ATENŢIONARE!

Acest material a fost elaborat de Cristian Oftez, în cadrul S.C. Centrul de Pregătire în Informatică S.A., partener de implementare a proiectului POSDRU /3/1.3/S/5.

Versiunea materialului de instruire: V2.0

9. Administrarea Serverelor

Drivere, module şi dispozitive în Linux

Având în vedere faptul că numărul de producători de dispozitive hardware de toate felurile a crescut constant, există în prezent mii de astfel de dispozitive (şi variază de la un simplu mouse până la plăci grafice de performanță care sunt foarte specializate), iar sistemele de operare existente au trebuit să țină pasul cu acestea, astfel încât să permite funcționarea corectă a acestora. Acest fapt este posibil datorită existenței driverilor, care oferă o interfață programabilă cu restul sistemului, astfel încât programatorii să poată folosi facilitățile oferite de diverse dispozitive ataşate sistemului fără să fie nevoiți să cunoască în adâncime modul de funcționare al acestora. În esență, driverele pot fi privite ca un mecanism de simplificare şi uniformizare a părții hardware, astfel încât aceasta să poată fi folosită mai uşor. Raționamentul din spatele acestei interfețe programabile foarte bine definite este simplu: acesta permite o dezvoltare modulară, în care drivere (fiind total independente de kernel) pot fi inițializate atunci când este nevoie de ele, simplificând astfel foarte mult procesul de creare al acestora. Aceste module sunt cunoscute în Linux ca **LKM**, sau **Loadable Kernel Module** şi reprezintă extensii ale kernel-ului. O greşeală comună în rândul utilizatorilor de Linux este concepția că aceste module fac parte din kernel în mod absolut, însă acesta poate fi împărțit în două părți importante: imaginea kernel-ului de „bază” (sau **base kernel**), adică imaginea care este încărcată în timpul procesului de **boot** şi aceste extensii (**LKM**), care comunică în mod direct cu partea de bază.

Caracterul modular oferă sistemului Linux un avantaj foarte mare printr-un grad de flexibilitate crescut, modulele putând fi oprite când nu este nevoie de ele sau încărcate numai atunci când este nevoie de acestea. Există două alternative în implementarea LKM-urilor: încărcarea acestora ca extensii, sau includerea lor în kernel-ul de bază. De multe ori, prima abordare este şi cea mai bună, din mai multe motive: în primul rând nu mai forțează utilizatorul să recompileze kernel-ul (şi deci reduce semnificativ probabilitatea apariției erorilor sau greşelilor în kernel-ul de bază), nu necesită reîncărcarea sistemului şi de foarte multe ori salvează memorie (**base kernel-ul** fiind încărcat mereu în memorie), încărcând modulele numai atunci când este nevoie de acestea.

Un alt avantaj major al LKM-urilor îl reprezintă uşurința cu care acestea pot fi depanate în cazul în care se întâmplă o eroare, de multe ori, erori în kernel-ul de bază ducând la imposibilitatea încărcării sistemului. Aceste module nu înregistrează o scădere în performanța comparativ cu cele incluse în kernel-ul de bază, însă există cazuri în care această abordare nu este posibilă şi kernel-ul trebuie recompilat. Vom aborda pe larg această problemă în secțiunea dedicată recompilarii unui kernel.

LKM-urile pot să îndeplinească o serie întreagă de funcții într-un sistem Linux, însă cele mai comune roluri sunt:

- **Drivere** – aşa cum am menționa mai devreme, acestea reprezintă o interfață care permite kernel-ului să comunice cu dispozitivele pentru care au fost create, fără a fi nevoie de cunostiinte detaliate despre modul în care acestea functioneza.
- **Funcții de sistem –** oferă programelor metode de a apela funcțiile puse la dispoziție de sistem cum ar fi crearea unui nou proces, citirea unui fişier sau chiar oprirea sistemului. Deşi majoritatea apelurilor de sistem sunt deja incluse în kernel-ul de bază, însă se pot crea noi apeluri sau chiar înlocui unele deja existente prin intermediul LKM-urilor.
- **Interpretoare de executabile –** care fac posibilă încărcarea în memorie şi rularea programelor executabile în diverse formate.
- **Drivere pentru sistemul de fişiere –** astfel încât sistemul de operare să poată interpreta conținutul fizic şi implicit şi cel logic (modul în care sunt organizate directoarele şi fişierele pe un mediu de stocare sau de pe rețea)

Urmând această arhitectură, sistemul Linux pune la dispoziția utilizatorului o serie de utilitare prin intermediul cărora să poată controla modulele existente.

Pentru **vizualizarea modulelor încărcate**, utilizatorul poate să folosească comanda **lsmod** (probabil prescurtat de la „list modules”). Acest utilitar, după cum menționează şi descrierea sa din manual (man), este unul oarecum trivial: unicul său rol este acela de formatare a informațiilor (aflate într-unul din fişierele speciale) într-o formă ce poate fi interpretată mai uşor, afişând pe ecran numele, dimensiunea şi dependințele tuturor modulelor de kernel încărcate.


```
Module                  Size  Used by
psmouse                32336  0
ppdev                   6468  0
lp                      8164  0
ipv6                  235396  16
cpufreq_conservative    5960  0
cpufreq_stats           3776  0
cpufreq_userspace       3172  0
cpufreq_powersave       1856  0
cpufreq_ondemand        6476  0
freq_table              4224  2 cpufreq_stats,cpufreq_ondemand
vboxvfs                36704  0
nls_base                6820  1 vboxvfs
loop                   12748  0
parport_pc             22500  0
parport                30988  3 ppdev,lp,parport_pc
serio_raw               4740  0
pcspkr                  2432  0
battery                10180  0
ac                      4196  0
button                  6096  0
vboxadd                58848  4 vboxvfs
i2c_piix4               7216  0
i2c_core               19828  1 i2c_piix4
:_
```

Să analizăm primul modul listat în poza de mai sus, cu numele de **psmouse**. Acesta reprezintă modulul responsabil de interfațarea cu dispozitive periferice de tip mouse, conectate prin PS2. Putem observa faptul că modulul respectiv are dimensiunea de **32336** bytes, iar câmpul următor, cu valoarea 0, reprezintă numărul de entități ce depind de acest modul.

Pentru **înlăturarea unui modul** se pot folosi atât comenzile **modprobe** dar şi **rmmod**, însă doar primă este recomandată. Modul de utilizare al acesteia este destul de simplu, iar utilizatorul trebuie să specifice doar doi parametri: numele modului şi parametrul care specifică faptul că acesta doreşte înlăturarea modulului (-r). Dacă ar fi să urmăm exemplul de mai sus şi am dori să excludem modulul **psmouse**, ar trebui executată comanda **modprobe -r psmouse**. Efectul înlăturării unui modul este de regulă instant, având ca urmare (în acest caz) pierderea funcționalității mouse-ului.

În cazul în care utilizatorul doreşte **punerea în funcțiune a unui modul**, din diverse motive, acesta poate utiliza tot comanda **modprobe [nume_modul]**. În exemplul de mai sus, putem folosi **modprobe psmouse** pentru a putea folosi mouse-ul din nou.

Încărcarea automată a modulelor

Sistemul poate fi configurat astfel încât să încarce un modul LKM în mod automat atunci când kernel-ul are nevoie de acesta prima oară. Această încărcare automată poate fi realizată în principiu prin două mecanisme: fie prin **kernel module loader** (care face parte din kernel-ul Linux), fie prin versiunea mai veche a acestuia: un daemon numit **kerneld**. Amândouă variantele folosesc utilitarul **modprobe** (deci implicit şi **insmod**) pentru a-şi duce sarcina la capăt.

Modul de funcționare al Kernel Module Loader

Aşa cum am menționat mai devreme, KML (Kernel Module Loader) face parte directă din kernel şi are scopul de a încărca module LKM în mod automat atunci când este nevoie de acestea. Când necesitatea unui modul devine aparență şi acesta trebuie încărcat, KLM crează un nou proces (al cărui proprietar este contul root) care va încerca să încarce modulul dorit prin executarea utilitarului **modprobe** (acesta aflându-se la adresa **/sbin/modprobe** în mod implicit) cu o serie de parametri specifici: **-s** (forțează notarea oricăror erori în rapoartele de sistem), **-k** (modul „autoclean”) şi desigur, numele modulului ce trebuie încărcat.

De multe ori, ultimul parametru pasat utilitarului **modprobe** nu este neapărat numele unui fişier de tip modul, ci mai degrabă un nume simbolic, un alias pentru simplificarea numelor folosite. Lista acestora se găseşte de regulă în fişierul **/etc/modprobe.d/aliases**.

Eliminarea automată a modulelor

În acelaşi fel în care modulele pot fi încărcate în mod automat, există şi un mecanism de înlăturare a acestora, numit **autoclean**. Motivul principal pentru care acest mecanism a fost implementat, este înlăturarea modulelor care nu au mai fost folosite pentru perioade prelungite de timp (de regulă peste 1-2 minute) salvându-se altfel memorie şi timp de procesor. Kernel Module Loader nu utilizează acest mecanism spre deosebire de **kerneld**, însă poate fi forțat prin executarea comenzii **rmmod –all** la intervale diferite de timp. În cazul în care utilizatorul doreşte să beneficieze de acest mecanism, dar nu doreşte să folosească **kerneld**, acesta poate programa execuția comenzii **rmmod -all** ca un **cron job**, astfel încât acesta să fie executat automat la intervale de timp fixe.

Interfațarea cu structurile de date ale kernel-ului

Kernel-ul Linux are două roluri majore în cadrul unui sistem: să intermedieze şi să controleze accesul cu diferitele dispozitive conectate la sistem şi să gestioneze interacțiunea proceselor (adică când şi cum vor interacționa) cu aceste dispozitive.

Directorul **/proc** reprezintă o ierarhie de fişiere speciale, prin intermediul cărora se pot obține o multitudine de informații despre starea actuală a kernel-ului. Acest director constituie şi principalul mecanism prin care aplicațiile şi utilizatorii pot obține informații despre sistem.

Respectând structura unui sistem Linux (unde totul este un fişier) şi analizând conținutul directorului **/proc** putem trage o concluzie interesantă: acesta este de fapt un pseudo sistem de fişiere dat fiind faptul că introduce un nou tip de fişier: fişierul virtual. Acesta au o calitate deosebită prin faptul că deşi mare parte din acestea au dimensiunea de zero bytes, la afişare vor prezenta o cantitate mare de informații. În esență, aceste fişiere virtuale sunt mecanismul prin care kernel-ul Linux oferă informații despre sistem, folosind arhitectura cunoscută de „totul este un fişier”. Informația este structurată în directoare şi subdirectoare în funcție de categoria din care face parte. Deşi majoritatea fişierelor au dimensiunea zero, şi sunt read-only, acestea sunt reactualizate de kernel la intervale regulate. Totuşi, există câteva fişiere din directorul **/proc** care pot fi alterate (de regulă doar de contul root) modificând configurația kernel-ului. Pentru a afla de exemplu informații despre tipul de procesor instalat în sistem, utilizatorul trebuie doar să afişeze (în orice fel) fişierul **/proc/cpuinfo**, cu mențiunea că deşi o parte din aceste informații sunt uşor de înțeles de către utilizator, există fişiere care pot fi interpretate de oameni cu greutate – aici devenind aparent rolul utilitarelor existente precum **top, free**, etc.

Pentru a putea înțelege mai bine cum funcționează acest pseudo sistem de fişiere şi ce fel de informații ne oferă, vom analiza principalele zone ale acestuia. De regulă, fiind vorba de un alt sistem de fişiere decât cel de bază şi în baza celor discutate în secțiunile anterioare, acesta trebuie „montat”

într-un director pentru a putea fi folosit, acesta fiind de regulă chiar **/proc** din directorul rădăcina. Principalele categorii din **/proc** sunt:

- **/proc/[identificator_numeric]/** - Directorul /proc va conține o serie de subdirectoare al căror nume este un identificator numeric. Acestea simbolizează de fapt procese, iar numele directorului este dat de PID-ului fiecărui proces ce rulează în momentul curent. Directorul va conține o serie fişiere şi subdirectoare:
    - **/cmdline** conține linia de comandă cu care a fost lansat executabilul, incluzând şi lista de parametri.
    - **/cwd** este un link simbolic către directorul activ al procesului în cauză.
    - **/environ** conține informații despre variabilele de stare şi mediu cu care rulează procesul.
    - **/exe** reprezintă un link spre executabilul ce reprezintă procesul, având inclusă calea absolută.
    - **/fd/** este un subdirector ce conține link-uri către toate fişierele deschise de proces: 0 – standard input, 1 – standard output, 2 - standard error şi aşa mai departe. Fişierele sunt numite după descriptorul de fişiere folosit şi pentru a vedea exact de ce fişiere sunt legate este suficientă comanda **ls -l**.
    - **/stat** conține informații despre procese, şi este unul din fişierele pe care le consultă comanda **ps** pentru a obține informații despre procesele care rulează. Multe din informațiile conținute de acest fişier sunt de natură tehnică, şi utile în special dezvoltatorilor de software. Pentru informații suplimentare, apelați comanda **man proc**.
- **/proc/apm** – prescurtarea de la advanced power management, ne oferă informații despre starea bateriei în cazul în care aşa ceva există şi kernel-ul a fost compilat corespunzător.
- **/proc/bus –** conține subdirectoare pentru toate magistralele instalate
    - /**pci** este un subdirector ce conține informații despre magistralele pci, dispozitive şi drivere.
    - **/pci/devices –** informații despre dispozitivele conectate prin pci. Se recomandă folosirea comenzii **lspci** pentru vizualizarea conținutului acestui fişier.
- **/proc/cpuinfo –** reprezintă o sumarizare a arhitecturii sistemului.
- **/proc/diskstats –** este un fişier ce conține informații despre activitatea input/output a fiecărui disc fizic. Formatul acestui fişier este oarecum greu de citit, şi se recomandă folosirea comenzii **df** pentru aflarea informațiilor legate de discuri.
- **/proc/filesystem –** este o listă cu toate tipurile de sisteme de fişiere cu care kernel-ul a fost compilat, fiind folosită de comandă **mount** pentru ciclare, atunci când nu se specifică în mod special un sistem de fişiere.
- **/proc/kmsg -** reprezintă o interfață prin care utilizatorul poate să citească mesaje generate de kernel. Pentru a putea citi acest fişier, este nevoie de

drepturi de root, şi nu se recomandă accesarea simultană de către mai multe programe a acestui fişier. Pentru uşurință, conținutul acestui fişier este cel mai uşor de accesat prin intermediul comenzii **dmesg**.

- **/proc/meminfo –** conține informații despre memorie (atât liberă cât şi utilizată, fizică şi virtuală) şi este interpretată cel mai uşor prin comanda **free**.
- **/proc/mounts –** oferă informații despre sistemele de fişiere montate în momentul afişării.
- **/proc/net –** oferă o serie de fişiere virtuale, prin care se pot obține informații despre starea curentă a rețelei. Deşi mare parte din aceste informații sunt uşor de interpretat de către oameni, comanda **netsta**t poate formata informațiile într-o măsură mai bună. Directorul **/proc/net** conține următoarele puncte de interes:
  - **/dev –** un fişier ce conține informații despre starea dispozitivelor de comunicare, precum plăci de rețea. Pe lângă acestea, fişierul mai conține şi numărul de pachete trimise şi primite pentru fiecare interfață, cât şi numărul de erori şi alte informații statistice. Acest fişier este citit de comandă **ifconfig** pentru afişarea şi configurarea tuturor interfețelor de rețea.
  - **/tcp, /udp, /raw** conțin datele aferente socket-urilor de comunicație pentru fiecare protocol în parte. Aceste fişiere sunt de regulă folosite numai în scopul depanării sau monitorizării.
- **/proc/partitions –** afişează o tabelă ce evidențiază modul în care sunt partiționate hard-discurile.
- **/proc/sys –** acest director conține fişiere şi subdirectoare ce alcătuiesc mediul şi variabilele de kernel. Aceste variabile pot fi schimbate prin intermediul pseudo sistemului de fişiere /proc sau prin intermediul utilitarului **sysctl**.
- **/proc/uptime –** un fişier ce conține două valori numerice. Prima dintre acestea reprezintă timpul de funcționare al sistemului (măsurat în secunde zecimale) sau **uptime**, iar al doilea reprezintă timpul **idle** al sistemului (măsurat tot în secunde zecimale). Timpul de **idle** se referă la numărul cumulat de secunde în care procesul a fost inactiv (nu a executat nicio instrucțiune), şi în cazul sistemele desktop, acesta poate să fie foarte mare, atingând chiar valori de peste 90% din uptime. În cazul unui server dedicat, această cifră este un indicator al eficienței de utilizare al acestuia, valori mai mici simbolizând un raport mai bun.
- **/proc/version –** informații despre versiunea de kernel ce rulează. Acest fişier concatenează informațiile din mai multe fişiere precum **/proc/sys/kernel/ostype** şi **/proc/sys/kernel/osrelease**.
- **/proc/vmstat –** folosit pentru afişarea informațiilor legate de memoria virtuală.

Sistemul de gestiune a pachetelor

Unul dintre aspectele care sperie orice utilizator de calculatoare este gestiunea corectă a programelor: instalarea, dezinstalarea sau configurarea acestora, provoacă de multe ori probleme utilizatorilor, mai ales când aceştia nu au experiența cu fiecare program în parte.

În sistemele de tip UNIX, această problemă era chiar mai mare decât pe alte sisteme de operare, cel puțin în condițiile în care utilizatorul nu plănuia cu atenție ce şi cum anume instalează, fiecare utilizator fiind forțat să compileze singur programele pe care dorea să le folosească. Acest detaliu însemna că în lipsa unui plan de desfăşurare al software-ului instalat într-un calculator, utilizatorul putea instala programe de mai multe ori, cu versiuni diferite şi în locuri diferite (sau cu suprascrieri parțiale), ceea ce făcea gestiunea corectă a acestora aproape imposibilă. Când un utilizator dorea să instaleze un program într-un sistem de tip UNIX, acesta de regulă achiziționa codul sursă al acestuia în format **tar.gz** (**tar = tape archive, gz = gzip**), adică o structură de fişiere şi directoare, ce menținea informații despre sistemul de fişiere (permisiuni, date, şi proprietari) care era arhivata pentru distribuirea uşoară. După acest pas, utilizatorul trebuia să extragă conținutul arhivei, şi de regulă, să urmeze nişte script-uri de compilare relativ automate, incluse în arhivă (prin intermediul comenzii **make**, care se ocupă atât de configurare cât şi de compilarea programului). Dacă totul funcționa corect, şi dependințele de alte programe şi librării erau satisfăcute, programul era instalat - cu mențiunea că dacă utilizatorul ar fi vrut să facă o actualizare a acestui program la o versiune mai nouă, ar fi trebuit dezinstalat, şi apoi reinstalat şi reconfigurat.

Debian a fost gândit de la început să ofere o soluție pentru această problemă, şi astfel a apărut un sistem complet de gestiune a programelor, unul care să ofere un sistem rapid şi practic de instalare a pachetelor, abordând în mod automat problema dependințelor şi a reconfigurărilor în cazul actualizărilor de versiuni. Astfel a apărut pachetul Debian (fişiere cu extensia **.deb**) şi **apt** (Advanced Packaging Tool). Pe lângă toate aceste facilități, **apt** oferă şi posibilitatea procurării pachetelor necesare din **repository**-uri (arhive de pachete), care pot fi surse locale (CD-ROM, hard-disc, etc) sau puse la dispoziție prin intermediul rețelei. Toate acestea fac din **apt** un excelent utilitar pentru gestiunea software-ului dintr-un sistem Linux.

Vom analiza în continuare compoziția pachetelor specifice distribuției Debian, fişierele **.deb**. Acestea conțin mai mult decât o serie de fişiere ce pot fi instalate pe sistemul nostru: pe lângă fişierele de bază acestea conțin şi o informații ierarhizate despre dependințele pachetului (mai exact ce alte pachete trebuie instalate astfel încât cel în cauză să poată fi instalat). În baza acestor informații adiționale, sistemul responsabil de gestiunea pachetelor va ştii ce pachete să descarce (din arhiva de pachete – repository) pentru ca instalarea să poată fi terminată cu succes. De altfel sistemul este capabil să decidă exact ordinea de instalare a dependințelor (cât şi stabilirea acestora în funcție de pachețele deja instalate), cât şi să țină evidența versiunilor. Vom continua prin a detalia conceptele de **dependințe** şi **conflicte**.

Dependințele sunt în esență pachete necesare pentru instalarea unui alt pachet: dacă am lua trei pachete imaginare **a, b** şi **c** ca exemplu, iar **c** ar avea ca dependințe pachetele **a** şi **b**, atunci acesta nu ar putea fi instalat fără că pachetele **a** şi **b** să fie instalate în prealabil. Sistemul respectiv funcționează în ambele sensuri, atât pentru instalări dar mai ales pentru dezinstalari, când eliminarea unui pachet ar putea afecta multe altele.

Conflictele se referă la perechi de pachete, care instalate împreună ar putea cauza funcționarea anormală a unor programe. Din acest motiv, pachetele pot avea marcate alte pachete ca fiind **conflictuale**, şi prin urmare acestea nu pot fi instalate în acelaşi timp.

Întreg sistemul este gestionat de programul **dpkg** (**Debian package management system**), existând mai multe interfețe pentru a facilita uzul acestei aplicații. Printre cele mai comune interfețe pentru **dpkg** putem aminti **apt-get** sau varianta mai nouă **aptitude** (tot o aplicație de consolă, însă care oferă o interfață mai uşor de folosit decât cea bazată pe linia de comandă).

Instalarea pachetelor într-un sistem Debian

Fişierele Debian pot fi instalate folosint atât utilitarul **dpkg**, cât şi **apt-get**. În cazul în care utilizatorul dispune de un fişier Debian (.deb) în mod absolut (de exemplu **pachet.deb**) acesta ar putea dispune instalarea acestui pachet folosind comanda: **dpkg –i pachet.deb**

În cazul în care pachetul dorit nu există local pe sistemul în cauză, utilizatorul poate încerca să obțină acest pachet prin intermediul programului **apt-get** de la un server ftp, numit repository. Programul **apt-get** se va ocupa şi de descărcarea automată a dependințelor, şi tratarea oricăror conflicte ce pot să apară – ceea ce poate rezulta uneori în ştergerea unor pachete, cu o notificare foarte clară către utilizator. Fiind vorba de un utilitar în linie de comandă, sintaxa acestuia este una foarte simplă:

apt-get install nume_pachet

În urma executării comenzii anterioare, **apt** va determina o listă de pachete deja instalate pe sistem, pentru a putea stabili ce pachete lipsesc. Acest lucru este posibil prin interpretarea unui fişier esențial pentru sistemul Debian - **/var/lib/dpkg/status**. Compromiterea acestui fişier în orice fel, poate avea consecințe grave asupra unui sistem Debian, pornind de la incapacitatea de a folosi platforma de gestiune a pachetelor până la pieredea funcționalității sistemului de operare. Din acest motiv, sistemul ține o copie (ceva mai veche decât originalul, dar relativ similară) în **/var/lib/dpkg/status-old**.

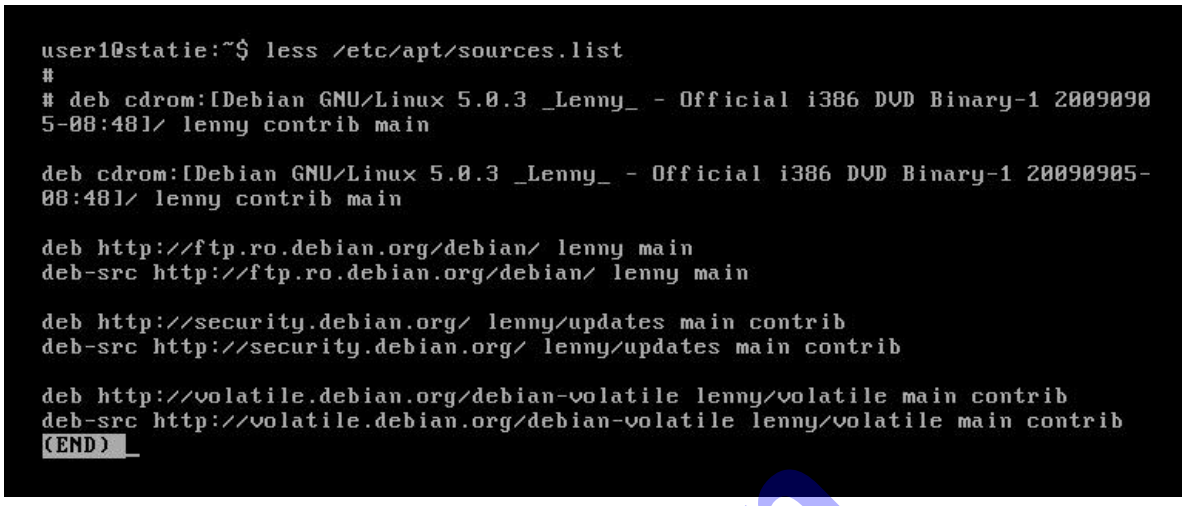
În urma stabilirii dependințelor lipsă, **apt** va descărca atât acele pachete, cât şi pachetul dorit şi va continua instalarea acestora, în ordinea relevantă. Toate pachetele descărcate prin intermediul utilitarului **apt** vor fi stocate în directorul **/var/cache/apt/archives**. Sistemul de gestiune al pachetelor este gândit astfel încât după descărcarea şi instalarea unui pachet, acesta să rămână stocat în sistem la adresa menționată anterior, pentru a nu forța descărcarea unui pachet în mod inutil în cazul instalărilor multiple. Ştergerea pachetelor are loc periodic, dar poate fi forțată de către utilizator fie prin

ştergerea manuală a conținutului directorului menționat anterior, fie prin rularea comenzii **apt-get clean**.

Arhive de pachete (repository)

Pentru a permite accesul facil la pachete comune, programul **apt-get** se foleste de arhive de pachete (numite repository). Acestea pot să varieze de la resurse locale (hard-disc, cdrom sau alte medii locale) la servere ftp de pe internet, acestea din urmă fiind şi cea mai utilizată variantă, din moment ce de regulă conțin versiuni actualizate ale tuturor pachetelor dorite.

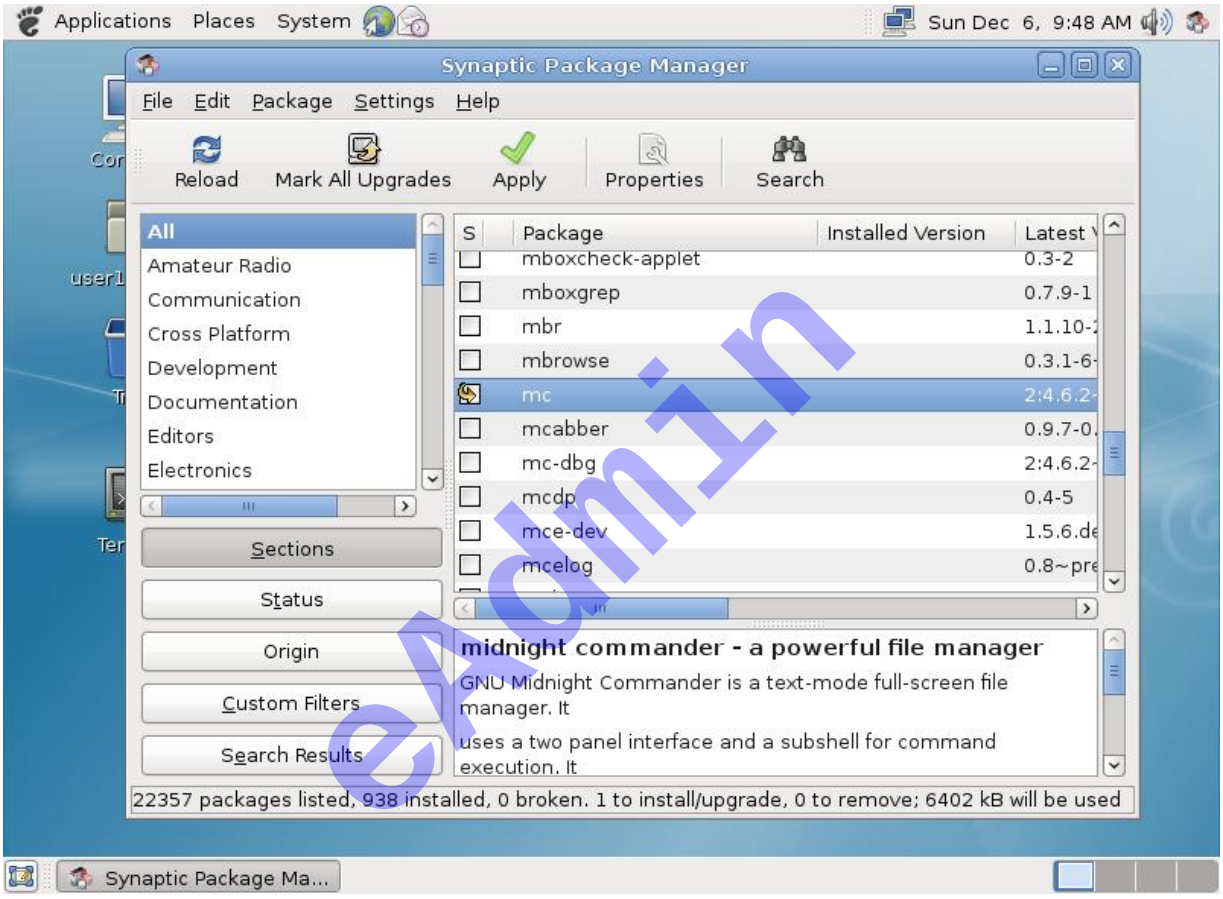
Lista de repositories poate fi modificată, putând fi adăugate sau şterse astfel de arhive, sistemul menținând un fişier text (**/etc/apt/sources.list**), cu un format bine stabilit în care aceste arhive de pachete sunt menționate. Este important de precizat faptul că acest fişier conține doar referințe către noi surse de software, şi în cazul serverelor ftp, este necesară o conexiune fizică cu acestea.



Analizând imaginea de mai sus putem observa un exemplu de cum ar putea să arate fişierul **sources.list**. Aşa cum am menționat, acesta conține o listă cu referințe către toate arhivele de pachete de pe care am putea să instalăm ceva. Sistemul apt foloseşte o bază de date proprie pentru a păstra o listă cu toate pachetele disponibile pentru descărcare (inclusiv versiunile disponibile pentru acestea) şi generează această bază de date în funcție de arhivele selectate prin intermediul fişierului **sources.list**. Deşi acest proces este automat într-o oarecare măsură, se recomandă forțarea actualizării acestei baze de date ori de câte ori se adaugă un nou repository, sau când se încearcă instalarea unui nou program, astfel încât să ne asigurăm că pachetul dorit există în lista de pachete disponibile, actualizat la cea mai nouă versiune disponibilă. Pentru a forța sistemul de gestiune să reactulizeze această bază de date, trebuie folosită comanda: **apt-get update**

Pe lângă varianta instalării pachetelor folosind consola sistemului, utilizatorul poate opta pentru folosirea interfeței grafice, prin intermediul programului **Synaptic Package Manager**. Acesta nu este altceva decât o interfață pentru sistemul **apt**, şi oferă în esență aceleaşi funcții: instalarea, ştergerea,

căutarea sau actualizarea pachetelor din sistem. Principala diferența dintre acesta şi utilizarea în consolă este un sistem în care schimbările pot fi adăugate într-o coadă de comenzi, urmând să fie executate într-o ordine anume. Principalele avantaje ale acestui program, sunt uşurința cu care poate fi folosit de către oricine, mesajele clare (utilizatorul este atenționat foarte clar ce efect va avea orice schimbare), afişarea modificărilor altor pachete ca urmare a deciziilor luate şi nu în ultimul rând facilitățile avansate de căutare a pachetelor. **Synaptic Package Manager** vine instalat în mod implicit cu Debian, şi poate fi lansat foarte uşor prin accesarea meniului **System -> Administration -> Synaptic Package Manager**.



Ca şi în cazul altor utilitare ce afectează sistemul la nivel global, pentru lansarea acestui program utilizatorul va fi rugat să furnizeze parola unui utilizator autorizat, în cazul de față fiind vorba de contul utilizatorului **root**. Utilizatorul poate să adauge comenzi de instalare, ştergere sau actualizare individual pentru pachete, însă schimbările nu vor fi procesate decât după ce acesta acționează butonul **Apply**. Ca un exercițiu, putem încerca să instalăm programul **mc (Midnight Commander)**, vizibil în poza de mai sus. Acesta este un program de gestiune a fişierelor în mod text, similar cu **Norton Commander** şi oferă multe facilități pentru o gestiune mai uşoară a sistemului de fişiere. Pe lângă rolul său primar, programul mai conține şi o serie de alte utilitare precum **mcedit**, un editor de text foarte util şi foarte uşor de folosit.

Pentru a instala pachetul **mc**, utilizatorul trebuie să localizeze mai întâi pachetul respectiv, fie parcurgând lista de pachete (vizibilă în partea din dreapta a ecranului), fie folosind facilitățile de căutare de pachete. În cazul în care utilizatorul foleste funcția de căutare, acesta trebuie să specifice o serie de parametri: un şir de caractere ce reprezintă textul căutat, şi câmpul în care programul să efectuează căutarea (opțiunile sunt destul de clare, însă dat fiind faptul că de regulă se caută nume de pachete, se recomandă căutarea după nume – şi nu după descriere – pentru a reduce numărul de rezultate).

După ce utilizatorul a identificat cu succes pachetul dorit, acesta poate să acționeze butonul din stânga al mouse-ului peste pachetul respectiv şi să selecteze una din opțiunile disponibile:

- **Mark for Installation** – pachetul va fi marcat pentru instalare în coadă de comenzi. În urma acționării butonului **Apply**, se va încerca instalarea acestuia şi a tuturor dependințelor acestuia.
- **Mark for Reinstallation** – va marca pachetul pentru a fi dezinstalat şi instalat din nou.
- **Mark for Upgrade** – se va încerca actualizarea pachetului respectiv la o versiune mai nouă, în cazul în care acesta este disponibilă într-una din arhivele de pachete menționate.
- **Mark for Removal** – marchează pachetul pentru ştergerea acestuia din sistem.
- **Mark for Complete Removal** – spre deosebire de varianta precedentă, aceasta va elimina complet pachetul cât şi eventuale fişiere de configurație.

Utilizatorul poate înşirui mai multe comenzi, iar **Synaptic** se va asigura că acestea vor fi executate în ordinea corespunzătoare şi va avea grijă să instaleze şi să afişeze numele tuturor dependințelor ce vor fi instalate. Este important de reamintit faptul că nici o schimbare nu va fi dusă la capăt până când utilizatorul nu acționează butonul **Apply**.

Ştergerea unui pachet

Pe lângă instalarea pachetelor, adminstratorul unui sistem Debian trebuie să poată şi să le elimine din sistem. Pentru ştergerea pachetelor se poate folosi tot interfața **apt-get** cu o serie de parametri: **apt-get remove nume_pachet**

În urma executării comenzii de mai sus, sistemul **apt** va afişa o serie de informații esențiale precum o atenționare, lista tuturor pachetelor ce vor fi şterse (**apt** se va ocupa automat şi de înlăturarea tuturor pachetelor ce depind de pachetul pe care încercam să-l ştergem, utilizatorul neavând altă opțiune), cât şi informații generale precum numărul de pachete ce vor fi instalate, şterse sau actualizate şi spațiul eliberat în urma acțiunii. Ultimul rând este de regulă o cerere de confirmare din partea utilizatorul – de regulă în formatul **Do you want to continue [Y/n]** -, specificând dacă acesta doreşte cu adevărat executarea comenzii.

OBSERVAȚIE: Urmând un fel de tradiție în Linux, marea parte a confirmărilor de tipul de mai sus, pot avea una, două sau mai multe opțiuni. Printre opțiunile prezente în exemplul de mai sus, putem observa faptul că litera **Y** (simbolizând yes) este scrisă cu majuscule, pentru a semnala opțiunea implicită; dacă utilizatorul nu specifică altă opțiune, cea implicită va fi selectată automat la apăsarea tastei ENTER.

Exemplul de mai sus poate fi tradus foarte simplu în programul **Synaptic Package Manager** prin selectarea opțiunii **Mark for Removal**. Pentru specificarea ştergerii complete a pachetului (adică inclusiv a fişierelor de configurație, în cazul în care acestea există), trebuie adăugat parametrul **--purge**: **apt-get --purge remove nume_pachet**

Căutarea şi afişarea informațiilor referitoare la pachete

Urmărind arhitectura prezentată până acum, sistemul de gestiune al pachetelor funcționează prin folosirea unei baze de date proprii cu informații despre pachetele disponibile, aceasta putând fi reactualizată prin comanda **apt-get update**. Programul folosit pentru menținerea bazei de date interne actuală este **apt-cache,** acesta permițând utilizatorilor (prin intermediul consolei) să afişeze şi să caute în lista de pachete disponibile. De regulă, principalul motiv pentru care se doreşte căutarea unui pachet este atunci când utilizatorul nu ştie exact numele pachetului, sau pur şi simplu doreşte afişarea tutoror pachetelor ce conțin unul sau mai mulți termeni în descriere.

Căutarea se poate face foarte simplu folosind programul **apt-cache** cu parametrul **search**: **apt-cache search sir_de_căutare** unde şirul de căutare reprezintă o expresie regulată (REGEX). În urma executării acestei comenzi, utilitarul va afişa toate pachetele ce conțin expresia căutată fie în numele pachetului, fie în descrierea acestuia. Informațiile afişate pot fi de multe ori derutante, doar prin prisma numărului de rezultate găsite, însă există metode pentru reducerea numărului de rezultate.

O altă metodă de căutare poate implica căutarea după mai multe expresii; abordarea poate să difere, fie prin folosirea expresiilor regulate (REGEX – accesați comanda **man regex** pentru mai multe informații), fie prin filtrarea rezultatului intermediar cu ajutorul comenzii **grep**: **apt-cache search mc | grep midnight**

În exemplul de mai sus putem observa prima parte a comenzii, care va efectua o căutare după şirul de caractere „mc”. Chiar dacă în cazul de față, dorim să aflăm informații doar despre aplicația Midnight Commander (prezentată mai sus), **apt-cache** va afişa numele şi descrierile tuturor pachetelor ce conțin şirul „mc” ducând astfel la un număr foarte mare de rezultate. Partea a doua va filtra rezultatul intermediar (cu ajutorul utilitarului **grep**); astfel se vor afişa doar pachetele care conțin atât textul „mc” cât şi textul „midnight”. Această modalitate de căutare poate fi extinsă în mai multe feluri, de exemplu, pentru a căuta o listă cu pachetele editoarelor de text disponibile: **apt-cache search editor | grep text**

Odată ştiut numele pachetului dorit, utilizatorul poate afla în continuare mai multe informații despre acestea, tot prin intermediul programului **apt-cache**. Acesta poate de exemplu să solicite informații detaliate despre un pachet, precum versiunea, dimensiunea, arhitectură, conflictele sau chiar dependințele acestuia: **apt-cache show mc**

Comanda de mai sus va afişa toate informațiile disponibile despre pachetul „mc”. A se observa faptul că în loc de directiva **search**, s-a folosit directiva **show** pentru a specifica faptul că utilizatorul doreşte informații, şi nu o căutare. Comanda poate fi folosită alternativ cu utilitarul **less** pentru ca rezultatul să fie mai uşor de citit.

Alternativ, utilizatorul poate să afişeze dependințele unui pachet (acestea fiind menționate şi în secțiunea de informații adiționale a pachetului) într-o formă mai uşor de citit folosind directiva **depends** în loc de **show**.

Căutarea pachetelor în linie de comandă poate fi evitată în cazul în care mediul de lucru beneficiază de o interfață grafică. Programul **Synaptic Package Manager** prezentat anterior, poate fi folosit cu uşurință de către aproape oricine, fiind foarte uşor să se afle informații adiționale precum dependințe sau conflicte.

Actualizarea surselor de pachete (Repositories)

Sistemul de gestiune al pachetelor apt foloseşte surse de pachete pentru descărcarea acestora, pentru a putea fi instalate, indiferent de mediul pe care care se află acestea (extern sau intern). Principalul avantaj al acestei infrastructuri este că atât utilizatorii cât şi dezvoltatorii de software pot apela la o sursă comună de programe. Această abordare nu este utilă numai pentru descărcarea inițială a unuia sau mai multor programe, ci şi pentru păstrarea acestora la versiuni actualizate; un dezvoltator se va asigura de regulă că pachetele sale sunt prezente fie pe servere ftp proprii, fie pe arhive comune, facilitând astfel accesul către acestea.

Arhitectura folosită este foarte flexibilă, permițând astfel utilizatorului să adauge şi să elimine astfel de surse de pachete după nevoi. Lista de arhive poate fi alterată de utilizator pentru a adăuga sau a scoate anumite surse în două feluri: fie alterând fişierul ce conține lista de arhive (**/etc/apt/sources.list**) manual cu un editor de text, fie folosind un utilitar – care în esență face acelaşi lucru, însă oferă o interfață mai uşor de folosit.

Formatul fişierului **/etc/apt/sources.list** este prestabilit, fiecare sursă de pachete fiind menționată pe o linie, şi este gândit să suporte o gamă foarte largă de medii şi surse. Fiecare linie descrie aşadar o sursă de pachete şi respectă următoarea sintaxă: **deb URI distribuție [componenta 1] [componenta 2] …**

Primul cuvânt poate fi **deb** sau **deb-src**, şi descrie fie o arhivă cu două nivele tipică distribuției Debian, fie o arhivă ce conține codul sursă în aceeaşi formă ca tipul **deb**.

URI specifică baza distribuției, sau mai exact locația de unde apt va putea descărca pachetele necesare. Aceasta poate fi şi sub forma unei adrese (de internet de exemplu), caz în care componentele menționate trebuie omise şi distribuția trebuie terminată cu ajutorul caracterului **„/”**. Principalele tipuri de surse recunoscute sunt următoarele:

- file – permite folosirea unui fişier oarecare din sistem drept o arhivă.
- cdrom – permite folosirea suportului CDROM pentru surse.
- http – foloseşte un server HTTP drept arhivă
- ftp – specifică un server FTP drept arhivă

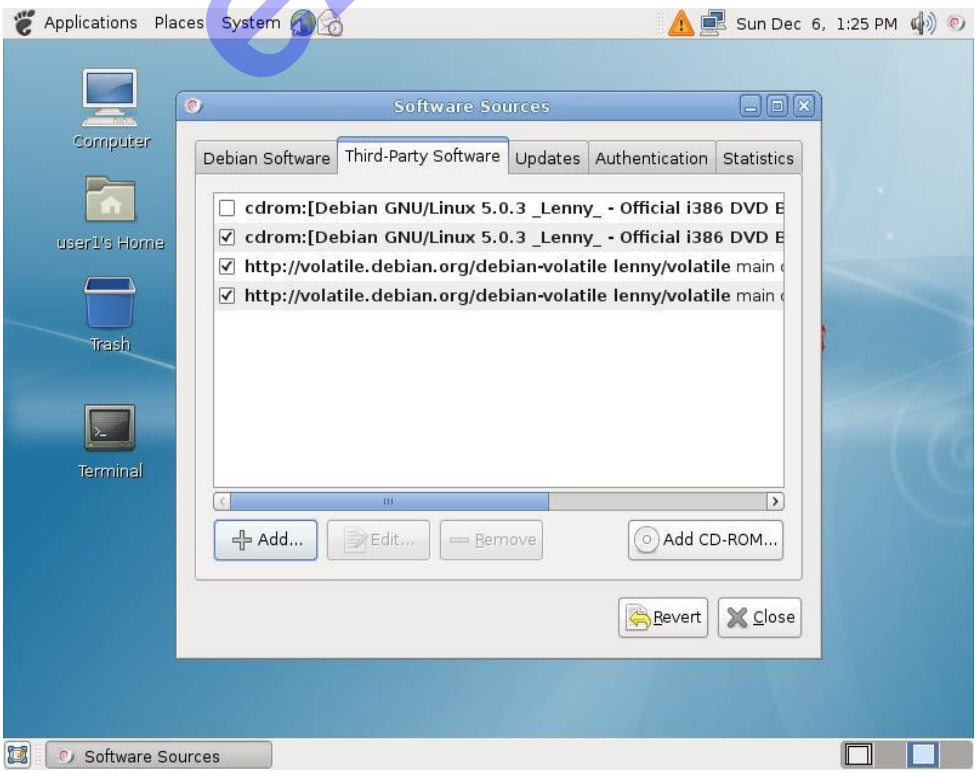
Pentru a clarifica tipurile menționate anterior, putem da câteva exemple:

Linia **deb http://archive.debian.org/debian-archive main** va menționa că sursă de pachete serverul HTTP de la adresa menționată, însă folosind doar zona **main**.

Linia **deb ftp://ftp.debian.org/debian stable main** va folosi ca sursă serverul ftp de la adresa menționată, sub directorul **/debian**, accesând doar zonele stable şi main.

Pentru mai multe informații, puteți accesa pagina de manual corespunzătoare comenzii apt prin intermediul comenzii **man sources.list**. Ca observație, utilizatorul trebuie să aibă grijă mereu de unde descărca pachete, fiindcă folosind un repository necunoscut acesta poate permite instalarea unor pachete compromise pe sistem. De asemenea, orice modificare a fişierului **sources.list** trebuie să fie urmată de comandă **apt-get update** astfel încă baza de date folosită de apt să poată fi actualizată.

Alternativ, pe lângă editarea manuală a fişierului **sources.list** putem folosi programul **Software Sources** (accesibil prin meniul **System -> Administration -> Software Sources**) care oferă o interfață mult mai uşor de folosit pentru modificarea listei de arhive.

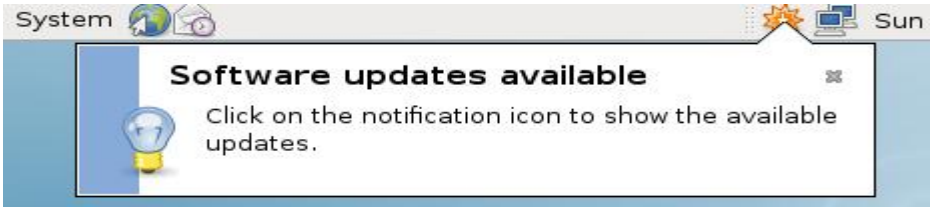


Observăm în lista din poză de mai sus, o înşiruire a tuturor surselor care sunt incluse de către apt, mai exact, vedem o variantă puțin interpretată a fişierului **sources.list**. Fiecare linie poate fi activată sau dezactivată prin bifarea, respectiv debifarea acesteia şi se traduce prin adăugarea caracterului „#” la începutul liniei respective în cadrul fişierului **sources.list**. Acest caracter simbolizează existența unui comentariu în cadrul fişierului, ducând la ignorarea liniei respective. Mecanismul respectiv este foarte util pentru a nu forța utilizatorul să piardă astfel de surse, oferindu-i în acelaşi timp posibilitatea excluderii acestora. Pe lângă dezactivarea unui repository, mai există şi opțiunea de adăugare de noi arhive prin acționarea butonului **Add**, fiind necesară doar linia apt, ce descrie sursa (formatul acesteia este cel prezentat mai devreme, în cadrul fişierului **sources.list**).

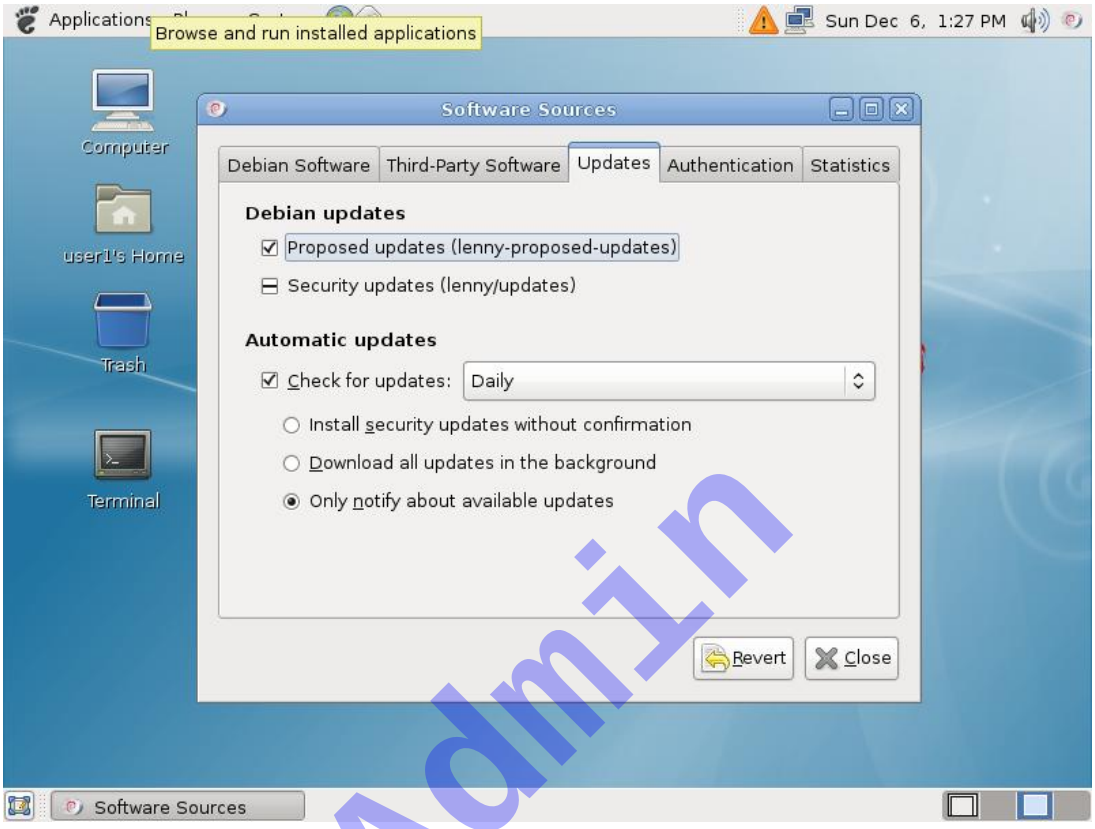
Actualizarea sistemului (Update)

Actualizarea versiunilor programelor ce rulează pe un sistem la un moment dat poate fi o sarcină dificilă chiar şi pentru utilizatorii experimentați: aceştia trebuie să monitorizeze permanent o serie de producători pentru a afla când vor lansa o versiune mai nouă (şi în general îmbunătățită) a aplicațiilor pe care le folosesc. Monitorizarea poate fi dificilă, dat fiind faptul că pot interacționa multe entități cu politici diferite: unii dezvoltatori de aplicații pot lansa versiuni noi zilnic, pe când alții o pot face şi la intervale de ani de zile. Pe lângă această problemă, administratorii mai au şi sarcina de a menține sistemul în stare funcțională cât mai mult timp posibil, iar actualizarea versiunii unui serviciu – precum cel de http – presupune mai întâi oprirea serviciului respectiv, şi apoi actualizarea sa. Combinată cu alte sarcini pe care administratorii le pot avea în cadrul unui sistem Linux, pot transforma această sarcină într-una dificilă, dar esențiala – mare parte din îmbunătățirile versiunilor mai noi se referă la neajunsuri anterioare sau vulnerabilități de securitate care nu pot fi ignorate.

Continuând această idee, s-a dezvoltat – inițial pentru distribuția Ubuntu, apoi inclusă şi în Debian – aplicația **Update Manager**, care are rolul de a automatiza procesul de actualizare a programelor instalate într-un sistem Linux. Aceasta îşi îndeplineşte rolul printr-un daemon de fundal, şi poate atenționa utilizatorul atunci când apar noi versiuni ale pachetelor existente, şi opțional le poate descărca şi instala. **Update Manager** este ca şi celelalte aplicații de gestiune a pachetelor, o interfață pentru sistemul APT, însă nu poate fi folosit pentru adăugarea sau ştergerea pachetelor. Programul rulează automat şi discret, în distribuțiile mai recente Debian, necesitând un minim de interacțiune cu utilizatorul, însă poate fi lansat accesând meniul **System -> Administration -> Update Manager**. Utilizatorul este atenționat mereu prin intermediul unei pictograme în panoul de sistem:



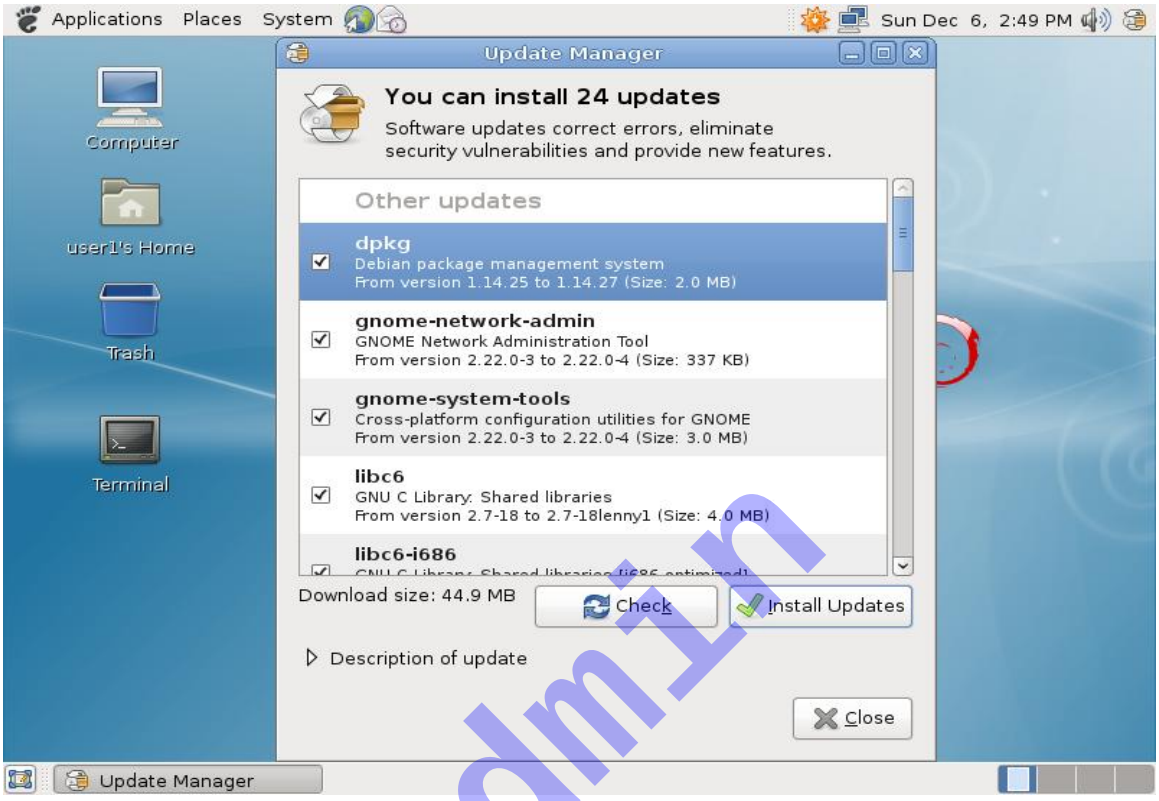
Este important de ştiut faptul că, precum şi celelalte programe bazate pe sistemul APT, **Update Manager** detectează existența unor noi versiuni şi descarcă, precum şi instalează noile pachete folosind APT. Mai mult, pentru a căuta noi versiuni, acesta trebuie să aibă acces la arhivele de pachete menționate în fişierul **sources.list**, sau prin cadrul programului **Software Sources**.



În imaginea precedentă, putem observa câteva dintre opțiunile pe care **Software Sources** ni le oferă în relația cu sistemul automat de actualizare:

- **Proposed updates** - în esență, pachetele noi trebuie să treacă printr-o perioadă de testare ceva mai lungă până să fie incluse în arhivele majore. Pachetele marcate ca fiind „proposed" sunt de regulă ceva mai noi, şi nu au beneficiat de întregul mecanism de testare. Ca atare, o primă consecință a bifării acestei opțiuni va fi aceea că sistemul va beneficia de mai multe actualizări, mai repede. Deşi pentru un sistem normal de tip „desktop" această opțiune poate fi folosită cu încredere, nu se recomandă bifarea în cazul unui server aflat în producție.
- **Security updates** – marchează pentru verificare, descărcare şi instalare, toate actualizările referitoare la securitate. Această opțiune este una foarte importantă, deoarece se va ocupa de repararea oricăror probleme de securitate şi nu poate fi dezactivată prin mijloace convenționale.
- **Check for updates** – setează intervalul la care daemon-ul responsabil cu actualizarea să caute pachete noi.
- **Install security updates without confirmation** – orice versiune nouă de pachet va fi instalat în mod automat, fără nici un fel de confirmare din partea utilizatorului.

- **Download all updates în the background –** va specifica faptul că utilizatorul doreşte să descarce noile pachete în fundal
- **Only notify about available updates** – este şi modul implicit de funcționare, în care utilizatorul va fi atenționat de existența unor noi pachete, acesta putând să aleagă ulterior dacă doreşte să le instaleze sau nu.



Utilizatorului îi este prezentată o fereastră în care poate obține informații detaliate despre actualizarea fiecărui pachet, prin selectarea pachetului dorit, şi vizualizarea secțiunii **„Description of update”**, localizată în partea de jos a ferestrei. Pe lângă aceste informații generale, mai se precizează şi dimensiunea totală ce va fi descărcată. Pachetele pot fi actualizate preferențial, prin bifarea sau debifarea fiecărei intrări din lista vizibilă şi nu se va efectua nici o schimbare asupra sistemului până nu se acționează butonul „**Install Updates**”.

OBSERVAȚIE: Actualizarea oricărui program presupune întreruperea acestuia în cazul în care rulează. Ca urmare imediată, se recomandă salvarea oricăror documente, şi oprirea tuturor programelor pe toată durata actualizării.

Actualizarea pachetelor se poate face şi fără existența unei interfețe grafice, prin intermediul utilitarului **apt-get**, folosind directiva **upgrade**. Se recomandă folosirea parametrului **–u** pentru afişarea informațiilor despre pachetele ce vor fi actualizate – în caz contrar, aceasta făcându-se în fundal: **apt-get –u upgrade**

O altă caracteristică foarte importantă a sistemului de gestiune APT este aceea că permite actualizarea întregii distribuții, prin executarea comenzii: **apt-get dist-upgrade**

Procese Linux

Aşa cum am menționat anterior, sistemul Linux este un sistem de operare multi-user cu capabilități multi-tasking. Dar ce înseamnă de fapt aceşti doi termeni? Pentru a răspunde la această întrebare, aceştia trebuie analizați.

Termenul multi-user se referă în mod concret la capacitatea accesării unui sistem informatic sau al unui program de către mai mulți utilizatori concurenți. Putem trage concluzia că Linux este un sistem de operare multi-user din moment ce permite utilizarea resurselor disponibile de către mai mulți utilizatori simultani, însă astfel se iveşte o nouă problemă: nevoia de multi-tasking.

Spre deosebire de alte sisteme de operare tip DOS (Disk Operating System), precum MS-DOS, care sunt single-user, single-tasking, mai precis nu permit accesul mai multor utilizatori simultani, şi nici execuția în paralel a mai multor programe, o platformă multi-user ar avea nevoie de capabilități multi-tasking pentru a opera eficient, altfel utilizatorii conectați la sistem ar trebui să îşi desfăşoare activitatea pe rând, unul după altul. Deşi DOS este în esență un sistem single-tasking (adică nu permite execuția mai multor programe simultan) şi numai un singur program poate fi încărcat în memorie la un moment dat, există mecanisme de evitare a acestei probleme: TSR sau Terminate and Stay Resident.

Acest mecanism forța sistemul de operare să păstreze un program în memorie şi să modifice vectorul de întreruperi al sistemului astfel încât acesta să poată fi apelat chiar şi după terminarea sa, şi a fost considerat o formă primitivă de multi-tasking, reprezentând o platformă pentru construirea unor driveri ce nu erau suportați nativ de sistem, asigurând totuşi funcționarea sistemului. Dezavantajul acestui sistem era faptul că programele TSR trebuiau păstrate într-o zonă de memorie foarte limitată – primii 640KB de memorie convențională, rezultatul fiind reprezentat atât de numeroase conflicte cu alte programe de tip TSR cât şi lupta continuă de a micşora dimensiunea acestora astfel încât să păstreze cât mai multă memorie liberă.

Deşi o soluție simplă la nivelul sistemelor de atunci, TSR nu este nici pe departe un sistem adevărat de gestionare a proceselor, cum este cel oferit de kernel-ul Linux.

În realitate, conceptul de multi-tasking este de fapt mult mai complicat şi poate fi abordat în multe feluri, fiind de fapt execuția mai multor procese individuale folosind o resursă comună – un procesor. În cazul unui calculator cu un singur procesor, este imposibil ca acesta să execute două instrucțiuni diferite în exact acelaşi moment în timp, însă multe sisteme de operare moderne permit execuția simultană a mai multor aplicații - precum vizualizarea paginilor de interne în timp ce utilizatorul asculta muzică – folosind un singur procesor. Mult simplificat, mecanismul poate fi explicat prin programarea în timp a permisiunii de execuție a fiecărui proces în parte – fiecare fiind executat pe rând de către procesor pentru un timp stabilit. Prin

alocarea timpului de procesor între procese diferite la intervale foarte mici, se crează iluzia de execuție în paralel – însă, realitatea este că părți din fiecare program sunt executate secvențial.

Există mai multe implementări ale acestui sistem de partajare a timpului de execuție, însă un sistem de gestiune al întregului proces este esențial. Primele tentative de execuție în paralel se numeau „multiprogramming”, şi au apărut ca urmare a nepotrivirii dintre viteza mare de execuție a unui procesor comparativ cu echipamentele periferice. Dezavantajul major al acestui sistem era însă faptul că nu se putea garanta în niciun fel execuția în timp util al unei aplicații, maximizând însă gradul de utilizare al procesorului.

O altă variantă de multi-tasking, ceva mai evoluată decât precedentă, este modelul time-sharing (sau multi-tasking cooperativ), unde utilizatorul putea executa programe diferite simultan, ca şi când acestea erau singurele care rulau de pe sistem. Printre sistemele de operare care folosesc acest mecanism se numără primele variante de Windows şi Mac OS. Prin acest model, fiecare program în execuție pe sistem se angajează să acorde timp de proces şi celorlalte procese, dezavantajul major fiind acela că în cazul execuției unei aplicații prost gândite, întregul sistem se poate bloca.

Pentru a sumariza cele spuse, putem concluziona că un proces reprezintă un program în execuție şi are ataşate o serie de informații specifice precum instrucțiunile programului, resurse folosite (precum fişiere deschise), unul sau mai multe fire de execuție şi alte informații necesare procesului de execuție în paralel.

Caracteristicile proceselor

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 naştere procesului curent (cunoscut şi ca proces părinte)
- **Terminalul ataşat** – prescurtat şi TTY, reprezintă terminalul la care procesul curent este legat.
- **RUID –** Real User ID, reprezintă identificatorul utilizatorului care a lansat aplicația. Similar există şi EUID (sau Effective User ID) pentru identificarea drepturilor reale la resursele sistemului.
- **RGID şi EGID** – similar cu RUID şi EUID, doar că se referă la identificatorul grupului de utilizatori.
- **factorul nice** – folosit pentru a determina, aşa cum sugerează şi numele, „factorul de prietenie” al procesului cu scopul stabilirii priorității de execuție (bazat pe factorul nice şi istoricul de utilizare al procesorului)

Procesele pot rula în mod normal în două moduri: **foreground** respectiv **background**. Dintre procesele interactive, marea majoritate funcționează în modul foreground. Diferența dintre cele două moduri se rezumă strict la interacțiunea cu interfața cu utilizatorul, mai exact la ocuparea consolei din

care respectivul proces a fost lansat. În acest mod, consola nu va fi eliberată până la terminarea programului care o ocupă. În cazul comenzilor prezentate până acum, vorbim tot despre procese ce rulează în foreground – de exemplu, şi comanda **ls** face parte din această categorie, însă datorită timpului foarte scurt de rulare acest fapt este doar o iluzie. Un contra exemplu ar fi execuția comenzii **less** care nu va termina execuția odată ce a terminat afişarea pe ecran, ci va aştepta în permanență comenzi de la utilizator.

Ca alternativă, programele pot rula şi în modul **background** pentru a permite terminalului de pe care rulează să primească şi alte comenzi în acelaşi timp, execuția având loc în fundal şi fiind guvernată de aceleaşi reguli ca orice alt proces. Acestea sunt procesele care de regulă au un timp de execuție mare sau nu necesită interacțiunea cu utilizatorul. Serviciile ce rulează pe un calculator fac şi ele parte din această categorie, însă există câteva diferențe.

Dacă interacțiunea cu programe care ruleaz ă în foreground este evidentă (acestea de multe ori aşteptând gesturile utilizatorului), interacțiunea cu programele ce rulează pe fundal (background) poate fi ceva mai dificilă. Pentru a putea gestiona aceste procese, shell-ul ne pune la dispoziție o serie de utilitare:

- **comanda jobs** va afişa toate comenzile ce rulează în fundal. Fiecare proces ce rulează în fundal va avea asociat un număr întreg reprezentând identificatorul şi poate fi folosit pentru referentierea procesului respectiv. Acest identificator este diferit de câmpul PID (sau process identifier) şi se referă doar la consola curentă.
- comanda **kill** este folosită pentru terminarea oricărui proces, specificat prin menționarea PID-ului procesului țintă ca parametru.
- comanda **fg %n** va aduce procesul cu identificatorul **n** înapoi pe fundal.
- similar, comanda **bg %n** va reactiva un proces suspendat în fundal.

Procesele pot fi afectate şi prin combinații de taste precum **CTRL+Z** care va suspendă (dar nu termină) procesul ce rulează în foreground. Similar, combinația **CTRL+C** este foarte cunoscută pentru funcția sa de terminare a procesului curent.

Orice comandă executată în consolă va rula implicit în modul foreground, însă pentru a forța o comandă să ruleze în fundal trebuie adăugat caracterul & după comandă:

```
user1@computer1:~$ top &
[1] 3180
user1@computer1:~$ jobs
[1]+  Stopped                 top
user1@computer1:~$ _
```

În exemplul de mai sus observăm cum prin apelarea unui program ce nu îşi termină execuția decât la comanda utilizatorului (**top**) împreună cu **&** acesta este trecut pe fundal. Prin apelarea comenzii **jobs** se vor afişa o serie de informații despre procesele din fundal.

Servicii

O categorie foartă importantă în orice sistem de operare modern, serviciile oferă diverse funcționalități fie local, fie unei rețele de calculatoare. Acestea sunt ca orice alt program într-un sistem Linux, cu mențiunea că sunt denumiți **Daemons** şi că rulează în continuu, de regulă pornind odată cu sistemul. Programele de acest tip rulează de regulă în fundal şi aşteptă cereri către serviciile pe care le oferă, însă despre servicii vom vorbi pe larg în capitolele următoare.

Ciclul de viață al proceselor

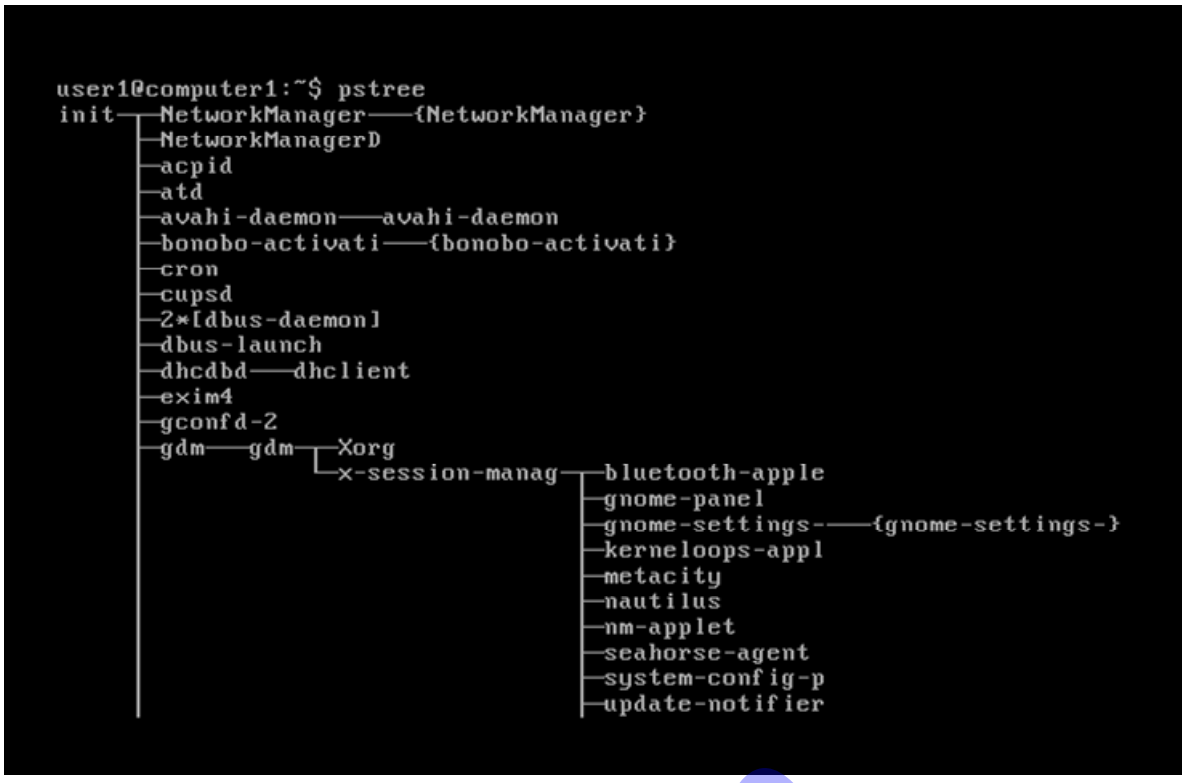
Ca orice entitate dinamică, procesele au un ciclu de viață foarte bine definit, acestea născându-se, evoluând şi terminându-se după reguli bine puse la punct. Nevoia creării unui proces nou este evidentă: acest lucru se întâmplă ori de cât ori este nevoie să fie instantiat un nou program.

Procesele în Linux, urmează o structură de organizare foarte bine definită: aceea de arbore, cu procese ce reprezintă rădăcina arborelui, nodurile şi frunzele acestuia. Procesele în cadrul sistemului Linux pot fi create numai din alte procese, acestea devenind **procese fiu** (sau **children**). Urmând structura de arbore, fiecare proces poate fi reprezentat printr-un nod, rădăcina fiind părintele tuturor proceselor create. Nodurile din arborele de procese, pot avea la rândul lor alte procese de tip fiu, creându-se astfel o structură ierarhică foarte bine definită.

Crearea unui proces implică apelarea mecanismului numit **fork**, prin care acesta crează o copie identică a sa, însă cu un **PID** diferit. După terminarea mecanismului de forking, spațiul de adresare al procesului este înlocuit cu cel reprezentativ pentru noul proces (procedeu numit **exec**) şi astfel, noul proces este gata de execuție. În ceea ce priveşte execuția propriu-zisă, o practică comună este aceea de transformare în **daemon** a proceselor fiu ale unui program, pentru ca acestea să poată rula chiar şi după terminarea procesului părinte. Cazuri excepționale pot avea loc atunci când un proces fiu îşi termină execuția, în timp ce procesul părinte nu aşteaptă acest lucru; acest tip de procese se numesc procese **zombie**.

Terminarea execuției unui proces în mod normal (dacă nu este terminat forțat) are loc printr-un mecanism ce returnează un rezultat procesului părinte (numit **exit status**). Acest rezultat reprezintă un număr, ce poate oferi indicii despre modul în care procesul şi-a derulat activitatea, mecanismul fiind împrumutat din limbajul **C**, unde orice funcție poate returna un rezultat. Codurile de ieşire pot fi interpretate de către procesul tată şi este specific fiecărui program. În momentul în care execuția unui proces tată este terminată, procesele sale fii pot fi terminate sau nu, în funcție de natura acestora, însă este bine de ştiut că nu există un o regulă fixă.

Afişarea structurii ierarhice în care sunt organizate procesele poate fi făcută în mai multe feluri, probabil cea mai comună fiind apelarea comenzii **pstree**:

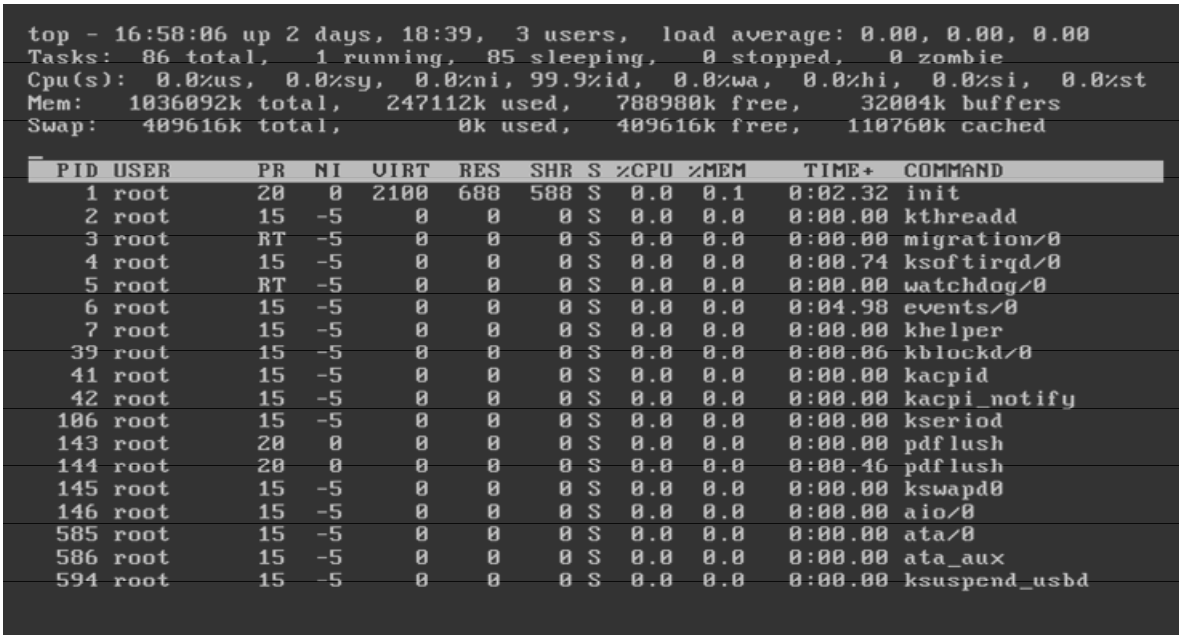


Durata de funcționare a sistemului; uptime şi downtime

Unul din avantajele majore al sistemului de operare Linux este stabilitatea sa, motiv pentru care este şi atât de răspândit în aplicații unde este nevoie de siguranță şi durată mare de funcționare. O astfel de aplicație ar putea fi un calculator care se ocupă de servicii bancare şi pe care banca respectivă nu-şi poate permite să-l oprească deloc, sau unde siguranța datelor procesate este foarte importantă. În terminologia folosită pentru servere, apar astfel doi termeni legați de disponibilitatea acestora: **uptime** şi **downtime**. Primul se referă la durata de funcționare a sistemului de când a fost pornit, pe când cel de-al doilea vine în completare şi măsoară timpul în care serverul respectiv nu a fost operațional. Deşi nu există o modalitate corectă de a măsura downtime-ul local (acesta fiind măsurat extern de către alte sisteme), se poate determina uptime-ul unui sistem, ştiindu-se exact momentul în care sistemul a pornit cât şi ora curentă. Pentru afişarea uptime-ului se foloseşte comanda **uptime**:

```
computer1:/etc/rc3.d# uptime
 16:57:28 up 2 days, 18:38,  3 users,  load average: 0.00, 0.00, 0.00
computer1:/etc/rc3.d# _
```

Pe lângă comanda **uptime**, se mai poate folosi şi comanda **top**, utilă nu numai pentru determinarea timpului de funcționare al calculatorului, cât şi pentru afişarea proceselor care rulează în acel moment, totul în timp real. În plus, pe lângă aceste informații, comanda **top** va mai afisa şi alți parametrii importanți ai sistemului, precum numărul de procese şi informații despre memorie.

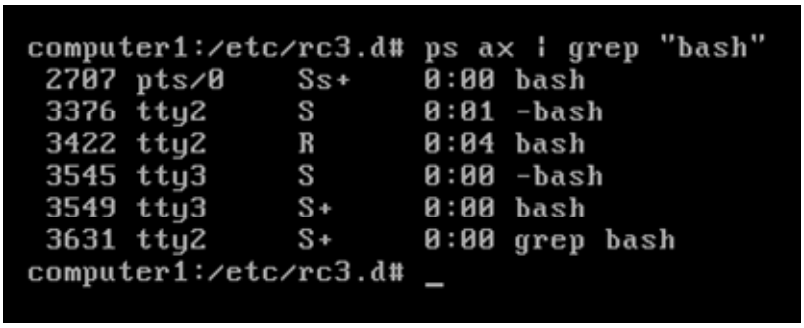


Terminarea forțată a proceselor

În ciuda stabilității foarte mări a sistemelor Linux, uneori se întâmplă că procesele ce rulează să se blocheze sau să funcționeze anormal, făcând astfel oprirea lor prin mijloacele convenționale imposibilă. Linux oferă o stabilitate foarte mare a sistemului, acesta putând să continue operarea normală chiar şi în cazul în care unul din procesele în curs de derulare se blochează. În acest caz, pentru a aduce sistemul la o stare de normalitate, se folosesc mecanisme de terminare forțată a proceselor.

Unic pentru fiecare proces ce rulează la un moment dat, este atributul PID (sau identificatorul numeric unic) al procesului, care poate fi folosit pentru a comanda terminarea sa. Pentru aflarea PID-ului unui proces, se pot folosi mai multe tehnici care implică folosirea următoarelor comenzi:

- comanda **ps** afişează o listă a tuturor proceselor ce rulează în mod curent. Dat fiind faptul că foarte multe procese pot rula la un moment dat (având ca rezultat o listă de procese ce se întinde pe mai multe ecrane), se recomandă folosirea împreună cu comanda **less**. Parametrii cei mai utili sunt -ax (unde parametrul **a** forțează afişarea tuturor proceselor iar **x** elimină restricția de afişare numai a proceselor ce rulează dependent de o consolă)
  EXEMPLU: ps ax | less va lista toate procesele, câte unul pe linie, afişând totodată şi PID-ul fiecăruia.
- comanda **top** prezentată anterior poate fi şi ea folosită pentru afişarea proceselor ce rulează în momentul curent, însă este mai puțin folositoare datorită cantității limitate de informații pe care le poate afişa la un moment dat.
- o altă metodă de folosire a comenzii **ps** este în concordanță cu comanda **grep** pentru filtrarea rezultatelor la un număr mai mic: **ps ax | grep „nume_proces”**. Acest mod de apelare va filtra rezultatele, afişând informații numai despre procesele al căror nume conține şirul de caractere menționat între ghilimele.



Această comandă poate fi extrem de utilă dat fiind faptul că mai multe procese cu nume identifice pot rula la un moment dat.

- comenzile **pidof** şi **pgrep** pot fi folosite pentru afişarea tuturor identificatorilor PID ce rulează sub numele menționat în parametru. Urmărind exemplul anterior, comanda **pgrep bash** sau **pidof bash** ar fi afişat PID-urile vizibile în poză. De regulă, comanda **pgrep** este mai utilă, fiindcă permite o libertate mai mare în specificarea şirului de caractere ce reprezintă numele procesului căutat.

OBSERVAȚIE: folosirea comenzilor împreună cu caracterul „|” simbolizează folosirea nu a uneia, ci a două comenzi. Pe scurt, acest caracter specifică faptul că datele primei comenzi afişate pe ecran(output-ul), vor reprezenta datele de intrare (sau input) pentru comanda **grep**, care are rolul de a filtra text. Ca urmare, deşi comanda **ps ax** ar returna informații despre toate procesele existente, afişarea propriu-zisă pe ecran se rezumă la doar câteva, datorită comenzii **grep**.

Odată aflat PID-ul unui proces, se poate trece la terminarea acestuia prin intermediul comenzii **kill**. Aceasta poate primi o serie de parametrii, însă de regulă este folosit doar cu unul singur: PID-ul procesului țintă (dat fiind exemplul anterior, comanda **kill 3376** ar comanda terminarea procesului cu PID-ul respectiv).

Modul în care kill funcționează este prin folosirea sistemului de semafoare aferent sistemului de control al proceselor. Comandă va trimite o cere de terminare normală procesului țintă, care va returna (ca orice alt proces) un rezultat. Există cazuri în care procese blocate nu vor răspunde unor astfel de comenzi, când este necesară terminarea forțată prin includerea parametrului **-9** (**kill -9 3376**). Terminarea proceselor în acest mod nu este recomandată, dat fiind faptul că poate întrerupe procese importante în mod necontrolat.

Pe lângă comanda **kill** (care necesită neapărat PID-ul procesului țintă), se mai poate folosi şi comanda **pkill** care va lua ca parametru direct numele procesului. Spre deosebire de comandă **pgrep,** comanda **pkill** nu va afişa PID-urile proceselor, ci le va comanda terminarea.

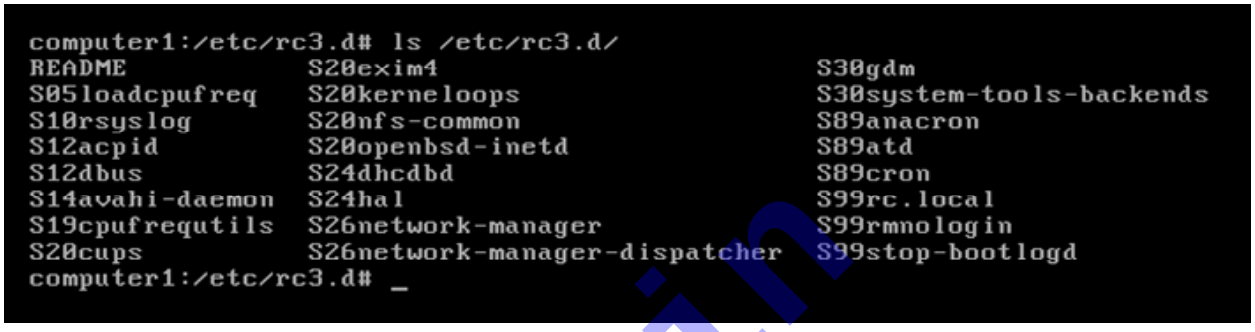
Procesul INIT

După cum se vede în structura de procese prezentată în poza anterioară, procesul rădăcină, care este părintele tuturor proceselor ce rulează la un moment dat, se numeşte **init**. Procesul init are rolul de a inițializa sistemul într-un mod prestabilit, după ce kernel-ul sistemului a fost încărcat în

totalitate. Mai exact, init va efectua sarcini precum montarea sistemului de fişiere, pornirea serviciilor aferente sistemului, sau chiar încărcarea interfeței grafice.

În Linux, init este executat cu un parametru, cunoscut şi ca **run-level**, prin care se specifică modul în care se doreşte inițializarea sistemului. Valoarea acestui parametru este un număr întreg între 1 şi 6, şi semnalează ce componente ale sistemului vor fi inițializate, fiecare nivel având o serie de script-uri ce vor fi executate. Aceste scripturi sunt de obicei stocate în directoare dedicate fiecărui nivel (de regulă în **/etc/rc[n].d** unde **[n]** reprezintă run level), iar fişierul principal de configurare în **/etc/inittab**.

De menționat este faptul că script-urile conținute în directoarele ce reprezintă fiecare run-level nu sunt de fapt fişiere propriu-zise, ci legături către alte scripturi (care se află de regulă în **/etc/init.d**). Motivul acestei implementări este că se poate modifica comportamentul fiecărui run-level independent de celelalte, fără să existe 6 copii diferite ale scripturilor folosite.



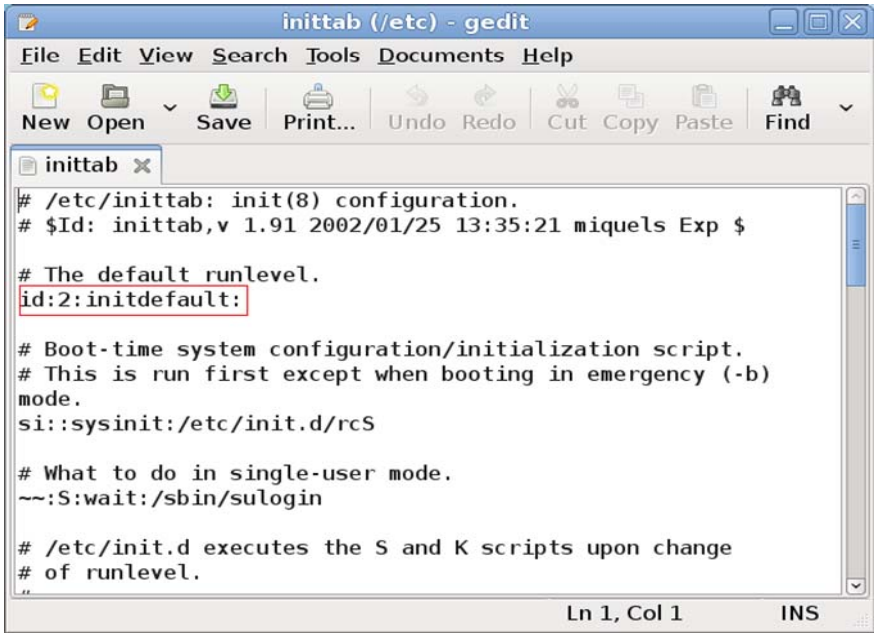
În poza de mai sus, se poate observa lista tuturor scripturilor atribuite nivelului 3. Se poate observa că acestea urmează un algoritm de denumire caracteristic, pentru a putea determina ordinea în care aceste sunt executate (alfabetic). În cazul în care se doreşte executarea unui script pentru pornirea unui serviciu înaintea altuia, numele primului trebuie să fie înaintea celui de-al doilea din punct de vedere alfabetic.

După terminarea inițializării procesului init, acesta este clonat prin procedeul **fork**, devenind un nou proces, numit **getty.** Acesta are rolul de a porni consolele virtuale aferente sistemului, de regulă în număr de 6 (acest număr fiind configurabil în /etc/inittab).

Fişierul de configurare **inittab** poate de asemenea să specifice modul în care sistemul va reacționa la combinația de taste CTRL+ALT+DEL (reset în mod implicit) – se doreşte închiderea corectă a sistemului spre deosebire de închiderea imediată.

Nivele run-level pot să însemne lucruri diferite în funcție de distribuția folosită, însă în Debian, există următoarele nivele:

- **nivelul 0 –** halt (shutdown)
- **nivelul 1 –** single user
- **nivelele 2-5 –** multi-user cu console şi mod grafic (distribuția Debian nu face nici o diferență între modurile 2,3,4 şi 5)
- **nivelul 6** – reboot



Fire de execuție (THREADS)

Firele de execuție, cunoscute şi sub numele de threads (din limba engleză), ca şi procesele, reprezintă un mecanism de execuție în paralel a instrucțiunilor, facilitate esențială pentru orice sistem de operare modern, cu capabilități multi-tasking.

Deşi termenul este folosit foarte des drept sinonim pentru proces, există diferențe foarte mari între acestea. Procesele în Linux se referă la instanțe ale unor programe în execuție, adică inclusiv variabile, informații de stare şi în general întregul spațiu de memorie al respectivului program, pe când firele de execuție se referă la execuția în cadrul unui proces. Deşi firele de execuție şi procesele funcționează în mod diferit în funcție de sistemele de operare pe care rulează, există totuşi un punct comun: firele de execuție pot să împartă accesul la informațiile de stare caracteristice unui proces fără să fie nevoie să apeleze la mecanisme de comunicare complicate precum comunicarea prin fişiere, pipes sau sockets. Firele de execuție pot efectua schimbul de informații în mod direct prin intermediul variabilelor sau al altor structuri de memorie. Consecința directă a acestui fapt este că schimbarea contextului de execuție se poate face mai rapid pentru fire de execuție decât pentru procese.

Principalul avantaj al firelor de execuție este acela al performanței în cazul aplicațiilor; o bază de date de exemplu, care este caracterizată de un volum foarte mare de operații de intrare/ieşire poate obține o îmbunătățire foarte mare a performanței dacă foloseşte o abordare orientată pe fire de execuție. Un alt exemplu de utilizare al firelor de execuție ar putea fi un program care necesită interacțiunea permanentă cu utilizatorul în timp ce se execută operații pe fundal; abordarea normală în acest caz este implementarea proceselor costisitoare (din punct de vedere al timpului de execuție) folosind conceptul de „worker thread” (sau fir de execuție „lucrător”), care va rula în mod paralel cu firul de execuție principal al programului, păstrând însă responsivitatea acestuia.

Principalul dezavantaj al firelor de execuție este dificultatea de a le implementa în mod corect, fapt datorat mecanismului de funcționare mai greu de înțeles, cât şi diverselor conflicte ce pot să apară datorită accesului mai multor entități la resurse comune.

Concluzionând, procesele pot fi reprezentate drept un fir de execuție alături de spațiul sau de adresare şi alte informații, şi pot fi compuse din mai multe fire de execuție.

Conceptul de pipe

În cadrul sistemelor de operare de tip UNIX, conceptele de **pipe** şi **pipeline** se referă la mecanisme de redirectare ale fluxurilor de date între procese, având că principală întrebuințare comunicarea între procese diferite, sau implementarea unor filtre. Un exemplu bun de folosire al conceptului poate fi regăsit în cadrul unor comenzi prezentate anterior: **ps ax | less**. În acest caz, avem de-a face cu două comenzi separate, **ps** (cu parametrii menționați, listează procesele existente în momentul execuției) şi **less** (care permite parcurgerea şi navigarea în cadrul textelor lungi). Caracterul „|” poate fi observat între cele două comenzi, şi reprezintă mecanismul obişnuit de creare al unui pipe. Comanda anterioară ar avea ca efect, afişarea numelor tuturor proceselor ce rulează la momentul execuției în sistem şi ar depăşi cu uşurință cantitatea maximă de informații ce poate fi afişată într-un singur ecran.

Comanda anterioară poate fi rezumată astfel: redirecționează fluxul standard output (ceea ce se afişează pe ecran) rezultat din execuția comenzii **ps** direct către aplicația **less**, care îl va putea prezenta utilizatorului într-o formă uşor de citit. Un alt exemplu de folosire al pipe-urilor este filtrarea datelor, această abordare fiind uneori mai eficientă decât căutarea manuală. Filtrarea se poate face redirectand fluxul standard output spre un program care în loc să ofere facilități de navigare al conținutului primit, va afişa numai informațiile dorite, pe baza unor parametrii. Un astfel de program este **grep**, care folosit simplu cu un singur parametru va afişa doar liniile ce conțin textul specificat: comanda **ps ax | grep bash** va redirecționa rezultatul (lista de procese) către comandă **grep**, care în baza parametrului specificat, va afişa doar acele linii care conțin şirul de caractere „bash”.

Redirecționarea fluxurilor de intrare/ieşire

Urmând filozofia Linux conform căreia totul este un fişier, chiar şi tastatura, mouse-ul sau alte dispozitive sunt reprezentate prin fişiere. Principalul avantaj al acestei abordări este uşurința cu care se pot interfața dispozitivele de către programatori, cât şi posibilitatea de a trimite, a primi sau a manipula datele primite de la dispozitive.

De exemplu, fişierul **/dev/input/mice** este folosit pentru a primi informații de la mouse, şi folosind comanda **cat /dev/input/mice** (urmat de acționarea mouse-ului) într-un terminal vom putea observa schimbările. Comanda **cat** are rolul de a concatena conținutul unui fişier pe standard output (adică pe ecran).

Linux foloseşte 3 fluxuri importante de intrare/ieşire, codificate numeric: 0 – standard input (tastatură), 1 – standard output (ecran), 2 – standard error.

Shell-ul ne permite să redirecționăm aceste fluxuri pentru o flexibilitate foarte mare, folosind operatorii „**>**” (redirectarea fluxului de ieşire) şi „**<**” (redirectarea fluxului de intrare). Comanda **ps ax > fişier.txt** ar avea ca rezultat, un efect similar cu cel observat la pipe-uri: **ps ax** ar afişa toate procesele ce rulează, însă rezultatul în loc să fie afişat pe standard output, va fi redirectat către fişierul **fişier.txt** pentru examinare ulterioară. În cazul în care se apelează o comandă care poate genera o eroare (de exemplu, comanda ls pe un fiser inexistent), ar trebui redirectat fluxul de standard error: **ls fişier_inexistent.txt 2> fişier.txt** .

În cadrul sistemului Linux există o serie de fişiere speciale, ce pot fi folosite în diverse moduri. Unul dintre aceste fişiere este **/dev/null**, un fişier cu dimensiunea 0, în care orice modificare este imediat ştearsă. O utilizare a acestui fişier ar putea fi ascunderea mesajelor de eroare: **ls fişier_inexistent 2> /dev/null** ar trimite mesajul de eroare către /dev/null, fiind total ignorat.

O altă abordare a redirectarii poate fi folosită pentru tipărirea unui fişier prin intermediul imprimantei: comanda **cat fişier.txt > /dev/lp0** ar trimite conținutul fişierului **fişier.txt** către fişierul **lp0** care reprezintă imprimanta sistemului.

În mod invers, fluxul de intrare poate fi redirectat, astfel încât să reprezinte un fişier. Operatorul de redirectare al fluxului de ieşire către un fişier este în mod absolut distructiv, în sensul că va rescrie tot fişierul țintă cu noul conținut. Pentru a adăuga date la un fişier deja existent (modul append) trebuie folosit operatorul „**>>**”.

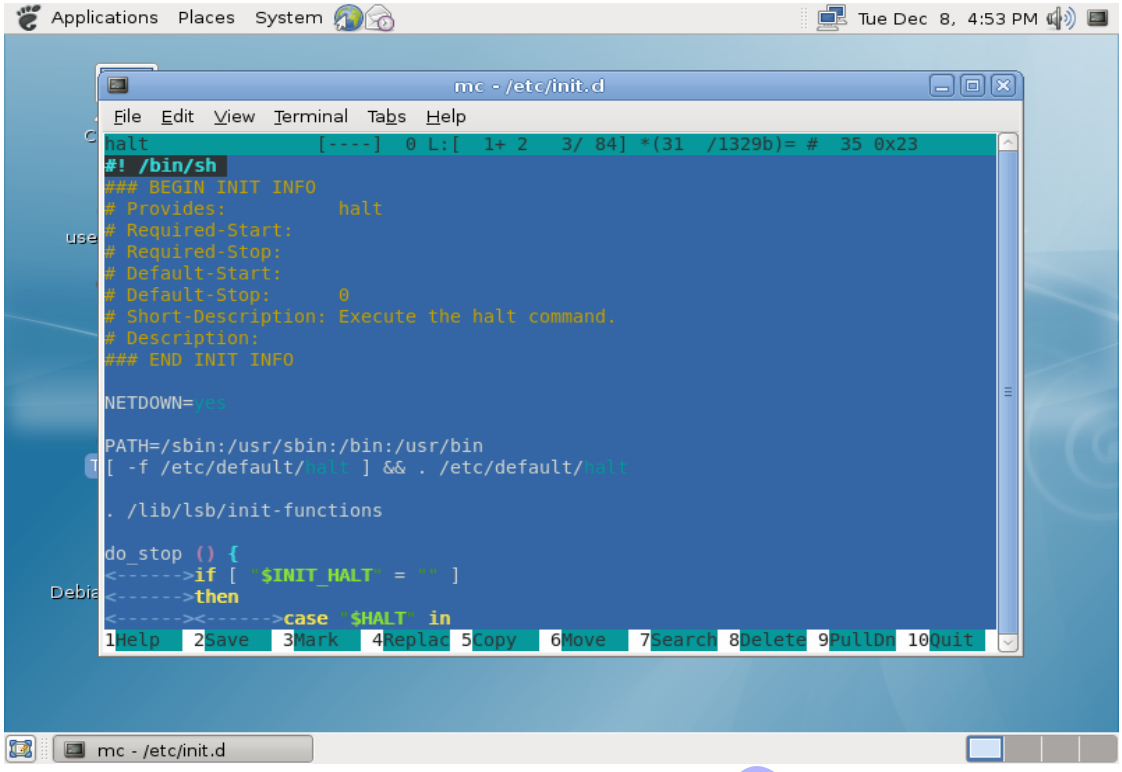
Script-uri bash

Bash, sau Bourne-Again Shell, este un interpretor de comenzi pentru sistemul de operare Linux. Acesta este shell-ul implicit pentru o serie foarte mare de distribuții Linux, printre care şi Debian, şi pe lângă facilitățile de interpretare a comenzilor lansate prin intermediul consolei, oferă şi posibilitatea scrierii unor script-uri pentru diverse sarcini, în general în scopul automatizării unor procese, sau pentru îmbunătățirea facilităților existente ale unui program. Un script este un fişier text, ce poate fi executat, al cărui conținut respectă un limbaj de scripting (în cazul de față **bash**). Fiind vorba de fişiere text, şi nu binare (compilate în cod maşină), scripturile nu pot fi executate propriu-zis; acestea sunt de fapt interpretate de un program specializat, de cele mai multe ori fiind vorba de interpretorul în a cărui limbaj a fost conceput respectiv script. În continuare vom studia procesul de creare al script-urilor **bash**, cât şi o parte din facilitățile pe care acest interpretor le oferă.

Conceptul din spatele scripturilor este simplu: comenzile din script vor fi interpretate linie cu linie, secvențial (sau cu salturi în cazul script-urilor mai complicate). Orice script bash, este de fapt un fişier, cu drepturi de execuție, şi pot fi lansate în mai multe feluri. Să presupunem în acest scop, că avem un script numit **s1** în directorul curent, pe care îl putem rula folosind unul din mijloacele de mai jos:

- Apelând una din comenzile **bash s1** sau **sh s1**. Această modalitate dezvăluie faptul că se foloseşte un interpretor pentru execuție. În acest caz, fişierul nu trebuie să aibă neapărat drepturi de execuție.
- Lansând direct scriptul cu ajutorul comenzii **./s1**. În acest caz, shell-ul îşi va da seama că este vorba de un script bash, şi va instantia interpretorul pentru acest fişier. Pentru a seta drepturi de execuție asupra unui script pentru orice utilizator este necesară rularea comenzii **chmod a+x nume_script**.

În continuare vom analiza compoziția unui fişier de tip script bash.



Se recomandă folosirea editorului **mcedit** pentru vizualizarea, modificarea sau crearea scripturilor bash (sau a multor altor tipuri de script-uri) datorită facilităților de sublienere colorată a sintaxei (syntax highlighting) oferite de acesta.

Orice script bash, va avea pe prima linie secvența de caractere „#!” urmată de locația interpretorului pentru care a fost scris. În scopuri didactice, vom crea în continuare un script simplu pornit de la reguli menționate anterior:

Pasul 1. Crearea fişierului se poate face cu ajutorul comenzii **touch**:

```
touch exemplu
```

Pasul 2. Setarea permisiunilor. Reamintim pe această cale, că se recomandă ca orice script să aibă drepturi de execuție, cel puțin pentru utilizatorii care ar trebui să-l poată rula.

```
chmod a+x exemplu
```

Pasul 3. Editarea fişierului în vederea introducerii comenzilor ce vor alcătui script-ul. Se recomandă folosirea editorului **mcedit** pentru acest pas. Mai jos, se poate observa un script foarte simplu prin care se şterge conținutul directorului **tmp** (presupunând că acesta există) din cadrul directorului home al utilizatorului curent, urmat de un mesaj de atenționare:

```
#!/bin/bash
rm ~/tmp/* 2>/dev/null
echo „Conținutul directorului temporar a fost golit.”
```

Se poate observa că întregul script este compus din comenzi obişnuite. Prima linie a script-ului semnalează faptul că avem de-a face cu un script şi include şi calea către interpretorul bash. A doua linie este ceva mai greu de înțeles la început, dar poate fi interpretată în felul următor: **rm** este comanda folosită pentru ştergerea fişierelor, caracterul **~** reprezintă simbolul pentru directorul home al uşerului curent, iar calea **~/tmp/*** se referă la toate

fişierele din cadrul directorului **tmp**, conținut în directorul home al utilizatorului curent. Ultima parte a liniei este o redirecționare; în secțiunile anterioare am discutat despre redirecționarea fluxurilor principale: standard out (1), standard error (2), respectiv standard în (3). Utilitarele pot trimite un mesaj de eroare către fluxul standard pentru a fi afişate pe ecran. În cazul de față, eroarea cea mai probabilă ar fi fost lipsa oricărui fişier din directorul **tmp**, utilizatorul fiind avertizat de acest lucru. Redirecționarea **2>/dev/null** nu face alteceva decât să redirecționeze toate mesajele de eroare generate de către **rm** către dispozitivul special **/dev/null** (care are rolul de a ignora orice fel de date scrise în el) astfel încât acestea să nu mai fie afişate pe ecran. Ultima linie face uz de comandă **echo** folosită pentru afişarea unui text pe ecran, în cazul de față fiind vorba despre o atenționare simplă.

În cazul în care exemplul de mai sus este greu de înțeles la început (mai ales pentru începători) o soluție bună este simplificarea sa:

```
#!/bin/bash
rm ~/tmp/*
```

În varianta simplificată, am scăpat de orice fel de tratare a mesajelor de eroare, cât şi de atenționarea asupra terminării operației de golire a directorului **tmp**. Aşa cum am menționat mai devreme, comenzile vor fi executate secvențial, iar **prompt**-ul (controlul tastaturii) va fi redat consolei după terminarea completă a execuției.

În continuare vom trata câteva din facilitățile mai avansate oferite de interpretorul bash.

Variabilele sunt nume simbolice al căror rol este să păstreze o valoare ce poate fi schimbată. Exemplul următor arată cum se pot folosi variabilele pentru stocarea şi afişarea unei valori:

```
#!/bin/bash
var=”un text”
echo $var
```

Ca urmare a execuției script-ului anterior, se va afişa pe ecran prin comanda **echo** conținutul variabilei **var.** Ca observație, semnul $ trebuie folosit în afişarea acesteia (acesta este modul prin care interpretorul bash îşi dă seama că are de-a face cu o variabilă şi nu un text oarecare).

Parametrii în linie de comandă reprezintă o facilitate importantă oferită de interpretor. Ca şi multe alte executabile ce se folosesc de parametrii primiți prin cadrul liniei de comandă, la fel şi scripturile pot beneficia de parametrii pasați în linia de comandă. Deşi linia de comandă este una singură, parametrii folosiți sunt de regulă separați prin spații. Accesul la aceşti parametri din cadrul unui script bash se face prin intermediul unor variabile speciale $1, $2, până la $9. $0 va reprezenta numele script-ului mereu.

```
#!/bin/bash
echo „Parametrul 1 este $1”
echo „Parametrul 2 este $2”
```

Lansând scriptul prin **./nume_script exemplu script**, interpretorul va afişa pe ecran:

Parametrul 1 este exemplu
Parametrul 2 este script

Citirea datelor de la tastatură se poate face în momentul execuției unui script folosind comanda **read**.

```
#!/bin/bash
echo „Introduceți numele dvs:”
read nume
echo „Bine ai venit $nume!”
```

Verificarea condițiilor prin cuvintele cheie if / else / fi oferă utilizatorilor posibilitatea de a verifica anumite condiții în cadrul script-urilor şi de a lua decizii particularizate pentru fiecare caz în parte. În exemplul următor vom putea examina verificarea de condiții multiple, cât şi verificarea existenței unui fişier:

```
#!/bin/bash
fişier=”exemplu”
echo „Doriți afişarea rezultatului pe ecran? DA/NU”
read ecran
if [ -f $fişier ]; then
if [ $ecran == „DA” ]; then
        echo „Fişierul $fişier exista.”
fi
else
if [ $ecran == „DA” ]; then
        echo „Fişierul $fişier nu exista”
fi
fi
```


Exemplul anterior demonstrează cum se pot pune condiții multiple şi cum se poate verifica existența unui fişier (a se observa spațiile dintre parantezele pătrate atât după cât şi înaintea terminării condiției) - **fără aceste spații script-ul nu va funcționa**.

Verificarea se face cu ajutorul cuvântului cheie **if** care poate fi folosit cu o serie de parametri. Şirul **–f $fişier** este folosit pentru verificarea existenței fişierului al cărui nume este conținut de variabila $fişier. În cazul în care se doreşte verificarea existenței unui director, trebuie folosit parametrul **–d**.

O structură **if** va urma un set de reguli: începe cu **if**, continua cu instrucțiunile care se vor executa dacă condiția este împlinită, opțional poate continua cu **else** (care va conține o secțiune de instrucțiuni executate în cazul în care condiția nu este împlinită) şi obligatoriu se va termina în **fi** care semnalizează sfârşitul structurii.

După cum se poate observa în exemplul anterior, în cadrul structurilor **if** pot exista alte structuri **if,** numite „nested”.

Crearea copiilor de rezervă. Backup.

Copiile de rezervă sunt folosite în general pentru salvarea datelor importante, pentru a putea fi recuperate în urma unor evenimente neaşteptate – precum defectarea unui disc, erori software sau chiar furt.

În general se recomandă includerea tuturor datelor în copiile de rezervă, în cazul în care spațiul dedicat acestei soluții permite acest lucru, şi în cazul în care timpul folosit este disponibil. Copiile întregi reprezintă cea mai sigură soluție pentru siguranța datelor în eventualitatea unor dezastre, însă acestea pot dura foarte mult, în funcție de dimensiunea lor.

Pentru planificarea unui backup parțial, utilizatorul trebuie să aleagă în primul rând ce fişiere doreşte să salveze. Printre fişierele importante (şi posibil de neînlocuit) într-un sistem Linux, putem aminti:

- Baze de date – în cazul serverelor MYSQL, acestea pot fi găsite în **/var/lib/mysql**
- Fişiere personale şi documente – localizate de obicei în **/home**
- Fişiere de configurare – directorul **/etc**

Copiile de siguranță pot fi realizate în mai multe feluri, fie salvând manual toate fişierele dorite, fie prin intermediul unor utilitare specializate. Indiferent de metoda de backup folosită, utilizatorul va trebui să planifice în prealabil ce resurse doreşte să salveze.

Salvarea manuală a datelor poate fi realizată în mai multe feluri şi variază de la copierea manuală a fişierelor, până la folosirea unor utilitare precum **dd** sau **tar**. În cazul copiilor de siguranță realizate manual, se recomandă însă folosirea unor utilitare precum cele menționate mai devreme în defavoarea copierii manuale, motivul principal fiind alterarea informațiilor legate de sistemul de fişiere – permisiuni, proprietar, etc.

Utilitarul tar, prezentat anterior poate fi folosit (împreună cu facilitatea de compresie **gzip**) pentru realizarea unei copii fidele a unei structuri de fişiere şi directoare. Un exemplu bun ar fi: **tar –czf nume_copie.tar.gz /etc /home**

Comanda anterioară va crea o arhivă comprimată, reprezentând copia de siguranță a directoarelor **/etc /home**. Recuperarea datelor în eventualitatea pierderii lor, se poate realiza prin comanda: **tar –zxfp nume_copie.tar.gz**

Diferența față de dezarhivarea şi decomprimarea normală a unei arhive este parametrul **–p**, prin care utilitarul va fi forțat să dezarhiveze structura de fişiere menținând permisiunile originale.

Utilitarul dd este unul dintre programele originale UNIX folosite pentru scrierea datelor în mod „raw”. Acesta este şi în ziua de azi un program foarte util, datorită facilităților pe care le oferă: poate fi folosit pentru scrierea preferențială pe discuri, pentru extragerea parțială a unor secțiuni din fişiere binare şi chiar pentru copierea la nivel de dispozitiv.

Modul de folosire al utilitarului **dd** este unul ceva mai complicat, prin faptul că se bazează pe o serie de parametri specificați în formatul **parametru=valoare**. Restrâns la cazul de față, **dd** poate fi folosit pentru crearea copiilor de siguranță a unor partiții întregi (sau secțiuni din acestea).

Copia de siguranță în acest caz, va fi un fişier imagine, care reprezintă o copie fidelă a partiției după care a fost făcută. Acest fişier imagine poate fi comprimat, urmând să fie păstrat, sau poate fi scris pe o partiție. Această metodă de backup este ceva mai cuprinzătoare, putând fi folosită pentru reîncărcarea întregului sistem, însă, datorită faptului că se foloseşte pentru crearea unei copii fidele a unei partiții, procesul poate dură considerabil mai mult.

Principiul de funcționare al utilitarului este simplu: acesta se bazează pe o sursă de intrare a datelor, folosită pentru citirea datelor ce vor fi copiate (fiind folosit foarte des pentru partiții întregi, sursa de intrare va fi de multe ori un fişier de tip dispozitiv) şi o sursă pentru datele de ieşire, sau mai exact destinația unde se vor scrie datele copiate. Sursa de ieşire poate fi un fişier, în cazul în care se doreşte crearea unei imagini, sau poate fi fişier de tip dispozitiv, în cazul în care se doreşte scrierea directă pe o partiție.

Parametrii cei mai folosiți de către **dd** sunt **if (input file)** şi **of (output file).** Un exemplu bun de creare a unei imagini fidele a partiției **/dev/sda1** ar fi: **dd if=/dev/sda1 of=/(nume_imagine) bs=4096 conv=notrunc,noerror**

Aceasta poate fi tradusă în felul următor: crează o imagine cu numele specificat, având ca sursă partiția **/dev/sda1**, cu dimensiunea blocului de date de 4096 bytes, şi cu parametrii **notrunct** (adică fişierul destinație nu va fi truncat) şi **noerror** (adică procesul va continua şi în cazul în care apare o eroare). Pentru a facilita stocarea imaginii astfel create, aceasta poate fi comprimată cu ajutorul utilitarului **gzip**. Comanda anterioară poate fi adaptată astfel încât fişierul rezultat va fi direct o arhivă: **dd if=/dev/sda1 bs=4096 conv=notrunct,noerror | gzip –c > /(nume_imagine).gz**

Comanda poate fi interpretată în felul următor: redirecționează datele de ieşire ale utilitarului **dd** spre utilitarul **gzip** (care va redirecționa la rândul său fişierul comprimat către standard output prin parametrul **–c**) care va redirecționa din nou ieşirea către fişierul final, reprezentând imaginea comprimată a partiției.

Restaurarea unei partiții având fişierul imagine inițial (dacă acesta nu este comprimat) se poate face cu ajutorul comenzii: **dd if=/(nume_imagine) of=/dev/sda1 bs=4096 conv=notrunct,noerror**

În cazul în care fişierul imagine a fost comprimat după creare, comanda devine: **gzip –dc /(nume_imagine).gz | dd of=/dev/sda1**

OBSERVAȚIE: Principalul dezavantaj al utilitarul **dd** față de utilitare special gândite pentru back-up este acela că nu cunoaşte detalii despre sistemul de fişiere folosit, şi ca urmare, va face o copie fidelă a întregii partiții, inclusiv a blocurilor de date nefolosite. De asemenea, indiferent de utilitarul folosit pentru crearea copiei de siguranță, se recomandă alegerea unor nume sugestive pentru acestea, astfel încât administratorul să-şi poată da seama cu uşurință care este conținutul unei copii şi cât de actuală este aceasta.

Cron. Automatizarea procesului de creare a copiilor de siguranță.

Procesul de crearea a copiilor de siguranță poate fi automatizat într-o foarte mare măsură, prin programarea evenimentelor folosind daemon-ul **cron**. Rolul acestuia este de a programa şi executa comenzi la intervale şi date prestabilite.

Cron funcționează prin interpretarea fişierului **/etc/crontab**, un fişier text ce descrie când şi ce comenzi sunt programate pentru executare automată. Fiecare utilizator îşi poate defini singur sarcinile automatizate, folosind utilitarul **crontab**, dat fiind faptul că aceştia nu au acces direct la fişierul **/etc/crontab**. Deşi în mod implicit toți utilizatorii îşi pot defini propriul set de comenzi programate, administratorul poate să restricționeze accesul la acest serviciu prin intermediul fişierelor **cron.allow** şi **cron.deny** localizate în directorul **/etc**. Utilizatorii listați în primul fişier vor avea dreptul de a programa sarcini automate, pe când cei listați în cel de-al doilea fişier nu vor avea acest drept.

Programarea sarcinilor se face de regulă prin intermediul utilitarului **crontab.** Pentru afişarea listei de comenzi planificate trebuie executată comanda **crontab –l**. Formatul tipic pentru fişierele crontab este simplu: fiecare linie reprezintă programarea pentru o comandă, şi are 6 câmpuri separate prin spații. Acestea sunt în ordine:

- Numărul de minute după oră [0;59]
- Ora [0;23]
- Ziua lunii [1;31]
- Luna [1;12]
- Ziua săptămânii [0 şi 7 pentru Duminică, progesiv pentru restul]
- Comandă ce va fi executată

Fiecare coloană poate conține mai multe valori separate prin virgulă sau o linie orizontală şi se accepte caracterul „*” cu semnificația **oricare**. Fiecare utilizator îşi poate edita propriul fişier crontab, prin executarea comenzii **crontab –e.** Utilizatorul root este singurul care are dreptul să editeze fişierele crontab ale altor utilizatori prin adăugarea parametrului **–u nume_utilizator**, de exemplu **crontab –u user1 –e**. Fiecare utilizator îşi poate programa astfel execuția unor comenzi la intervale prestabilite. Urmează câteva exemple de linii crontab explicate:

- **40 * * * * echo „salut” | wall** - va emite un mesaj de atenționare către toți utilizatorii la fiecare 40 de minute (0:40, 1:40, 2:40, etc...)
- **1-59 * * * * echo „salut” | wall** - va emite un mesaj de atenționare către toți utilizatorii o dată la fiecare minut.
- **0 12 * * 5 /bin/salut** - va executa scriptul / programul **/bin/salut** în fiecare zi de vineri (5 = vineri) la ora 12:00 AM.

Administratorul poate programa şi activați la nivel de sistem, prin editarea fişierului **/etc/crontab**, păstrând sintaxa anterioară, cu mențiunea că acesta are o coloană în plus inserată între coloanele „ziua săptămânii” şi „comanda”. Acest câmp adițional reprezintă utilizatorul pentru care se va executa comanda programată.

În această idee, procesul de creare a copiilor de siguranță poate fi automatizat, la ore prestabilite. În cazul în care administratorul doreşte executarea unei copii de siguranță (de orice tip) folosind oricare dintre utilitarele prezentate mai sus, tot ce are de făcut este de adăugat linia corespunzătoare în fişierul **/etc/crontab**.

Gestionarea datelor

Datele de pe un sistem Linux se supun la aceleaşi reguli şi riscuri ca orice date: proasta gestionare, pierdere sau furt. În continuare vom studia câteva tehnici diferite pentru o mai bună gestiune a datelor pe un sistem Debian.

Primul pas în vederea asigurării confidențialității datelor, pentru a preveni accesul neautorizat sau furtul este **criptarea.** Procesul implică configurarea unei partiții astfel încât conținutul acesteia să fie criptat în mod transparent şi în timp real. Configurarea unei partiții pentru criptare presupune pierderea datelor existente de pe acea partiție, astfel încât se recomandă un back-up înainte de acest proces.

În continuare, vom prezenta paşii necesari pentru obținerea unei partiții criptate:

Pasul 1. Instalarea pachetului cryptsetup

Pachetul cryptsetup conține utilitarul necesar pentru realizarea partiției criptate. Instalarea se face apelând comanda **apt-get install cryptsetup**.

Pasul 2. Pregătirea partiției pentru criptare

Acest pas presupune scrierea unor seturi de date pe partiția ce urmează să fie creată, pentru a ajuta procesul de criptare. Urmând acest past înaintea criptării efective, partiția va fi protejată împotriva atacurilor. Comanda recomandată pentru acest pas este una care îndeplineşte raportul optim securitate / timp generare: **badblocks –vfw (nume_partiție) 4096**

Aplicația **badblocks** este folosită pentru examinarea discurilor fizice în scopul detectării blocurilor defecte. În acest scop, acest program va genera nişte şabloane pe discul ce urmează să fie testat. Parametrii **–vfw** forțează scrierea acestor şabloane pe disc, generând în esență un punct de plecare pentru facilitarea procesului de criptare.

Pasul 3. Criptarea propriu-zisă

În acest moment, partiția prelucrată este gata pentru a fi criptată. Pentru a continua, vom stabili câțiva parametri necesari pentru criptare: tipul de cifru va fi „**aes**” şi dimensiunea cheii în biți (în cazul de față vom folosi o cheie pe 128 de biți). Pentru folosirea acestor parametri, şi pentru realizarea criptării este necesară executarea comenzii (atenție, partiția în cauză NU trebuie să fie montată): **cryptsetup luksFormat (nume_partiție) –c aes –s 128**

În cazul în care utilizatorul primeşte un mesaj de eroare în genul „**Failed to setup dm-crypt key mapping**”, înseamnă că modulul de kernel corespunzător nu este încărcat. Pentru remedierea acestei probleme, trebuie încărcat modulul **dm-crypt** prin executarea comenzii: **modprobe dm-crypt**.

Dacă totul a decurs normal, utilizatorul va fi rugat să introducă o frază de

autentificare ce va fi folosită în procesul de criptare. Se recomandă alegerea unei fraze cât mai lungi pentru garantarea unui nivel minim de securitate.

Pasul 4. Deschiderea partiției

În acest moment, avem pregătită o partiție criptată, însă aceasta nu este încă gata pentru folosire. Pentru deschiderea partiției, putem folosi tot utilitarul **cryptsetup** cu parametrii corespunzători:

cryptsetup luksOpen (nume_partiție) (nume_criptat)

În urma execuției comenzii anterioare, utilizatorul va fi rugat să introducă frază de autentificare stabilită la pasul anterior. Dacă fraza este corectă, atunci partiția va fi prelucrată de către sistemul **device mapper**. Acesta este un cadru prin care se poate face o asociere între două dispozitive – în esență crează un fel de legătură către un fişier de tip dispozitiv ce va fi folosit pentru montarea partiției criptate. Numele fişierului creat este cel specificat că parametrul în cadrul comenzii anterioare, şi va putea fi localizat la adresa: **/dev/mapper/(nume_criptat)**.

Pasul 5. Crearea sistemului de fişiere pe partiția criptată

În acest moment partiția criptată este gata, însă nu poate fi folosită până nu va fi formatată. Pentru a crea sistemul de fişiere standard Linux (ext3) pe noua partiție, trebuie folosită comanda:

mke2fs –j /dev/mapper/(nume_criptat)

Pasul 6. Montarea partiției

Odată generată, partiția poate fi montată ca orice altă partiție, însă cu avantajul că toate datele vor criptate într-un mod transparent pentru utilizator. Montarea se face în mod normal, cu mențiunea că nu se va folosi numele real al partiției pentru montare, ci legătura pusă la dispoziție prin **device mapper.**

Pentru automatizarea procesului, astfel încât partiția criptată să fie montată în mod automat la fiecare pornire a sistemului trebuie modificate două fişiere. Primul este **/etc/crypttab,** folosit pentru generarea device map-ului, prin adăugarea liniei următoare la sfârşitul fişierului.

(nume_criptat) (nume_partiție) none luks

Al doilea fişier ce trebuie modificat este **/etc/fstab** pentru ca procesul de montare automată să poată fi finalizat. Montarea fără fişierul generat de device mapper nu este posibilă, dar dincolo de acest punct, procesul de montare este unul obişnuit prin includerea liniei următoare în **/etc/fstab**

/dev/mapper/(nume_criptat) (punct_de_montare) ext3 defaults 0 0

În cazul în care utilizatorul optează pentru montarea automată a partiției în timpul procesului de boot, acesta va fi rugat să introducă frază de autentificare. În cazul în care totul a decurs corect, sistemul beneficiază în acest moment de o partiție criptată conform parametrilor aleşi.

OBSERVAȚII: Toate comenziile lansate mai devreme necesită drepturi de root. Deşi criptarea partiției principale (directorul root) este posibilă, acest procedeu este mai complicat, o alternativă bună la această problemă fiind stocarea datelor importante pe partiția criptată. În cazul examinării hard disc-

ului cu partiția criptată cu utilitare externe, aceasta va figura ca fiind ne-formatata.

Compresia şi arhivarea datelor

Conceptul de arhivare şi compresare a datelor nu este unul nou. Datele comprimate şi/sau arhivate sunt mult mai uşor de gestionat şi de transmis pe diverse medii de comunicare, distribuțiile Linux folosindu-se în mod curent de ambele procedee, datorită multiplelor avantaje oferite.

Putem începe prin a sublinia diferențele dintre cele două procese: cel de arhivare şi cel de compresie a datelor. Arhivarea datelor înseamnă unirea mai multor fişiere şi directoare în cadrul unui singur fişier, pentru o gestiune mai uşoară, păstrând totuşi o serie de informații despre structura de fişiere anterioară, astfel încât aceasta să poată fi dez-arhivata. Probabil cel mai cunoscut format de arhive în lumea Linux este formatul **tar** (tape archive), creat la începuturile sistemelor UNIX, şi standardizat de abia în 1988. Acesta a fost gândit inițial pentru scrierea secvențială pe dispozitive de stocare cu bandă magnetică, în scopul creării copiilor de rezervă, dar este folosit în prezent pentru gruparea unei structuri de fişiere şi directoare într-un singur fişier principal, păstrând informații despre sistemul de fişiere inițial (informații precum permisiuni, proprietari, grupuri, date, etc).

Compresia în schimb, reprezintă un proces de codificare reversibilă a fişierelor, astfel încât acestea să ocupe mai puțin spațiu fizic decât ar ocupa în mod normal. Deşi există multe tipuri de algoritmi de comprimare a datelor, unele specializate pe anumite tipuri de date (fişiere video, foto, sunet, etc), toate au în comun acelaşi principiu de funcționare: codificarea datelor existente într-o manieră mai eficientă din punct de vedere al spațiului folosit pentru stocare. În general utilizatorii vor observa că programele de compresie vor funcționa cel mai bine când sunt aplicate pe surse de date al căror conținut nu este foarte aleatoriu (text, imagini, etc). În cadrul sistemelor Linux, cele mai răspândite formate compresate sunt **zip**, **bz2** şi **gzip**.

O tactică comună este combinarea celor două procedee pentru a beneficia de facilitățile oferite de acestea; de aici vine şi cunoscutul format **tar.gz**, care nu este altceva decât o arhivă comprimată.

Crearea arhivelor se poate realiza cu ajutorul programului **tar**, inclus în toate distributiileLinux. Sintaxa este una destul de simplă: **tar [parametri] [fişiere]**. Vom examina în continuare câteva exemple în care se foloseşte comanda **tar:**

Pentru arhivarea unor fişiere: tar –cf arhiva.tar fişier_1 fişier_2 …

În exemplul anterior, programul va arhiva toate fişierele menționate într-o singură arhivă numită **arhiva.tar**. Parametrul **–c** semnalează faptul că dorim să creăm o nouă arhivă, iar **–f** înseamnă că se va lucra cu fişiere.

Pentru arhivarea unui director: tar –cf arhiva.tar cale_director

Comprimarea fişierelor se realizează de obicei cu ajutorul utilitarului **gzip**. Sintaxa este foarte simplă: **gzip (nume_fişier)**. Este important de menționat faptul că programul **gzip** va înlocui fişierul original cu varianta sa

comprimată. Dacă numele fişierului iniţial ar fi fost (de exemplu) **test.txt** în urma acţionării comenzii **gzip test.txt**, fişierul test.txt ar fi fost înlocuit cu fişierul **test.txt.gz**.

Decompresia datelor se face prin includerea parametrului **–d** în linia de comandă: **gzip –d nume_fişier.gz**

Comprimarea arhivelor se poate face printr-o combinaţie a utilitarelor prezentate mai devreme. Utilitarul **tar** ştie să arhiveze fişierele arhivă cu ajutorul utilitarului **gzip**: **tar –czf nume_arhiva.tar.gz director**

Parametrii **–c** şi **–f** au aceeaşi semnificaţie ca şi în exemplele anterioare, iar parametrul **–z** specifică faptul că se doreşte comprimarea arhivei folosind **gzip**. Despachetarea unui fişier de tip **tar.gz** (acest lucru implică mai întâi decompresia şi apoi dezarhivarea sa) se poate face prin comanda:

tar –zxf nume_arhiva.tar.gz

Parametrul **–x** specifică faptul că se doreşte dezarhivarea fişierului tar, rezultat în urma decomprimării.

Sistemul Debian nu oferă în mod nativ compresie la nivel de partiție, însă dacă luăm în considerare progresele tehnologice din ultimii ani, care au redus costurile dispozitivelor de stocare semnificativ, în timp ce capacitățile de stocare au crescut foarte mult, atunci putem concluziona că pierderea de performanță asociată cu compresia datelor în timp real, raportată la nivelul de spațiu recuperat nu este justificabilă.

Implementarea cotelor de disc

Implementarea cotelor de disc este o necesitate în orice sistem multi-user. Riscurile asociate cu supraîncărcarea discurilor sistemului nu pot fi ignorate, putând să genereze o serie întreagă de erori, mergând de la imposibilitatea despachetării fişierelor temporare (necesare pentru foarte multe activități) şi ajung până la cazuri de blocare a sistemului. Din acest motiv, activitatea utilizatorilor poate fi restricționată într-un sistem Linux, astfel încât aceştia – conform politicii de sistem curente – să nu poată depăşi un anumit nivel de spațiu ocupat.

Sistemul de cote permite configurarea cotelor de disc pentru două categorii: stabilirea cotelor individuale pentru fiecare utilizator cât şi stabilirea cotelor de grup – ambele vor trebui luate în considerare în momentul stabilirii cotelor, pentru o gestiune corectă a sistemului de fişiere.

În cazul cotelor individuale, administratorul va stabili o limită superioară de spațiu de stocare alocat pentru fiecare utilizator, pe când în cazul cotelor de grup, se va stabili limita superioară de spațiu alocat întregului grup.

Pentru configurarea sistemului cu suport pentru stabilirea cotelor de disc, trebuie urmați următorii paşi:

Pasul 1. Montarea partițiilor în mod corespunzător

Pentru ca sistemul de cote să funcționeze corespunzător, partițiile pe care se doreşte stabilirea cotelor de disc trebuie montate cu o serie de parametri speciali. Aceştia se vor seta în cadrul fişierului **/etc/fstab**, în cea de-a cincea

coloană „<options>”, prin includerea cuvintelor **usrquota**, respectiv **grpquota**. În exemplul de mai jos, vom monta partiția principală a sistemului cu suport pentru cote prin modificarea liniei corespunzătoare în fişierul **/etc/fstab**:**/dev/sda1/ext3errors=remount-ro,usrquota,grpquota 0 1**
Semnificația liniei de mai sus a fost explicată în cadrul procesului de montare automată, singura modificiare a procesului de montare pentru partiția **/dev/sda** fiind includerea parametrilor **usrquota** şi **grpquota**.

Pasul 2. Încărcarea modulului kernel corespunzător
Pentru ca întregul sistem să funcționeze, modulul de kernel **quota_v2** trebuie încărcat. Acest proces se poate realiza temporar, pentru instanța curentă a sistemului prin intermediul comenzii **modprobe quota_v2**, iar în cazul în care se doreşte încărcarea modulului în mod automat la fiecare reinițializare a sistemului, linia „**quota_v2”** trebuie adăugată la sfârşitul fişierului **/etc/modules** prin intermediul comenzii **echo `quota_v2` >> /etc/modules**.

Pasul 3. **Crearea fişierelor de configurare**
Pentru ca întreg procesul de instalare să decurgă în mod normal, administratorul trebuie să creeze două fişiere ce vor fi folosite pentru descrierea cotelor. În exemplul curent, dat fiindcă vom implementa cotele pe partiția principală montată în rădăcină, fişierele trebuie create în acelaşi loc. În acest scop, trebuie executate următoarele comenzi:

```
touch /quota.user
touch /quota.group
chmod 600 /quota.*
```

Aceste fişiere trebuie create ca root, şi vor fi folosite pentru determinarea cotelor individuale şi de grup. Ultima comandă are rolul de a bloca orice fel de acces la fişierele respective, prin setarea permisiunilor 600 (adică read-write doar pentru owner – root).

Pasul 4. Instalarea pachetelor necesare
Pachetele necesare pentru implementarea sistemului de cote sunt **quota**, respectiv **quotatool**. Acestea pot fi instalate cu ajutorul comenzii: **apt-get install quota quotatool**.

Pasul 5. Pregătirea sistemului
Dacă dorim ca sistemul de cote să fie implementat fără o repornire a sistemului, partiția în cauză va trebui să fie remontată prin comanda **mount –o remount /** . Pentru a verifica dacă aceasta a fost remontată cu parametrii necesari (menționați la punctul 1), este suficientă executarea comenzii **mount**, care va afişa toate partițiile montate, cât şi opțiunile individuale pentru fiecare din acestea.

Pasul 6. Stabilirea cotelor
Odată ce toate părțile sistemului de cote au fost instalate în mod corespunzător, administratorul poate începe să configureze cotele pentru utilizatori şi grupuri. Sistemul funcționează folosind două principii de limite,

numite limită „soft” şi limită „hard”. Limita „soft” se referă la o limită temporară pe care o poate depăşi pentru o perioadă fixă de timp (implicit 7 zile), pe când limita „hard” se referă la limită absolută pe care un utilizator nu o poate depăşi în niciun fel.

Pentru stabilirea limitelor individuale administratorul poate să folosească comanda: **quotatool –u (nume_utilizator) –bq (limita soft) –l (limita hard) (mount point)**

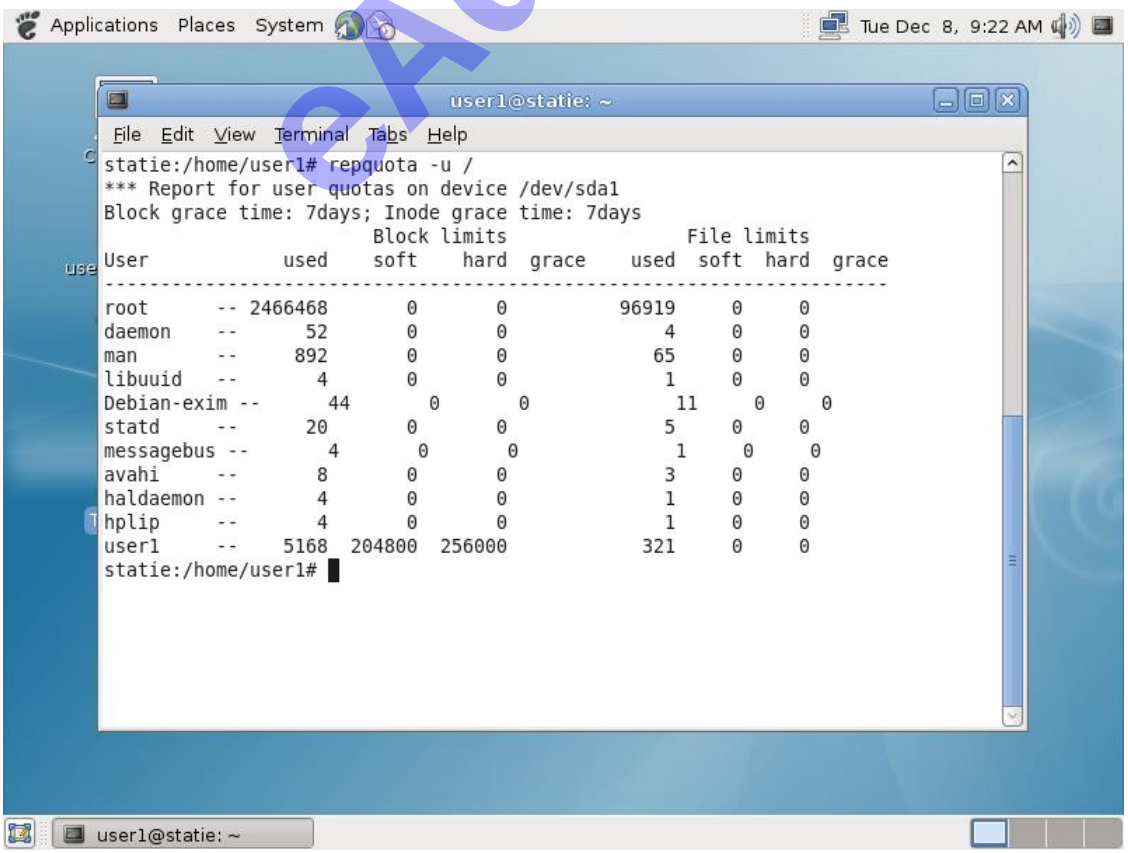
Comanda pentru stabilirea unei cote soft de 200Mb, respectiv 250 limita hard pentru utilizatorul **user1** pe partiția principală a sistemului: **quotatool –u user1 –bq 200M –l 250M /**

Pentru stabilirea limitelor de grup administratorul poate folosi comanda anterioară, însă trebuie schimbat parametrul **–u** (user) cu **–g** (grup). Pentru identificarea utilizatorilor sau a grupurilor, administratorul poate folosi atât numele conturilor cât şi identificatoarele numerice (**uid** respectiv **gid**).

Pasul 7. Verificarea sistemului de cote

În cazul în care sistemul a fost implementat corect, în acest moment utilizatorul **user1** ar trebui să fie limitat la 250Mb spațiu de stocare. Administratorul poate oricând să verifice dacă sistemul de cote funcționează, prin apelarea utilitarului **quotacheck –vagum**.

Pentru vizualizarea unui raport privind cotele şi spațiul de stocare folosit de fiecare utilizator (şi/sau grup) utilizatorul poate folosi comanda **repquota (mount point).** Raportat la exemplul curent, pentru a vedea cota utilizatorului **user1** pe partiția principală cu punctul de montare */*, administratorul ar trebui să folosească comanda: **repquota –u /**



În poza anterioară, se poate observa o listă cu cotele impuse pentru fiecare utilizator, cât şi un raport cu spațiul ocupat de fiecare cont. Timpul de grație (grace period) este setat în mod implicit la 7 zile, semnificând că utilizatorul poate păstra datele peste limita sa soft timp de 7 zile. În cazul în care utilizatorul şi-a atins cota, orice tentativă de scrierere nouă vă fi oprită de către sistem, iar utilizatorul va fi atenționat printr-un mesaj de genul „**Disk quota exceeded**”.

OBSERVAȚII: În cazul în care timpul de grație al unui utilizator expiră, sistemul nu va şterge automat nimic, însă utilizatorul respectiv nu va mai avea dreptul să efectueze nici un fel de operație de scriere până când spațiul pe care îl ocupă nu va fi redus sub limita soft. De asemenea, în cazul în care administratorul doreşte să elimine cota unui utilizator, acesta va trebui să configureze cota utilizatorului respectiv la valoarea zero.

Configurarea server-ului Xorg

Server-ul **Xorg** face posibilă funcționarea mediului grafic în cadrul unui sistem Linux, şi configurarea sa corectă reprezintă un punct foarte important pentru toți ce doresc funcționalitatea unui mediu grafic - deşi instalarea ghidată a sistemului a dus la configurarea automată a server-ului **Xorg**.

În majoritatea cazurilor nu va fi necesară intervenția manuală asupra configurării server-ului **X**, însă există cazuri în care s-ar putea ca utilizatorul să-şi dorească activarea unor moduri funcționale speciale sau a unor rezoluții care nu sunt disponibile în mod normal. Fişierul de configurare asociat server-ului **X** este **/etc/X11/xorg.conf.**

Fişierul de configurare este structurat în secțiuni ce urmăresc următoarea sintaxă:

```
Section    [nume_secțiune]
Opțiuni…
…
EndSection
```

Secțiunile sunt folosite pentru delimitarea opțiunilor şi restricționarea efectului la un punct de interes, şi pot lua următoarele valori:

- **Files**
- **Module**
- **InputDevice**
- **Device**
- **VideoAdaptor**
- **Monitor**
- **Modes**
- **Screen**
- **ServerLayout**

La rândul lor, secțiunile pot fi împărțite în sub-sectiuni prin intermediul directivei **SubSection**, în general pentru definirea unui set de caracteristici legate de secțiunea de care aparțin. Un astfel de exemplu, este următoarea

secvență folosită pentru configurarea rezoluției monitorului la una mai puțin convențională; pentru aceasta toate modificările trebuie făcute în fişierul **xorg.conf** în cadrul secțiunii **„Screen”**. În cadrul acesteia se va defini o sub-sectiune intitulată „**Display”:**

```
Section „Screen”
Identifier   „Default Screen”
Monitor      „Configured Monitor”
SubSection „Display”
        Depth       16
        Modes       „800x600” „1440x900”
EndSubSection
EndSection
```

În secvența de mai sus, am definit două moduri de funcționare asociate monitorului descris prin directiva **Monitor** – mai exact am definit rezoluțiile **800x600** respectiv **1440x900.** Important de menționat este că o bună parte a răspunderii pentru modul de funcționare a interfei grafice îl are şi display manager-ul – fie că este vorba de **gdm, kdm, xdm** sau alte astfel de programe.