, împărțirea la zero	Da	Da	
9	KILL	Semnalul ce indică terminarea imediată a procesului	Nu	Nu	
11	SEGV	Segmentation Violation - semnalul este trimis către un proces când se face o referință la memoria virtuală nevalidă sau o eroare de segmentare, adică sa efectuat o încălcare a segmentării	Da	Da	
13	PIPE	Sa încercat transferul datelor utilizând o înlanțuire de comenzi sau o coadă FIFO, dar nu există niciun proces care să poată prelua aceste date	Da	Da	

15	TERM	Software Termination - cerere de a termina procesul (finalizarea programului)	Da	Da	
17	CHLD	Semnalul este trimis unui proces când un proces copil se termină, este întrerupt sau este relansat după ce a fost întrerupt	Da	Da	
18	CONT	Continuarea execuției procesului suspendat	Da	Da	
19	STOP	Semnalul indică sistemului de operare să oprească un proces pentru reluarea ulterioară	Nu	Nu	
20	TSTP	Semnalul este format de terminal, de obicei, de către utilizator executând clic pe Ctrl + Z	Da	Da	<Ctrl>+ <Z>

Tratarea semnalelor se realizează prin asocierea unei funcții (**handler**) unui semnal. Funcția (rutină, *handler*) reprezintă o secvență de cod, care va fi apelată în momentul în care procesul recepționează semnalul respectiv. Semnalele de sistem au *handler*-ele sale, de obicei. Pentru prelucrarea semnalului utilizatorului, evident utilizatorul va scrie și o funcție ce va prelucra acest semnal.

Pentru prelucrarea semnalelor în **sh** (**bash**) , este utilizată comanda **trap**:

trap action signal

Comanda are doi parametri: acțiunea (action) executată la recepționarea semnalului și semnalul pentru care va fi efectuată acțiunea specificată. În mod obișnuit, acțiunea este indicată prin apelarea funcției (cod de script) scrise anterior apelului.

Cu comanda **trap** puteți să schimbați handler-ul pentru unele semnale de sistem (cu excepția celor ale căror interceptare este interzisă). În acest caz, handler-ul executat este indicat de argumentul **action** în comanda **trap**.

Să considerăm o pereche de script-uri (tabelul 6.3) pentru a demonstra transferul controlului de la un proces la altul.

Tabelul 6.3 – Script-uri

Generator	Handler
<pre>#!/bin/bash while true; do read LINE case \$LINE in STOP) kill -USR1 \$(cat .pid) ;; *) : ;; esac done</pre>	<pre>#!/bin/bash echo \$\$ > .pid A=1 MODE="Work" usr1() { MODE="STOP" } trap 'usr1' USR1 while true; do case \$MODE in "Work") let A=\$A+1 echo \$A ;;</pre>

	<pre> "STOP") echo "Stopped by SIGUSR1" exit ;; esac sleep 1 done </pre>
--	--------------------------------------------------------------------------------------

Script-ul "Generator" va citi liniile din consolă într-o buclă infinită și va fi inactiv (folosind operatorul :) pentru orice linie de intrare, diferită de linia STOP. Citind linia STOP, script-ul, va trimite semnalul utilizatorului **USR1** procesului "Handler". Script-ului „Generator” este necesar să-i fie transmis PID-ul procesului "Handler". Transferul se petrece printr-un fișier ascuns. În procesul "Handler", PID-ul procesului este determinat folosind variabila **\$\$**.

Script-ul "Handler" afișează o secvență de numere naturale, până când va fi recepționat semnalul **USR1**. În acest moment, se lansează procesul handler **usr1 ()**, care modifică valoarea variabilei **MODE**. În următorul pas al ciclului, se va afișa un mesaj de finalizare a lucrului datorită apariției semnalului, iar script-ul va fi finalizat.

6.3 Sarcină pentru lucrarea de laborator

Executați următorii pași:

- Toate script-urile și fișierele pentru extragerea rezultatelor le veți crea în subdirectorul lab6.
- Scrieți script-uri, care vor rezolva următoarele probleme:

- 1) Creați și executați o singură dată script-ul (în acest script, nu puteți utiliza operatorul de condiție și operatorii de control a proprietăților și valorilor), care va încerca să creeze directorul **test** în directorul **home**. În cazul în care directorul va fi creat, script-ul va scrie în fișierul *~/raport* un mesaj "**catalog test was created successfully**" și va crea în directorul **test** un fișier numit *Data_Ora_Lansarii_Scriptului*. Apoi, indiferent de rezultatele etapei anterioare, script-ul trebuie să interogheze, folosind comanda **ping**, adresa *www.traiasca_moldova.md* și în cazul în care **host**-ul nu este disponibil, adăugați un mesaj de eroare în fișierul *~/raport* (se recomandă - >, >>, ~, date, ||, &&).
- 2) Modificați script-ul din 1) pentru încă o singură executare peste 2 minute. Controlați conținutul fișierului *~/raport* și afișați liniile noi apărute (se recomandă - at, tail).
- 3) Modificați script-ul din 1) ca să ruleze la fiecare 5 minute din oră, în zilele pare ale săptămânii.
- 4) Creați două procese de fundal care efectuează același ciclu infinit de calcul (de exemplu, înmulțirea a două numere). După lansarea proceselor, prevedeați posibilitatea de a utiliza consolele virtuale din care au fost lansate. Folosind comanda **top**, analizați procentul de utilizare a microprocesorului de către aceste procese. Rezervați primului proces lansat o rată de utilizare a resurselor microprocesorului nu mai mare de 20% (se recomandă - nice, kill, cpulimit).
- 5) Procesul "Generator" transmite informații procesului "Handler" utilizând un fișier ascuns. Procesul "Handler" trebuie să efectueze următoarele procesări asupra liniilor noi în acest fișier: dacă linia conține un singur caracter "+", procesul "Handler" comută modul în *adunare* și așteaptă introducerea datelor numerice. Dacă

șirul conține un singur caracter "*", atunci procesul "Handler" comută modul în *înmulțire* și așteaptă introducerea datelor numerice. În cazul când linia conține un întreg, atunci procesul "Handler" execută operația curentă activă (modul activ curent) asupra valorii variabilei calculate (curente) și valoarea recent introdusă (de exemplu, adună sau înmulțește rezultatul calculat anterior cu numărul recent introdus). Când lanșați script-ul, modul se va seta în *adunare*, iar variabila calculată – în 1. Dacă este primit QUIT, script-ul afișează un mesaj despre finalizare și finalizează activitatea. Dacă sunt primite alte valori, script-ul finalizează activitatea cu un mesaj de eroare - *date de intrare eronate*.

6) Procesul "Generator" citește liniile de la consolă până când va fi introdus **TERM**. În acest caz, acesta trimite un semnal de sistem **SIGTERM** procesului "Handler". Procesul "Handler" (ca în exemplul, ce afișează, în buclă infinită, un număr natural în fiecare secundă) trebuie să intercepteze semnalul de sistem SIGTERM și să finalizeze activitatea, afișând mesajul despre finalizarea activității cu primirea semnalului de la alt proces.

7) Procesul "Generator" citește linii de la consolă în buclă infinită. Dacă linia citită conține un singur caracter "+", procesul "Generator" trimite procesului "Handler" un semnal **USR1**. Dacă linia conține un singur caracter "*", "Generator" trimite procesului "Handler" semnalul **USR2**. Dacă șirul conține cuvântul **TERM**, "Generator" trimite un semnal **SIGTERM** procesului "Handler". Alte valori ale liniilor sunt ignorate. Procesul "Handler" adună 2 sau multiplică cu 2 valoarea curentă a numărului procesat (valoarea inițială este 1) în funcție de semnalul primit de la utilizator și afișează rezultatul pe ecran. Calcularea și afișarea sunt efectuate o dată pe secundă. Primind semnalul **SIGTERM**, "Handler" își finalizează

activitatea și afișează mesajul despre finalizarea activității cu primirea semnalului de la alt proces.

- c) Prezentați profesorului script-urile și primiți întrebări sau sarcină pentru susținerea lucrării de laborator. Baremul – rezolvarea 1, 2 – nota 6; 1-4 –nota 8; 1-7 - nota 10.

6.4 Sarcină pentru lucrul neauditorial

Subiecte de pregătire către lucrarea de laborator 7: Explicați comenzile – *cd, cp, ls, file, fiind, ln, mkdir, mv, pwd, rm, cat, readlink*.

Задания к лабораторной работе №6

Все скрипты и файлы для хранения результатов будут созданы в каталоге Lab6.

Напишите скрипты, для решения следующих задач:

- 1) Создайте и однократно выполните скрипт (в этом скрипте нельзя использовать условный оператор и операторы проверки свойств и значений), который будет пытаться создать директорию **test** в домашней директории. Если создание директории пройдет успешно, скрипт выведет в файл **~/report** сообщение вида **"catalog test was created successfully"** и создаст в директории **test** файл с именем **Дата_Время_Запуска_Скрипта**. Затем независимо от результатов предыдущего шага скрипт должен опросить с помощью команды **ping** хост **www.cto_tam.ru** и, если этот хост недоступен, дописать сообщение об ошибке в файл **~/report** (рекомендуются - >, >>, ~, date, ||, &&).
- 2) Задайте еще один однократный запуск скрипта из пункта 1 через 2 минуты. Организуйте слежение за

файлом **~/report** и выведите на консоль новые строки из этого файла, как только они появятся (рекомендуются - at, tail).

- 3) Задайте запуск скрипта из пункта 1 каждые 5 минут каждого часа в день недели, в который вы будете выполнять работу.
- 4) Создайте два фоновых процесса, выполняющих одинаковый бесконечный цикл вычисления (например, перемножение двух чисел). После запуска процессов должна сохраниться возможность использовать виртуальные консоли, с которых их запустили. Используя команду **top**, проанализируйте процент использования ресурсов процессора этими процессами. Добейтесь, чтобы тот процесс, который был запущен первым, использовал ресурс процессора не более чем на 20% (рекомендуются - nice, kill, cputlimit).
- 5) Процесс «Генератор» передает информацию процессу «Обработчик» с помощью файла. Процесс «Обработчик» должен осуществлять следующую обработку новых строк в этом файле: если строка содержит единственный символ «+», то процесс «Обработчик» переключает режим на *сложение* и ждет ввода численных данных. Если строка содержит единственный символ «*», то обработчик переключает режим на *умножение* и ждет ввода численных данных. Если строка содержит целое число, то обработчик осуществляет текущую активную операцию (выбранный режим) над текущим значением вычисляемой переменной и считанным значением (например, складывает или перемножает результат предыдущего вычисления со считанным числом). При запуске скрипта режим устанавливается в сложение, а вычисляемая переменная приравнивается к 1. В случае получения строки **QUIT** скрипт выдает сообщение о плановой остановке и завершает работу. В случае

получения любых других значений строки скрипт завершает работу с сообщением об ошибке входных данных.

- 6) Процесс «Генератор» считывает строки с консоли, пока ему на вход не поступит строка **TERM**. В этом случае он посылает системный сигнал **SIGTERM** процессу обработчику. Процесс «Обработчик» (как и в примере, выводящий в бесконечном цикле натуральное число каждую секунду) должен перехватить системный сигнал **SIGTERM** и завершить работу, предварительно выведя сообщение о завершении работы по сигналу от другого процесса.
- 7) Процесс «Генератор» считывает с консоли строки в бесконечном цикле. Если считанная строка содержит единственный символ «+», он посылает процессу «Обработчик» сигнал **USR1**. Если строка содержит единственный символ «*», генератор посылает обработчику сигнал **USR2**. Если строка содержит слово **TERM**, генератор посылает обработчику сигнал **SIGTERM**. Другие значения входных строк игнорируются. Обработчик добавляет 2 или умножает на 2 текущее значение обрабатываемого числа (начальное значение принять на единицу) в зависимости от полученного пользовательского сигнала и выводит результат на экран. Вычисление и вывод производятся один раз в секунду. Получив сигнал **SIGTERM**, «Обработчик» завершает свою работу, выведя сообщения о завершении работы по сигналу от другого процесса.

Предъявите скрипты преподавателю и получите вопрос или задание для защиты лабораторной работы. Выполненная лабораторная работа будет оценена следующим образом: выполнение п.1,2 – 6 баллов, 1-4 – 8, 1-7 – 10.

Задачи для неаудиторной работы

Вопросы для подготовки студентов к лабораторной работе 7:

Объясните команды – *cd*, *cp*, *ls*, *file*, *fiind*, *ln*, *mkdir*, *mv*, *pwd*, *rm*, *cat*, *readlink*.

7 Manipularea fișierelor în SO GNU/Linux

7.1 Scopul lucrării: Studiarea comenzilor pentru manipularea fișierelor și directoarelor, utilizarea mecanismului de legături, adresarea directă și indirectă a directoarelor.

7.2 Indicații metodice

Reamintim că conceptul de fișier este un concept fundamental pentru SO GNU/Linux. Alături de fișierele de date sunt utilizate fișiere speciale pentru a realiza interfețe de acces la dispozitive externe, pentru afișarea datelor despre starea resurselor și proceselor a sistemului de operare, despre configurarea componentelor sistemului de operare și aplicații personalizate, etc. Și directorul este, de asemenea, un tip special de fișiere care stochează numele și descriptorii subdirectoarelor și fișierelor incluse.

Principalele comenzi pentru manipularea fișierelor și directoarelor:

- **cd** (change directory) - schimbă directorul;
- **cp** fișier_sursă fișier_destinație – (copy) copiază un fișier (director);
- **ls** – afișează conținutul directorului;
- **file** – indică tipul fișierului;
- **find** – caută fișiere;
- **ln** - creează legături între fișiere;
- **mkdir** nume_director - (make directory) creează un director cu numele indicat;
- **mv** fișier_sursă fișier_destinație - (move) mută (redenumeste) un fișier sau director;
- **pwd** (print working directory) - afișează calea absolută a directorului curent;
- **rm** (remove) – șterge fișierul;
- **rmdir** nume_director - (remove directory) șterge directorul indicat;

- **cat** (concatenate) – concatenează și tipărește fișiere în consolă.

Fișiere de legătură. O legătură nu este altceva decât o cale prin care potrivim două sau mai multe nume de fișiere în același set de date. Sunt două căi prin care obținem acest lucru:

- Legături fizice (*hard links*): nu au echivalent în Windows, asociază două sau mai multe nume de fișiere aceluiasi nod. Legăturile fizice împart aceleași blocuri de date pe discul fix, în timp ce continuă să se comporte ca fișiere independente. Există un dezavantaj: legăturile fizice nu pot trece peste limita partițiilor, deoarece numărul asociat nodului este unic doar pentru partiția în cauză.
- Legături simbolice - numite și soft links (*symlink*): un fișier mic care este un indicator către alt fișier, precum un shortcut din Windows. O legătură simbolică conține traseul către fișierul țintă, în schimbul locului fizic de pe discul fix al fișierului țintă. Deoarece nodurile nu sunt folosite în această metodă, legăturile simbolice pot traversa partiții.

Fiecare fișier obișnuit este, în principiu, o legătură fizică. Legăturile fizice nu pot traversa partițiile, deoarece ele au ca referință nodurile (*inodes*), iar numărul unui nod este unic doar în interiorul unei partiții date.

Comanda prin care facem legături este **ln**. Pentru a realiza o legătură hard:

```
ln path_to_file      path_to_hard_link
```

Pentru a realiza o legătură simbolică se folosește opțiunea **-s**:

```
ln -s path_to_file      path_to_soft_link
```

Legăturile simbolice sunt fișiere mici, pe când legăturile fizice au aceeași mărime ca fișierul original.

```
$ touch a
$ ln -s a sym
$ ls -l
total 2
-rw-rw-r-- 1 student student 0 oct 7
22:42 a
lrwxrwxrwx 1 student student 1 oct 7
22:42 sym -> a
```

Observați că ultima linie începe cu **1**, semnificând ca intrarea este un link simbolic. Mai mult, **ls -l** ne arată și către ce fișier indică link-ul - *a*.

```
$ echo "abc" > a
$ cat a
abc
$ cat sym
abc
```

Link-ul se comportă ca și fișierul către care indică. Putem să le folosim ca fiind același fișier. Ce se întâmplă dacă mutăm fișierul destinație?

```
$ mv a ..
$ cat sym
cat: sym: No such file or directory
```

Link-ul nu mai funcționează dacă am mutat fișierul către care indică. Această problemă nu poate fi rezolvată cu link-uri simbolice. Dacă vrem să putem muta fișierul destinație, trebuie să folosim link-uri hard.

Pentru a citi starea link-ului simbolic, precum și numele fișierului la care se referă, este utilizată comanda **readlink**.

7.3 Sarcină pentru lucrarea de laborator

Executați următorii pași:

- a) Toate script-urile și fișierele pentru extragerea rezultatelor le veți crea în subdirectorul *lab7*.
- b) Scrieți script-uri, care vor rezolva următoarele probleme:
 - 1) Script-ul **rmtrash**.
 - Script-ului i se transmite un parametru - numele fișierului ce va fi creat în directorul curent (*lab7*).
 - Script-ul verifică dacă a fost creat un director ascuns *trash* în directorul *home/user* (\$HOME) al utilizatorului. Dacă nu este creat, îl creează.
 - Apoi, în directorul curent, script-ul creează un fișier cu numele parametrului. Creează și un hard link către fișierul creat și directorul ascuns *trash*, cu un nume unic ce constă din cifre (**utilizați (date +%s)**) și șterge fișierul din directorul curent.
 - Apoi, în fișierul ascuns **trash.log** din directorul *home/user* al utilizatorului este plasată o linie care conține calea completă a fișierului șters și numele link-ului hard creat.
 - 2) Script-ul **untrash**.
 - Script-ului i se transmite un parametru - numele fișierului care urmează să fie restabilit (fără calea completă - numai numele).
 - Script-ul în fișierul *trash.log* găsește toate înregistrările ce conțin nume de fișiere ce vor fi ca parametri pentru script și va afișa aceste nume de fișiere câte unul, cu o cerere de confirmare.
 - Dacă utilizatorul acceptă cererea, se execută o încercare de a restabili fișierul cu calea completă specificată anterior (creați un link hard în directorul corespunzător către fișierul din *trash* și ștergeți fișierul corespunzător din *trash*). Dacă directorul specificat în calea completă către fișier nu mai există, fișierul este

restabilit în directorul `home/user` al utilizatorului cu afișarea mesajului corespunzător.

3) Script-ul **backup**.

- Script-ul va crea în `/home/user/` directorul cu numele `Backup-YYYYMM-DD`, unde `YYYY-MM-DD` – data la care a fost rulat script-ul, dacă nu există nici un director în `/home/user/` cu un nume care să corespundă unei date mai mici de 7 zile față de cea curentă. Dacă în `/home/user/` este deja un director activ de backup (creat nu mai devreme de 7 zile de la data lansării script-ului), atunci nu se creează un nou director. Puteți utiliza comanda `date` pentru a determina data curentă.
- Dacă a fost creat un nou director, script-ul va copia în acest director toate fișierele din directorul `/home/user/source/` (pentru testarea script-ului creați un director și un set de fișiere în el). După aceasta, script-ul va scrie următoarele informații în fișierul `/home/user/backup-report`: o linie cu informații despre crearea noului director cu copiile de rezervă cu indicarea numelui și datei creării acestuia; lista fișierelor din `/home/user/source/` care au fost copiate în acest director.
- Dacă directorul nu a fost creat (există un director activ de backup), scriptul trebuie să copieze toate fișierele din `/home/user/source/` în acest director folosind următoarele reguli: dacă nu există nici un fișier cu acest nume în directorul de backup, el este copiat din `/home/user/source`. Dacă există un fișier cu acest nume, dimensiunea acestuia este comparată cu dimensiunea fișierului cu același nume din directorul de backup curent. Dacă dimensiunile sunt egale, fișierul nu este copiat. Dacă dimensiunile sunt diferite, atunci fișierul este copiat cu crearea automată a copiei versiunii, astfel încât ambele versiuni ale fișierului să

apară în directorul curent de backup (fișierul existent este redenumit, adăugându-i-se o extensie suplimentară ".YYYY-MM-DD" (data la care scriptul a fost pornit), iar cel copiat își păstrează numele). După copiere, se scrie o linie în fișierul /home/user/backup-report despre efectuarea modificărilor în directorul curent de backup, cu indicarea numelui său și data efectuării modificărilor.

- 4) Script-ul **upback**. Scriptul trebuie să copieze în directorul /home/user/restore/ toate fișierele din directorul de backup (care are cea mai recentă dată în nume), cu excepția fișierelor cu versiunile anterioare.

c) Toate scripturile trebuie să prelucreze corect orice parametru de intrare, precum și valoarea lui. Nu este permisă afișarea mesajelor de eroare de la comenzile separate, utilizate în script. În cazul unor date incorecte de intrare sau imposibilitatea de a efectua operațiunea, utilizatorului i se va transmite un mesaj special, care va fi generat în script. Testați script-urile înainte de a fi prezentate profesorului.

d) Prezentați script-urile profesorului și primiți întrebări sau sarcină pentru susținerea lucrării de laborator. Baremul – rezolvarea 1 – nota 6; 1, 2 –nota 8; 1-4 - nota 10.

7.4 Sarcină pentru lucrul neauditorial

Subiecte de pregătire către lucrarea de laborator 8: Explicați comenzile – *echo, copy, xcopy, dir, cd, md, rd, rm, fiind, sort, mem, diskpart, at, sc, call, if, for*.

Задания к лабораторной работе №7

Все скрипты и файлы для хранения результатов будут созданы в каталоге lab7.

Напишите скрипты, для решения следующих задач:

1) Скрипт **rmtrash**

- a) Скрипту передается один параметр – имя файла в текущем каталоге (lab7) вызова скрипта.
- b) Скрипт проверяет, создан ли скрытый каталог **trash** в домашнем каталоге пользователя. Если он не создан – создает его.
- c) Далее, в текущем каталоге, скрипт создает файл с именем параметра. Создает жесткую ссылку на созданный файл и скрытый каталог **trash** с уникальным именем, состоящим из цифр (**используйте - (date +%s)**) и удаляет файл из текущего каталога.
- d) Затем в скрытый файл **trash.log** в домашнем каталоге пользователя помещается запись, содержащая полный исходный путь к удаленному файлу и имя созданной жесткой ссылки.

2) Скрипт **untrash**

- a) Скрипту передается один параметр – имя файла, который нужно восстановить (без полного пути – только имя).
- b) Скрипт по файлу **trash.log** должен найти все записи, содержащие в качестве имени файла переданный параметр, и выводить по одному на экран полные имена таких файлов с запросом подтверждения.
- c) Если пользователь отвечает на подтверждение положительно, то предпринимается попытка восстановить файл по указанному полному пути (создать в соответствующем каталоге жесткую ссылку на файл из **trash** и удалить соответствующий файл из **trash**). Если каталога, указанного в полном пути к файлу, уже не существует, то файл восстанавливается в домашний

каталог пользователя с выводом соответствующего сообщения.

3) Скрипт **backup**

- a) Скрипт создаст в **/home/user/** каталог с именем **Backup-YYYYMM-DD**, где YYYY-MM-DD – дата запуска скрипта, если в **/home/user/** нет каталога с именем, соответствующим дате, отстоящей от текущей менее чем на 7 дней. Если в **/home/user/** уже есть «действующий» каталог резервного копирования (созданный не ранее 7 дней от даты запуска скрипта), то новый каталог не создается. Для определения текущей даты можно воспользоваться командой **date**.
- b) Если новый каталог был создан, то скрипт скопирует в этот каталог все файлы из каталога **/home/user/source/** (для тестирования скрипта создайте такую директорию и набор файлов в ней). После этого скрипт выведет в режиме дополнения в файл **/home/user/backup-report** следующую информацию: строка со сведениями о создании нового каталога с резервными копиями с указанием его имени и даты создания; список файлов из **/home/user/source/**, которые были скопированы в этот каталог.
- c) Если каталог не был создан (есть «действующий» каталог резервного копирования), то скрипт должен скопировать в него все файлы из **/home/user/source/** по следующим правилам: если файла с таким именем в каталоге резервного копирования нет, то он копируется из **/home/user/source**. Если файл с таким именем есть, то его размер сравнивается с размером одноименного файла в действующем каталоге резервного копирования. Если размеры совпадают, файл не копируется. Если размеры отличаются, то файл копируется с автоматическим созданием

версионной копии, таким образом, в действующем каталоге резервного копирования появляются обе версии файла (уже имеющийся файл переименовывается путем добавления дополнительного расширения «.YYYY-MM-DD» (дата запуска скрипта), а скопированный сохраняет имя). После окончания копирования в файл **/home/user/backup-report** выводится строка о внесении изменений в действующий каталог резервного копирования с указанием его имени и даты внесения изменений, затем строки, содержащие имена добавленных файлов с новыми именами.

- 5) Скрипт **upback**. Скрипт должен скопировать в каталог **/home/user/restore/** все файлы из актуального на данный момент каталога резервного копирования (имеющего в имени наиболее свежую дату), за исключением файлов с предыдущими версиями.

Все скрипты должны корректно обрабатывать любые передаваемые им входные параметры и значения. Не допускается вывод сообщений об ошибках от отдельных команд, использующихся в скрипте. В случае некорректных входных данных или невозможности выполнить операцию, пользователю должно выводиться отдельное сообщение, формирующееся в скрипте. Перед предъявлением результатов выполнения лабораторной работы преподавателю необходимо провести тестирование разработанных скриптов.

Выполненная лабораторная работа будет оценена следующим образом: выполнение п.1 – 6 баллов, 1,2 – 8, 1-4 – 10.

Задачи для неаудиторной работы

Вопросы для подготовки студентов к лабораторной работе
8: Объясните команды – *echo*, *copy*, *xcopy*, *dir*, *cd*,

*md, rd, rm, fiind, sort, mem, diskpart, at, sc,
call, if, for.*

8 Utilizarea consolei în SO Microsoft Windows Server

8.1 Scopul lucrării: linia de comandă Microsoft Windows Server, utilizarea comenzilor ce gestionează structura de directoare și sistemul de fișiere, gestionarea serviciilor și driver-elor.

8.2 Indicații metodice

Spre deosebire de SO GNU/Linux, în SO Microsoft Windows, principalul este o interfață grafică care oferă acces la toți parametrii variabili ale sistemului de operare și aplicații. Cu toate acestea, în sistemele de operare din această familie, este posibilă utilizarea liniei de comandă și scrierea script-urilor pentru automatizarea sarcinilor de administrare.

Interpretorul liniei de comandă este lansat de executabilul *cmd.exe*, care este localizat în directorul **%SystemRoot%\SYSTEM32**, unde **%SystemRoot%** este variabila de mediu care conține calea către directorul de sistem Windows (de exemplu, C: / Windows).

Interpretorul de comenzi poate executa două tipuri de comenzi: interne și externe. Comenzile interne sunt recunoscute direct de către interpretor. Orice comandă care nu se referă la comenzile interne este interpretată de către linia de comandă ca fiind externă. Comenzile externe sunt executate de utilitarele care, de regulă, sunt localizate în același director ca și *cmd.exe*.

Sintaxa unui apel de comandă include numele comenzii, urmată de spații și parametrii transmiși comenzii. În SO Microsoft Windows, parametrii sunt indicați după *slash* /, de exemplu, /A sau /X. Orice comandă acceptă parametrul /?.

Interpretorul *cmd.exe* acceptă redirecționarea comenzilor de intrare și ieșire, similare cu SO GNU/Linux: >, >>, <, |.

Comenzile de bază necesare pentru efectuarea lucrării de laborator:

- **echo** - afișează un mesaj text;
- **copy** - copiază unul sau un grup de fișiere. Comanda nu permite copierea fișierelor în subdirectoare;
- **xcopy** - copiază fișiere și directoare, inclusiv subdirectoare;
- **dir** - afișează o listă de fișiere și subdirectoare ale directorului;
- **cd** - schimbă directorul curent;
- **md** - creează un director;
- **rd** - elimină un director gol;
- **rm** - șterge fișierul;
- **find** - caută linia de text specificată într-un fișier sau într-un grup de fișiere. Rezultatul - este afișarea tuturor liniilor fișierelor care conțin modelul specificat;
- **sort** - sortează fluxul de date de intrare;
- **mem** - afișează informații despre memoria utilizată și liberă;
- **diskpart** - este un interpretor de comenzi independent pentru gestionarea unităților de stocare (partiții, disk-uri). Poate fi controlat prin comenzi sau script-uri. Fișierul **diskpart** este un fișier text cu extensia **.txt**. Comanda **diskpart** poate fi apelată cu parametrul **/s**;
- **at** (schedule service command line interface) - lansează programe și comenzi la data și ora specificată. Comanda **at** poate fi utilizată numai când serviciul de planificare este în desfășurare. Comanda **at**, care este apelată fără parametri, afișează o listă a tuturor comenzilor și programelor care vor fi lansate împreună cu aceasta. Pentru a apela comanda **at**, utilizatorul trebuie să fie membru al grupului local de administratori;
- **sc** - gestionează serviciile și este un program utilizat pentru comunicarea cu Service Control Manager. Utilizând diferite

- setări, puteți configura un anumit serviciu, puteți afișa starea curentă a serviciului, opriți și porniți serviciul, ș.a.m.d.;
- **call** – apel la un script din altul, fără a finaliza primul script;
 - **if** – operatorul de condiție.
 - **for** – operator ciclul iterativ.

Sintaxa structurii *if*:

- **if [not] ERRORLEVEL <Number> <Command> [else <Expression>]**

Condiția este adevărată dacă comanda anterioară, executată de cmd.exe, sa terminat cu un cod egal sau mai mare decât Number.

- **if [not] <String1>==<String2> <Command> [else <Expression>]**

*Condiția este adevărată dacă șirurile **String1** și **String2** sunt identice. Șirurile pot fi expresii literale sau variabile (de exemplu, % 1). Nu este necesar să includeți șirurile literale între ghilimele.*

- **if [not] exist <FileName> <Command> [else <Expression>]**

Condiția este adevărată, dacă există fișierul cu numele FileName.

Sintaxa structurii *for*:

```
for {%%|%}<Variable> in (<Set>) do  
<Command> [<Command-parameters>]
```

- **{%%|%}<Variable>** - Parametru obligatoriu. Utilizați un singur simbol (%) dacă comanda **for** este apelată din linia de comandă și două simboluri (%%) dacă

comanda este apelată dintr-un fișier `batch` (script). Variabilele trebuie să fie reprezentate literal, cum ar fi `%A`, `%B` sau `%C`.

- **<Set>** - Parametru obligatoriu. Specifică unul sau mai multe fișiere, directoare sau șiruri de text, sau o serie de valori care vor fi procesate de comanda specificat. Parantezele sunt obligatorii.
- **<Command>** - Parametru obligatoriu. Specifică comanda care se va executa pentru fiecare fișier, director sau șir de text sau pe domeniul valorilor incluse în **Set**.

Lista de comenzi de mai sus este incompletă.

8.2 Sarcină pentru lucrarea de laborator

Toate script-urile și fișierele pentru extragerea rezultatelor le veți crea în subdirectorul *lab8*.

Scrieți script-uri, care vor rezolva următoarele probleme:

- 1) Manipularea fișierelor și directoarelor.
 - Creați un director pe partiția **C:** \ cu numele LAB8. În acesta, creați fișiere cu informații despre versiunea sistemului de operare, memorie liberă și ocupată, hard disk-uri conectate la sistem.
 - Creați un subdirector **TEST**, copiați conținutul directorului LAB8 în el.
 - Creați cu o singură comandă un fișier în care veți plasa toate fișierele din LAB8.
 - Ștergeți toate fișierele din directorul curent, cu excepția celui ultim creat, specificând numele fișierelor șterse.
 - Creați un fișier text cu o listă de comenzi și parametri utilizați pentru executarea p. 1.
- 2) Lansarea și eliminarea proceselor.

- Creați un fișier executabil care va copia orice fișier din directorul C:\cd\ cu un volum mai mare de 2 MB în directorul \\hostname\temp.
- Programați lansarea fișierului peste 1 minut.
- Verificați dacă a început procesul de copiere; dacă procesul a început, stopați-l.
- Comparați fișierele sursă și destinație. Verificați integritatea lor.
- Continuați procesul de copiere din momentul întreruperii.
- Creați un fișier text cu o listă de comenzi și parametri utilizați pentru executarea p. 2.

3) Lucrul cu serviciile (services).

- Obțineți un fișier care conține lista serviciilor care rulează pe sistem.
- Creați un fișier *batch* ce va asigura:
 - stoparea serviciilor DNS-client;
 - cu o întârziere de timp, va crea un fișier care va conține o listă actualizată de servicii care rulează în sistem;
 - porniți un alt fișier de comandă care va compară fișierele cu listele de servicii până și după stoparea serviciilor DNS-client și creați fișierul cu indicația diferenței;
 - restabiliți serviciile.
- Creați un fișier text cu o listă de comenzi și parametri utilizați pentru executarea p. 3.

4) Căutarea și sortarea informațiilor din fișiere.

- Formați o listă cu toate numele de drivere încărcate în sistem și plasați-o într-un fișier DRIVERS, într-o formă tabelară.
- Sortați datele în ordine alfabetică, inversă.
- Creați un fișier text cu o listă de comenzi și parametri utilizați pentru executarea p. 4.

Prezentați script-urile profesorului și primiți întrebări sau sarcină pentru susținerea lucrării de laborator. Baremul – rezolvarea 1 – nota 6; 1, 2 –nota 8; 1-4 - nota 10.

8.4 Link-uri utile:

Comenzi CMD - <https://www.microsoft.com/en-us/download/details.aspx?id=56846>

Задания к лабораторной работе №8

Все скрипты и файлы для хранения результатов будут созданы в каталоге Lab8.

Напишите скрипты, для решения следующих задач:

1. Работа с файлами и директориями

1. Создать каталог на диске **C:** с именем **LAB8**. В нем создать файлы с информацией о версии операционной системы, свободной и загруженной памяти, жестких дисках, подключенных в системе. Имена файлов должны соответствовать применяемой команде.

2. Создать подкаталог **TEST**, в него скопировать содержимое каталога **LAB8**.

3. Создать одной командой файл с содержимым всех файлов каталога **LAB8**.

4. Удалить все файлы в текущем каталоге, кроме созданного последним, указав явно имена удаляемых файлов.

5. Создать текстовый файл со списком использованных команд и параметрами, использованными для выполнения п.п. 1.1–1.4.

2. Запуск и удаление процессов

1. В ручную узнать **имя_хостового_компьютера** (свойства компьютера).

2. Создать исполняемый файл, производящий копирование любого файла из дериктории **C:\cd** объемом более 2 Мбайт на ресурс **\\имя_хостового_компьютера\temp** с поддержкой продолжения копирования при обрыве.

3. Настроить запуск файла по расписанию через 1 минуту.

4. Проверить запуск копирования; если процесс появился, принудительно завершить его.

5. Сравнить исходный и конечный файл. Проверить их целостность.

6. Продолжить копирование с места разрыва.

7. Создать текстовый файл со списком использованных команд с параметрами, использованными для выполнения п.п. 2.1–2.5.

3. Работа со службами

1. Получить файл, содержащий список служб, запущенных в системе.

2. Создать командный файл обеспечивающий:

- а) остановку служб **DNS-client**;**
- б) с временной задержкой, создание файла, содержащего обновленный список служб, запущенных в системе;**
- с) запуск другого командного файла, сравнивающего файлы, полученные в пп. 3.1 и 3.2, и создающего разностный файл;**
- д) восстановление работы служб.**

3. Создать текстовый файл со списком использованных команд и параметрами, использованными для выполнения пп. 3.1–3.2.

4. Поиск и сортировка информации в файлах

1. Поместить список всех имен драйверов, загруженных в системе, в файл **DRIVERS**, в табличной форме.

2. Отсортировать полученные в п.п. 4.1 данные в обратном порядке по алфавиту.

3. Создать текстовый файл со списком использованных команд и параметрами, использованными для выполнения п.п. 4.1–4.2.

Выполненная лабораторная работа будет оценена следующим образом: выполнение п.1 – 6 баллов, 1,2 – 8, 1-4 – 10.

Bibliografia

1. <http://www.linuxcommand.org/index.php>.
2. <http://tille.garrels.be/training/tldp/ITL-Romanian.pdf>.
3. <http://ac.upg-ploiesti.ro/cursuri/so/laborator/lab04.html>.
4. <http://ac.upg-ploiesti.ro/cursuri/so/laborator/lab05.html#GREP>.
5. <https://www.techonthenet.com/linux/commands/>
6. <https://linux.die.net/man/>
7. <https://www.microsoft.com/en-us/download/details.aspx?id=56846>.
8. Shell scripting, -<http://andrei.clubicisco.ro>