

Universitatea din Bucureşti
Facultatea de Matematică şi Informatică – Informatică ID

Proiect Sisteme de Operare

Coordonator:
Prof. Marin Vlada

Student:
[REDACTED]

Iunie 2019

Cuprins

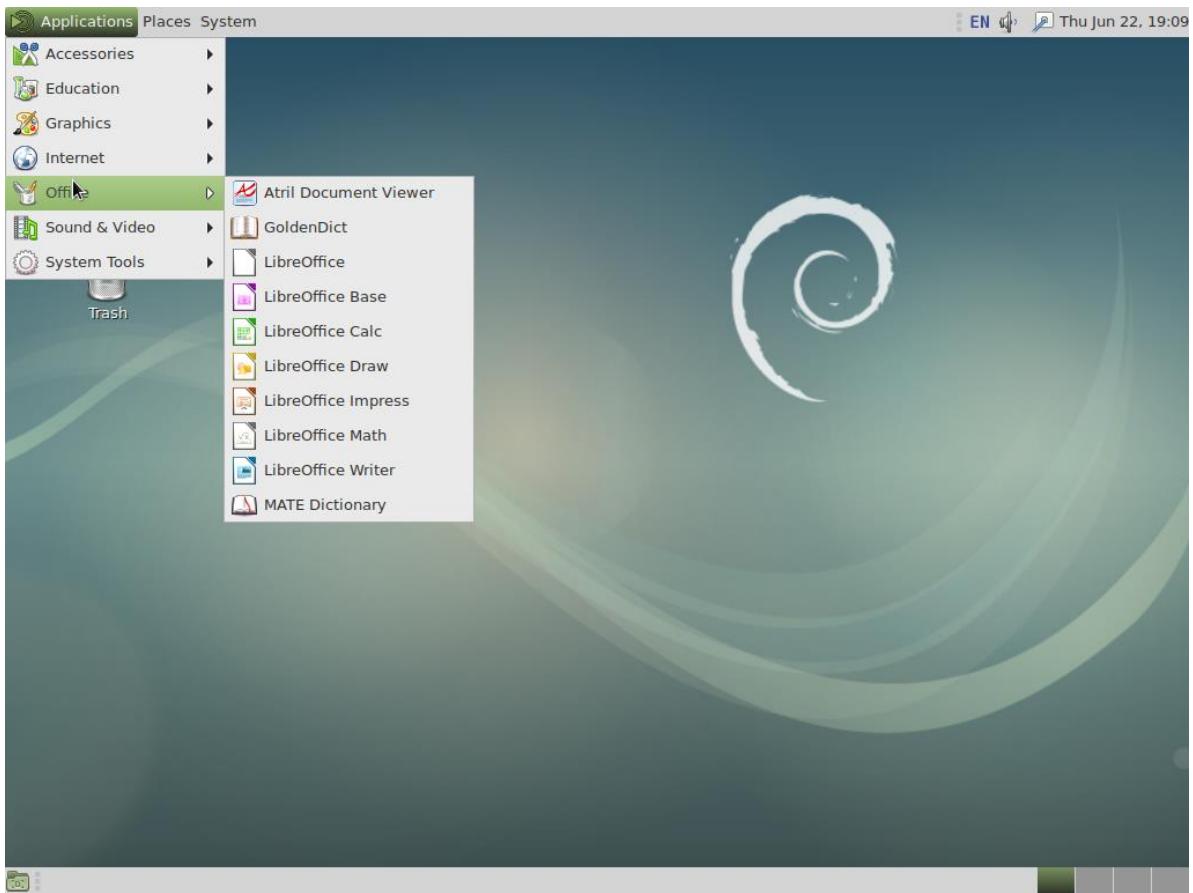
A.Distribuții Linux	3
Debian	3
Red Hat Enterprise Linux	3
Suse Linux	4
Ubuntu	5
Slackware	7
Mandriva	10
B.Mecanisme și Tehnici de Protecție a Sistemelor de Operare	11
Introducere în Securitate	11
Criptografia	11
Criptografia cu chei simetrice	11
Criptografia cu chei asimetrice	12
Modelul matricei de control a accesului	16
Modelul de securitate Bell-La Padula	16
Modelul de securitate Biba	20
Modelul Harrison-Ruzzo-Ullman	21
Modele ale securității multilaterale	22

A.

Distribuții Linux

Debian

Debian GNU/Linux este un sistem de operare compus din software liber, și o distribuție populară și foarte influentă între distribuțiile GNU/Linux. Această distribuție este menținută la zi datorită efortului voluntar depus de utilizatori de pe întreg mapamondul. Distribuția este testată riguros înainte de a fi lansată, pentru a asigura un grad ridicat de stabilitate. Nu este foarte ușor de configurat, această distribuție fiind orientată în principal către utilizatorul profesionist, dar are un foarte bun sistem de actualizare. Versiunile mai noi tend să fie orientate mai mult către utilizatorul obișnuit și beneficiază de asistență de calitate oferită de comunitatea de utilizatori. Sistemul de operare Debian poate fi folosit atât ca sistem de operare pentru calculatoarele desktop, cât și pentru servere.

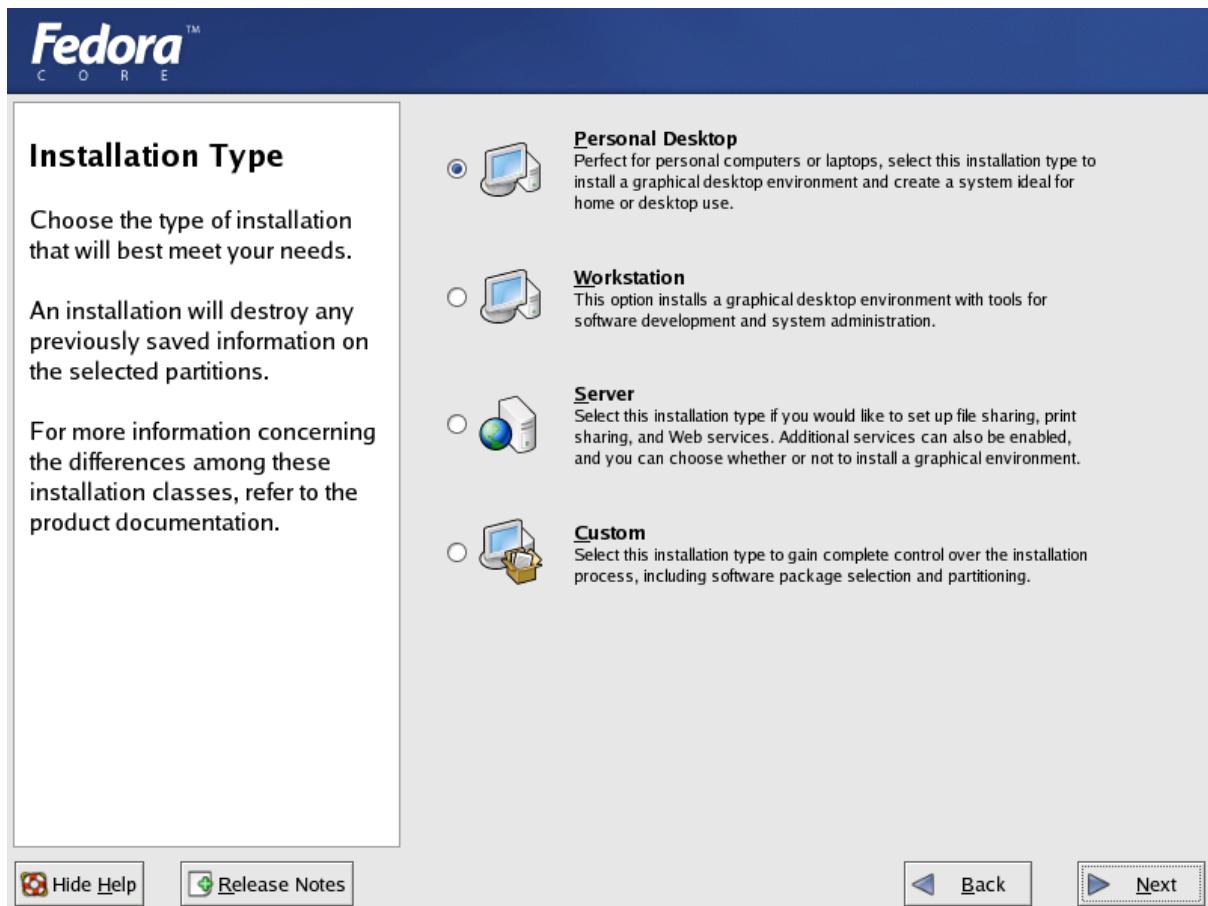


Red Hat Enterprise Linux

Red Hat Enterprise Linux (RHEL) este o distribuție Linux produsă de Red Hat, Inc pentru piața comercială. Red Hat Enterprise Linux a fost lansată în versiuni pentru servere x86, x86-64, Itanium, PowerPC și IBM System z și versiuni desktop pentru x86 și x86-64.

În anul 2003, Red Hat Enterprise Linux înlocuiește distribuția Red Hat Linux.

Red Hat Linux vine cu un program de instalare numit Anaconda, care se dorește a fi simplu de folosit, chiar și pentru începători. Aceasta are, de asemenea un instrument pentru configurarea firewall-ului numit Lokkit. În plus, de la versiunea 8.0 a distribuției Red Hat Linux, s-a mai adăugat suportul pentru codarea UTF-8 a fonturilor din sistem. Această modificare nu are efect asupra vorbitorilor de limbă engleză, dar este foarte importantă pentru internaționalizare și asigurarea suportului pentru alte limbi. Red Hat a scos anumite programe din cauza drepturilor de autor și a brevetelor, cum ar fi suportul pentru formatul MP3 și citirea partițiilor cu sisteme de fișiere NTFS.

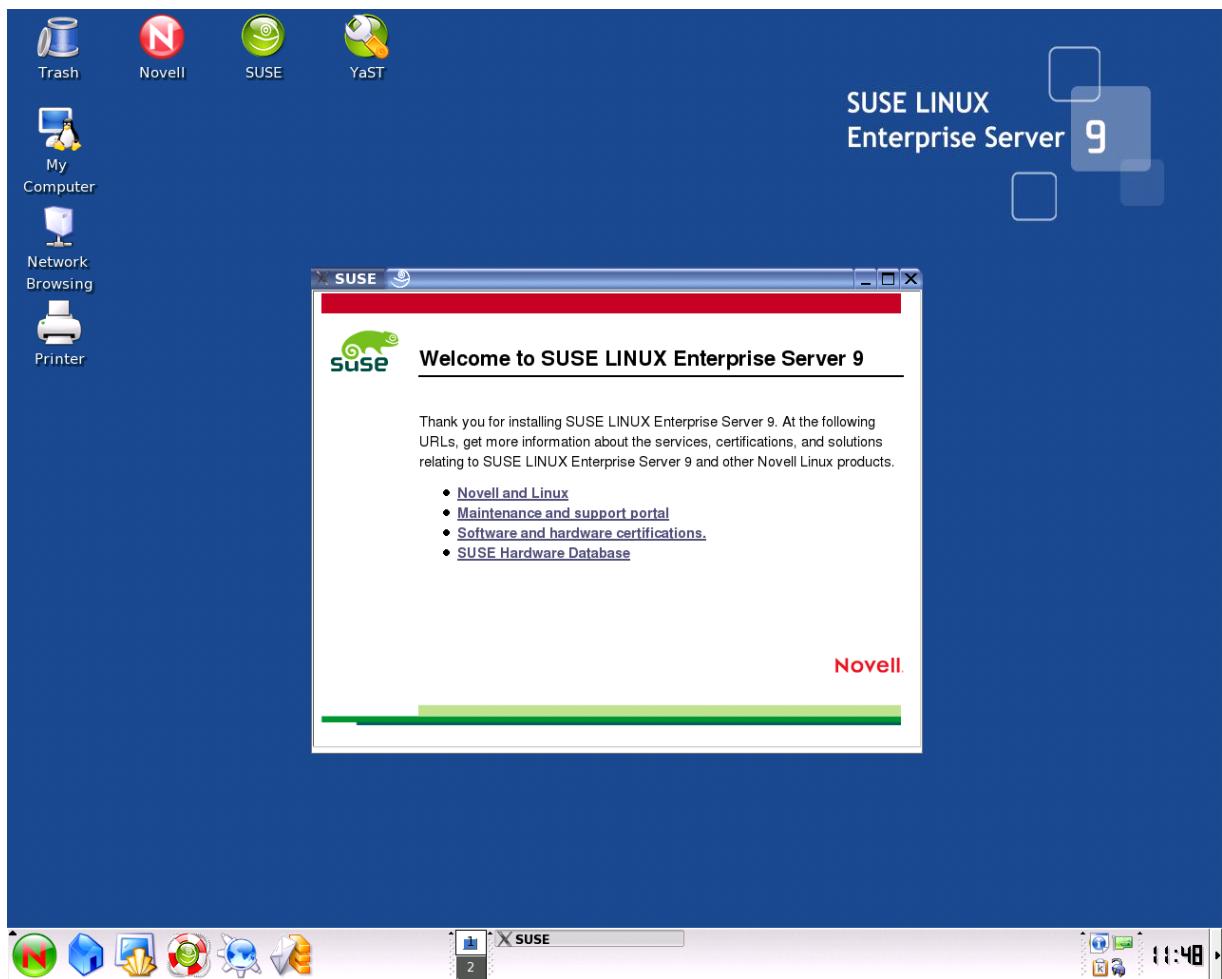


Suse Linux

SUSE Linux (în trecut ortografiat și S.u.S.E) este o distribuție Linux foarte populară, în special în Europa. Numele S.u.S.E este un acronim pentru "Software- und System-Entwicklung", în limba germană ("Dezvoltare de software și sisteme", în română).

Unul din punctele forte ale distribuției este suita de programe de instalare și configurare YaST, una dintre cele mai avansate și mai prietenoase dintre echivalentele sale din lumea Linux. SUSE are inclus foarte mult software într-o distribuție pe CD-uri (sau DVD). Poate adopta foarte repede noi versiuni de software, biblioteci, etc. Este orientată atât către profesioniști cât și către începători.

Distribuția lansată în 4 noiembrie 2015 sub numele openSUSE Leap 42.1 a inaugurat un nou tip de distribuție hibridă, bazată pe surse din SUSE Linux Enterprise (SLE), care posedă un nivel de stabilitate neîntâlnit în versiunile precedente.



Ubuntu

Ubuntu este un sistem de operare bazat pe Linux pentru computerele personale, servere și netbook-uri. Rudă apropiată a sistemului de operare Debian GNU/Linux, Ubuntu este ușor de instalat și folosit, des actualizat și neîngrădit de restricții legale. Ubuntu este sponsorizat de Canonical Ltd., o companie privată fondată de antreprenorul sud-african Mark Shuttleworth.

Numele sistemului de operare provine din limba zulusă, unde „ubuntu” este o ideologie ce poate fi definită pe scurt drept „credința într-o legătură universală ce unește întreaga omenire”. Sloganul adoptat, „Linux pentru ființe umane” încorporează unul din scopurile declarate ale proiectului, acela de a face din Linux un sistem de operare popular și ușor de folosit. Cea mai recentă versiune Ubuntu este 18.10.



Ușurință în folosire

Ubuntu folosește mediul de lucru Unity, al cărui scop este să ofere o interfață gratuită, simplă și intuitivă, dar în același timp și o pleiadă de aplicații și programe moderne. Suite de birou Libre Office, navigatorul web Mozilla Firefox și editorul grafic Gimp sunt câteva din programele distribuite implicit.

După instalarea sistemului de operare, utilizatorul este întâmpinat de un spațiu de lucru fără pictograme, în care culorile portocaliu și maro sunt predominante. Programele de uz general sunt instalate în meniul „Aplicații”. Locațiile importante și cele mai des frecventate sunt grupate în meniul „Places”. Modificarea parametrilor de funcționare se poate face cu ușurință din meniul „System”. Ferestrele deschise pot fi vizualizate în bara din josul ecranului.

Ubuntu este localizat în peste 40 de limbi, inclusiv limba română. Utilizatorii se pot folosi de unealta de traducere Rosetta pentru a contribui corecturi și/sau traduceri noi.

Din dorința de a-l face mai ușor de folosit, dezvoltatorii au pus accent pe folosirea comenzi sudo. Această comandă permite utilizatorilor să îndeplinească sarcini administrative fără a iniția o sesiune cu drepturi administrative.

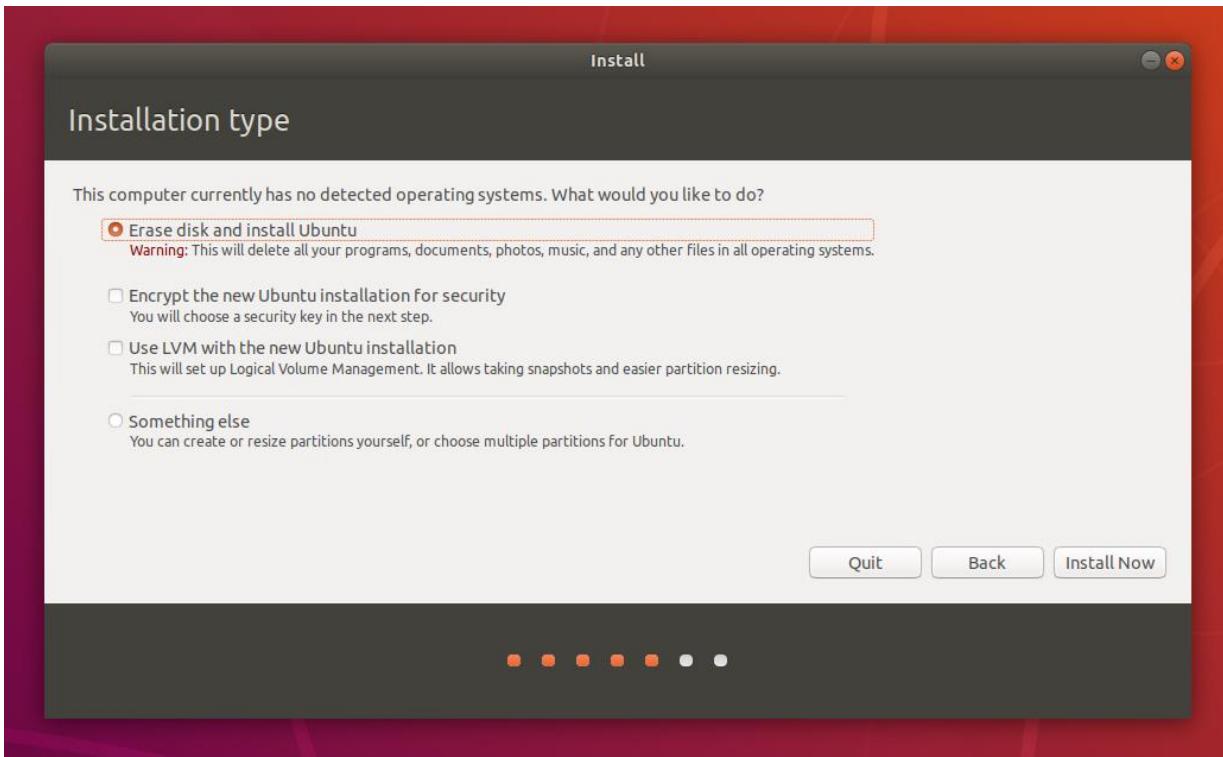
Metode de distribuție

Ubuntu poate fi descărcat ca imagine ISO și începe cu ușurință pe un CD. Alternativ, utilizatorii pot comanda gratuit prin poștă versiunile 6.06 și 7.04 folosind serviciul Shiplt oferit de Canonical. Multe distribuții Linux necesită descărcarea a numeroase imagini ISO și procesul lor de instalare poate dura câteva ore, dar Ubuntu poate fi instalat rapid, de pe un singur disc.

Începând cu versiunea 6.06, CD-ul de instalare Ubuntu poate fi folosit și ca LiveCD. În acest fel, utilizatorul poate experimenta cu programele disponibile și verifica compatibilitatea sistemului de operare cu hardware-ul existent fără a porni procesul de instalare. Există și un CD alternativ, pentru cei ce întâmpină probleme la instalarea de pe LiveCD, au un sistem slab performant sau pur și simplu doresc o instalare avansată folosind programul debian-installer.

LiveCD-ul conține și o colecție de software open source ce poate fi instalat pe sistemul de operare Microsoft Windows:

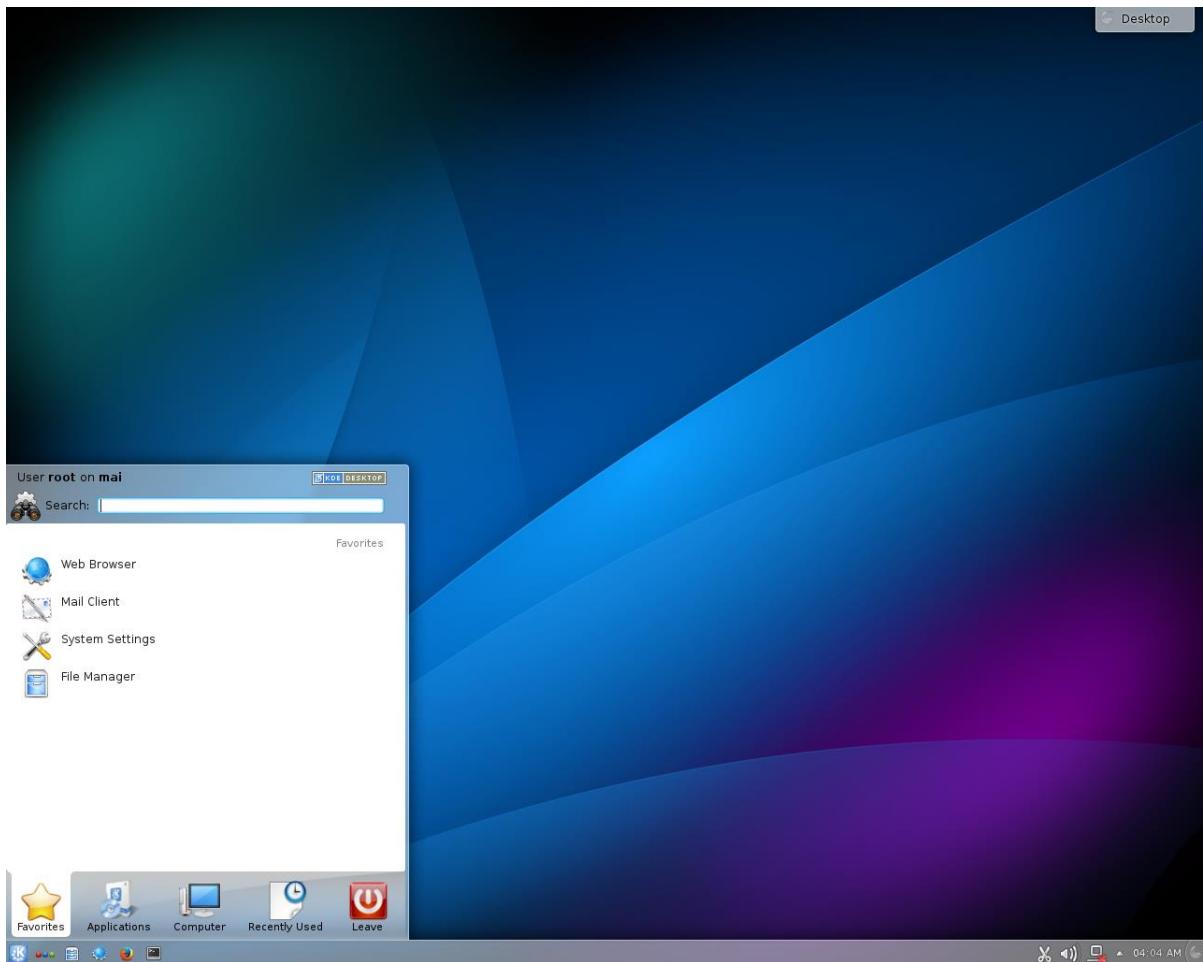
- Mozilla Firefox, Mozilla Thunderbird, AbiWord, Blender sau ClamWin.



Fiecare versiune este lansată în două variante: una destinată instalării pe computere personale de birou și laptopuri și o a doua pentru servere. Prima este disponibilă pentru arhitecturile x86 și x86-64, în timp ce la a doua se adaugă suport pentru arhitectura SPARC (fără LiveCD).

Slackware

Slackware este o distribuție Linux. Slackware folosește un mod de abordare diferit de distribuțiile renumite cum ar fi Red Hat, Debian, Gentoo, SUSE, sau Mandriva Linux. Poate fi descris cel mai bine ca fiind „UNIX-like”, având în vedere politica strictă de incorporare a aplicațiilor, și absența unor utilitare de configurare ce folosesc interfață grafică, precum în alte varietăți Linux. Printre partizanii Slackware există o vorbă, „când știi Slackware, știi Linux, când știi Red Hat, tot ce știi e Red Hat”.



Slackware Linux este un sistem de operare UNIX multitasking pe 32 sau 64 biți. Este foarte ușor de instalat și conține documentația online, având de asemenea un sistem bazat pe interfață grafică. Instalarea oferă posibilitatea alegerii sistemului X Window System, medii de dezvoltare în C/C++, Perl, Python, utilități pentru rețea cum ar fi mail, știri, web și servere ftp, GNU Image Manipulation, Mozilla Firefox, plus multe alte programe. Slackware Linux poate rula pe 486 sisteme (versiuni mai vechi) până la ultimele computere pe x86 (dar folosește optimizarea -mcpu=i686 pentru performanțe mai bune pe computere clasa i686 cum ar fi P3, P4, Duron/Athlon și ultimele procesoare multi-core x86) și de asemenea pe orice calculator ce rulează un procesor AMD64 compatibil.

Slackware menține o filosofie KISS. Folosește fișiere text pentru configurare în loc de programe de interfață grafică precum multe alte distribuții. Principalele caracteristici și particularități ale Slackware precum și filosofia din spatele distribuției sunt descrise după cum urmează.

Filosofia Slackware

Slackware este:

- O distribuție ce poate fi ușor instalată offline folosind un CD/DVD.
- O distribuție ce este lansată atunci când există o versiune stabilă și nu bazată pe un anumit calendar. Fiecare versiune lansată de Slackware Linux este foarte bine testată de echipa Slackware, precum și de comunitate, punând preț mai mult pe stabilitate decât pe ideea de software nou.
- O distribuție unde simplitatea este preferată în locul convenienței. Lipsa fișierelor tip helper din interfață grafică (comune în alte distribuții comerciale Linux) remarcă acest aspect.
- O distribuție unde administrarea și configurarea sistemului este făcută prin scripturi sau prin fișiere text ușor editabile, cu multe comentarii pentru a ușura configurarea fișierelor.

- O distribuție ce preferă software simplu sau software ce nu a fost modificat de către inițiatori.
- O distribuție ce urmează principiul „dacă nu este stricat, nu încerca să îl repari”.



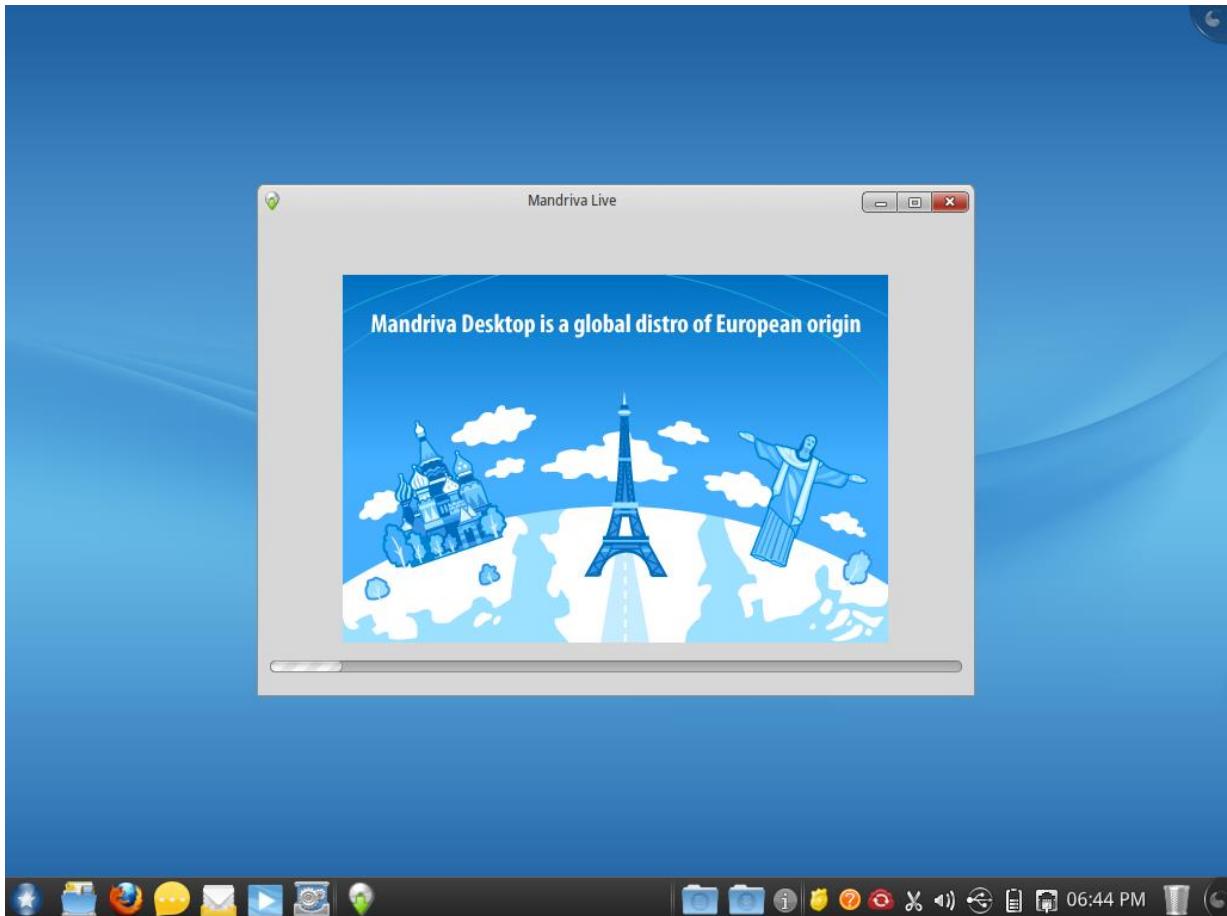
Mandriva

Mandriva Linux (în trecut Mandrakelinux sau Mandrake Linux, o achiziție Conectiva și Lycoris) este o distribuție Linux creată de Mandriva (în trecut Mandrakesoft). Prima versiune a fost bazată pe Red Hat Linux (versiunea 5.1) și KDE (versiunea 1.0) în iulie 1998. De atunci distribuția s-a distanțat de Red Hat și a inclus mai multe unelte originale în principal pentru a ușura configurarea sistemului. Mandriva Linux (în acea vreme se numea Mandrake Linux) a fost inițiativa lui Gaël Duval, care a fost de asemenea cofondator al Mandrakesoft.

Mandriva Linux este o distribuție recunoscută ca fiind prietenoasă și simplu de utilizat de către începători.

A fost creată în 1998 de compania franceză Mandrakesoft, sub numele Mandrake Linux, cu scopul de a face Linux-ul mai ușor de folosit. La acea vreme, Linux-ul era foarte bine cunoscut ca fiind un sistem de operare puternic și stabil, care însă cerea cunoștințe tehnice foarte bogate și folosirea liniei de comandă în foarte multe situații; Mandrakesoft a văzut acesta ca pe o oportunitate de a integra cele mai bune interfețe grafice și să contribuie cu propriile utilitare de configurație.

În 2005 Mandrakesoft a achiziționat firma braziliană Conectiva, producătorul distribuției Conectiva Linux. În aprilie 2005 firma și-a schimbat numele în Mandriva (Mandrake + Conectiva), iar distribuția a devenit cunoscută sub numele de Mandriva Linux.



B.

Mecanisme și Tehnici de Protecție a Sistemelor de Operare

Introducere în securitate

Securitatea informației este un concept mai larg care se referă la asigurarea integrității, confidențialității și disponibilității informației. Dinamica tehnologiei informației induce noi riscuri pentru care organizațiile trebuie să implementeze noi măsuri de control. De exemplu, popularizarea unităților de inscripționat CD-uri sau a memorilor portabile de capacitate mare, induce riscuri de copiere neautorizată sau furt de date.

Lucrul în rețea și conectarea la Internet induc și ele riscuri suplimentare, de acces neautorizat la date sau chiar frauda. Dezvoltarea tehnologică a fost acompaniată și de soluții de securitate, producătorii de echipamente și aplicații incluzând metode tehnice de protecție din ce în ce mai performante. Totuși, în timp ce în domeniul tehnologiilor informaționale schimbarea este exponențială,

componenta umană rămâne neschimbată. Asigurarea securității informațiilor nu se poate realiza exclusiv prin măsuri tehnice, fiind în principal o problemă umană.

Criptografia

Criptografia reprezintă o ramură a matematicii care se ocupă cu securizarea informației precum și cu autentificarea și restrictionarea accesului într-un sistem informatic. În realizarea acestora se utilizează atât metode matematice (profitând, de exemplu, de dificultatea factorizării numerelor foarte mari), cât și metode de criptare cuantică.

Domeniul modern al criptografiei poate fi împărțit în câteva domenii de studiu.

Tehnici utilizate în criptografie:

În prezent există două tipuri principale de tehnici utilizate în criptografie, și anume:

- criptografia prin cheie simetrice (chei secrete sau chei private) și,
- criptografia prin chei asimetrice (chei publice).

În cazul cheii simetrice, atât expeditorul cât și destinatarul mesajului folosesc o cheie comună secretă. În cazul cheii asimetrice, expeditorul și destinatarul folosesc în comun o cheie publică și, individual, câte o cheie privată.

Criptografia cu chei simetrice

Criptografia cu chei simetrice se referă la metode de criptare în care atât trimițătorul cât și receptorul folosesc aceeași cheie (sau, mai rar, în care cheile sunt diferite, dar într-o relație ce la face ușor calculabile una din celalaltă). Acest tip de criptare a fost singurul cunoscut publicului larg până în 1976.

Studiul modern al cifrurilor cu chei simetrice se leagă mai ales de studiul cifrurilor pe blocuri și al cifrurilor pe flux și al aplicațiilor acestora. Un cifru pe blocuri este, într-un fel, o formă modernă de cifru poli-alfabetic Alberti: cifrurile pe blocuri iau la intrare un bloc de text clar și o cheie, și produc la ieșire un bloc de text cifrat de aceeași dimensiune. Deoarece mesajele sunt aproape mereu mai lungi decât un singur bloc, este necesară o metodă de unire a blocurilor succesive. S-au dezvoltat câteva astfel de metode, unele cu securitate superioară într-un aspect sau altul decât alte cifruri. Acestea se numesc *moduri de operare* și trebuie luate în calcul cu grijă la folosirea unui cifru pe blocuri într-un criptosistem.

Data Encryption Standard (DES) și Advanced Encryption Standard (AES) sunt cifruri pe blocuri care sunt considerate standarde de criptografie de guvernul american (deși DES a fost în cele din urmă retras după adoptarea AES). În ciuda decăderii ca standard oficial, DES (mai ales în varianta triple-DES, mult mai sigură) rămâne încă popular; este folosit într-o gamă largă de aplicații, de la criptarea ATM la securitatea e-mail-urilor și accesul la distanță securizat. Multe alte cifruri pe blocuri au fost elaborate și lansate, cu diverse calități. Multe au fost sparte.

Cifrurile pe flux de date, în contrast cu cele pe blocuri, creează un flux arbitrar de material-cheie, care este combinat cu textul clar, bit cu bit sau caracter cu caracter. Într-un cifru pe flux de date, fluxul de ieșire este creat pe baza unei stări interne care se modifică pe parcursul operării cifrului. Această schimbare de stare este controlată de cheie, și, la unele cifruri, și de fluxul de text clar. RC4 este un exemplu de bine-cunoscut cifru pe flux.

Funcțiile hash criptografice (adesea numite *message digest*) nu folosesc neapărat chei, sunt o clasă importantă de algoritmi criptografici. Aceștia primesc date de intrare (adesea un întreg mesaj), și produc un hash scurt, de lungime fixă, sub forma unei funcții neinversabile. Pentru hash-urile bune, coliziunile (două texte clare diferite care produc același hash) sunt extrem de dificil de găsit.

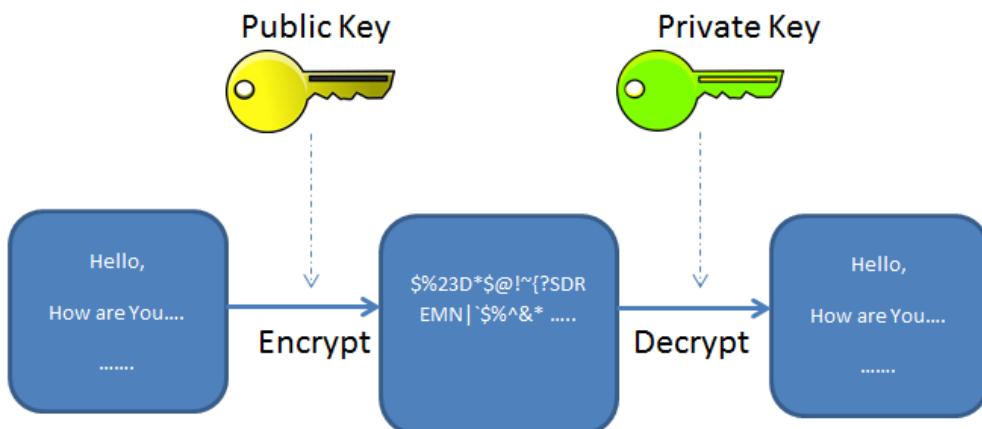
Criptografia cu chei asimetrice

Criptografia asimetrică este un tip de criptografie care utilizează o pereche de chei: o cheie publică și o cheie privată. Un utilizator care deține o astfel de pereche își publică cheia publică astfel încât oricine dorește să o poată folosi pentru a îi transmite un mesaj criptat. Numai deținătorul cheii secrete (private) este cel care poate decripta mesajul astfel criptat.

Matematic, cele două chei sunt legate, însă cheia privată nu poate fi obținută din cheia publică. În caz contrar, oricine ar putea decripta mesajele destinate unui alt utilizator, fiindcă oricine are acces la cheia publică a acestuia.

O analogie foarte potrivită pentru proces este folosirea cutiei poștale. Oricine poate pune în cutia poștală a cuiva un plic, dar la plic nu are acces decât posesorul cheii de la cutia poștală.

Criptografia asimetrică se mai numește criptografie cu chei publice.



Sistemele de criptare cu chei simetrice folosesc o singură cheie, atât pentru criptare cât și pentru decriptare. Pentru a putea folosi această metodă, atât receptorul cât și emițătorul ar trebui să cunoască cheia secretă. Aceasta trebuie să fie unică pentru o pereche de utilizatori, fapt care conduce la probleme din cauza gestionării unui număr foarte mare de chei. Sistemele de criptare asimetrice înlătură acest neajuns. De asemenea, se elimină necesitatea punerii de acord asupra unei chei comune, greu de transmis în condiții de securitate sporită între cei 2 interlocutori.

Cele două mari ramuri ale criptografiei asimetrice sunt:

1. *Criptarea cu cheie publică* – un mesaj criptat cu o cheie publică nu poate fi decodificat decât folosind cheia privată corespunzătoare. Metoda este folosită pentru a asigura confidențialitatea.
2. *Semnături digitale* – un mesaj semnat cu cheia privată a emițătorului poate fi verificat de către oricine, prin acces la cheia publică corespunzătoare, astfel asigurându-se autenticitatea mesajului.

O analogie pentru semnăturile digitale ar fi sigilarea unui plic folosind un sigiliu personal. Plicul poate fi deschis de oricine, dar sigiliul personal este cel care verifică autenticitatea plicului.

1. Criptarea cu cheie publică

Spre deosebire de sistemele de criptare bazate pe chei secrete, care presupun o singură cheie cunoscută de emițător și receptor, sistemele bazate pe chei publice folosesc două chei: una publică și una privată. Cheia publică este pusă la dispoziția oricărei persoane care dorește să trimită un mesaj criptat; Cheia privată este utilizată pentru decriptarea mesajului, iar nevoie de a face schimb de chei secrete este eliminată. Funcționarea sistemului:

- cheia publică nu poate decripta un mesaj criptat;
- se recomandă ca o cheie privată să nu derive dintr-o cheie publică;
- un mesaj care a fost criptat printr-o anumită cheie poate fi decriptat cu altă cheie;
- cheia privată nu este făcută publică.

Criptografia prin chei publice este posibilă în aplicațiile care funcționează într-un singur sens. O funcție în sens unic este aceea care este ușor de calculat într-o direcție, dar dificil de calculat în sens invers. Pentru o asemenea funcție, $y = f(x)$, este simplu de calculat valoarea lui y dacă se știe x , dar este foarte greu să-l determinăm pe x dacă-l cunoaștem pe y (de ex. cartea telefonică, dacă știm un număr de telefon, este foarte greu să găsim persoana).

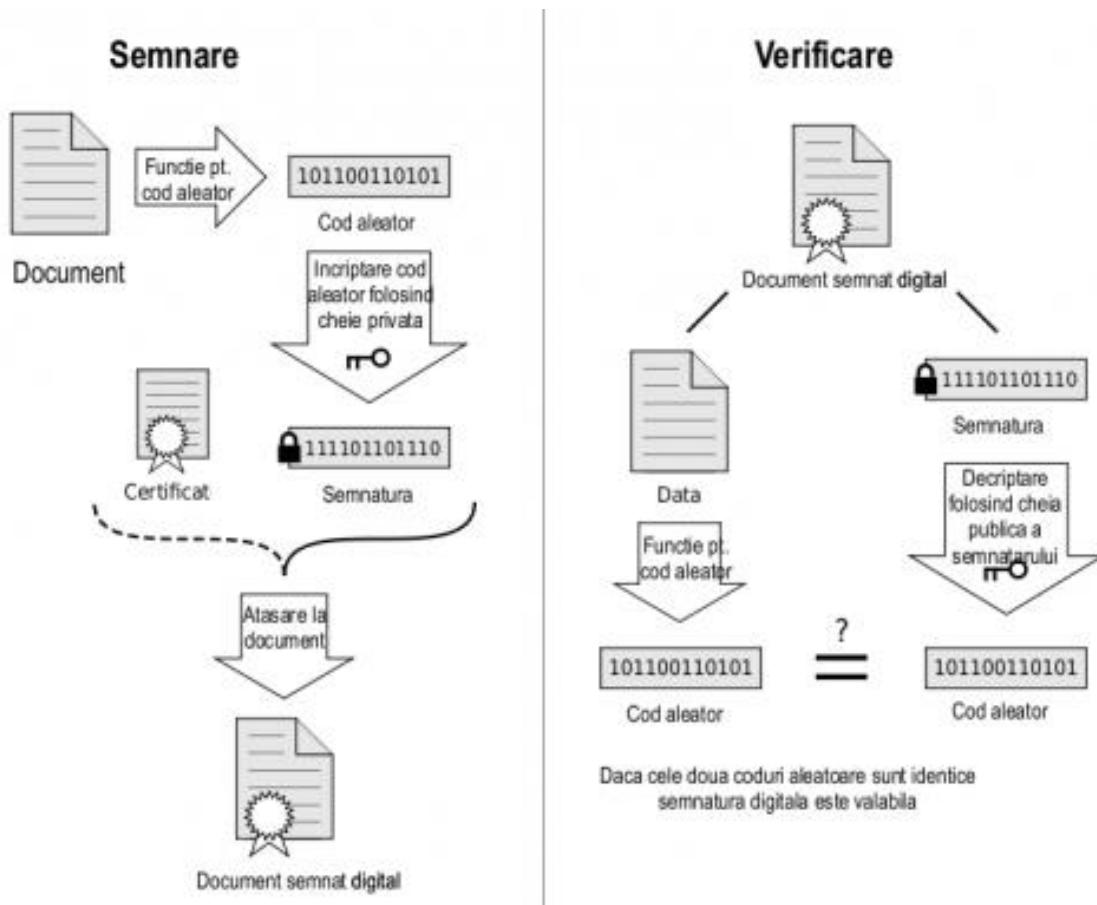
Pentru ca funcțiile cu sens unic să fie utile, ele trebuie să aibă o trapă, adică un mecanism secret care să permită realizarea cu ușurință a funcției inverse, astfel încât să se poată obține x dacă se știe y . În contextul criptografiei bazate pe chei publice este foarte dificil să se calculeze cheia privată din cheia publică dacă nu se știe trapa. De-a lungul anilor s-au dezvoltat mai mulți algoritmi pentru cheile publice. Unii dintre ei se folosesc pentru semnătura digitală, pentru criptare sau în ambele scopuri.

Din cauza calculelor numeroase solicitate de criptarea prin chei publice, aceasta este de la 1000 la 10.000 ori mai lentă decât criptarea prin chei secrete, au apărut metode hibride, care folosesc criptografia prin chei publice pentru transmiterea sigură a cheilor secrete utilizate în criptografia prin chei simetrice. Dintre algoritmii importanți ai cheilor publice, amintim Diffie-Hellman, RSA, El Gamal Knapsak și curba eliptică.

2. Semnăturile digitale

Inventarea criptografiei prin chei publice a adus două mutații importante. Prima permite transmiterea unui secret către o altă persoană fără să fie nevoie de o a treia persoană de încredere sau de un canal de comunicație off-line pentru a transmite cheia secretă. A doua mutație s-a produs pe planul calculării semnăturii digitale.

O semnătură digitală este un bloc de date (cifre binare) ce se atașează unui mesaj sau document pentru a întări încrederea unei alte persoane sau entități, legându-le de un anumit emițător.



Legătura este realizată astfel încât semnătura digitală poate fi verificată de receptor sau de o terță persoană și nu se poate spune că a fost uitată. Dacă doar o cifră binară nu corespunde, semnătura va fi respinsă în procesul de validare.

Semnătura digitală stabilește autenticitatea sursei mesajului. Dacă o persoană nu și divulgă cheia personală privată nimici nu poate să-i imite semnătura. O semnătura privat nu înseamnă și recunoașterea dreptului de proprietate asupra textului transmis, ci ea atestă faptul că persoana semnatării a avut acces la el și la semnat.

Atunci când semnarea este cuplată cu crearea documentului, semnătura digitală poate oferi o probă evidentă a originii documentului. În această categorie intră fotografiile făcute cu camere digitale bazate pe chei private, caz în care proba este de necontestat. Procedeul este folosit când se dorește realizarea protecției împotriva manipulării imaginilor cu calculatorul. În același mod pot lucra și camerele video sau alți senzori care și pot semna ieșirea pentru a-i certifica originea.

Deși semnătura digitală este implementată prin sistemul criptografiei cu chei publice, în cazul acesta componenta privată este folosită pentru semnarea mesajelor în timp ce componenta publică este folosită pentru a verifica semnătura.

Iată care este mecanismul realizării semnăturii digitale: Dacă (A) vrea să semneze un mesaj, va calcula o valoare rezumat a mesajului, care este determinată printre-o funcție publică de dispersie (hashing). În acest moment nu se folosesc chei. În pasul următor (A) va utiliza o cheie privată pentru semnătură KSApriv. pentru a calcula o transformare criptografică a valorii rezumat a mesajului. Rezultatul, care este semnătura sa pe mesaj, se atașează mesajului. Din acest moment, mesajul poate fi transmis unei alte persoane, de exemplu (B), sau poate fi stocat într-un fișier.

Când (B) primește mesajul, validează semnătura lui (A) cu ajutorul cheii ei publice pentru semnături KSApubl, ce va fi folosită ca intrare într-o funcție criptografică prin care se va testa dacă valoarea rezumat determinată de (B) este aceeași cu valoarea codificată prin semnătura lui (A). Dacă valoarea

coincide, (B) va accepta semnătura. De remarcat că nici o cheie a lui (B) nu a fost utilizată în procesul de validare a semnăturii transmise de (A), ci doar cheile lui (A).

Dacă (A) transmite cheia unui mesaj secret către (B), va folosi doar cheile lui (B). Dacă (A) dorește să trimită un mesaj, semnat și criptat, către B, procesul presupune utilizarea cheilor pentru semnături ale lui A (KSApriv și KSApub), a cheilor lui B de criptare (KBpub) și o cheie a mesajului, K. În sinteză, procesul este următorul:

- (A) generează o cheie aleatoare a mesajului, K. (A) cripteaază mesajul M cu cheia K, obținând mesajul criptat MC;
- (A) cripteaază cheia K folosind cheia publică a lui (B) de criptare KBpub, rezultând cheia criptată KC;
- (A) realizează o semnătură S folosind cheia sa privată pentru semnătură KSApriv.;
- (A) trimite către (B) următoarele: KS, MC și S;
- (B) folosește cheia sa privată de criptare, KBpriv, pentru a decripta KC și a obține cheia K;
- (B) folosește K pentru decriptarea MC și obține textul clar M;
- (B) folosește cheia publică pentru semnătură a lui (A), KSApub, pentru validarea semnăturii S.

Tot acest proces este utilizat de procesul de criptare a e-mail-urilor, aşa că (A) și (B) nu vor efectua manual operațiunile de mai sus, ci le va face calculatorul.

Modelul matricei de control a accesului

Prinț-o matrice de acces se oferă drepturi de acces pentru subiecte de încredere la obiectele sistemului. Drepturile de acces sunt de tipul *citește, scrie, execută* și.a. Un *subiect de încredere* este o entitate activă care își caută drepturile de acces la resurse sau obiecte. Subiectul poate fi o persoană, un program sau un proces. Un obiect este o entitate pasivă, cum sunt fișierele sau o resursă de stocare. Sunt cazuri în care un element poate fi, într-un anumit context, subiect și, în alt context, poate fi obiect.

Coloanele se numesc **Liste de control al accesului**, iar liniile, **Liste de competențe**. Modelul matricei de control al accesului acceptă controlul discreționar al accesului pentru că intrările în matrice sunt la discreția persoanelor care au autorizația de a completa tabelul. În matricea de control al accesului, competențele unui subiect sunt definite prin tripla (obiect, drepturi, număr aleator).

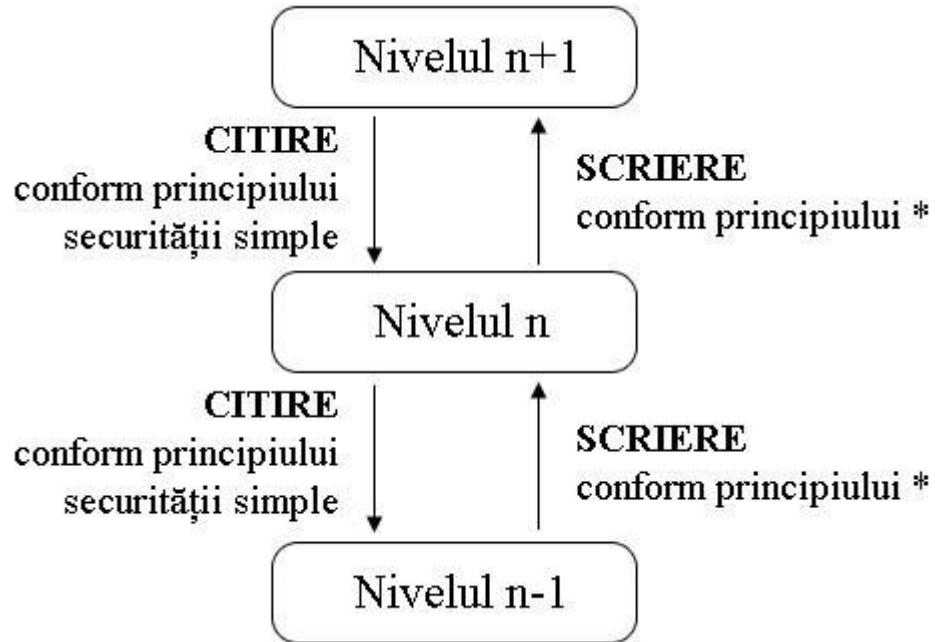
Modelul de securitate Bell-La Padula

Modelul Bell-La Padula este unul din cele mai cunoscute modele de securitate, propus de David Bell și Len La Padula în anul 1973. Constituie un model de reper în dezvoltarea tehniciilor de securitate a sistemelor informaționale, fiind utilizat la proiectarea sistemelor care manevrează informații distribuite pe mai multe niveluri. Din acest punct de vedere modelul Bell-La Padula este cunoscut ca modelul de securitate multi-nivel sau MLS (Multi-Level Secure).

Modelul de securitate Bell-La Padula este folosit în special pentru asigurarea confidențialității informațiilor. Formal, a introdus 3 principii:

- principiul securității simple, prin care nu-i este permis niciunui proces să citească date aflate pe un nivel superior lui. Este cunoscut și ca „Nu citi deasupra” (NRU – No Read Up);

- principiul *: niciun proces nu poate să scrie date pe un nivel aflat sub el. Este cunoscut și ca „Nu scrie dedesubt” (NWD – No Write Down);
- principiul securității discreționare: introduce o matrice de acces pentru a specifica controlul accesului discreționar. Este cunoscut și ca Trusted Subject (subiect de încredere). Prin acest principiu, subiectul de încredere violează principiul *, dar nu se abate de la scopul său.



Principiile modelului de securitate Bell-La Padula

Modelul fixează clasa de securitate pentru fiecare obiect $o \in O$ din sistem ($R(o \in O)$) și clasa de securitate pentru fiecare subiect $s \in S$ din sistem ($C(s \in S)$). Principiul acestui model constă din faptul că atât obiectele (mulțimea O) cât și subiectele (mulțimea S) sunt ierarhizate în raport cu o anumită combinație de parametri de securitate. Sunt utilizate două funcții ce au același codomeniu: mulțimea L ce reprezintă gama nivelurilor de securitate.

$R : O \rightarrow L$ – funcția de evaluare a obiectelor;

$C : S \rightarrow L$ – funcția de atribuire a utilizatorilor un domeniu.

Cele două funcții pot fi privite ca niște funcții ce calculează clasa de securitate a resurselor și respectiv a utilizatorilor.

Peste mulțimea L se definește o relație de ordine parțială astfel încât să existe o margine superioară și o margine inferioară I , cu următoarele proprietăți:

$$\forall x \in L : x \leq u;$$

$$\forall x \in L : I \leq x.$$

Astfel, $\forall x \in L : I \leq x \leq u$; unde I – marginea inferioară și u – marginea superioară. Putem analiza acum cele 3 principii introduse de modelul de securitate Bell-La Padula.

Principiul securității simple

Un subiect s va putea avea acces (drepturi de citire) la un obiect o numai dacă îndeplinește condiția:

$$C(s) \geq R(o)$$

Astfel un utilizator poate accesa informațiile pentru care posedă un nivel identic al clasei de securizare sau informațiile ce au clasa de securitate inferioară clasei utilizatorului. Utilizatorii nu pot citi informațiile ce au clasele de securitate superioare lor – principiul „Nu citi deasupra” – NRU).

Principiul *

Un subiect s va putea modifica (drepturi de scriere) un obiect o dacă îndeplinește condiția:

$$C(s) \leq R(o)$$

Un utilizator nu poate modifica un obiect al cărei clasă de securizare este inferioară utilizatorului – principiul „Nu scrie dedesubt” – NWD).

Un subiect s care are drept de acces de citire pentru un obiect o va putea să aibă drept de acces la scriere asupra unui alt obiect p numai dacă:

$$R(p) \geq R(o)$$

Această proprietate previne violarea securității informațiilor prin scurgerile de informații din nivelele superioare spre nivelele inferioare.

Principiul securității discreționare

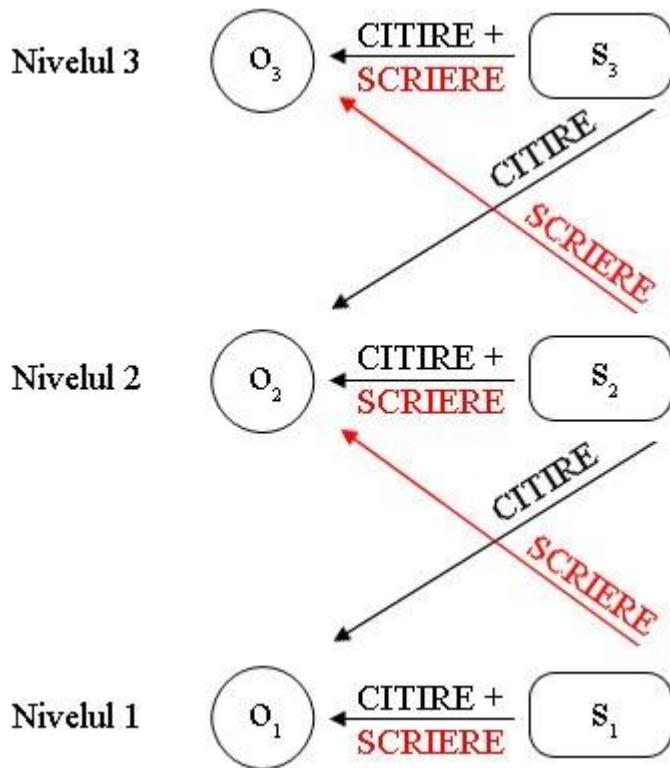
Principiul securității discreționare permite unui subiect s să scrie informația o la un nivel mai coborât ($C(s) \geq R(o)$) atâtă vreme cât s nu are acces la niciun obiect al cărui nivel să fie superior obiectului o.

Mulțimea operațiilor de acces este definită peste produsul cartezian $S \times O \times A$, unde:

- S – o mulțime de subiecți s;
- O – o mulțime de obiecte o;
- A – un set de operații de acces a.

Permișunile de acces sunt definite prin matricea de control al accesului M. Această matrice este realizată ținând cont de următoarele reguli:

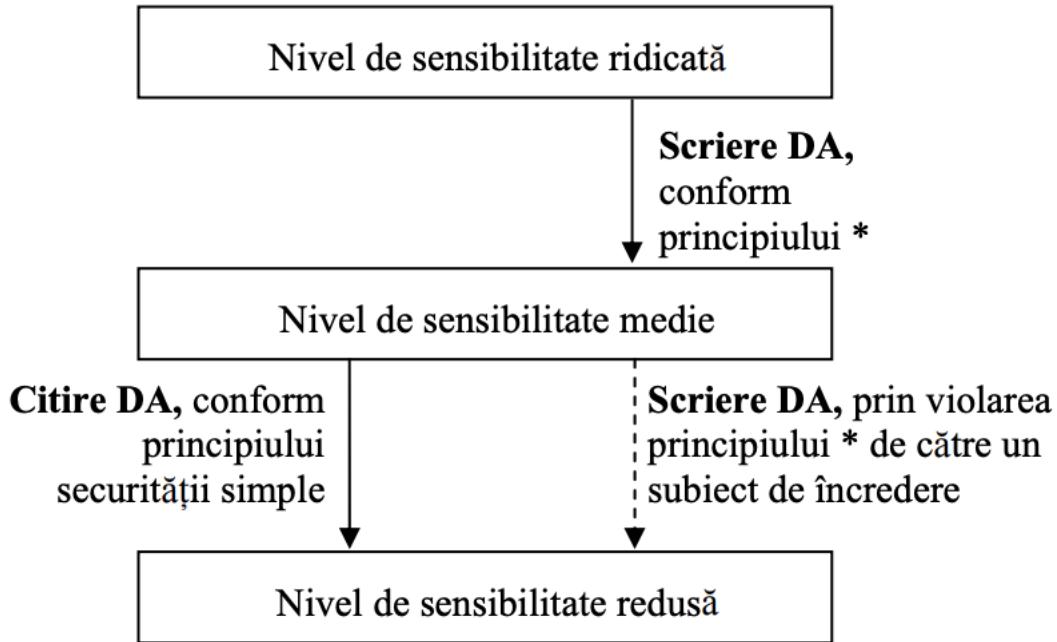
- pentru fiecare subiect se alocă câte un rând în matrice;
- pentru fiecare obiect se alocă câte o coloană în matrice;
- orice obiect din sistem are un proprietar;
- orice subiect din sistem are asociat un controlor.



Modelul Bell-La Padula

În model este introdusă noțiunea de stare sigură a unui sistem informațional și se demonstrează că fiecare tranziție de stare păstrează securitatea prin trecerea de la o stare sigură la o altă stare sigură. Un sistem este sigur dacă starea inițială a sistemului este sigură iar toate stările de tranziție sunt sigure.

O stare a sistemului este definită ca fiind sigură dacă singurele moduri de acces ale unui subiect la obiectele din sistem sunt în conformitate cu prevederile unei politici de securitate. Tranziția dintr-o stare a sistemului într-o altă stare se definește prin funcțiile de tranziție. Pentru a se determina corectitudinea accesării unui obiect de către un subiect, se compară domeniul subiectului cu clasificarea obiectului accesat. Astfel se observă dacă subiectul este autorizat pentru accesarea respectivului obiect.



Modelul Bell-La Padula cu cele trei principii

Modelul de securitate Bell-La Padula este considerat un model de referință pentru dezvoltarea tehniciilor de securitate a sistemelor informaționale, fiind primul model ce a introdus conceptul de securitate multi-nivel. Modelul este folosit cu precădere pentru confidențialitatea informației. Ca dezavantaj, modelul Bell-La Padula este static, fără o politică de reguli pentru crearea sau ștergerea subiecților și a obiectelor sau o politică de schimbare a modurilor de acces. De aceea modelul Bell-La Padula se poate aplica doar sistemelor ce asigură o securitate statică. Un alt dezavantaj este acela că un subiect aflat pe un nivel de securitate inferior poate detecta un obiect aflat pe un nivel de securitate superior. Acest lucru este un impediment atunci când se dorește ascunderea unui document, nu numai ascunderea conținutului acestuia.

Modelul de securitate Biba

Modelul Biba, al lui Ken Biba, se ocupă doar de integritatea sistemelor, nu și de confidențialitate. El se bazează pe observația că în multe cazuri confidențialitatea și integritatea sunt concepe duale: în timp ce prin confidențialitate se impun restricții celor ce pot citi un mesaj, prin integritate sunt controlați cei ce pot să scrie sau să modifice un mesaj.

În unele organizații guvernamentale sau comerciale există aplicații în care integritatea datelor este mult mai importantă decât confidențialitatea, ceea ce a făcut să apară modele formale ale integrității. Integritatea vizează trei scopuri principale:

- protejarea datelor împotriva modificărilor efectuate de utilizatorii neautorizați;
- protejarea datelor împotriva modificărilor neautorizate efectuate de utilizatori autorizați;
- asigurarea consistenței interne și externe a datelor.

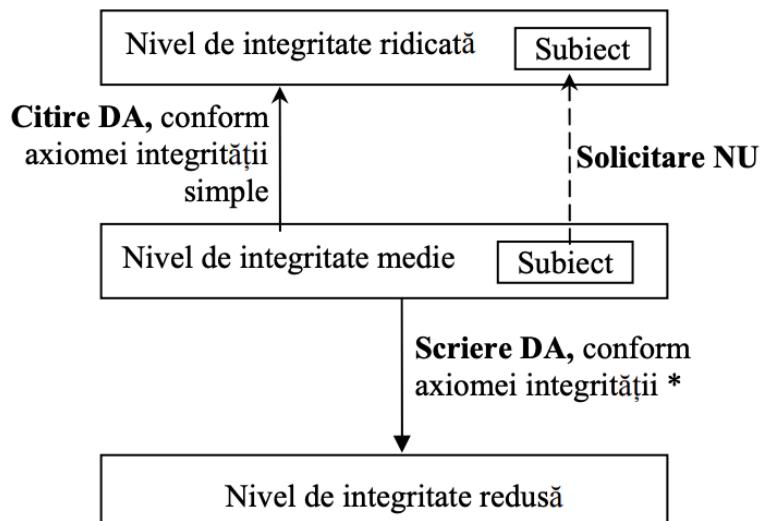
Modelul a fost realizat în 1977 ca unul al integrității datelor, așa cum modelul Bell-La Padula este cunoscut ca modelul confidențialității datelor. Modelul Biba este unul de tip rețea și folosește relația mai mic sau egal. O structură a rețelei este definită ca un ansamblu parțial ordonat cu cea mai mică limită superioară, LUB (Least Upper Bound), și cea mai mare limită inferioară, GLB (Greatest Lower

Bound). O rețea reprezintă un ansamblu de clase de integritate (CI) și de relații ordonate între aceste clase. Ea poate fi definită astfel:

$$(CI \leq LUB, GLB)$$

Așa cum Bell-La Padula operează cu niveluri diferite de sensibilitate, modelul Biba clasifică obiectele în diferite niveluri de integritate. Modelul enunță trei axiome ale integrității:

- axioma integrității simple. Ea stabilește că unui subiect aflat pe un anumit nivel de integritate nu-i este permis să observe (citească) un obiect de o integritate mai joasă (No Read Down, Nu citi dedesubt);
- axioma integrității * - stabilește că unui obiect situat pe un anumit nivel de integritate nu-i este permis să modifice (scrive) alt obiect situat pe un nivel mai înalt de integritate (No Write Up, Nu scrie deasupra);
- un subiect de pe un anumit nivel de integritate nu poate solicita un subiect situat pe un nivel de integritate superior.



Modelul Harrison-Ruzo-Ullman

Modelul de securitate Harrison Ruzo Ullman este un model general de securitate ce își propune să acopere deficiențele modelului Bell-La Padula. Acest model introduce un sistem de monitorizări.

Componentele modelului Harrison Ruzo Ullman sunt următoarele:

- S – o mulțime de subiecți s ; $s \in S$
- O – o mulțime de obiecte o ; $o \in O$
- R – o mulțime de reguli;
- M – o matrice de control al accesului.

Matricea de control al accesului este realizată în același fel ca și la celelalte modele de securitate – se ține cont de următoarele reguli:

- pentru fiecare subiect se alocă câte un rând în matrice;
- pentru fiecare obiect se alocă câte o coloană în matrice;
- orice obiect din sistem are un proprietar;
- orice subiect are asociat un controlor.

Modelul Harrison-Ruzzo-Ullman conține un număr de șase operații principale cu ajutorul cărora sunt gestionate obiectele și subiectii din sistem:

- crearea unui obiect sau a unui subiect;
- ștergerea unui obiect sau subiect;
- introducerea unor reguli în matricea de control al accesului;
- ștergerea unor reguli din matricea de control al accesului.

Atunci când un subiect $s_1 \in S$ creează un obiect o, va fi proprietarul acestui obiect și va deține implicit drepturile de citire și scriere asupra obiectului. Subiectul s_1 va putea să acorde discreționar, în raport cu obiectul o, drepturi de acces unui alt subiect $s_2 \in S$.

Modelul de securitate Harrison-Ruzzo-Ullman prezintă avantajul existenței sistemului de autorizări. Astfel un subiect poate crea diverse drepturi de acces pentru anumiți utilizatori din sistem asupra obiectelor față de care este proprietar.

Modele ale securității multilaterale

Deseori, în realitate, preocupările noastre s-au concentrat nu către prevenirea curgerii în jos a informațiilor, ci către stoparea fluxurilor între diferite compartimente. În astfel de sisteme, în locul frontierelor orizontale, aşa cum recomandă modelul Bell-La Padula, s-au creat altele verticale.

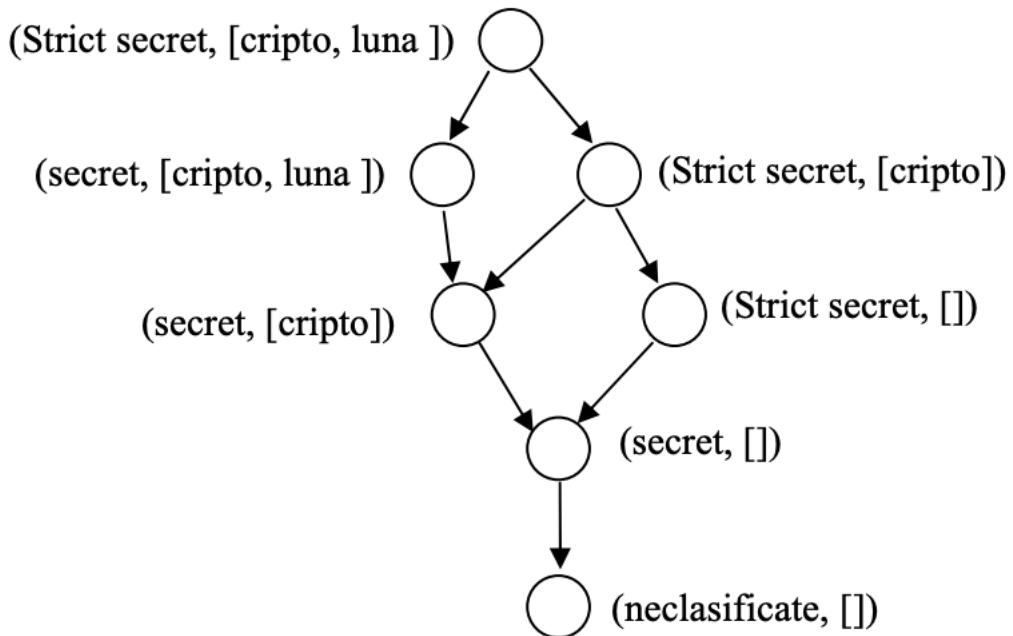
Acest control al fluxurilor informaționale laterale este unul organizațional, aşa cum este cel al organizațiilor secrete, pentru păstrarea în taină a numelor agenților care lucrează în alte țări, fără să fie cunoscuți de alte departamente speciale. La fel se întâmplă și în companii, unde separarea verticală a compartimentelor, după funcțiile îndeplinite (producție, comercială, personal-salarizare și.a.), conduce la o situație identică. Există cel puțin trei modele diferite de implementare a controlului accesului și de control al fluxurilor informaționale prin modelul securității multilaterale. Acestea sunt:

- compartimentarea, folosită de comunitatea serviciilor secrete;
- zidul chinezesc, folosit la descrierea mecanismelor utilizate pentru prevenirea conflictelor de interes în practicile profesionale;
- BMA (British Medical Association), dezvoltat pentru descrierea fluxurilor informaționale din domeniul sănătății, **conform** cu etica medicală.

Compartimentarea și modelul rețea

Ani mulți acest model a servit ca practică standard, în SUA și guvernele aliate, pentru restricționarea accesului la informații, prin folosirea cuvintelor-cod și a clasificărilor. Este arhicunoscut cuvântul-cod Ultra, folosit în cel de-al doilea război mondial, de către englezi și americani, pentru decriptarea mesajelor criptate de germani cu mașina Enigma. Cercul persoanelor cu acces la mesajele decriptate fiind foarte redus, numărul autorizațiilor pentru informații de pe cel mai înalt nivel de clasificare era mult mai mare. Prin folosirea cuvintelor-cod se creează o puternică subcompartimentare, chiar a categoriei strict secret și deasupra ei. Cuvintele-cod sunt folosite pentru crearea grupurilor de control al accesului printr-o variantă a modelului Bell-La Padula, numită modelul rețea. Clasificările, împreună cu cuvintele-cod, formează o rețea, conform figurii 5.6. Potrivit modelului, o persoană autorizată să aibă acces la informații SECRETE nu poate accesa informații SECRETE CRIPTO, dacă nu are și autorizație pentru CRIPTO. Ca un sistem să răspundă acestor cerințe, va trebui ca problemele

clasificării informațiilor, ale autorizării persoanelor și ale etichetelor ce însotesc informațiile să se transfere în politica de securitate pentru a defini țintele securității, modul de implementare și evaluare.



Model rețea cu etichete de securitate

Modelul zidului chinezesc

Modelul a fost realizat de Brewer și Nash. Numele provine de la faptul că firmele care prestează servicii financiare, cum sunt băncile de investiții, au normele lor interne pentru a preveni conflictul de interes, norme numite de autori zidul chinezesc. Aria de aplicare este, însă, mai largă. Se poate spune că toate firmele prestatoare de servicii au clienții lor și pentru a-i păstra se află într-o veritabilă competiție. O regulă tipică este următoarea: „un partener care a lucrat recent pentru o companie dintr-un anumit domeniu de activitate nu poate să aibă acces la documentele companiilor din acel domeniu”, cel puțin pentru o perioadă controlată de timp. Prin aceasta, caracteristica modelului zidului chinezesc constă într-un mix de libertate de opțiune și de control obligatoriu al accesului: oricine este liber să lucreze la orice companie, dar îndată ce a optat pentru una, se supune restricțiilor ce operează în domeniul respectiv de activitate.

Modelul zidului chinezesc introduce principiul separării obligațiilor de serviciu: un utilizator anume poate să prelucreze tranzacțiile A sau B, nu amândouă. Așadar, putem spune că modelul zidului chinezesc aduce elemente noi pe linia controlării accesului.

Modelul de securitate zidul chinezesc are următoarele componente:

- subiect: persoană angajată în organizație;
- obiect: datele referitoare la un client;
- datele organizației: datele tuturor clienților;
- clasa conflictelor de interes: organizațiile care sunt în competiție;
- etichete: datele organizației coroborate cu clasa conflictelor de interes;
- informații filtrate: acces fără restricții.

Proprietățile modelului zidului chinezesc sunt:

- accesul este permis dacă obiectul apelat se află în același set de date ale organizației pe care subiectul le-a accesat;
- un subiect poate să scrie într-un obiect numai dacă nu are acces de citire la un obiect ce face parte din datele organizației și nu sunt filtrate. Filtrarea informațiilor se face pentru a nu dezvăluia date importante despre obiecte;
- drepturile de acces ale subiecților se schimbă în mod dinamic, în funcție de context.

Modelul BMA (British Medical Association)

În domeniul medical sunt confruntări serioase privind tocmai sistemele de securitate a datelor pacienților. În efortul multor țări de a introduce carduri inteligente cu datele medicale personale, se înregistrează o puternică opoziție din partea publicului. Aceasta invocă vulnerabilitatea individului prin trecerea informațiilor despre anumite boli foarte grave, purtate până acum pe brățara de la mână, pe cartela intelligentă, ceea ce va face ca atunci când se va afla în avion, în țări străine, să fie foarte greu sau chiar imposibil să i se citească informațiile respective. O altă problemă se referă la păstrarea secretului datelor personale sau a unei părți dintre acestea. Cea mai mare temere vine din cauza proliferării practicilor de inginerie socială, putându-se afla cu multă ușurință date personale din baze de date medicale. Scopul modelului politicii de securitate BMA este acela de consolidare a principiului consimțământului pacientului și de a preveni accesul prea multor persoane la datele personale din bazele de date ce le conțin. Totul s-a rezumat la un nou sistem de codificare. Politica BMA se bazează pe nouă principii, formulate foarte pe scurt astfel:

- controlul accesului,
- deschiderea înregistrărilor,
- controlul modificărilor din liste,
- consimțământul și notificarea clientului,
- persistența,
- marcarea accesului pentru a servi ca probă în justiție,
- urmărirea fluxului informațiilor,
- controlul agregării informațiilor,
- încrederea în sistemele informatiche.

Unii autori susțin că politicile de securitate deseori înseamnă un abuz de mijloace pur manageriale, neglijându-se trei termeni precisi, utilizati pentru descrierea specificațiilor tehnice ale cerințelor sistemelor de securitate, prezentați în continuare. Modelul politicii de securitate este o declarație succintă a proprietăților sau principiilor securității, ca un sistem sau ca un tip generic de sistem.

Punctele sale esențiale pot fi consemnate în scris, pe cel mult o pagină, iar documentul respectiv prevede scopurile protecției unui sistem, agreate de întreaga comunitate sau de linia managerială a clienților. Un astfel de model constituie baza de pornire a analizelor formale matematice. Ținta securității este o descriere mult mai detaliată a mecanismelor de protecție oferite de o anumită variantă de implementare, precum și a modului în care ele concură la atingerea obiectivelor de control ale sistemului.

Ținta securității formează baza testării și evaluării produsului implementat. Profilul protecției, ca și ținta securității, exprimă o cale independentă de implementare, care să permită evaluări comparative ale produselor sau versiunilor lor.

Bibliografie

<http://atitech.unitbv.ro>

https://docs.slackware.com/ro:slackware:beginners_guide

<https://www.lifewire.com/unix-flavors-list-4094248>

<https://cybercrimesmd.wordpress.com/2014/05/12/cryptarea-simetrica-si-asimetrica/>

Brewer, D.F.C., Nash, M.J. (1989) The Chinese wall security policy

Universitatea din Bucureşti
Facultatea de Matematică-Informatică

Sisteme De Operare

Student: [REDACTED]

Anul II, Informatică ID

Bucureşti 2019

Cuprins

A - Partea I - Limbajul Shell scripting Unix/Linux

1.1. Introducere	3
1.2. Navigare filesystem.....	4
1.3. Administrarea fișierelor	6
1.4. Administrarea directoarelor	8
1.5. Variabile.....	10
1.6. Fișiere Script	12
1.7. Structuri de control.....	13
1.8. Aplicații.....	18

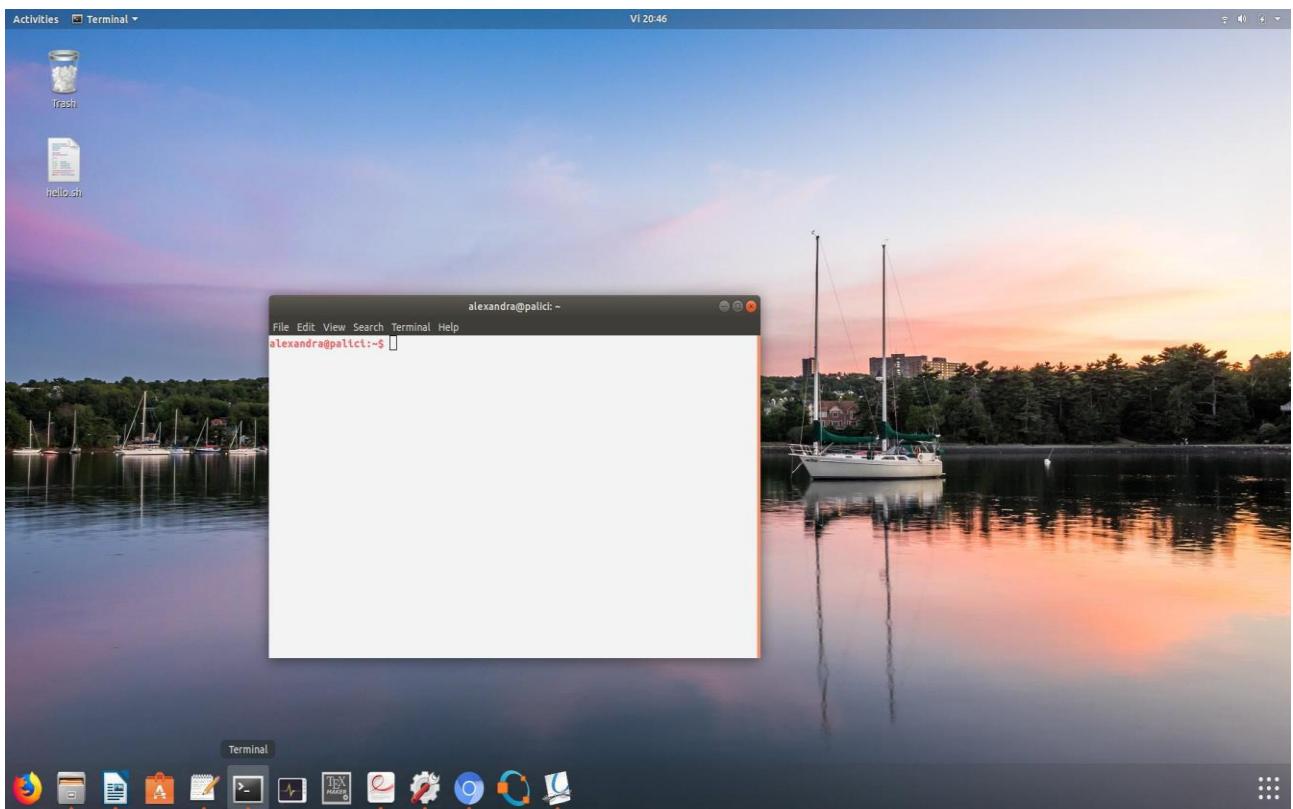
B - Partea II – Gestiunea proceselor in Unix/Linux

II.1. Introducere	19
II.2. Operații cu procese.....	20
II.3. Stările unui proces.....	22
II.4. Comutarea proceselor	23
II.5. Procese și Thread-uri in Unix/Linux.....	25
Bibliografie.....	28

A. Limbajul Shell scripting Unix/Linux

- ***Introducere***

GNU/Linux shell este o utilitate interactiva. Este folosit pentru a porni programe, a administra fisiere si procese in sistemul Linux. Partea centrala a shellului este consola de comanda care permite introducerea comenzilor prin text, interpreteaza comenzile si le executa in kernel.



Shell-ul contine un set de comenzi interne cu ajutorul caror userul poate copia, muta si redenumi fisiere, poate afisa sau opri programele care ruleaza. Shell-ul permite introducerea numelui unui program in consola dupa care il paseaza kernel-ului pentru a-l porni.

Exista diferite shell-uri cu diferite caracteristici disponibile intr-un sistem Linux. Unele sunt mai bune pentru a crea scripturi, altele pentru a administra procese. In mod predefinit shell-ul pentru majoritatea distributiilor Linux este bash. Acesta a fost dezvoltat in proiectul GNU de Bourne pentru a inlocui shell-ul Unix standard, Bourne Shell, de aici provine si numele din jocul de cuvinte “Bourne again shell”.

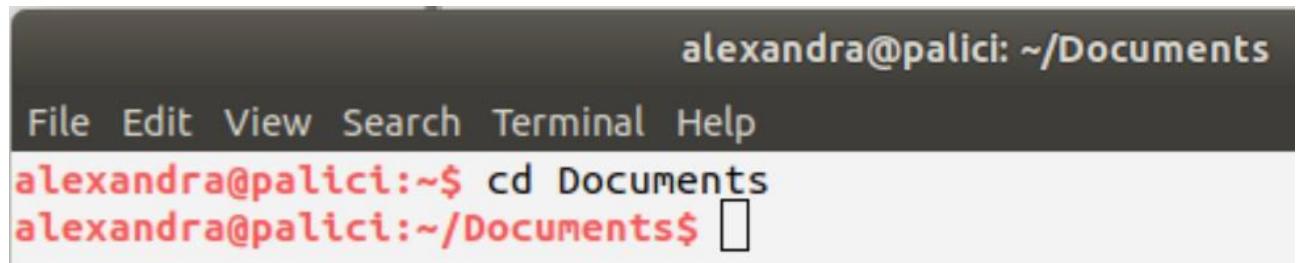
Alte shell-uri:

Shell	Descriere
ash	Are nevoie de putine resurse, compatibilitate cu bash
korn	Compatibil cu Bourne shell dar suporta caracteristici de programare avansate
tcsh	Incorporeaza elemente din limbajul C in script
zsh	Incorporeaza caracteristici din bash, tcsh si korn, caracteristici avansate de programare...

- **Navigare filesystem**

La deschiderea consolei, directorul predefinit este home.

Pentru a schimba directorul in care ne aflam folosim comanda “cd” urmata de numele destinatiei. Exemplu, pentru directorul Documents aflat in home (~):



The screenshot shows a terminal window with a dark header bar. The header bar contains the text "alexandra@palici: ~/Documents" in white. Below the header is a menu bar with options: File, Edit, View, Search, Terminal, Help. The main area of the terminal is white and contains red text. It shows the command "alexandra@palici:~\$ cd Documents" being typed and then executed, followed by the prompt "alexandra@palici:~/Documents\$". A small black square icon is located at the bottom right of the terminal window.

Daca nu se specifica destinatia, consola ne duce in directorul home.

Listarea fisierelor si directoarelor:

Pentru a lista continutul unui director, se foloseste comanda “ls” in directorul respectiv:

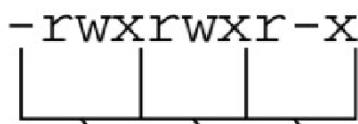
```
alexandra@palici: ~/Documents/FMI/Anul 2/sem_1
File Edit View Search Terminal Help
alexandra@palici:~/Documents/FMI/Anul 2/sem_1$ ls
CC                                         Probabilitati
Geometrie_comp                            SO
Licenta-Informatica-ID-sesiunea_2101-1002.pdf   TAP
ORAR_INFORMATICA_ID_sem_I_2018-2019.pdf      Tehnici_Web
alexandra@palici:~/Documents/FMI/Anul 2/sem_1$
```

“ls -l” pentru mai multe informatii despre fisiere:

```
alexandra@palici: ~/Documents/FMI/Anul2/sem_1
File Edit View Search Terminal Help
alexandra@palici:~/Documents/FMI/Anul2/sem_1$ ls -l
total 584
drwxrwxrwx  4 alexandra alexandra  4096 nov 15 19:34 CC
drwxr-xr-x  6 alexandra alexandra  4096 jan 28 09:42 Geometrie_comp
-rw-rw-r--  1 alexandra alexandra 469112 feb 28 09:16 Licenta-Informatica-ID-sesiunea_2101-1002.pdf
-rw-rw-r--  1 alexandra alexandra 92491 dec  6 15:29 ORAR_INFORMATICA_ID_sem_I_2018-2019.pdf
drwxrwxrwx  3 alexandra alexandra  4096 jan 13 20:04 Probabilitati
drwxr-xr-x  3 alexandra alexandra  4096 mar 28 20:14 SO
drwxrwxrwx  8 alexandra alexandra  4096 mar 20 18:53 TAP
drwxrwxrwx 12 alexandra alexandra  4096 jan 15 23:50 Tehnici_Web
alexandra@palici:~/Documents/FMI/Anul2/sem_1$
```

Exista foarte multi parametri pentru comanda “ls” in afara de “-l”, spre exemplu “-c” sorteaza dupa data ultimei modificari, “-u” afiseaza data ultimului acces etc.

Permsiuni ale fisierelor



permissions for everyone else

permissions for group members

permissions for the file owner

- - fisiere
- d directoare
- c caractere
- r pentru permisiunea de citire a obiectului
- w pentru scriere de citire a obiectului
- x pentru executare de citire a obiectului

```
alexandra@palici:~/SO$ ls
test1.sh
alexandra@palici:~/SO$ ls -l
total 4
-rw-r--r-- 1 alexandra alexandra 32 apr  4 20:11 test1.sh
alexandra@palici:~/SO$ chmod o+x test1.sh
alexandra@palici:~/SO$ ls -l
total 4
-rw-r--r-x 1 alexandra alexandra 32 apr  4 20:11 test1.sh
alexandra@palici:~/SO$ █
```

- u pentru user
- g pentru grup
- o pentru oricine altcineva
- a pentru toti cei mai sus mentionati

- ***Administrarea fisierelor***

Crearea fisierelor:

Pentru a crea un fisier gol se poate folosi comanda “touch”.

Copierea fisierelor:

Copierea se face prin comanda ”cp sursa destinatie”. Comanda ”cp” copiază fisierul sursă într-un nou fisier cu numele specificat ca destinatie. Dacă fisierul destinatie există deja, consola întreabă dacă se dorește suprascrierea fisierelor.

```
alexandra@palici:~/SO$ touch test
alexandra@palici:~/SO$ ls -l
total 0
-rw-r--r-- 1 alexandra alexandra 0 mar 28 20:46 test
alexandra@palici:~/SO$ cp test test1
alexandra@palici:~/SO$ ls -l
total 0
-rw-r--r-- 1 alexandra alexandra 0 mar 28 20:46 test
-rw-r--r-- 1 alexandra alexandra 0 mar 28 20:47 test1
alexandra@palici:~/SO$ 
```



Redenumirea fisierelor:

Redenumirea se face prin mutarea fisierului sau directorului prin comanda "mv":

```
alexandra@palici: ~/SO
File Edit View Search Terminal Help
alexandra@palici:~/SO$ ls
test test1
alexandra@palici:~/SO$ mv test test2
alexandra@palici:~/SO$ ls
test1 test2
alexandra@palici:~/SO$ 
```

Stergerea fisierelor:

"rm" este folosit pentru stergerea unui fisier:

```
alexandra@palici: ~/SO
File Edit View Search Terminal Help
alexandra@palici:~/SO$ ls
test1 test2
alexandra@palici:~/SO$ rm test2
alexandra@palici:~/SO$ ls
test1
alexandra@palici:~/SO$ 
```

Vizualizarea unui fisier text

Comanda “cat” este foarte buna pentru a afisa continutul unui fisier text. Ea poate fi folosita impreuna cu alti parametri pentru a afla mai multe informatii despre continutul fisierului. Spre exemplu “-n” afiseaza numerele liniilor:



```
Open ▾ test.txt
~/SO
Hello
World!
alexandra@palici: ~/SO
File Edit View Search Terminal Help
alexandra@palici:~/SO$ cat -n test.txt
1 Hello
2
3 World!
alexandra@palici:~/SO$ 
```

- **Administrarea directoarelor**

Crearea si stergerea directoarelor:

```
alexandra@palici: ~/SO
File Edit View Search Terminal Help
alexandra@palici:~/SO$ mkdir dir1
alexandra@palici:~/SO$ ls
dir1 test1
alexandra@palici:~/SO$ rmdir dir1
alexandra@palici:~/SO$ ls
test1
alexandra@palici:~/SO$ 
```

- **Monitorizarea programelor:**

Pentru a examina procesele care ruleaza se foloseste comanda basic “ps”:

```
alexandra@palici: ~/SO
File Edit View Search Terminal Help
alexandra@palici:~/SO$ ps
 PID TTY          TIME CMD
 998 pts/0        00:00:00 bash
11603 pts/0        00:00:00 ps
alexandra@palici:~/SO$ 
```

PID este proces ID, TTY terminalul, TIME timpul folosit de CPU pentru executie. “ps” afiseaza informatia corespunzatoare momentului executiei comenzii. Pentru a vizualiza dinamic procesele se foloseste “top”:

```

top - 22:27:28 up 9:10, 1 user, load average: 1,17, 0,74, 0,82
Tasks: 309 total, 1 running, 239 sleeping, 0 stopped, 0 zombie
%Cpu(s): 7,3 us, 1,7 sy, 0,0 ni, 90,9 id, 0,0 wa, 0,0 hi, 0,1 si, 0,0 st
KiB Mem : 16303748 total, 12187144 free, 1609268 used, 2507336 buff/cache
KiB Swap: 2097148 total, 2097148 free, 0 used. 13807132 avail Mem

```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
16794	alexand+	20	0	2090108	254128	105376	S	45,0	1,6	0:19.33	chrome
16005	alexand+	20	0	1078352	99672	74148	S	13,6	0,6	0:05.95	chrome
2867	alexand+	20	0	3857320	207236	74756	S	6,6	1,3	4:28.73	gnome-shell
2728	alexand+	20	0	645884	113024	95296	S	4,0	0,7	3:24.00	Xorg
2890	alexand+	9	-11	2222984	13756	10040	S	3,6	0,1	0:03.12	pulseaudio
15861	alexand+	20	0	4275140	230220	131052	S	2,3	1,4	0:04.86	chrome
16821	alexand+	20	0	1721524	94740	71900	S	0,7	0,6	0:00.73	chrome
16865	alexand+	20	0	52860	4304	3616	R	0,7	0,0	0:00.17	top
8	root	20	0	0	0	0	I	0,3	0,0	0:04.31	rcu_sched
2685	root	20	0	1400536	11152	9604	S	0,3	0,1	0:07.50	teamviewerd
23755	root	20	0	0	0	0	I	0,3	0,0	0:04.85	kworker/0:2
1	root	20	0	225644	9532	6784	S	0,0	0,1	0:07.05	systemd
2	root	20	0	0	0	0	S	0,0	0,0	0:00.01	kthreadd
4	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	kworker/0:+
6	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	mm_percpu_+
7	root	20	0	0	0	0	S	0,0	0,0	0:00.09	ksoftirqd/0
9	root	20	0	0	0	0	I	0,0	0,0	0:00.00	rcu_bh

Pentru a opri un proces se poate folosi “kill” cu argument PID sau “killall” cu numele proceselor.

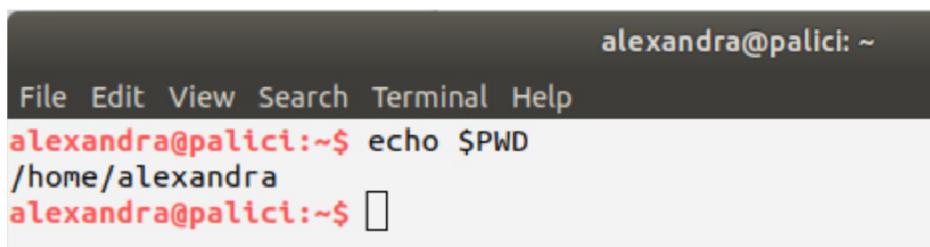
- **Variabile**

1. Variabilele de sistem permit stocarea de informatii despre user, sistem, shell etc.
2. Variabilele sunt de tipul global sau local.
3. Variabilele de mediu globale sunt vizibile din sesiunea shell, si din orice proces copil pe care shell-ul le genereaza.
Variabilele locale sunt disponibile numai în shell-ul care le creează.

```

USER=alexandra
DESKTOP_SESSION=ubuntu
QT_IM_MODULE=xim
TEXTDOMAINDIR=/usr/share/locale/
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/a15ae8c7_869e_4d15_86e9_cd08cd328097
PWD=/home/alexandra
HOME=/home/alexandra
TEXTDOMAIN=im-config
SSH_AGENT_PID=2764
QT_ACCESSIBILITY=1
XDG_SESSION_TYPE=x11
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share:/usr/share:/var/lib/snapd/desktop
XDG_SESSION_DESKTOP=ubuntu
LC_ADDRESS=ro_RO.UTF-8
LC_NUMERIC=ro_RO.UTF-8
GTK_MODULES=gail:atk-bridge
WINDOWPATH=2
TERM=xterm-256color
SHELL=/bin/bash
VTE_VERSION=5202
QT_IM_MODULE=ibus
XMODIFIERS=@im=ibus
IM_CONFIG_PHASE=2
XDG_CURRENT_DESKTOP=ubuntu:GNOME
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
GNOME_TERMINAL_SERVICE=:1.95
XDG_SEAT=seat0
SHLVL=1
LC_TELEPHONE=ro_RO.UTF-8
GDMSESSION=ubuntu
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
LOGNAME=alexandra
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
XDG_RUNTIME_DIR=/run/user/1000
XAUTHORITY=/run/user/1000/gdm/Xauthority
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
LC_IDENTIFICATION=ro_RO.UTF-8
SESSION_MANAGER=local/palici:@/tmp/.ICE-unix/2669,unix/palici:/tmp/.ICE-unix/2669
LESSOPEN=| /usr/bin/lesspipe %
GTK_IM_MODULE=ibus
LC_TIME=ro_RO.UTF-8
_=~/usr/bin/printenv
alexandra@palici:~$ 

```



“alias” permite crearea unui nume alias pentru comenzi pentru a facilita scrierea lor.
Lista de alias-uri:

```
alexandra@palici: ~
File Edit View Search Terminal Help
alexandra@palici:~$ alias -p
alias alert='notify-send --urgency=low -i "$( [ $? = 0 ] && echo terminal || echo error)" "$(history|tail -n1|sed -e '\''s/^[\s*][0-9]+\s*//;s/[;&]\s*alert$/'\')"
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
```

- **Fisiere Script**

Un Script se scrie intr-un fisier text in care se introduc comenzi. Mai intai trebuie specificat Shell-ul folosit:

```
#!/bin/bash
```

"#" este folosit pentru a comenta.

Fiecare comanda este scrisa pe randuri diferite sau pot fi separate de ";". Shell-ul va procesa fiecare comanda in ordinea aparitiei.

The screenshot shows a terminal window titled "test1.sh" with the following content:

```
alexandra@palici:~/SO
File Edit View Search Terminal Help
alexandra@palici:~/SO$ ls -l
total 4
-rw-r--r-x 1 alexandra alexandra 32 apr  4 20:11 test1.sh
alexandra@palici:~/SO$ chmod u+x test1.sh
alexandra@palici:~/SO$ ls -l
total 4
-rwxr--r-x 1 alexandra alexandra 32 apr  4 20:11 test1.sh
alexandra@palici:~/SO$ ./test1.sh
hello world
alexandra@palici:~/SO$
```

Below the terminal window, there is a file manager interface showing a file named "test1.sh".

- **Structuri de control**

Shell-ul proceseaza fiecare comanda individual in ordinea in care apar in script. Structurile de control permit alterarea ordinii in care comenzi sunt executate.

Structura if-then:

```
if command  
then  
    commands  
fi
```

Shell-ul ruleaza comanda de pe linia if, daca exit status este zero atunci se executa comenzile de sub then, altfel nu se executa si se trece la urmatoarea comanda in script.

```
alexandra@palici:~/S0$ ./test1  
luni 6 mai 2019, 20:49:48 +0300  
it worked  
alexandra@palici:~/S0$ cat test1  
#!/bin/bash  
# testing the if statement  
if date  
then  
echo "it worked"  
fi  
alexandra@palici:~/S0$ █
```

Daca nu vrem asta, si vrem ca shell-ul sa execute o alta comanda in cazul in care exit status e zero, putem folosi:

If-then-else:

```
if command  
then  
    commands  
else  
    commands  
fi
```

```
alexandra@palici:~/S0$ ./test2
alexandra:x:1000:1000:Alexandra,,,:/home/alexandra:/bin/bash
The files for user alexandra are:
/home/alexandra/.bash_history /home/alexandra/.bashrc
/home/alexandra/.bash_logout
alexandra@palici:~/S0$ cat test2
#!/bin/bash
# testing the else section
testuser=alexandra
if grep $testuser /etc/passwd
then
echo The files for user $testuser are:
ls -a /home/$testuser/.b*
else
echo "The user name $testuser doesn't exist on this system"
fi
alexandra@palici:~/S0$ █
```

Nesting ifs:

Atunci cand trebuie sa verificam mai multe situatii in script, putem folosi “elif” in loc de a repeta “if-then”:

```
if command1
then
    commands
elif command2
then
    more commands
fi
```

Comanda Test:

Comanda test permite testarea mai multor conditii intr-un “if-then”:

```
if test condition
then
    commands
fi
```

Comenzi compuse:

Constructia "if-then" permite folosirea logicii Boole pentru a combina conditii:

- [condition1] && [condition2]
- [condition1] || [condition2]

```
alexandra@palici:~/SO$ ./test3
I can't write to the file
alexandra@palici:~/SO$ cat test3
#!/bin/bash
# testing compound comparisons
if [ -d $HOME ] && [ -w $HOME/testing ]
then
echo "The file exists and you can write to it"
else
echo "I can't write to the file"
fi
alexandra@palici:~/SO$
```

Comanda case:

Comanda "case" poate fi folosita atunci cand vrem sa evaluam valoarea unei variabile. Aceasta comanda compara variabila specificata cu diferite pattern-uri. Daca variabila se potriveste cu pattern-ul respectiv, shell-ul efectueaza comenziile specificate:

```
case variable in
pattern1 | pattern2) commands1;;
pattern3) commands2;;
*) default commands;;
esac
```

```
alexandra@palici:~/SO$ ./test4
Welcome, alexandra
Please enjoy your visit
alexandra@palici:~/SO$ cat test4
#!/bin/bash
# using the case command
case $USER in
alexandra | barbara)
echo "Welcome, $USER"
echo "Please enjoy your visit";;
testing)
echo "Special testing account";;
jessica)
echo "Don't forget to log off when you're done";;
*)
echo "Sorry, you're not allowed here";;
esac
alexandra@palici:~/SO$
```

Comanda for:

Shell-ul permite, prin comanda “for”, crearea buclelor iterative printr-o serie de valori.

Format:

```
alexandra@palici:~/SO$ cat test
#!/bin/bash
# basic for command
for test in Alabama Alaska Arizona Arkansas California Colorado
do
    echo The next state is $test
done
alexandra@palici:~/SO$ ./test
The next state is Alabama
The next state is Alaska
The next state is Arizona
The next state is Arkansas
The next state is California
The next state is Colorado
alexandra@palici:~/SO$ █
```

Limbajul C pentru comenzi:

Shell-ul permite folosirea metodei limbajului C pentru comanda “for”:

```
alexandra@palici:~/SO$ cat test
#!/bin/bash
# testing the C-style for loop
for (( i=1; i <= 10; i++ ))
do
    echo "The next number is $i"
done
alexandra@palici:~/SO$ ./test
The next number is 1
The next number is 2
The next number is 3
The next number is 4
The next number is 5
The next number is 6
The next number is 7
The next number is 8
The next number is 9
The next number is 10
alexandra@palici:~/SO$ █
```

Comanda while:

```
while test command  
do  
    other commands  
done
```

```
alexandra@palici:~/SO$ cat test  
#!/bin/bash  
# while command test  
var1=10  
while [ $var1 -gt 0 ]  
do  
echo $var1  
var1=$(( $var1 - 1 ))  
done  
alexandra@palici:~/SO$ ./test  
10  
9  
8  
7  
6  
5  
4  
3  
2  
1  
alexandra@palici:~/SO$ █
```

Comanda until:

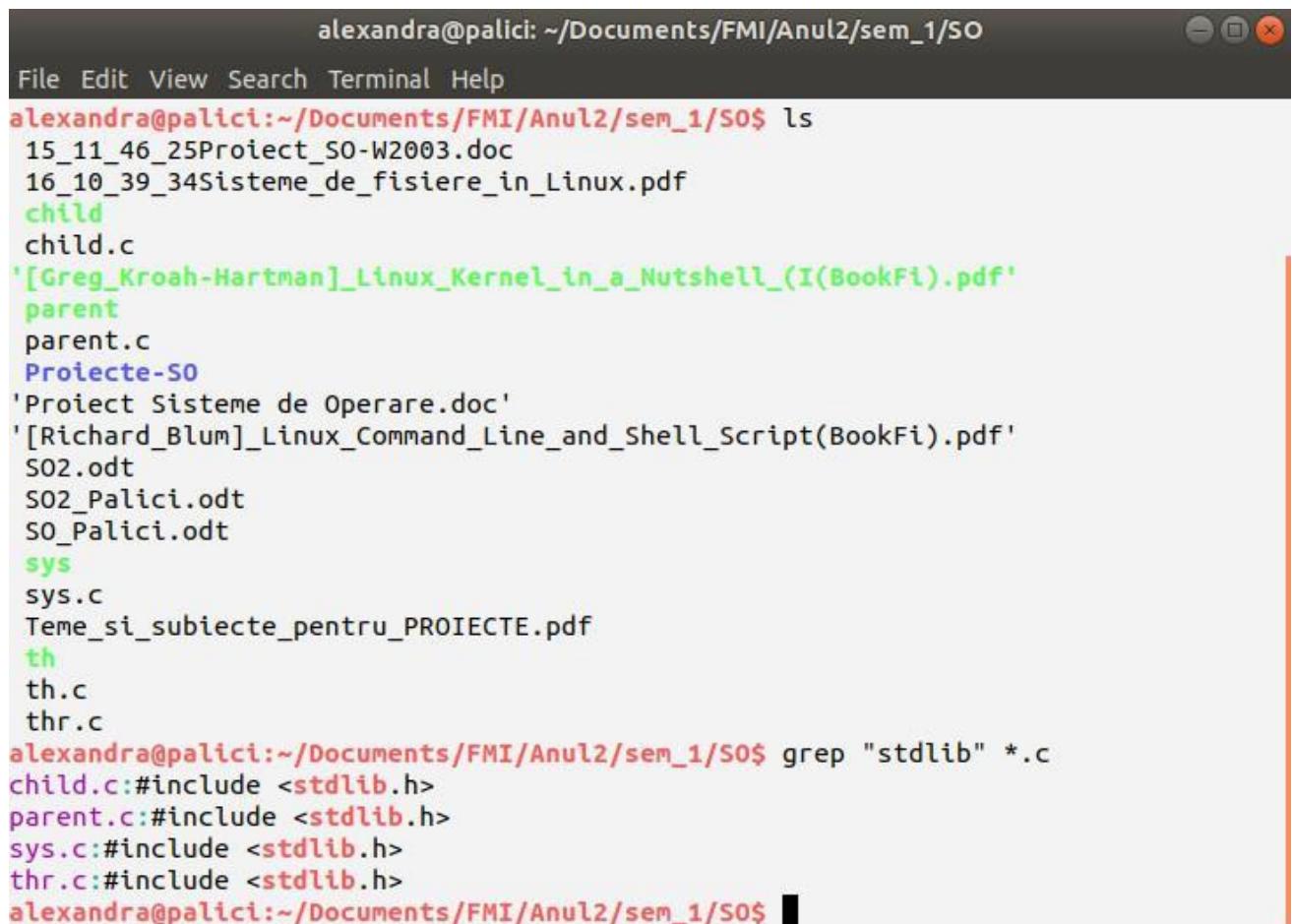
```
until test commands  
do  
    other commands  
done
```

```
alexandra@palici:~/SO$ cat test  
#!/bin/bash  
# using the until command  
var1=100  
until [ $var1 -eq 0 ]  
do  
echo $var1  
var1=$(( $var1 - 25 ))  
done  
alexandra@palici:~/SO$ ./test  
100  
75  
50  
25  
alexandra@palici:~/SO$ █
```

Aplicatii

Shell-ul are diverse functionalitati in particular administarea fisierelor. De exemplu prin comanda “grep” putem identifica aparitiile unui sir de caractere in cadrul fisierelor dintr-un director. Exemplu:

Cautam “stdlib” in toate fisierele din directorul curent cu extensia .c



The screenshot shows a terminal window with the following content:

```
alexandra@palici: ~/Documents/FMI/Anul2/sem_1/SO
File Edit View Search Terminal Help
alexandra@palici:~/Documents/FMI/Anul2/sem_1/SO$ ls
15_11_46_25Proiect_SO-W2003.doc
16_10_39_34Sisteme_de_fisiere_in_Linux.pdf
child
child.c
'[Greg_Kroah-Hartman]_Linux_Kernel_in_a_Nutshell_(I(BookFi).pdf'
parent
parent.c
Proiecte-SO
'Proiect Sisteme de Operare.doc'
'[Richard_Blum]_Linux_Command_Line_and_Shell_Script(BookFi).pdf'
SO2.odt
SO2_Palici.odt
SO_Palici.odt
sys
sys.c
Teme_si_subiecte_pentru_PROIECTE.pdf
th
th.c
thr.c
alexandra@palici:~/Documents/FMI/Anul2/sem_1/SO$ grep "stdlib" *.c
child.c:#include <stdlib.h>
parent.c:#include <stdlib.h>
sys.c:#include <stdlib.h>
thr.c:#include <stdlib.h>
alexandra@palici:~/Documents/FMI/Anul2/sem_1/SO$
```

B. Gestiunea proceselor in Unix/Linux

- **Introducere**

Un **proces** este un program în execuție. Procesele sunt unitatea primitivă prin care sistemul de operare alocă resurse utilizatorilor. Orice proces are un spațiu de adrese și unul sau mai multe fizice de execuție.

La crearea unui nou proces cu fork- procesul copil va avea o copie a resurselor procesului părinte.

Dacă se dorește înlocuirea imaginii procesului copil acesta poate fi schimbată prin apelarea unei funcții din familia exec*.

Un proces se poate crea și prin folosirea funcției de bibliotecă "system".

Apelul acestei funcții are ca efect execuția ca o comandă shell a comenzii reprezentate prin sirul de caractere command.

Rularea unui program executabil Exemplu:

The screenshot shows a Linux desktop environment. In the background, a file manager window is open, displaying four files named test, test2, test3, and test4. In the foreground, a terminal window is active, showing the following session:

```
alexandra@palici: ~/Documents/FMI/Anul2/sem_1/SO
File Edit View Search Terminal Help
alexandra@palici:~/Documents/FMI/Anul2/sem_1/SO$ gcc sys.c -o sys
alexandra@palici:~/Documents/FMI/Anul2/sem_1/SO$ ./sys
test test2 test3 test4
alexandra@palici:~/Documents/FMI/Anul2/sem_1/SO$ 
```

Echivalent cu:

```
alexandra@palici:~/Documents/FMI/Anul2/sem_1/SO$ sh -c "ls /home/alexandra/SO"
test test2 test3 test4
alexandra@palici:~/Documents/FMI/Anul2/sem_1/SO$ █
```

- **Operatii cu procese**

- Crearea unui proces

În UNIX singura modalitate de creare a unui proces este prin apelul de sistem **fork**. Efectul este crearea unui nou proces - procesul copil, copie a celui care a apelat *fork* - procesul părinte. Copilul primește un nou PID de la sistemul de operare.

- Așteptarea terminării unui proces

Familia de funcții **wait** suspendă execuția procesului apelant până când procesul (procesele) specificate în argumente fie s-au terminat fie au fost opriți.

- Terminarea unui proces

Pentru terminarea procesului curent, Linux pune la dispoziție apelul de sistem **exit**.

Activities Terminal ▾

Open ▾

parent.c: the parent program

```
// parent.c: the parent program

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>

int main (int argc, char **argv)
{
    int i = 0;
    long sum;
    int pid;
    int status, ret;
    char *myargs [] = { NULL };
    char *myenv [] = { NULL };

    printf ("Parent: Hello, World!\n");

    pid = fork ();

    if (pid == 0) {
        // I am the child
        execve ("child", myargs, myenv);
    }

    // I am the parent
    printf ("Parent: Waiting for Child to complete.\n");
    if ((ret = waitpid (pid, &status, 0)) == -1)
        printf ("parent:error\n");

    if (ret == pid)
        printf ("Parent: Child process waited for.\n");
}
```

child.c: the child program

```
// child.c: the child program

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

#define A 500
#define B 600
#define C 700

int main (int argc, char **argv)
{
    int i, j;
    long sum;

    // Some arbitrary work done by the child
    printf ("Child: Hello, World!\n");

    for (j = 0; j < 30; j++ ) {
        for (i = 0; i < 900000; i++) {
            sum = A * i + B * i * i + C;
            sum %= 543;
        }
    }

    printf ("Child: Work completed!\n");
    printf ("Child: Bye now.\n");
}

exit (0);
}
```

File Edit View Search Terminal Help

alexandra@pallici:~/Documents/FNI/Anul2/sem_1/50\$ gcc parent.c -o parent
alexandra@pallici:~/Documents/FNI/Anul2/sem_1/50\$ gcc child.c -o child
alexandra@pallici:~/Documents/FNI/Anul2/sem_1/50\$./parent
Hello, World!
Parent: Waiting for Child to complete.
Child: Hello, World!
Child: Work completed!
Child: Bye now.
Parent: Child process waited for.
alexandra@pallici:~/Documents/FNI/Anul2/sem_1/50\$

C ▾ TabWidth:8 ▾ Ln 21, Col 19 ▾ INS

C ▾ TabWidth:8 ▾ Ln 1, Col 5 ▾ INS

- **Starile unui proces**

La orice moment de timp un proces se poate afla intr-una din urmatoarele stari:

- *blocat (waiting)* - se zice ca procesul doarme (sleeps); in aceasta stare un proces este blocat in asteptarea unui eveniment (o operatie I/O, primirea unui semnal etc.)
- *gata de executie (ready)* - un proces este pregatit pentru a fi rulat; trebuie doar sa i se acorde accesul la procesor
- *in executie (running)* - procesul ruleaza si are acces complet la procesor
- *zombie* – executia procesului a fost opresa dar acesta inca apare in lista proceselor

In vizualizarea dinamica a proceselor prin comanda “top” in terminal, coloana S indica statusul procesului.

```

alexandra@palici: ~
File Edit View Search Terminal Help
top - 22:49:15 up 13:19, 1 user, load average: 2,34, 1,69, 1,16
Tasks: 337 total, 3 running, 261 sleeping, 0 stopped, 0 zombie
%Cpu(s): 21,1 us, 3,5 sy, 0,0 ni, 74,9 id, 0,3 wa, 0,0 hi, 0,2 si, 0,0 st
KiB Mem : 16303724 total, 8914520 free, 2642628 used, 4746576 buff/cache
KiB Swap: 2097148 total, 2097148 free, 0 used. 12227936 avail Mem

      PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
22305 alexand+  20   0 1821744 179180 110140 R  71,9  1,1  21:08.97 chrome
31786 alexand+  20   0 1939100 266512 105672 S  50,0  1,6   0:35.13 chrome
2013 alexand+  20   0 1092536 168340 101960 S  26,8  1,0   7:22.32 chrome
3508 alexand+  20   0 3966788 234760  86808 R  26,5  1,4  16:13.97 gnome-shell
3371 alexand+  20   0 688492 124148 103464 S  7,9  0,8  11:28.56 Xorg
4400 alexand+  9 -11 2222984 14264 10540 S  3,6  0,1   1:48.11 pulseaudio
1871 alexand+  20   0 4667600 307512 143332 S  3,3  1,9   4:51.18 chrome
2018 alexand+  20   0 1558052 169416  57472 S  2,0  1,0   0:31.52 chrome
486 root      -51   0      0      0      0 S  0,7  0,0   1:15.03 irq/51-DLL+
16697 alexand+  20   0 1678208  80376  64072 S  0,7  0,5   0:14.48 chrome
19559 alexand+  20   0 52960   4588   3724 S  0,7  0,0   0:02.05 top
22279 alexand+  20   0 1750800 130028  87136 S  0,7  0,8   0:23.83 chrome
23332 root      20   0      0      0      0 D  0,7  0,0   0:01.39 kworker/u1+
32508 alexand+  20   0 52968   4524   3656 R  0,7  0,0   0:00.27 top
1 root      -51   0      0      0      0 S  0,3  0,1   0:24.85 systemd
444 root      -51   0      0      0      0 S  0,3  0,0   0:18.66 irq/138-iw+
548 root      -2   0      0      0      0 S  0,3  0,0   0:13.09 i915/signa+

```

Pentru a opri un proces se poate folosi "kill" cu argument PID sau "killall" cu numele proceselor.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
29663	alexand+	20	0	1889832	290256	152876	S	6,6	1,8	0:07.62	firefox
29836	alexand+	20	0	1543580	108904	83420	S	1,3	0,7	0:00.53	WebExtensi+
29905	alexand+	20	0	1583952	148168	108576	S	1,3	0,9	0:01.41	Web Content
22305	alexand+	20	0	1826592	176760	111724	S	1,0	1,1	24:43.49	chrome
31177	alexand+	20	0	1519952	81732	63792	S	0,7	0,5	0:00.16	Web Content
635	systemd+	20	0	71192	6696	5480	S	0,3	0,0	0:01.84	systemd-re+
1507	root	20	0	0	0	0	I	0,3	0,0	0:04.15	kworker/0:0
27187	alexand+	20	0	52972	4608	3732	R	0,3	0,0	0:00.49	top
1	root	20	0	225876	9688	6760	S	0,0	0,1	0:26.13	systemd
2	root	20	0	0	0	0	S	0,0	0,0	0:00.05	kthreadd
4	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	kworker/0:+
6	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	mm_percpu_+
7	root	20	0	0	0	0	S	0,0	0,0	0:00.42	ksoftirqd/0
8	root	20	0	0	0	0	I	0,0	0,0	0:19.13	rcu_sched
9	root	20	0	0	0	0	I	0,0	0,0	0:00.00	rcu_bh
10	root	rt	0	0	0	0	S	0,0	0,0	0:00.06	migration/0
11	root	rt	0	0	0	0	S	0,0	0,0	0:00.06	watchdog/0

- **Comutarea proceselor**

Inlocuirea unui proces cu un altul in cadrul unui procesor se numeste comutare de context (task switching). Aceasta este realizata de planificatorul de procese (scheduler). Acesta alege din coada de procese **ready** procesul cel mai potrivit pe baza anumitor criterii si ii ofera acestuia procesorul.

Procesele disponuie de o prioritate la scheduling. La o comutare de context, un proces prioritari va acapara procesorul in dauna unuia mai putin prioritari. In Linux/Unix prioritatea unui proces poate fi alterata cu ajutorul comenzii "**nice**".

Intucat un singur proces poate folosi CPU la un anumit moment de timp, este necesara alocarea de timp CPU de catre kernel pentru fiecare proces in parte.

In acest context prioritatea unui proces se defineste ca timpul CPU alocat procesului relativ la celalalte procese. Prioritatea este o valoare intreaga de la -20 (prioritate mare) la +20 (prioritate mica). Valoarea default pentru prioritate cu care shell-ul porneste toate procesele este 0.

```

alexandra@palici: ~/SO
File Edit View Search Terminal Help
0 1000 3151 2921 20 0 206432 6904 poll_s Sl  tty2 0:02 /usr/lib/ib
0 1000 6250 2710 20 0 593984 25888 poll_s Sl+ tty2 0:00 update-noti
0 1000 6253 2710 20 0 1250984 150668 poll_s SLL+ tty2 0:02 /usr/bin/gn
1 1000 6498 13246 20 0 1810824 191388 futex_ Sl+ tty2 0:28 /snap/chrom
1 1000 6513 13246 20 0 1365892 45580 futex_ Sl+ tty2 0:00 /snap/chrom
0 1000 8588 2710 20 0 791044 32036 poll_s Sl+ tty2 0:00 /usr/lib/de
0 1000 13042 13033 20 0 31240 5148 wait Ss pts/0 0:00 bash
4 1000 13130 2833 20 0 3974572 240536 poll_s Sl+ tty2 0:31 /snap/chrom
4 1000 13244 13130 20 0 411116 44944 wait S+ tty2 0:00 /snap/chrom
5 1000 13246 13244 20 0 411116 12528 poll_s S+ tty2 0:00 /snap/chrom
0 1000 13269 13130 20 0 1014940 114560 poll_s Sl+ tty2 0:40 /snap/chrom
0 1000 13275 13130 20 0 1616556 89612 futex_ Sl+ tty2 0:04 /snap/chrom
1 1000 13451 13246 20 0 1780904 177104 futex_ Sl+ tty2 0:27 /snap/chrom
0 1000 13744 13033 20 0 31240 5140 wait Ss pts/1 0:00 bash
1 1000 14133 13246 20 0 1784944 178988 futex_ Sl+ tty2 0:17 /snap/chrom
1 1000 14148 13246 20 0 1728560 123876 futex_ Sl+ tty2 0:09 /snap/chrom
1 1000 14161 13246 20 0 1791740 153780 futex_ Sl+ tty2 0:36 /snap/chrom
1 1000 14173 13246 20 0 1388244 75804 futex_ Sl+ tty2 0:00 /snap/chrom
1 1000 14199 13246 20 0 1387396 74900 futex_ Sl+ tty2 0:00 /snap/chrom
1 1000 14205 13246 20 0 1384640 71268 futex_ Sl+ tty2 0:00 /snap/chrom
1 1000 14235 13246 20 0 1386240 74024 futex_ Sl+ tty2 0:00 /snap/chrom
0 1000 14486 13744 30 10 13992 1804 wait_w SN+ pts/1 0:00 ./test1
4 1000 15133 13042 20 0 37556 1452 - R+ pts/0 0:00 ps al
alexandra@palici:~/SO$ 
```

alexandra@palici: ~/SO

```

File Edit View Search Terminal Help
alexandra@palici:~/SO$ g++ test.cpp -o test1
alexandra@palici:~/SO$ nice -n 10 ./test1

```

Pentru a schimba prioritatea se poate folosi comanda "nice":

```

0 1000 17079 13744 39 19 13992 1716 wait_w SN+ pts/1 0:00 ./test2
4 1000 17084 13042 20 0 37556 1480 - R+ pts/0 0:00 ps al
alexandra@palici:~/SO$ 
```

alexandra@palici: ~/SO

```

File Edit View Search Terminal Help
alexandra@palici:~/SO$ g++ test.cpp -o test1
alexandra@palici:~/SO$ nice -n 10 ./test1
^C
alexandra@palici:~/SO$ g++ test.cpp -o test2
alexandra@palici:~/SO$ nice -n 20 ./test2

```

Prioritatea este scazuta cu 10 respectiv 20 in exemplele anterioare.

Prioritatea poate fi scăzută până la o valoare de +20.

```
0 1000 18382 13744 39 19 13992 1736 wait_w SN+ pts/1 0:00 ./test3
4 1000 18383 13042 20 0 37556 1504 - R+ pts/0 0:00 ps al
alexandra@palici:~/SO$
```

```
alexandra@palici: ~/SO
```

File Edit View Search Terminal Help

```
alexandra@palici:~/SO$ g++ test.cpp -o test1
alexandra@palici:~/SO$ nice -n 10 ./test1
^C
alexandra@palici:~/SO$ g++ test.cpp -o test2
alexandra@palici:~/SO$ nice -n 20 ./test2
^C
alexandra@palici:~/SO$ g++ test.cpp -o test3
alexandra@palici:~/SO$ nice -n 30 ./test3
```

Utilizatorilor normali de sistem le este interzis să crească prioritatea unui proces.

```
4 0 20981 20980 10 -10 13992 1888 - S<+ pts/1 0:00 ./test4
4 1000 21621 13042 20 0 37556 1500 - R+ pts/0 0:00 ps al
alexandra@palici:~/SO$
```

```
alexandra@palici: ~/SO
```

File Edit View Search Terminal Help

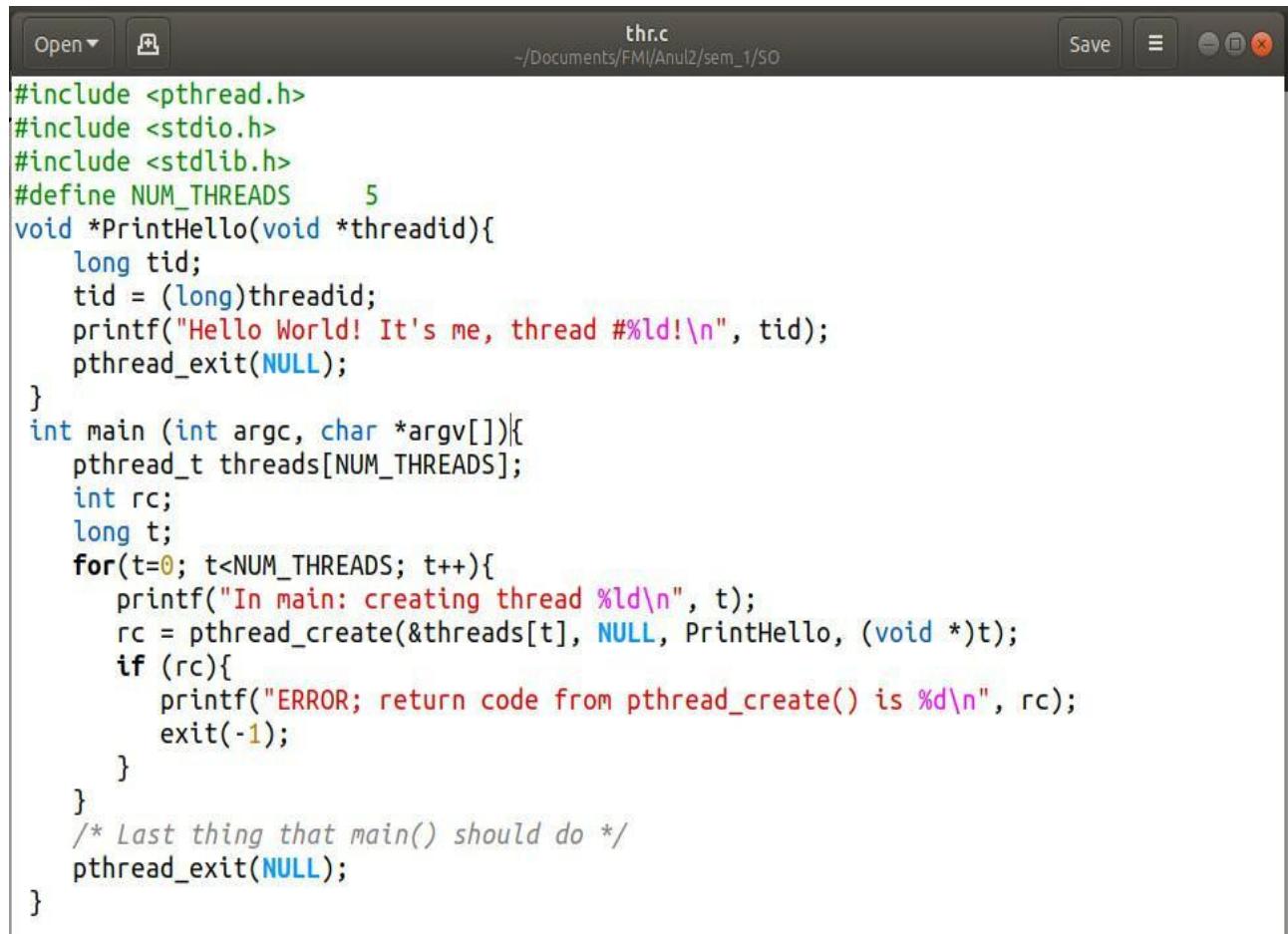
```
alexandra@palici:~/SO$ g++ test.cpp -o test4
alexandra@palici:~/SO$ nice -n -10 ./test4
nice: cannot set niceness: Permission denied
^C
alexandra@palici:~/SO$ sudo nice -n -10 ./test4
[sudo] password for alexandra:
```

- **Procese și Thread-uri în UNIX/Linux**

Conceptul de **thread** (fir de execuție) definește cea mai mică unitate de procesare ce poate fi programată spre execuție de către sistemul de operare. Este folosit în programare pentru a eficientiza execuția programelor, executând porțiuni distincte de cod în paralel în interiorul aceluiași proces.

Diferenta dintre procese si thread-uri este ca procesele sunt executate in mod independent si sincronizarea dintre ele este determinata doar de kernel, pe cand threadurile sunt sincronizate in cadrul procesului parinte.

Interactia dintre doua procese este realizata doar prin metode de comunicare standard intre procese, pe cand threadurile generate de acelasi proces comunica usor intrucat au resurse comune de memorie etc.



The screenshot shows a code editor window with the file name "thr.c" and the path "/Documents/FMI/Anul2/sem_1/SO". The code is written in C and uses the Pthreads library to create multiple threads. It defines a function PrintHello that prints a message and exits. The main function creates 5 threads, each calling PrintHello with its thread ID. If an error occurs during thread creation, it prints an error message and exits with code -1. The code also includes a comment indicating the last thing main() should do is pthread_exit(NULL);

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS      5
void *PrintHello(void *threadid){
    long tid;
    tid = (long)threadid;
    printf("Hello World! It's me, thread #%ld!\n", tid);
    pthread_exit(NULL);
}
int main (int argc, char *argv[]){
    pthread_t threads[NUM_THREADS];
    int rc;
    long t;
    for(t=0; t<NUM_THREADS; t++){
        printf("In main: creating thread %ld\n", t);
        rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
        if (rc){
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }
    /* Last thing that main() should do */
    pthread_exit(NULL);
}
```

```
alexandra@palici: ~/Documents/FMI/Anul2/sem_1/SO
File Edit View Search Terminal Help
alexandra@palici:~/Documents/FMI/Anul2/sem_1/SO$ gcc -pthread thr.c -o th
alexandra@palici:~/Documents/FMI/Anul2/sem_1/SO$ ./th
In main: creating thread 0
In main: creating thread 1
Hello World! It's me, thread #0!
In main: creating thread 2
Hello World! It's me, thread #1!
In main: creating thread 3
Hello World! It's me, thread #2!
In main: creating thread 4
Hello World! It's me, thread #3!
Hello World! It's me, thread #4!
alexandra@palici:~/Documents/FMI/Anul2/sem_1/SO$
```

In exemplul de mai sus sunt create 5 fire de executie folosind functia **pthread_create()**. Fiecare fir de executie printeaza un mesaj “Hello World” si actiunea lui este terminata la chemarea functiei **pthread_exit()**.

Bibliografie:

- Linux Command Line and Shell Scripting Bible, Richard Blum
- <https://ocw.cs.pub.ro/courses/>
- <https://www.softprayog.in/programming/creating-processes-with-fork-and-exec-in-linux>
- <https://www.tecmint.com/linux-process-management/>
- <http://www.ubm.ro/~adip/Sisteme%20de%20operare/>
- <https://computing.llnl.gov/tutorials/pthreads/>