5.PROCESE SI SERVICII. RULARE AUTOMATIZATA

5.1. Procese....2
5.1.1. Notiunea de proces si caracteristici....2
5.1.2. Relatii intre procese....3
5.1.3. Vizualizarea listei de procese si a caracteristicilor acestora....4
5.1.3.1. Comanda ps....4
5.1.3.2. Comanda pstree....5
5.1.3.3. Comanda top....5
5.1.4. Prioritizarea proceselor....7
5.1.4.1. Despre executie concurenta si prioritati....7
5.1.4.2. Comenzile nice si renice....8
5.1.5. Comunicarea inter-proces....8
5.1.5.1. Notiunea de semnal....8
5.1.5.2. Comenzile kill, pkill si killall....9
5.1.5.3. Comanda trap....10
5.1.5.4. Comanda nohup....10
5.2. Rularea mai multor comenzi in paralel din linia de comanda....10
5.2.1. Descrierea problemei si solutii disponibile....10
5.2.2. Job control....10
5.2.3. Screen....11
5.3. Servicii....12
5.3.1. Conceptul de serviciu si sisteme de initializare....12
5.3.2. Managementul serviciilor SysV....13
5.3.2.1. Principii....13
5.3.2.2. Managementul serviciilor in timpul rularii sistemului de operare....13
5.3.2.3. Managementul serviciilor la bootare....14
5.3.3. Managementul serviciilor systemd....14
5.4. Rularea automatizata a proceselor....14
5.4.1. Solutii existente si diferente intre acestea....14
5.4.2. Cron....15
5.4.2.1. Principii si structura fisierelor de configurare....15
5.4.2.2. Configurare globala....15
5.4.2.3. Configurare per-user....17
5.4.3. Anacron....17
5.4.4. Atd....18
5.5. BIBLIOGRAFIE....19

InfoAcademy

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de autor.

5.1. Procese

5.1.1. Notiunea de proces si caracteristici

Un program este format dintr-unul sau mai multe fisiere memorate in sistemul de fisiere. Cel putin unul dintre acestea este un executabil, cu urmatoarele proprietati:

- are un format ce depinde de sistemul de operare pe care este gandit sa ruleze. Executabilele de Windows au alta structura decat cele de Linux, ceea ce face ca un executabil creat pentru un sistem de operare sa nu functioneze pe altul
- contine cod masina – instructiuni intelese de catre procesorul masinii pe care programul va rula.

Un fisier executabil nu ruleaza direct de pe hard-disk – el trebuie mai intai incarcat in RAM, astfel incat procesorul/procesoarele sa aiba acces la instructiunile sale (de aici rezulta si necesitatea ca, in Linux, fisierul sa aiba nu numai permisiune de executie, ci si de citire). Odata fisierul incarcat, sistemul de operare va lua masurile necesare pentru a executa codul continut, avand grija insa sa ofere aceeasi posibilitate si altor programe care se aflau deja in executie sau care vor fi pornite ulterior si vor rula concurent cu programul pornit.

Sistemele de operare ale zilelor noastre sunt multi-tasking – au capabilitatea de a rula mai multe programe simultan (sau cel putin dandu-ne iluzia simultaneitatii). Aceasta se realizeaza prin gestionarea resurselor sistemului (procesor, memorie) de catre kernel-ul sistemului de operare. Atunci cand avem un singur procesor single-core, acesta nu poate executa mai multe programe simultan; sistemul de operare il va „oferi” pe rand, pentru un interval de timp limitat, fiecarui program care ruleaza, comutarea de la un program la altul facandu-se atat de rapid incat senzatia noastra este de avans simultan al tuturor programelor concurente. In consecinta, „viata” unui program consta dintr-o executie fragmentata – un ansamblu de scurte, multiple perioade de timp in care procesorul ii apartine.

Chiar si atunci cand sistemul de operare ruleaza pe o statie cu mai multe procesoare sau core-uri, situatia este asemanatoare cu cazul single-core: numarul de programe care ruleaza simultan este in general mult superior numarului de procesoare, astfel incat intotdeauna vor exista mai multe programe care trebuie sa se execute concurent pe acelasi procesor. De aceea nu vorbim despre o execuție „simultana” sau „in paralel”, ci una concurenta - programele care ruleaza concurează pentru accesul la resursele sistemului.

Pentru a putea tine evidenta programelor care se executa si a le aloca spatii de memorie si timp de procesor, kernel-ul opereaza cu notiunea de **proces** – acesta reprezentand contextul in care ruleaza o anumita instanta (exemplar) a unui program. Kernelul mentine o tabela de procese in care fiecare proces are caracteristici precum:

- un process id (PID). Este un numar intreg intre 0 si 65535 prin care sistemul de operare identifica in mod unic fiecare proces. PID-ul intervine si in unele comenzi de lucru cu procesele (vezi mai jos)
- un spatiu de memorie. Sistemul de operare are grija ca procesele sa nu interfere unul cu altul din punct de vedere al zonei de memorie ocupate (ele putand totusi sa foloseasca zone de memorie comune daca au nevoie explicit de acest lucru)
- un UID si un GID. Acestea determina contextul de securitate al procesului, din doua puncte de vedere:
  - unele operatii in sistemul de operare nu ii sunt accesibile decat lui root (UID-ul 0) – spre exemplu, deschiderea de porturi din zona joasa (<=1024).
  - in functie owner-ul, group-ul si permisiunile fiecarui fisier din sistemul de operare, si de raportul intre acestea si UID/GID ale procesului, rezulta permisiunile procesului in sistem (sa nu uitam ca multe fisiere reprezinta dispozitive hardware, capete de conexiune etc)

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

- codul programului pe care il executa – asa-numita *imagine a procesului*. Daca programul reprezinta o colectie pasiva de instructiuni, atunci procesul este un exemplar live al sau.
- terminalul la care este conectat procesul. In general, un proces pornit dintr-o consola sau dintr-un terminal virtual ramane „conectat" la acel terminal, in sensul in care input-ul procesului este preluat de la tastatura acelui terminal, iar output-ul este afisat pe ecranul terminalului in cauza. Exista insa si procese neconectate la un terminal – exemple: procesele pornite la bootarea sistemului de operare (inaintea aparitiei consolelor virtuale si a terminalelor), procesele corespunzatoare serverelor Linux (care, chiar si atunci cand sunt pornite din linia de comanda, se detaseaza de terminalul curent, inapoind promptul utilizatorului)

Un acelasi program poate fi pornit de mai multe ori, rezultand mai multe procese – cate unul pentru fiecare lansare in executie a programului in cauza. Exista si scenariul in care un singur program poate determina pornirea mai multor procese, in masura in care programul in cauza are diferite componente executate ca procese separate.

5.1.2. Relatii intre procese

Un proces poate crea la randul sau un altul. Spre exemplu, atunci cand executam o comanda din shell-ul Linux, procesul in care ruleaza interpretorul de comenzi porneste un al doilea proces in care este lansata comanda dorita. In aceasta relatie, procesul initial este denumit in general *proces parinte*, iar procesul nou pornit este denumit *subproces* sau *proces fiu*. Cu exceptia primului proces pornit in sistemul de operare, toate procesele sunt de fapt subprocese; de aceea, fiecare proces are atasat un PPID (Parent PID).

Primul proces pornit in sistemul de operare porneste o serie de subprocese; acestea pornesc la randul lor alte subprocese s.a.m.d, rezultand in final o structura arborescenta in care fiecare nod are unul sau mai multi descendenti: arborele de procese ("process tree").

Desfasurarea normala a relatiei parinte-subproces este urmatoarea:

- procesul parinte creeaza subprocesul, adaugand un nou rand in tabela de procese. Daca parintele este conectat la un terminal, subprocesul va mosteni de la el cele trei canale de comunicatie standard (input, output si error); asa se face ca o comanda rulata dintr-un terminal isi va afisa automat output-ul in acelasi terminal
- subprocesul ruleaza pana cand executia sa se incheie si resursele alocate lui sunt eliberate (memorie, fisiere deschise etc). La final el genereaza un cod de iesire, memorat pe randul sau din tabela de procese: 0 indica incheierea cu succes a executiei, un cod nenul indica esecul. Din acest moment el se gaseste in starea de "zombie" - nu mai ruleaza si nu foloseste resurse de sistem, insa asteapta ca procesul sau parinte sa-i citeasca codul de iesire
- procesul parinte citeste codul de iesire al subprocesului, dupa care randul corespunzator celui din urma este eliminat din tabela de procese

Exista doua abateri de la desfasurarea normala prezentata mai sus:

- atunci cand procesul parinte se incheie inainte de terminarea subprocesului, acesta din urma devine *proces orfan.* Procesele orfane sunt automat "înfiate" de catre alt proces; nu exista subproces fara parinte
- atunci cand procesul parinte omite sau intarzie sa citeasca rezultatul de iesire al subprocesului, acesta ramane in starea de zombie pentru o perioada mai lunga. Acest lucru este evidentiat in comenzile de vizualizare a proceselor, si poate uneori indica un defect de programare al procesului parinte (desi exista si cazuri in care aceasta intarziere este intentionata). Procesele zombie persistente pot fi inchise fie prin trimiterea semnalului SIGCHLD catre parinte (vezi mai jos capitolul dedicat semnalelor), fie prin inchiderea procesului parinte, care cauzeaza aducerea subprocesului in starea de orfan si preluarea sa de

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

catre un alt proces care are grija sa citeasca periodic codurile de iesire ale subproceselor sale, eliminand astfel inregistrarile corespunzatoare din tabela de procese

5.1.3. Vizualizarea listei de procese si a caracteristicilor acestora

5.1.3.1. Comanda ps

Comanda **ps** (process status) realizeaza afisarea tabelara a listei de procese. Apelata fara argumente ea nu afiseaza decat procesele pornite din terminalul curent (si acestea cu un minim de detalii), de aceea este in general necesar sa recurgem la optiuni; iata cateva categorii de optiuni utile:

- controlul listei de procese afisate
  - **-A** sau **-e** – determina afisarea tuturor proceselor, nu doar cele pornite din terminalul curent
  - **-u** username – afiseaza numai procesele userului solicitat (cele care ruleaza cu UID-ul sau)
- gradul de detaliu al output-ului poate fi controlat cu optiuni precum:
  - **-f** (full) – afiseaza coloane suplimentare precum: UID cu care ruleaza procesul, PPID (parent PID), STIME(momentul pornirii procesului), TIME(timp de procesor consumat de catre acel proces de la pornire si pana in momentul rularii comenzii ps), TTY (terminalul la care este atasat procesul), C (CPU – procentul de timp de procesor folosit de catre acest proces de-a lungul existentei sale)
  - **-l** (long format) – adauga coloane precum: F (flags – indicatori atasati procesului; 4 indica privilegii de root), S (starea curenta a procesului: R - running, Z - zombie, S - sleeping etc), PRI (prioritatea procesului), NI (nice value – valoare setabila de catre utilizator si care participa in calculul prioritatii procesului)
  - **-F** (extra full) – afiseaza coloane precum RSS (resident segment size – cantitatea de memorie fizica (RAM) folosita de catre proces), PSR (procesorul caruia i-a fost asignat procesul – util pentru scenarii multi-procesor sau multi-core)
  - **-o** *coloana1*, *coloana2*,... - permite specificarea manuala a listei de coloane afisate, ceea ce ofera acces la o intreaga serie de parametri neinclusi in optiunile de mai sus. Lista completa de parametri poate fi gasita in manpage-ul comenzii ps, sectiunea STANDARD FORMAT SPECIFIERS

```
student@Desktop:~$ ps
  PID TTY          TIME CMD
 5507 pts/2    00:00:00 bash
 5560 pts/2    00:00:00 ps
student@Desktop:~$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
ionut     5507  5472  0 13:36 pts/2    00:00:00 bash
ionut     5548  5507  0 13:51 pts/2    00:00:00 ps -f
student@Desktop:~$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  5507  5472  0  80   0 -  1669 -      pts/2    00:00:00 bash
0 R  1000  5547  5507  0  80   0 -  1070 -      pts/2    00:00:00 ps
student@Desktop:~$ ps -F
UID        PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
ionut     5507  5472  0  1669  3144   0 13:36 pts/2    00:00:00 bash
ionut     5549  5507  0  1127  1028   0 13:51 pts/2    00:00:00 ps -F
student@Desktop:~$ ps -o pid,cls,pri,ni,command
  PID CLS PRI  NI COMMAND
 5507  TS  19   0 bash
 5550  TS  19   0 ps -o pid,cls,pri,ni,command
```

Nota: *dimensiunile sunt afisate din oficiu in kilobytes.*

- formatul output-ului poate fi controlat cu:

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

- **-H** – prezinta procesele in mod ierarhic, evidentiind relatiile dintre ele (subprocesele vor fi afisate imediat dedesubtul procesului lor parinte, cu alineat)
- **w** (wide) – atunci cand linia corespunzatoare unui proces este prea lunga pentru a incapea in terminal, ea este in mod normal trunchiata. Optiunea w (fara semnul minus in fata!) determina afisarea liniei intregi

5.1.3.2. Comanda pstree

Comanda **pstree** afiseaza arborele de procese intr-o forma care evidentiaza relatiile dintre procese in detrimentul altor informatii:

```
student@Desktop:~$ pstree
init─┬─NetworkManager───{NetworkManager}
     ├─NetworkManagerD
     ├─Xvnc4
     ├─acpid
     ├─apache2───5*[apache2]
     ├─artsd
[...restul output-ului a fost omis...]
```

Optiunile comenzii permit afisarea de informatii suplimentare fata de output-ul de mai sus:

- **-a** – afisarea argumentelor din linia de comanda; poate fi combinata cu **–l** (linii lungi)
- **-c** – afisare ne-compactata. Atunci cand un proces are mai multe subprocese cu nume identic, ele nu mai sunt reprezentate ca o singura ramura (cum este cazul celor 5 subprocese apache2 din exemplul de mai sus) ci ca ramuri diferite
- **-h** – iluminarea ramurii din arbore a procesului curent (cea pe care ruleaza comanda pstree)
- **-p** – este afisat si PID-ul pentru fiecare proces in parte
- **-u** – este afisat si UID-ul/username-ul pentru fiecare proces

Nota: *comanda pstree nu exista din oficiu pe toate distributiile Linux/Unix, de aceea este recomandabil ca utilizatorul sa stapaneasca cu prioritate utilizarea comenzii ps.*

5.1.3.3. Comanda top

Spre deosebire de comenzile prezentate anterior, **top** realizeaza o afisare continua a proceselor, improspatata periodic, cu posibilitatea ordonarii dupa diferite criterii (similar lui Task Manager sub Windows):

```
top - 15:35:34 up  3:46,  4 users,  load average: 0.21, 0.25, 0.22
Tasks: 145 total,   2 running, 143 sleeping,   0 stopped,   0 zombie
Cpu(s):  1.7%us,  1.3%sy,  6.3%ni, 90.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   1946288k total,  1842684k used,   103604k free,   431904k buffers
Swap:   208804k total,      852k used,   207952k free,   649320k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3478 root      20   0  245m  95m  22m S  1.3  5.0   8:09.70 Xorg
 4447 student   20   0  394m 241m  16m S  1.0 12.7   6:25.39 opera
   41 root      15  -5     0    0    0 S  0.7  0.0   0:00.62 kacpid
 5472 student   20   0  101m  22m  11m S  0.7  1.2   0:09.22 gnome-terminal
 6318 student   20   0  2392 1160  884 R  0.7  0.1   0:00.16 top
    1 root      20   0  2100  688  588 S  0.0  0.0   0:00.86 init
    3 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/0
[...restul output-ului a fost omis...]
```

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

Nota: *top este gandit strict pentru ce ii spune numele - topuri, asadar ordonarea proceselor dupa o anumita caracteristica si vizualizarea celor "fruntase". Nu trebuie sa ne asteptam ca top sa afiseze lista completa de procese, caci nu acesta este scopul sau.*

Output-ul lui top este format din zona de statistici (cea de sus) si zona de procese (dedesubt). Prima dintre ele ofera informatii generale despre procese, procesor si memorie. Procesorul isi imparte "existenta" intre:

- **us** (user) – procentul de timp petrecut executand cod al aplicatiilor utilizatorilor
- **sy** (system) – procentul de timp timp petrecut executand cod de kernel. In general executia unei aplicatii presupune atat cod al executabilului respectiv, cat si apeluri catre functii de kernel (si deci executare de cod care nu este prezent in executabilul aplicatiei)
- **ni** (nice) – procentul de timp petrecut executand procese prioritizate (vezi mai jos capitolul dedicat prioritizarii proceselor)
- **id** (idle) – procentul de timp in care procesorul nu a avut nimic de executat
- **wa** (waiting) – procentul de timp in care procesorul a fost nevoit sa stationeze (idle) in asteptarea incheierii unei operatii I/O. Se intampla, de exemplu, pe statii cu activitate intensa a unui hard-disk lent si care au memorie insuficienta
- **hi** (hardware interrupts) – procentul de timp in care procesorul a deservit intreruperi hardware
- **si** (software interrupts) – analog, dar pentru intreruperi software (softirq)
- **st** (stolen) – pe sistemele care folosesc virtualizare (procesoare virtuale), reprezinta procentul de timp in care un procesor virtual a fost nevoit sa astepte eliberarea unui procesor real

In privinta memoriei, ea este impartita in memorie RAM si memorie swap, ultima dintre ele fiind creata pe hard-disk si reprezentand o extensie de rezerva a memoriei fizice. Impreuna ele formeaza memoria virtuala, adresabila ca un tot unitar de catre kernel. Informatiile despre utilizarea memoriei oferite de top se refera la:

- memorie fizica (RAM)
    - **total** – reprezinta cantitatea de memorie RAM prezenta in sistem (fizic instalata)
    - **free** – reperezinta cantitatea de RAM neutilizata in momentul rularii comenzii top
    - **used** – cantitatea de RAM ocupata la momentul rularii comenzii top. Memoria ocupata este compusa din:
        - memorie utilizata de catre aplicatii
        - memorie folosita de catre kernel:
            - **buffers** – diverse zone de memorie tampon in care kernelul isi stocheaza informatie
            - **cached** – cache-ul de hard-disk alocat de kernel (zona de RAM prin intermediul careia se desfasoara operatiile cu sistemul de fisiere)
- memoria swap – cantitatea totala, folosita si respectiv disponibila de memorie swap

Trebuie inteles faptul ca memoria RAM care figureaza ca *used* nu este consumata exclusiv de aplicatii, ci reprezinta suma dintre memoria aplicatiilor, buffere si cache-ul de hard-disk. Kernel-ul va ajusta cache-ul de hard-disk in functie de necesitati: daca constata ca are RAM suficient, va utiliza mai mult spatiu pentru cache (spatiul poate chiar depasi o treime din RAM, in functie de configuratia hardware), insa daca aplicatiile cer memorie, kernelul va micsora dimensiunea cache-ului de hard-disk si va distribui memoria rezultata aplicatiilor. De aceea putem considera ca memoria libera (cea disponibila aplicatiilor) este de fapt formata din cea *used* din care se scad bufferele si cea cached.

Nota: *informatiile oferite de **top** pot fi obtinute separat si cu alte comenzi. Spre exemplu, **free** afiseaza statistici legate de memorie, in care este prezentata memoria libera in ambele variante (tinand sau nu cont de buffere si cache). Folosind optiunile -b, -k, -m, -g dimensiunile pot fi afisate in B/KB/MB/GB, iar cu -h va fi aleasa automat reprezentarea optima:*

InfoAcademy

```
student@Desktop:~$free -h
             total       used       free     shared    buffers     cached
Mem:          7,7G       7,6G       134M       725M       519M       2,7G
-/+ buffers/cache:       4,3G       3,4G
Swap:          18G         0B        18G
```

Comanda top dispune de o multitudine de shortcut-uri – mentionam doar cateva:

- lista de procese prezentata de top poate fi ordonata dupa diferitele coloane. In general se poate folosi SHIFT+initiala coloanei; de exemplu, **SHIFT-p** va ordona dupa procentul de timp de procesor, **SHIFT-m** dupa cantitatea de memorie ocupata etc. Alternativ, se poate apasa **f** sau **F**, dupa care se va putea alege din sageti campul dupa care sa se faca ordonarea, setarea sa facandu-se cu **s**. In oricare dintre cazuri, daca se doreste ordonare descrescatoare dupa campul ales se va apasa **SHIFT-r**
- procesele pot fi reprioritizate apasand **r**; va fi solicitat apoi PID-ul si noua prioritate dorita
- procesele pot fi inchise fortat apasand **k** ; vor fi solicitate PID-ul si tipul de semnal trimis procesului (vezi mai jos capitolul dedicat semnalelor)

Nota: *lista completa de shortcut-uri disponibile se obtine apasand h (help).*

5.1.4. Prioritizarea proceselor

5.1.4.1. Despre executie concurenta si prioritati

Executia concurenta a mai multor procese pe acelasi procesor presupune organizarea acestora de catre un scheduler (planificator de procese). Schedulerul este cel care, date fiind mai multe procese dornice de a rula, trebuie sa aleaga la un moment dat un anume proces care va fi executat pe procesorul in cauza. Alegerea este influentata de mai multi factori, printre care si prioritatea procesului. Prioritatea este un numar; cu cat numarul este mai mic, cu atat prioritatea procesului este mai buna si acesta va fi preferat de catre scheduler in detrimentul altora.

Sistemul de operare insusi este compus dintr-un ansamblu de procese care conlucreaza, o buna parte dintre ele fiind critice pentru corecta sa functionare. Acesta este motivul pentru care a fost gandita o separare din punct de vedere al prioritatilor: o aplicatie obisnuita, de utilizator, nu poate capata o prioritate mai buna decat o aplicatie de sistem (ceea ce ar putea duce la potentiala instabilitate a sistemului). Utilizatorul Linux poate doar influenta prioritatea unui proces, nu o poate seta direct.

Fiecare proces are doua caracteristici distincte:

- prioritate - numarul folosit de catre schedulerul de procese pentru a decide daca sa prefere acel proces in detrimentul altora
- nice value - un numar intre -20 si +19, care indica gradul de “niceness” (buna purtare) al procesului fata de celelalte. Nice value este setabil de catre user si reprezinta un ingredient ce intra in calculul prioritatii; valoarea -20 determina prioritatea cea mai buna, +19 cea mai slaba.

Nota: *prioritatea reala si nice value-ul unui proces pot fi vizualizate cu **ps -l/ps -o**, **chrt** sau **top**. Comanda **chrt** poate fi utilizata si pentru modificarea prioritatii reale a unui proces - a se folosi insa cu precautie!*

Nice value-ul primului proces pornit de catre kernel este 0; subprocesele mostenesc valoarea nice a procesului parinte, de aceea multe dintre procesele ce ruleaza in sistem vor avea nice-ul 0. Pe de alta parte, atat kernel-ul, cat si utilizatorul pot porni procese cu alt nice value decat cel mostenit de la parintele sau

Nota: *cu setarile din oficiu, numai root poate impune valori negative pentru nice-ul unui proces. Daca dorim ca utilizatorii obisnuiti sa aiba si ei aceasta posibilitate, putem edita fisierul /etc/security/limits.conf.*

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

Trebuie inteles ca nu exista neaparat o corespondenta directa intre valoarea nice a procesului si timpul de procesor pe care acesta il va primi. Spre exemplu, nu avem garantia ca un proces cu nice -4 va primi de doua ori mai mult timp de procesor decat unul cu nice -2. Felul in care timpul acordat procesului depinde de nice value difera de la un sistem de operare la altul si este de asemenea dependent de felul in care este configurat scheduler-ul.

5.1.4.2. Comenzile nice si renice

Comanda **nice** este folosita pentru a porni un proces cu o valoare nice diferita de cea a parintelui sau. Spre exemplu, daca lansam o comanda care arhiveaza continutul unui director si dorim ca aceasta sa impieteze cat mai putin asupra functionarii restului sistemului, putem porni procesul sau cu o valoare nice mai mare.

Sintaxa comenzii nice presupune specificarea numarului de trepte cu care se modifica valoarea nice a procesului fata de cea default (prin intermediul optiunii -n) si apoi a comenzii ce se doreste lansata in executie prioritizat:

```
nice -n+5 comanda_arhivare
# se porneste procesul care ruleaza comanda_arhivare; daca prioritatea shell-ului era 3,
# atunci noul proces va avea prioritatea 8
```

Atunci cand dorim schimbarea “din mers” a valorii nice pentru un proces deja pornit, putem folosi comanda **renice**. Aceasta primeste ca argument noua valoare nice si, cu ajutorul optiunii -p, PID-ul procesului vizat.

Atentie! Utilizatorii non-root nu pot seta/schimba valoarea nice decat pentru procese proprii si nu pot descreste valoarea nice a unui proces, chiar daca ei sunt cei care au crescut-o initial!

```
student@Desktop $ renice +10 -p 8319
8319: old priority -3, new priority 10
student@Desktop $ renice +5 -p 8319
renice: 8319: setpriority: Permission denied
```

5.1.5. Comunicarea inter-proces

5.1.5.1. Notiunea de semnal

Un semnal este o intrerupere software directionata catre un proces, avand rolul de a-l instiinta pe acesta asupra unui eveniment aparut. Semnalul poate fi trimis de catre un alt proces, de catre kernel sau de catre acelasi proces siesi. Evenimentele semnalate pot fi:

- erori – situatii neasteptate in urma carora executia procesului nu mai poate continua (ex: diviziune cu 0)
- evenimente externe procesului – sosirea unor date de intrare, expirarea unui timer, terminarea unui subproces
- incercari de comunicare din partea altui proces – spre exemplu, apasarea unei combinatii de taste in terminalul la care este conectat procesul (ex: apasarea combinatiei CTRL-c) sau trimiterea manuala a uni semnal folosind comenzi dedicate (vezi mai jos)

Semnalele sunt niste constante cu valoare numerica, fiecare cu o anumita semnificatie. Setul de semnale definite in sistem poate fi consultat cu *man 7 signal* si o lista a lor poate fi obtinuta cu comanda *kill –l*. Iata numele si semnificatia catorva semnale uzuale:

- erori

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

   - SIGFPE (floating point exception) – generat in cazul erorilor aritmetice (ex: diviziune cu 0)
   - SIGILL – instructiune ilegala
   - SIGSEGV (segmentation violation) – generat cand procesul incearca sa foloseasca memorie ce nu-i apartine
- terminare de proces (in paranteza se gaseste numarul semnalului)
   - SIGTERM (15) – este o cerere "politicoasa" de incheiere a procesului ce-l primeste; acesta poate sa ignore, sa trateze sau sa blocheze temporar acest tip de semnal
   - SIGKILL (9) – determina incheierea imediata si neconditionata a procesului destinatar; spre deosebire de SIGTERM, are actiune predeterminata - nu poate fi ignorat sau tratat in alt fel
   - SIGINT (2) – trimis catre aplicatia aflata in foreground la apasarea combinatiei CTRL-c in linia de comanda; reprezinta cererea de intrerupere a aplicatiei. Semnalul este ignorabil de catre aplicatie
   - SIGHUP (1) – hang up. Este trimis unui proces daca terminalul la care acesta este conectat se inchide inainte de incheierea executiei procesului. Efectul este in general inchiderea procesului. Pentru a rula un proces imun la semnalul SIGHUP poate fi folosita comanda *nohup* (vezi mai jos). In cazul proceselor care nu sunt conectate la un terminal (ex: serverele) acest semnal este deseori folosit cu un alt scop: un server care primeste semnalul SIGHUP isi va reciti din mers fisierul de configurare

Observatie: *ne explicam acum mesajele care apar la oprirea unei statii Linux: "sending all processes the TERM signal" – se incearca intai oprirea "politicoasa" a proceselor active – iar apoi, dupa un scurt interval de timp, "sending all processes the KILL signal" pentru a le inchide pe cele care nu au reactionat in timp util la SIGTERM.*

5.1.5.2. Comenzile kill, pkill si killall

Aceste comenzi sunt folosite pentru trimiterea explicita de semnale proceselor folosind linia de comanda:

- **kill** accepta ca argument PID-ul procesului
- **pkill** primeste ca argument un fragment de nume de proces sau un regex, trimitand semnalul dorit tuturor proceselor al caror nume contine fragmentul dorit sau respecta formatul specificat in regex (a se folosi cu prudenta!)
- **killall** primeste ca argument numele comenzii vizate si trimite semnalul solicitat tuturor proceselor ce ruleaza acea comanda. Util pentru a inchide toate instantele unei aplicatii ce genereaza mai multe procese.

In cazul tuturor comenzilor de mai sus, semnalul ce se doreste a fi trimis se specifica sub forma de optiune a comenzii; poate fi folosit numarul semnalului, numele său complet, sau numele partial (fara prefixul SIG). Atunci cand nu se specifica un semnal, cel implicit este SIGTERM:

```
# trimitere SIGHUP catre procesul ce ruleaza httpd; aflam intai PID-ul
student@Desktop$ pgrep httpd
17323
student@Desktop$ kill –HUP 17323
# ...echivalent cu a scrie kill –SIGHUP 17323 sau kill -1 17323

#SAU, daca se doreste trimiterea semnalului SIGKILL catre toate procesele httpd, avem variantele:
student@Desktop$ pkill -KILL htt
student@Desktop$ killall -9 httpd

# trimitere SIGTERM catre procesul skype - variante:
student@Desktop$ kill 17323
student@Desktop$ kill -TERM 17323
student@Desktop$ kill -SIGTERM 17323
```

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

5.1.5.3. Comanda trap

Aceasta comanda este folosita in scripturile de shell pentru a captura semnale si a le trata. Sintaxa sa este:

```
trap comanda tip_semnal
```

unde *comanda* va fi executata la aparitia unui semnal de tipul specificat. Daca *comanda* este sirul vid (“”), semnalul este ignorat (capturat si netratat).

5.1.5.4. Comanda nohup

Este folosita cand dorim ca o comanda sa nu reactioneze la primirea semnalului HUP (hang up), altfel spus sa continue sa ruleze si dupa deconectarea terminalului din care a fost pornita. Sintaxa sa este:

```
nohup comanda &
```

Rularea comenzii in background trebuie specificata explicit cu &.

nohup redirectioneaza automat output-ul comenzii catre fisierul *nohup.out* din directorul curent; daca nu poate scrie in acesta, va folosi *$HOME/nohup.out*. In acest fel, executia comenzii dorite nu va fi afectata de eventuala inchidere a terminalului. Procedeul este util atunci cand utilizatorul se conecteaza pe un server de la distanta, porneste un proces de durata (compilare a unui soft, arhivare etc) si apoi se deconecteaza, urmand a se reconecta ulterior pentru a verifica progresul operatiunii.

5.2. Rularea mai multor comenzi in paralel din linia de comanda

5.2.1. Descrierea problemei si solutii disponibile

Deseori in linia de comanda Linux apare necesitatea executiei mai multor comenzi in paralel. Poate dorim sa pornim o arhivare sau o compilare in timp ce lucram in linia de comanda Linux, sau poate ca pur si simplu dorim sa ascultam niste muzica in timp ce ne desfasuram activitatea in terminal. Aceasta nu este o problema in configurarile standard ale distributiilor Linux – odata ce ne aflam in fata computerului, avem la dispozitie cele 6 console virtuale sau, daca mediul grafic este instalat, putem oricand porni mai multe instante de emulator de terminal; problema apare insa atunci cand, din varii motive, suntem constransi sa folosim un singur terminal – fie fiindca sistemul de operare a fost configurat de asa natura, fie fiindca suntem conectati de la distanta pe o statie care nu accepta mai multe conexiuni simultane de pe aceeasi adresa IP.

Pentru a rula mai multe comenzi in paralel din acelasi terminal avem la dispozitie cel putin cateva solutii:

- daca folosim inetrpretorul bash, acesta ne pune la dispozitie o facilitate numita *job control* (descrisa mai jos), care permite rularea comenzilor in background in mod controlat
- pentru interpretoare de comenzi care nu au aceasta facilitate, exista utilitarul *screen* care creeaza terminale virtuale in cadrul unui unic terminal real

5.2.2. Job control

In bash, comenzile pot fi rulate in background, pe durata intregii lor executii sau numai pe portiuni. Aceasta inseamna ca aplicatia rulata se detaseaza temporar de terminalul curent si ii inapoiaza utilizatorului promptul, astfel incat acesta poate lansa alte comenzi in timpul executiei celei initiale (spre deosebire de rularea obisnuita, cand utilizatorul recapata posibilitatea de a introduce alte comenzi abia dupa incheierea executiei celei curente).

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

Comenzile trimise in background pot fi readuse in foreground, oprite temporar, trimise inapoi in background etc.

O comanda poate fi rulata in background inca de la inceputul executiei sale, adaugand un caracter & la sfarsitul liniei de comanda:

```
# o operatie de arhivare care poate dura mult
student@Desktop$ tar cjf arhiva.tar.bz2 * &
[1] 1115
```

Numarul dintre parantezele patrate este Job ID-ul – modalitatea shell-ului de a tine evidenta comenzilor rulate in background. Cel de-al doilea numar reprezinta PID-ul procesului creat.

Pentru a rula mai multe comenzi in paralel, ele pot specificate in cadrul aceleiasi linii de comanda, insa fiecare urmata de un caracter &, ca in exemplul de mai jos. Fiecare comanda va fi pornita cat de repede posibil, fara a astepta executia celei anterioare. Promptul revine si el de indata ce au fost pornite toate comenzile implicate:

```
student@Desktop $ comanda1 & comanda2 & comanda3 &
```

Lista de comenzi aflate in desfasurare in background poate fi vizualizata cu comanda jobs:

```
student@Desktop$ jobs
[1]-   tar cjf arhiva.tar.bz2 * &
[2]+   make -j2 &
```

O comanda aflata in background poate fi adusa inapoi in foregound folosind comanda **fg** urmata de job id-ul comenzii, precedat de caracterul % ca mai jos:

```
student@Desktop$ fg %1
tar cjf arhiva.tar.bz2 *
[...urmeaza output-ul comenzii de arhivare...]
```

O comanda aflata in foreground poate fi oprita cu combinatia de taste CTRL-z; aceasta trimite un semnal SIGSTOP procesului, care se va opri temporar pana cand primeste semnalul de continuare (SIGCONT). Reluarea procesului din punctul ramas se poate efectua pornind job-ul in cauza in foreground sau background:

```
student@Desktop$ tar cjf arhiva.tar.bz2 *
^Z
[1]+  Stopped                 tar cjf arhiva.tar.bz2 *
student@Desktop$ bg %1
[1]+   tar cjf arhiva.tar.bz2 * &
```

5.2.3. Screen

Comanda *screen* este folosita pentru a crea terminale virtuale in cadrul unuia fizic. Odata rulata, aceasta comanda deschide un prim terminal virtual. De aici pot fi efectuate diferite operatii, descrise mai jos.

Pentru a accesa comenzile screen (cele care creeaza noi terminale, schimba terminalul curent etc) este necesara o combinatie de taste mai aparte. In configuratia din oficiu, se apasa intai CTRL-a si apoi, separat, tasta corespunzatoare comenzii dorite. Iata cateva combinatii utile:

- CTRL-a c – creeaza un nou terminal virtual
- CTRL-a SPACE – comuta pe urmatorul terminal virtual din lista

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

- CTRL-a p – comuta pe terminalul precedent din lista
- CTRL-a “ – afiseaza lista terminalelor virtuale create, cu posibilitatea comutarii rapide pe cel dorit
- CTRL-a d – realizeaza detasarea de sesiunea screen. Sesiunea ramane pornita – in sensul in care toate aplicatiile care rulau in terminalele virtuale vor continua sa ruleze – insa utilizatorul poate inchide acum terminalul fizic din care a pornit sesiunea screen, putandu-se reconecta la acea sesiune din alt terminal, fie el consola, emulator de terminal sau chiar terminal deschis de pe o alta masina prin SSH!

5.3. Servicii

5.3.1. Conceptul de serviciu si sisteme de initializare

Sistemul de operare este compus dintr-o multitudine de programe care conlucreaza. La bootare, boot loaderul incarca kernel-ul iar acesta porneste un prim proces. Acel prim proces porneste la randul sau o serie de alte procese, coordonat fiind cu ajutorul unuia sau mai multor fisiere de configurare. Procesele pornite ruleaza si ele alte procese s.a.m.d pana cand bootarea sistemului de operare este completa si utilizatorul se regaseste in fata promptului de login, fie el el consola sau fereastra grafica.

Problema este ca aceste aplicatii componente ale sistemului de operare nu sunt pornite la bootare si atat; in activitatea uzuala unele dintre ele trebuie oprite, restartate, diagnosticate acolo unde este cazul etc. Desigur, am putea opri procesele corespunzatoare trimitandu-le semnale, si le-am putea reporni apeland direct executabilul sau scriptul in cauza cu optiunile potrivite; este nevoie insa de o interfata de management de nivel mai inalt, mai flexibila. Sa luam exemplul softurilor de tip server (web, mail etc): procesul de configurare/reconfigurare a unui server presupune uneori multiple reporniri ale acestuia, in conditiile in care serverul este compus din multiple excutabile si creeaza la pornire mai multe procese. A inchide toate acele procese manual si a reporni serverul de mana devine un proces ineficient, consumator de timp si cu probabilitate ridicata de a gresi.

In acest scop a fost creat conceptul de *serviciu* - o aplicatie sau facilitate a sistemului de operare care poate fi usor manipulata de catre administrator, in mod automatizat. La nivel de sistem de operare exista o interfata unitara pentru administrarea serviciilor, cu comenzi si fisiere de configurare atasate, care permite operatii precum:

- pornire/oprire/restartare a unui serviciu. In felul acesta folosim aceleasi comenzi pentru a manipula diversele servere/servicii, nu mai avem nevoie sa cunoastem “bucataria interna” a acelui soft
- aflarea status-ului unui serviciu. Administratorul are nevoie sa stie daca un serviciu este sau nu pornit si, in cazul in care pornirea sa a esuat, care este motivul
- managementul serviciilor pornite automat la bootare
- managementul interdependentelor intre servicii. Serviciile tind sa fie conditionate (sa depinda) unele de altele; spre exemplu, serverul web depinde de serviciul de retea. Deseori dorim ca atunci cand se opreste un serviciu sa fie oprite si cele care depind de el; de asemenea, daca el nu este pornit, sa nu poata porni nici cele care depind de el

Istoric vorbind au existat diverse moduri de a organiza bootarea sistemului de operare si managementul de servicii; acestea poarta numele de “stiluri de initializare” sau “sisteme de initializare” (init systems):

- stilul **BSD** (pre-SystemV). Este unul simplu, mai putin structurat, intalnit rar in Linux
- stilul **System V** - multi ani modalitatea standard de management a startup-ului si serviciilor, prezent in majoritatea Linux-urilor insa inlocuit treptat de solutii mai performante
- stilul **Upstart** - inlocuitor al lui System V promovat de catre Canonical (firma din spatele distributiei Ubuntu) dar care a pierdut teren in favoarea lui systemd
- stilul **systemd** - un sistem care, spre deosebire de System V, paralelizeaza pornirea diverselor componente ale sistemului de operare, reducand timpii de asteptare

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

5.3.2. Managementul serviciilor SysV

5.3.2.1. Principii

In System V, intreaga initializare a sistemului de operare si managementul de servicii se realizeaza prin scripturi de shell. Fiecare serviciu definit in sistem are un script corespunzator in directorul **/etc/init.d**. In momentul instalarii unui nou serviciu/server va fi creat si scriptul corespunzator pentru managementul acestuia.

Nota: *afirmatia anterioara este valabila numai in cazul instalarii din pachete precompilate; ea nu se aplica pentru compilarea si instalarea din surse. (vezi capitolul despre management software)*

Fiecare script din */etc/init.d* poate fi apelat cu un numar de argumente standard, fiecare argument reprezentand o operatie:

- **start** - pornirea serviciului
- **stop** - oprirea serviciului
- **restart** - repornirea serviciului (utila in cazul serverelor, pentru recitirea eventualelor modificari efectuate intre timp in fisierul de configurare)
- **status** - afisarea starii serviciului si a eventualelor informatii de diagnostic

Ca minim, utilizatorul Linux trebuie sa stapaneasca doua mari categorii de operatii cu servicii:

- pornire/oprire/restartare. Efectul operatiei dureaza doar pana la reboot; exista un set de configurari separat care dicteaza ce servicii pornesc automat la bootare
- administrarea serviciilor care pornesc automat la bootarea sistemului de operare

5.3.2.2. Managementul serviciilor in timpul rularii sistemului de operare

Exista doua modalitati de a manipula un serviciu:

- prin apelarea directa a scriptului sau, ca in exemplul de mai jos. Desi in general functioneaza, pot exista cazuri in care alternativa este superioara (vezi subpunctul urmator)

```
# repornire server web
/etc/init.d/apache2 restart
# oprire firewall
/etc/init.d/firewall stop
```

- prin apelarea unei comenzi dedicate. Solutia prezinta avantaje multiple: in primul rand, aceste comenzi tin cont de anumite restrictii/interdependente intre servicii; in al doilea rand, ele continua sa functioneze si in perioada de tranzitie, cand distributia in cauza migreaza de la SystemV la systemd (operatia solicitata fiind automat delegata catre systemd pentru serviciile deja migrate). Exemple:

```
# in distributiile din familia RedHat, dar si in Ubuntu si derivate
service apache2 restart
# in Debian si derivate
invoke-rc.d apache2 restart
```

Dupa cum se observa, comenzile dedicate primesc doua argumente - numele serviciului (care este de fapt numele de script din /etc/init.d) si operatia dorita, aceeasi ca in cazul apelarii directe a scriptului.

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

5.3.2.3. Managementul serviciilor la bootare

Administratorul Linux trebuie sa fie capabil sa execute cel putin urmatoarele operatii:

- determinarea listei de servicii

```
# in familia Redhat
chkconfig --list
# in familia Debian
service --status-all
```

- activarea/dezactivarea pornirii automate a unui serviciu la bootare. **Atentie!** Aceste comenzi dezactiveaza serviciul incepand cu urmatoarea pornire insa, daca momentan el ruleaza, nu va fi oprit!

```
# familia RedHat
chkconfig firewall off
chkconfig apache2 on
# familia Debian
update-rc.d firewall disable
update-rc.d apache2 enable
```

5.3.3. Managementul serviciilor systemd

Aceleasi operatii de mai sus pot fi efectuate si in systemd dupa cum urmeaza:

- pornire/oprire servicii:

```
systemctl start apache2
systemctl stop firewall
```

- listare servicii

```
systemctl list-unit-files
```

- activare/dezactivare serviciu la bootare

```
systemctl enable firewall
systemctl disable apache2
```

5.4. Rularea automatizata a proceselor

5.4.1. Solutii existente si diferente intre acestea

Intr-un sistem de operare exista o multitudine de operatii legate de mentenanta sau administrare care trebuie executate periodic, automat; la acestea se adauga diferite task-uri ocazionale. In acest scop, in Linux exista felurite programe menite a facilita rularea periodica sau one-time de procese in background:

- **cron** - este softul traditional Linux/Unix care permite rularea periodica de task-uri, sub forma de fisiere executabile sau script-uri. Job-urile pot fi programate de catre administratorul de sistem sau de catre useri, cu granularitate de minut. Daca statia nu este pornita la ora stabilita pentru rularea unui job, acea rulare nu va mai avea loc - cron nu are un sistem de evidenta a task-urilor ratate, ci le ruleaza doar pe cele a caror ora de incepere gaseste statia pornita. In acest fel, una sau mai multe executii consecutive ale unui task pot fi omise

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

- **anacron** - complement al lui cron, gandit pentru task-urile de mentenanta periodice care nu trebuie ratate. Spre deosebire de cron, care este gandit mai degraba pentru statii pornite permanent (ex: servere), anacron ruleaza dupa bootare task-urile ratate cat timp statia a fost stinsa
- **atd** - soft care permite executia one-time a unei comenzi sau succesiuni de comenzi de catre utilizatorii unui sistem (ex: setarea unui reminder care sa fie afisat la o anumita data/ora sau peste un anumit interval de timp din momentul programarii sale)

Nota: *numim in acest material* task *sau* job *una dintre operatiile executate in mod automat, planificat, de catre aceste utilitare la cererea noastra.*

5.4.2. Cron

5.4.2.1. Principii si structura fisierelor de configurare

Cron este numele generic dat unei familii de softuri cu scop/configurare asemanatoare. Traditional, *cron* era daemonul Unix responsabil cu rularea periodica de task-uri; versiunea cea mai raspandita in ziua de astazi in Linux este un urmas al sau numit initial *vixie-cron* si rebotezat ulterior *ISC cron*. Exista si alte variante mai putin populare dar cu facilitati sporite (*dcron*, *fcron* etc).

Functionarea lui cron se bazeaza pe un daemon numit *crond* care ruleaza o data pe minut si isi citeste fisierele de configurare, executand task-urile care sunt programate pentru acea ora. Task-urile pot fi programate fie de catre administratorul de sistem, fie de catre utilizatori; in acest scop exista doua mari categorii de fisiere de configurare:

- globale - editabile numai de catre administratorul sistemului
  - **/etc/crontab** - fisierul principal de configurare, utilizabil numai de catre root
  - **/etc/cron.d/** - pentru o mai buna organizare/structurare a configurarii, in loc sa definim absolut toate task-urile in /etc/crontab, putem crea fisiere individuale per-task in /etc/cron.d/, folosind aceeasi sintaxa ca in cel principal. Fiecare fisier va purta un nume sugestiv pentru task-ul pe care il realizeaza, usurand astfel administrarea configurarii
  - **/etc/cron.hourly, /etc/cron.daily, /etc/cron.weekly, /etc/cron.monthly, /etc/cron.yearly** - daca exista, in aceste directoare pot fi plasate task-uri ce trebuie sa se execute in fiecare ora/zi/saptamana/luna/an
- per-user, administrabile cu comanda crontab (vezi mai jos)
  - **/var/spool/cron/crontabs/*username*** - fisierul cu task-uri definite de userul *username*

Nota: *crond isi citeste fisierele de configurare la fiecare rulare, asadar nu este nevoie sa repornim serviciul cron dupa efectuarea de modificari.*

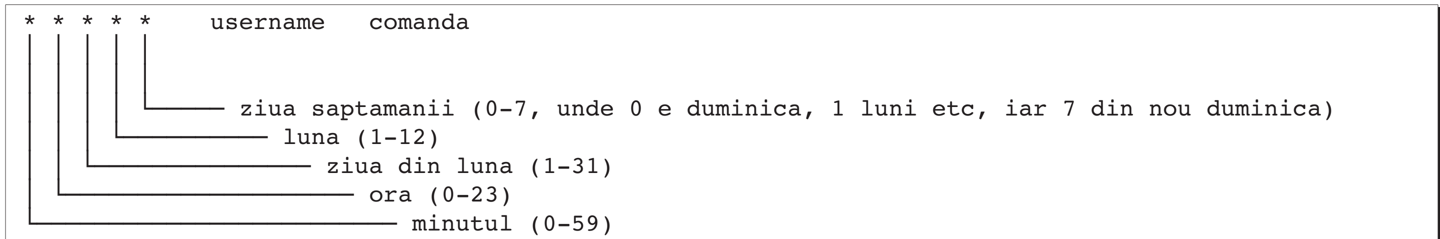
Formatul fisierelor de configurare globale si per user este identic, cu o singura exceptie: in cele globale se specifica username-ul cu identitatea caruia trebuie rulat task-ul, pe cand la cele per-user este folosita automat identitatea userului in cauza.

5.4.2.2. Configurare globala

Configurarea globala se bazeaza pe urmatoarele fisiere/directoare:

1. fisierul **/etc/crontab** (”cron table”) si fisierele subordonate din directorul **/etc/cron.d/**. Formatul fiecarei linii pentru aceste fisiere presupune urmatoarele 7 campuri, separate prin spatiu sau TAB:

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.



Dupa cum a fost explicat anterior, *crond* isi citeste fisierele de configurare in fiecare minut. Pentru fiecare task gasit, daca valorile campurilor corespund cu data/ora curenta, task-ul va fi rulat.

Valoarea fiecaruia dintre cele 5 campuri poate fi specificata in diferite moduri:

- valoare exacta (ex: **10**)
- nume. In cazul zilei si lunii se poate utiliza numele prescurtat la 3 litere din limba engleza (**sun, mon, tue** etc pentru zile, respectiv **jan, feb, mar** etc pentru luni)
- domenii de valori, folosind caracterul - (minus). Ex: **1-4**
- liste de valori, folosind caracterul , (virgula). Ex: **1,3**
- combinatii de cele de mai sus. Exemplu: **1,3,5-9**
- domeniul de valori complet al campului, folosind caracterul *****. Spre exemplu, daca vrem ca un task sa se execute indiferent de valoarea orei (adica sa nu impunem restrictii asupra acelui camp) vom scrie in campul de ora * in loc de 0-23
- domenii de valori cu increment (pas) specificat, folosind caracterul /. Spre exemplu:
  - daca dorim executie la fiecare trei minute insa doar in primul sfert al fiecarei ore, vom folosi pentru campul de minut valoarea **0-15/3** (0-15 este domeniul de valori, iar 3 este pasul): vor rezulta valorile 0, 3, 6, 9, 12 15. Acelasi efect putea fi obtinut prin intermediul unei liste (0,3,6,9,12,15) insa scriind mai mult
  - daca dorim executia din 3 in 3 minute permanent, putem scrie pentru campul de minut ***/3**

Cu unele exceptii, comanda este bine sa aiba specificata calea absoluta. Daca se doreste executia mai multor comenzi, acestea se vor separa prin ; (punct si virgula). Output-ul comenzii este suprimat (nu este afisat pe ecran); daca se doreste, el poate fi trimis catre o adresa email configurabila (pentru detalii vezi *man 5 crontab*).

Sa luam mai intai un exemplu real - rularea task-urilor din directoarele */etc/cron.daily*, */etc/cron.hourly* etc:

```
17 *   * * *  root    cd / && run-parts --report /etc/cron.hourly
25 6   * * *  root   test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6   * * 7  root   test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6   1 * *  root   test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
```

Task-urile de mai sus vor rula dupa cum urmeaza:

- primul task ruleaza in fiecare ora, la 17 minute dupa ora fixa (3:17, 4:17 etc). Cron va declansa task-ul ori de cate ori ceasul arata "si 17 minute"
- al doilea task ruleaza la ora 6:25 in fiecare zi. Cron va executa acest job ori de cate ori ora este 6 si minutul este 25 (celelalte campuri nu au niciun fel de restrictii)
- al treilea task ruleaza la ora 6:47 insa de data aceasta avem restrictie asupra zilei saptamanii: numai duminica. Rezulta deci o rulare saptamanala, duminica la ora 6:47
- al patrulea task ruleaza ori de cate ori ora este 6:52 si ziua din luna este 1. Asadar lunar, pe 1 ale lunii, la ora 6:52

Alte exemple:

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

```
# ceas cu cuc (task ce ruleaza la ora fixa)
0             *      *      *      *      root   echo "Cucuuuu"|wall
# (comanda wall trimite mesaj catre toti userii logati)

# ceas cu cuc care nu ne deranjeaza noaptea(ruleaza la ora fixa dar numai intre orele specificate)
0             9-21   *      *      *      root   echo "Cucuuuu"|wall
# cuc intermediar, care anunta fiecare sfert de ora cu exceptia orelor fixe
15-45/15      *      *      *      *      root   echo 'Viata ta s-a scurtat cu inca 15 minute!'|wall
# rulare task din 5 in 5 minute in zilele de luni
*/5           *      *      *      mon    user1  comanda1
# rulare task la ora 9 dimineata in zilele lucratoare
0             9      *      *      1-5    sef    echo 'La munca!!'|sendmail angajati@firma.ro
# rulare task intre 8PM si 6AM, din 2 in 2 ore, la "si un sfert"
15     0-5,20-23/2   *      *      *      root
```

Nota: *in locul celor 5 campuri ce specifica data si ora se accepta si shortcut-uri de forma @reboot, @hourly (echivalent cu "0 * * * *"), @daily (echivalent cu "0 0 * * *") etc.*

2. directoarele **/etc/cron.hourly, /etc/cron.daily, /etc/cron.weekly, /etc/cron.monthly, /etc/cron.yearly**. Fiecare fisier din aceste directoare reprezinta un task rulat cu periodicitatea indicata de numele directorului său; in consecinta continutul fisierului va fi format chiar din comanda/comenzile de rulat, fara alte elemente de configurare. Rularea task-urilor din aceste directoare este realizata prin intermediul unor job-uri definite in /etc/crontab, care stabilesc si ora executiei (vezi unul din exemplele de mai sus)

Spre exemplu, daca dorim sa monitorizam evolutia zilnica a spatiului liber din partitia /, vom crea un fisier numit *monitorizare_spatiu* in directorul */etc/cron.daily*, ii vom adauga permisiune de executie, iar continutul sau va fi format din comanda *df -h / >> /var/log/spatiu_liber.*

5.4.2.3. Configurare per-user

Configurarea per-user presupune crearea de task-uri in fisierele /var/spool/cron/crontabs/<username>. Formatul unui astfel de fisier este identic cu cel global, cu o singura exceptie: nu mai exista campul de username, deoarece la rulare este folosita automat identitatea proprrietarului fisierului crontab.

Desi posibila, editarea directa a acestor fisiere nu este recomandata; se prefera in schimb utilitarul **crontab**, ce permite urmatoarele operatii:

- **crontab -e** → editarea fisierului crontab al userului curent
- **crontab -r** → stergerea fisierului crontab al userului curent
- **crontab -l** → listarea fisierului crontab al userului curent

Nota: *utilizarea comenzii crontab poate fi restrictionata per-user folosind fisierele /etc/cron.allow si/sau /etc/cron.deny (vezi man crontab).*

5.4.3. Anacron

Scopul lui *anacron* este sa garanteze executia periodica a unor task-uri esentiale de mentenanta/administrare pe statii care nu sunt pornite permanent. El vine ca o completare a lui *cron*, insa cu urmatoarele diferente esentiale fata de acesta:

- daca statia nu este pornita in momentul stabilit pentru rularea unui task, acesta va fi rulat la scurt timp dupa bootare (timpul in cauza fiind configurabil)
- perioada de repetitie a executiei se poate stabili doar in zile (nu mai exista cele 5 campuri din fisierele crontab)

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

- task-urile anacron pot fi definite numai de catre root, nu mai exista task-uri per user

Fisierul de configurare este */etc/anacrontab*, cu urmatoarea sintaxa:

```
perioada     delay        job_identifier      comanda
```

Perioada este numarul de zile intre doua repetitii consecutive; daca la expirarea unei astfel de perioade statia este inchisa, jobul in cauza va rula dupa urmatoarea bootare, si mai exact la *delay* minute dupa pornirea sistemului. Job identifier-ul este o simpla descriere a task-ului in cauza, memorata ca atare si in loguri la rulare.

Exemplu de fisier anacrontab:

```
1       5       cron.daily    run-parts --report /etc/cron.daily
7       10      cron.weekly   run-parts --report /etc/cron.weekly
@monthly        15      cron.monthly run-parts --report /etc/cron.monthly
```

5.4.4. Atd

Atd este un program rezident in memorie care ruleaza task-uri primite de la utilizatori la momentele de timp specificate de catre acestia. Pentru a programa un astfel de task si pentru management-ul task-urilor in general, utilizatorii folosesc urmatoarele comenzi:

1. **at *moment*** - permite crearea unui nou task. Dupa rularea comenzii utilizatorul scrie comenzile ce compun task-ul si incheie intregul set cu CTRL-d

```
student@server:~$ at 15:22
warning: commands will be executed using /bin/sh
at> echo Hello world|wall
at> <EOT>   ← CTRL-d
job 8 at Sun Oct 11 15:22:00 2015
```

Momentul executiei poate fi specificat intr-o multitudine de feluri - iata cateva:

- ora.
  - exacta: **18:24** (daca ora a trecut deja, executia va avea loc la aceeasi ora in ziua urmatoare)
  - moment din zi: **now** (momentul de timp curent), **midnight** (12 noaptea), **noon** (12 ziua), **teatime** (4PM)
  - ora urmata de deplasament temporal: **now+5 minutes**
- ora urmata de data: **18:24 jan 6 2017**

Nota: *alternativ, comanda de executat poate fi citita dintr-un fisier, folosind optiunea -f (vezi* man at*)*

2. **atq** - listeaza task-urile definite de userul curent (sau de toti userii, daca comanda este rulata de root)

```
student@server:~$ atq
8       Sun Oct 11 15:22:00 2015 a user1
```

Numarul de la inceputul liniei reprezinta job number-ul (util, spre exemplu, pentru a-l sterge), iar ultimul camp este username-ul care a programat job-ul.

3. **at -c *job_number*** - vizualizeaza un job deja definit
4. **atrm *job_number*** - sterge job-uri din lista

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

Nota: *ca si in cazul cron, administratorul sistemului poate restrictiona lista de useri ce au acces la facilitatea at, prin intermediul fisierelor /etc/at.allow si /etc/at.deny (vezi* man at.allow *si* man at.deny*)*

5.5. BIBLIOGRAFIE

- Procese, prioritati
  - detalii tehnice despre calculul prioritatii
    - http://www.informit.com/articles/article.aspx?p=101760 (vezi Calculating priority and timeslice)
    - http://superuser.com/a/877353
- Cron si anacron
  - http://www.tuxradar.com/content/automate-linux-cron-and-anacron
  - http://www.thegeekstuff.com/2011/05/anacron-examples/ (vezi tabel comparativ de la sfarsit)
- Management servicii
  - comparatie intre sisteme de initializare: http://0pointer.de/blog/projects/why.html
  - upstart: http://upstart.ubuntu.com/cookbook/
  - systemd
    - https://www.linux.com/learn/tutorials/524577-here-we-go-again-another-linux-init-intro-to-systemd
    - http://www.linux.com/learn/tutorials/527639-managing-services-on-linux-with-systemd
    - http://linoxide.com/linux-command/systemd-vs-sysvinit-cheatsheet/

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.