Alexandru-Nicolae Ion

Master IISC Anul I

Linux Kernel

Sisteme de Operare Avansate

[Introducere 2](#_Toc377592755)

[Versiuni de Linux 3](#_Toc377592756)

[Concepte de sisteme de operare 5](#_Toc377592757)

[Sisteme multi-utilizator 6](#_Toc377592758)

[Procese 6](#_Toc377592759)

[Arhitectura Kernel-ului 8](#_Toc377592760)

[Cerinte pentru creearea si utilizarea Kernelului 9](#_Toc377592761)

[Unelte pentru creearea Kernel - ului 9](#_Toc377592762)

[Compilatorul 9](#_Toc377592763)

[Linker 9](#_Toc377592764)

[Make 10](#_Toc377592765)

[Unelte pentru utilizarea Kernel –ului 10](#_Toc377592766)

[Uil-Linux 10](#_Toc377592767)

[Module-init-tools 11](#_Toc377592768)

[Alte unelte 11](#_Toc377592769)

[Structurile de date ale Kernelului 13](#_Toc377592770)

[Liste inlantuite 13](#_Toc377592771)

[Liste inlantuite individual si de doua ori 13](#_Toc377592772)

[Liste inlantuite circular 14](#_Toc377592773)

[Parcurgerea unei liste inlantuita 15](#_Toc377592774)

[Cozi 15](#_Toc377592775)

[Kfifo 16](#_Toc377592776)

[Harti (maps) 17](#_Toc377592777)

[Arbori binari 17](#_Toc377592778)

[Arbori binari de cautare 18](#_Toc377592779)

[Echilibrarea arborilor binari de cautare 18](#_Toc377592780)

[Sincronizarea kernel-ului 19](#_Toc377592781)

[Contexte de executie si locking 19](#_Toc377592782)

[Preemptivitate 19](#_Toc377592783)

[Cand sincronizarea este necesara si cand nu este necesara 20](#_Toc377592784)

[Concluzii 22](#_Toc377592785)

[Bibliografie 23](#_Toc377592786)

# Introducere

Linux este un membru al marii familii de sisteme de operare bazate pe sistemul **UNIX**. Un nou venit care a experimentat o crestere brusca si spectaculoasa a populariatii incepand cu sfarsiut anilor 1990, Linux se alatura unor sisteme bine cunoscute pe piata software, bazate pe **UNIX**. Dintre acestea putem amintii *System V Release 4* dezvoltat de ***AT&T,*** sistemul *4.4BSB* dezvoltat la Univesitatea din California, *Digital UNIX* creat de *Digital Equipment Corporation* ( acum numit **H**ewlett-**P**ackard), *AIX* produs de ***IBM***, *HP-UX* dezvoltat de ***H****ewlett-****P****ackard*, *Solaris*  produs de *Sun Microsystems* si *Max OS X* produs de *Apple Computer.*

Pe langa Linux, mai exista cateva kerneluri opensource,ale caror cod sursa poate fi accesat de oricine, bazate pe Unix, cum ar fi FreeBSD, NetBSD si OpenBSD.

Linux a fost initial dezvoltat de catre Linus Torvalds in 1991 ca un sistem de operare pentru calculatoarele personale IBM compatibile, bazate pe arhitectura microprocesorului Intel 80386. Linus ramane profund implicat in procesul de imbunatatire a Linux - ului, mentinandu-l up-to-date cu evolutia diferitelor componente hardware si prin coordonarea activitatii a sute de dezvoltatori Linux din intreaga lume. De-a lungul anilor, dezvoltatorii au lucrat pentru a face Linux disponibil si pe alte arhitecturi, inclusiv Hewlett-Packard Alpha, Itanium Intel, AMD AMD64, PowerPC, si zSeries IBM.

Unul dintre cele mai importante beneficii ale Linux-ului este ca acesta nu este un sistem de operare comercial: codul sursa aflat sub GNU General Public License (GPL) este deschis si disponibil pentru oricine pentru a putea fi studiat sau modificat.

Tehnic vorbind, Linux este un adevarat nucleu Unix, desi acesta nu este un sistem de operare Unix complet, deoarece nu include toate aplicatiile Unix, cum ar fi utilitatile sistemului de fisiere, sistemele de ferestre si desktop-uri grafice, comenzile de administrator de sistem, editoare de text, compilatoare, si asa mai departe. Cu toate acestea, pentru ca cele mai multe dintre aceste programe sunt disponibile in mod liber sub GPL, ele pot fi instalate in orice sistem bazat pe Linux.

Deoarece kernel-ul Linux necesita atat de mult software suplimentar pentru a oferi un mediu util, multi utilizatori Linux prefera sa se bazeze pe distributii comerciale, pentru a obtine codul inclus intr-un sistem standard de Unix. Alternativ, codul poate fi obtinut de pe diverse site-uri. Mai multe distributii pun codul sursa Linux in */usr/src* in directorul */linux*.

Diferitele sisteme bazate pe Unix aflate pe piata, dintre care unele au o lunga istorie si arata lungi semne de practici arhaice, difera in multe privinte importante. Toate variantele comerciale au fost derivate fie din *SVr4* sau *4.4BSD*, si toate au tendinta de a conveni asupra unor standarde comune, cum ar fi Sistemele de Operare Portabile *IEEE* bazate pe Unix (POSIX) si X/ Open`s Common Applications Environment (CAE).

Standardele actuale specifica doar o interfata de programare a aplicatiilor (API), care este, un mediu bine definit, in care programele de utilizator ar trebui sa ruleze. Prin urmare, standardele nu impun nicio restrictie in materie de optiuni de design intern ale unui nucleu compatibil.

Pentru a defini o interfata comuna de utilizator, kernelurile Unix de multe ori schimba idei de design si caracteristici fundamentale. In acest sens, Linux este comparabil cu alte sisteme de operare Unix-like.

Versiunea 2.6 a kernel-ului Linux isi propune sa fie in conformitate cu standardul IEEE POSIX. Aceasta, desigur, inseamna ca cele mai multe programe Unix existente pot fi compilate si executate pe un sistem Linux, cu foarte putin efort sau chiar fara a fi nevoie de patch-uri pentru codul sursa. Mai mult decat atat, Linux include toate caracteristicile unui sistem de operare modern, Unix, cum ar fi memoria virtuala, un sistem de fisiere virtual, procese usoare, semnale de Unix, comunicatii interprocese SVr4, suport pentru sisteme multiprocesor simetric (SMP), si asa mai departe.

Cand Linus Torvalds a scris primul kernel, el s-a folosit de cateva carti clasice de explicau functionalitatea Unix, cum ar fi cartea lui Maurice lui Bach de proiectare a sistemului de operare Unix (Prentice Hall, 1986). De fapt, Linux are inca unele prejudecati fata de linia de baza Unix descrisa in cartea lui Bach (de exemplu, SVR2). Cu toate acestea, Linux nu ramane la nici o varianta particulara. In schimb, incearca sa adopte cele mai bune caracteristici si optiuni de design de la mai multe kernel –uri diferite Unix.[1]

## Versiuni de Linux

Kernel-ul Linux a avut trei scheme diferite de numerotare.

Prima schema a fost utilizata in perioada premergatoare lui "1.0". Prima versiune a kernel-ului a fost de 0,01. Aceasta a fost urmata de 0,02, 0,03, 0,10, 0,11, 0,12 (prima versiune GPL), 0,95, 0,96, 0,97, 0,98, 0,99 si apoi 1,0. De la 0,95 au existat multe patch – uri eliberate intre versiuni.

Pana la versiunea de kernel 2.5, Linux a identificat kernel - urile printr-o schema de numerotare simpla. Fiecare versiune a fost caracterizata de trei numere, separate prin punct. Primele doua numere au fost folosite pentru a identifica versiunea, al treilea numar a fost folosit pentru a identifica release - ul. Primul numar al versiunii, si anume 2, a ramas neschimbat din 1996. Al doilea numar al versiunii identifica tipul de nucleu: in cazul in care acesta a fost par, aceasta denota o versiune stabila, altfel, aceasta denota o versiune de dezvoltare.

Versiunea a fost schimbata doar atunci cand au fost facute schimbari majore in codul si in conceptul de kernel, acest lucru realizandu - se de doua ori in istoria de kernel-ului: in 1994 (versiunea 1.0) si in 1996 (versiunea 2.0). Versiunea 3.0 a fost lansata in 2011, dar nu a fost o schimbare majora in conceptul de kernel. Revizuirea majora a fost atribuita in conformitate cu sistemul traditional de numerotare al versiunii de sistem par si-impar. Release - ul a fost schimbat la aparitia de patch-uri de securitate, remedieri de erori, introducerea de noi caracteristici sau la implementarea de drivere in kernel.

Dupa cum sugereaza si numele, versiunile stabile au fost temeinic verificate de catre distribuitorii de Linux si de kernel. O noua versiune stabila a fost lansata doar pentru a remedia bug-uri si pentru a adauga noi drivere de dispozitiv. Versiunile de dezvoltare, pe de alta parte, difera in mod semnificativ de la una la alta, dezvoltatorii kernel-ului au fost liberi sa experimenteze solutii diferite, care conduc uneori la schimbari drastice de kernel. Utilizatorii care se bazeaza pe versiuni de dezvoltare pentru rularea de aplicatii s-ar putea confrunta cu surprize neplacute atunci cand realizeaza actualizarea kernel –ului lor la o versiune mai noua.

In timpul dezvoltarii versiunii 2.6 de Linux kernel o schimbare semnificativa in schema de numerotare a versiunilor a avut loc. Practic, al doilea numar nu mai identifica versiunile stabile sau de dezvoltare.

In 2004, dupa ce versiunea 2.6.0 a fost lansata, dezvoltatorii kernel-ului au tinut mai multe discutii cu privire la release si schema de versiune si in cele din urma Linus Torvalds si altii au decis ca ar fi benefic un ciclu mai scurt intre release -uri. Timp de aproximativ sapte ani, primele doua numere au ramas "2.6", iar al treilea numar a fost incrementat cu fiecare versiune noua, care a fost lansata dupa doua-trei luni. Un al patrulea numar a fost uneori adaugat,pentru a evidentia repararea bug-urilor si sau de securitate la versiunile de kernel. S-a renuntat la sistemul par-impar de alternanta intre stabil si instabil.

La data de 29 mai 2011, Linus Torvalds a anuntat ca versiunea de kernel va fi incrementata la 3.0 la eliberarea versiunii urmatoare lui 2.6.39, pentru a comemora 20 de ani de Linux. A continuat practica de release pe baza de timp introdus cu 2.6.0, dar folosind al doilea numar. Un numar suplimentar (in prezent al treilea numar) va fi adaugat atunci cand este necesar pentru a desemna schimbari de securitate si remedieri de bug-uri, ca de exemplu 3.0.18.

Prima utilizare a celui de al patrulea numar avut loc atunci cand o eroare grava, care a necesitat fixare imediata, a fost intalnita in codul NFS in versiunea 2.6.8. Cu toate acestea, nu au fost suficient de multe alte modificari pentru a legitima lansarea unei noi vevizii minore (care ar fi fost 2.6.9). Deci, 2.6.8.1 a fost lansat, cu singura modificare fiind fixarea acelei eroari. Cu 2.6.11, aceasta a fost adoptata ca fiind noua politica oficiala de versionare.[2]

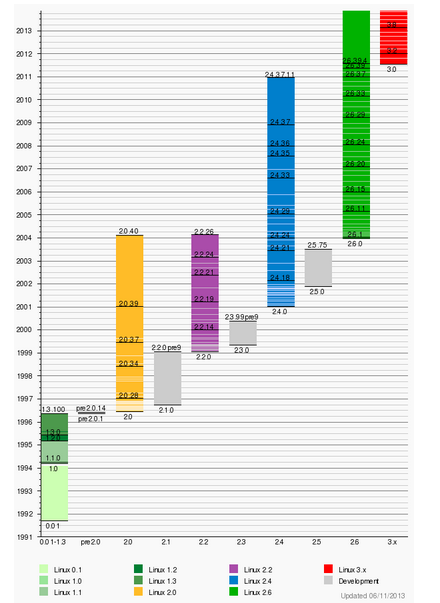


Figura 1.1 Versiuni de Linux Kernel

## Concepte de sisteme de operare

Fiecare sistem de computer include un set de programe de baza numite sistemul de operare. Cel mai important program in acest se este numit kernel. Acesta este incarcat in memoria RAM, atunci cand sistemul se incarca si contine multe proceduri esentiale, care sunt necesare sistemului sa functioneze. Celelalte programe sunt utilitati mai putin importante, ele pot oferi o mare varietate de experiente interactive pentru utilizator, precum realizarea cerintelor pentru care sistemul a fost cumparat, dar forma si capacitatile esentiale ale sistemului sunt determinate de kernel. Kernel-ul ofera facilitati cheie pentru programelor aflate in sistem si determina multe dintre caracteristicile software-ului mai abstract. Prin urmare, vom folosi de multe ori termenul de "sistem de operare", ca un sinonim pentru "kernel".

Sistemul de operare trebuie sa indeplineasca doua obiective principale:

* Sa interactioneze cu componentele hardware, manageriaza toate elemente low-level programabile incluse in platforma hardware.
* Sa ofere un mediu de executie pentru aplicatiile care ruleaza pe sistemul de calculator (asa-numitele programe de utilizator).

Unele sisteme de operare permit tuturor programelor utilizatorului sa interactioneze direct cu componentele hardware (un exemplu tipic este MS-DOS). In contrast, un sistem de operare Unix-like ascunde toate detaliile de nivel scazut in ceea ce priveste organizarea fizica a calculatorului de aplicatiile rulate de utilizator. Atunci cand un program doreste sa foloseasca o resursa hardware, aceasta trebuie sa emita o cerere pentru sistemul de operare. Kernel-ul evalueaza cererea si, in cazul in care alege sa acorde resurse, interactioneaza cu componentele hardware adecvate in numele programului de utilizator.

Pentru a pune in aplicare acest mecanism, sistemele de operare moderne se bazeaza pe disponibilitatea unor caracteristici hardware specifice, care sa interzica programelor utilizatorului de a interactiona direct cu componentele hardware low-level sau pentru a accesa locatii de memorie arbitrare. In special, hardware-ul introduce cel putin doua moduri diferite de executie pentru CPU: un mod neprevilegiat pentru programele utilizatorului si un mod privilegiat pentru kernel. Unix numeste aceste moduri Modul Utilizator si respectiv Modul Kernel.

### Sisteme multi-utilizator

Un sistem multi-utilizator este un computer care este capabil sa execute simultan si in mod independent mai multe aplicatii care apartin mai multor utilizatori. Executarea simultana inseamna ca aplicatiile pot fi active in acelasi timp, si lupta pentru diverse resurse, cum ar fi CPU, memorie, hard disk-uri, si asa mai departe. In mod independent inseamna ca fiecare aplicatie poate indeplini sarcina fara nici o preocupare pentru cererile facute de alti utilizatori. Trecerea de la o aplicatie la alta, desigur, incetineste executia fiecareia dintre ele si afecteaza timpul de raspuns perceput de catre utilizatori.

Sisteme de operare cu utilizatori multiplii trebuie sa includa mai multe caracteristici:

* Un mecanism de autentificare pentru verificarea identitatii utilizatorului
* Un mecanism de protectie impotriva programelor care ar putea intrerupe functionarea corecta a altor aplicatii aflate in sistem.
* Un mecanism de protectie impotriva programelor malware care ar putea interfera sau spiona activitatea altor utilizatori
* Un mecanism de contabilitate, care limiteaza cantitatea de unitati de resurse alocate pentru fiecare utilizator

Pentru a asigura mecanisme de protectie in conditii de siguranta, sisteme de operare trebuie sa utilizeze protectia hardware-ul asociat cu modul privilegiat CPU. Altfel, un program de utilizator ar putea accesa direct circuitele de sistem si de a depasi limitele impuse. Unix este un sistem multi-utilizator care impune protejarea hardware-ul al resurselor de sistem.

### Procese

Toate sistemele de operare folosesc o abstractie fundamentala: procesul. Un proces poate fi definit fie ca "o instanta a unui program in executie" sau ca "contextul de executie" al unui program care ruleaza. In sistemele de operare traditionale, un proces executa o singura secventa de instructiuni intr-un spatiu de adrese, spatiul de adrese fiind un set de adrese de memorie la care procesul poate sa faca referinta. Sisteme de operare moderne permit procese cu multiple fluxuri de executie, care sunt, mai multe secvente de instructiuni executate in acelasi spatiu de adrese.

Sistemele multi-user trebuie sa puna in aplicare un mediu de executie in care mai multe procese pot fi active concomitent si lupta pentru resursele sistemului, in special pentru CPU. Sistemele care permit procese active concurente se spune ca sunt multiprogramming sau multiprocesare. Este important sa se faca o distinctie intre programe si procese. Mai multe procese pot executa acelasi program simultan, in timp ce, acelasi proces se pot executa mai multe programe secvential.

Pe sistemele uniprocesor, doar un singur proces poate detine CPU, si, prin urmare, doar un singur flux de executie poate progresa la un moment dat. In general, numarul de procesoare este intotdeauna limitat, si, prin urmare, doar cateva procese pot progresa la un moment dat . O componenta din sistemul de operare numit planificatorul (scheduler) alege procesul care poate progresa. Unele sisteme de operare permit numai procese *nonpreemptable*, ceea ce inseamna ca planificatorul este invocat doar atunci cand un proces renunta in mod voluntar la ocuparea CPU - ului. Dar procesele unui sistem multi - utilizator trebuie sa fie *preemptibile*,asta inseamna ca sistemul de operare urmareste cat de mult detine CPU -ul fiecare proces si periodic activeaza planificatorul.

Unix este un sistem de operare cu multiprocesare cu procese preemptibile. Chiar si atunci cand nu este logat nici un utilizator si nici o cerere nu se executa, mai multe procese de sistem monitorizeaza dispozitivele periferice. In particular, mai multe procese asculta la terminalele sistemului asteptand pentru logarea unui utilizator. Atunci cand un utilizator introduce un nume de conectare, procesul de ascultare ruleaza un program care valideaza parola de utilizator. In cazul in care identitatea utilizatorului este recunoscuta, procesul creeaza un alt proces care ruleaza un shell in care sunt introduse comenzi. Atunci cand un afisaj grafic este activat, un proces executa managerul de ferestre, si fiecare fereastra de pe ecran este de obicei condusa de un proces separat. Atunci cand un utilizator creeaza un shell grafic, un proces executa ferestrele grafice si un al doilea proces executa shell - ul in care utilizatorul poate introduce comenzile. Pentru fiecare comanda data de utilizator, procesul de shell creeaza un alt proces care executa programul corespunzator.

Sistemele de operare Unix - like, adopta un model de *proces / kernel*. Fiecare proces are iluzia ca este singurul proces de pe masina, si are acces exclusiv la serviciile sistemului de operare. Ori de cate ori un proces face un apel de sistem, hardware-ul schimba modul de privilegiul de la Modul Utilizator in Modul Kernel, iar procesul incepe executarea unei proceduri de nucleu cu un scop strict limitat. In acest fel, sistemul de operare actioneaza in contextul executarii procesului, pentru a satisface cererea. Ori de cate ori cererea este pe deplin satisfacuta, procedura de kernel forteaza hardware-ul pentru a reveni la Modul Utilizator si procesul continua executarea ei de la instructiunea care urma in urma apelului de sistem.

### Arhitectura Kernel-ului

Cele mai multe kernel-uri Unix sunt monolitice: fiecare strat al kernel-ului este integrat in intregul program al kernel-ului si ruleaza in Modul Kernel, in numele procesului curent. In contrast, sistemele de operare microkernel cer un set foarte mic de functii de la kernel,in general incluzand, cateva primitive de sincronizare, un planificator simplu, si un mecanism de comunicare inter-procese. Mai multe procese de sistem care ruleaza peste microkernel implementeaza alte functii,cum ar fi repartitoare de memorie, drivere de dispozitiv, si manipulatoare de apeluri de sistem.

Desi cercetarea academica pe sistemele de operare este orientata spre microkernels, astfel de sisteme de operare sunt in general mai lente decat cele monolitice , pentru ca trecerea mesajului explicit intre diferitele straturi ale sistemului de operare are un cost . Cu toate acestea, sistemele de operare microkernel ar putea avea unele avantaje teoretice peste cele monolitice. Microkerne-urile forteaza programatorii de sistem sa adopte o abordare modulara, pentru ca fiecare strat din sistemul de operare este un program relativ independent, care trebuie sa interactioneze cu alte straturi prin interfete software bine definite. Mai mult decat atat, un sistem de operare microkernel existent poate fi usor adaptat pentru alte arhitecturi destul de usor, deoarece toate componentele dependente de hardware sunt, in general, incapsulate in codul microkernel. In cele din urma , sistemele de operare microkernel au tendinta de a face o utilizare mai buna a memoriei cu acces aleator ( RAM ) decat cele monolitice , deoarece procesele de sistem, care nu implementeaza functionalitatile necesare ar putea fi schimbate sau distruse.

Pentru a atinge multe din avantajele teoretice ale microkernelurilor fara a introduce sanctiuni de performanta, kernel-ul Linux ofera *module*. Un modul este un fisier obiect al carui cod poate fi legat de (si deconectat de la), kernel in timpul rularii. Codul obiectului, de obicei, consta dintr-un set de functii care pune in aplicare un sistem de fisiere, un driver de dispozitiv, sau alte caracteristici din stratul de sus al kernel-ului. Modulul, spre deosebire de straturile externe ale sistemelor de operare microkernel, nu se executa ca un proces specific. In schimb, este executat in modul Kernel, in numele procesului actual, la fel ca orice alta functie de nucleu legata static.

# Cerinte pentru creearea si utilizarea Kernelului

Acest capitol descrie programele de care avem nevoie pentru a configura un kernel, de al construi,si de a-l boota cu succes . Este o idee inteligenta consultarea dosarului cu documentatia pentru a verifica numarul specific al versiunii pe care ar trebui sa aveti de fiecare instrument descris in acest capitol. Acest capitol a fost bazat pe kernel-ul 2.6.18, si descrie versiunile de instrumente care lucreaza cu nucleul. Daca folositi un kernel, va rugam sa verificati daca aveti versiunile necesare specificate in acest dosar, sau lucrurile s-ar putea sa nu functioneze in mod corespunzator si poate fi foarte greu pentru a determina ce a mers prost.

## Unelte pentru creearea Kernel - ului

Cele mai multe distributii Linux ofera o optiune de instalare de pachete de hacking kernel. In cazul in care distributia aleasa ofera aceasta optiune, este mai usor de instalat aceasta optiune in loc de a incerca urmarirea tuturor programelor individuale care sunt necesare pentru aceasta sarcina.

Numai trei pachete care sunt necesare pentru a construi cu succes un nucleu: un compilator, un linker, si un utilitar *make*. Aceasta sectiune descrie continutul fiecarui pachet.

### Compilatorul

Kernel-ul Linux – ului este scris in limbajul de programare C, iar in unele locuri cu o cantitate mica de limbaj de asamblare. Pentru a construi kernel-ul, trebuie sa fie utilizat compilatorul *gcc C*. Cele mai multe distributii Linux au un pachet numit *gcc* care ar trebui sa fie instalat. Daca doriti sa descarcati compilatorul pentru construi kernelul, il puteti gasi la <http://gcc.gnu.org>.

Odata cu aparitia versiunii de nucleu 2.6.18, versiunea 3.2 a gcc-ului este cea mai veche pe care se poate construi in mod corespunzator un nucleu de lucru. Obtinerea celei mai recente versiuni de GCC nu este intotdeauna o idee buna. Unele dintre cele mai noi versiuni CCG nu construi kernel-ul in mod corespunzator, astfel incat in exceptia cazului in care doriti sa ajutati la depanarea de bug-uri de compilare, aceasta nu este recomandat sa incercati versiuni noi de compilatoare.

Pentru a determina ce versiune de *gcc* este instalata pe sistem trebuie apelata urmatoarea comanda:

***$ gcc –version***

### Linker

Compilatorul de C, nu face toata compilarea pe cont propriu. Este nevoie de un set suplimentar de instrumente cunoscute sub numele de *binutils* pentru a face conectarea si montarea de fisiere sursa. Pachetul *binutils* contine, de asemenea utilitare utile care pot manipula fisiere obiect intr-o multime de moduri utile, cum ar fi, pentru vizualizarea continutului unei biblioteci.

Pachetul suplimentar de instrumente *binutils* poate fi gasit intr-un pachet de distributie numit (nu surprinzator) binutils. Pentru descarcarea si instalarea pachetului, acesta poate fi gasit la <http://www.gnu.org/software/binutils>.

Odata cu aparitia versiunii de nucleu 2.6.18, versiunea de binutils, 2.12 este cea mai veche versiune, care poate lega cu succes kernelul. Pentru a determina ce versiune de binutils este instalata pe sistemul de operare, trebuie executata urmatoarea comanda:

***$ ld –v***

### Make

*Make* este un instrument care parcurge structura surselor kernel-ului pentru a determina care fisiere trebuiesc a fi compilate, iar apoi apeleaza compilatorul si alte uneltele de compilare pentru a face munca in construirea kernel-ului. Kernel-ul necesita versiunea GNU a utilitarului *make*, care poate fi gasita intr-un pachet numit *make*.

Acest utilitar poate fi gasit la adresa <http://www.gnu.org/software/make>.

De la versiunea de nucleu 2.6.18, versiunea 3.79.1 a utilitarului *make* este cea mai veche pe care se poate construi corect kernel-ul. Este recomandata instalarea ultimei versiuni stabile a utilitarului *make*, deoarece versiunile mai noi sunt mai rapide in procesarea fisierelor.

Pentru a determina versiunea utilitarului *make*  instalat pe sistem trebuie rulata urmatoarea comanda:

***$ make –version***

## Unelte pentru utilizarea Kernel –ului

In timp ce versiunea de kernel care se afla in executie nu afecteaza de obicei, orice aplicatie de utilizator, exista un numar mic de programe pentru care versiunea de kernel este importanta. Aceasta sectiune descrie o serie de instrumente care sunt, probabil, deja instalate pe sistemul Linux. Daca va actualizati nucleul la o versiune diferita de cea care a venit cu distributia de Linux, unele dintre aceste pachete ar putea avea nevoie, de asemenea, sa fie modernizate, pentru ca sistemul sa functioneze in mod corespunzator.

### Uil-Linux

Pachetul util-linux este o colectie de utilitare mici, care indeplinesc o gama larga de sarcini diferite. Cele mai multe dintre aceste utilitati se ocupa de montarea si de crearea de partitii de disc si manipularea ceasului hardware din sistem.

Ultima versiune a pachetului util-linux poate fi gasita accesand urmatoarea adresa, <http://www.kernel.org/pub/linux/utils/util-linux>.

Este recomandat sa fie instalata cea mai recenta versiune a acestui pachet, pentru ca sunt adaugate noi functionalitati in kernel. Bind Mounts este un exemplu de o optiune introdusa in kernel-urile mai noi, si este nevoie de o versiune mai noua de util-linux, in scopul de a functiona corect.

Pentru a determina versiunea pachetului *util-linux­* instalata pe sistem se ruleaza comanda:

***$ fdformat –version***

### Module-init-tools

Pentru a folosi modulele de kernel Linux este nevoie de pe pachetul ***module-init-tools***. Un modul de nucleu sau kernel este o bucata de cod incarcabila care pot fi adaugata sau eliminata din kernel in timp ce kernel-ul este pornit. Este util pentru a compila drivere pentru dispozitive ca module si apoi incarcarcate doar cele care corespund hardware-ului prezent in sistem. Toate distributiile Linux folosesc module, pentru a incarca doar driverele necesare si optiunile pentru sistemul bazat pe hardware-ul prezent, in loc de a fi nevoit sa construiasca toate driverele si optiunile posibile in kernel intr-o bucate mare. Modulele salveaza memorie prin inarcarea doar codului care este necesar pentru a controla corect masina.

Procesul de incarcare al modulului de kernel a suferit o schimbare radicala in versiunea de nucleu 2.6. Linker-ul pentru modul (codul care rezolva toate simbolurile si stie cum sa puna toate piesele impreuna in memorie) este construit in kernel, ceea ce face ca instrumentele de utilizatorului destul de mici. Distributiile mai vechi au un pachet numit *modutils* care nu functioneaza in mod corespunzator cu nucleul 2.6. Pachetul de module-init-tools necesar pentru ca kernel-ul 2.6 pentru sa functioneze corect cu module.

Pachetul *module-init-tools* poate fi gasit la adresa <http://www.kernel.org/pub/linux/utils> /org/pup/linux/utils/kernel/module-init-tools.

Se recomanda ca cea mai recenta versiune a acestui pachet sa fie instalata, ca noile functionalitati adaugate la kernel-ul sa poata fi utilizate de catre versiunile mai noi ale acestui pachet.

Pentru determinarea versiunii pachetului instalat pe sistem se foloseste comanda:

***$ depmod –V***

### Alte unelte

Exista cateva alte programe importante care sunt strans legate de versiunea de kernel. Aceste programe nu sunt de obicei necesare pentru ca kernel-ul sa functioneze corect, dar ele permit accesul la diferite tipuri de hardware si functii.

#### udev

**Udev** este un program care permite Linux-ului sa oferere un sistem persistent de numire al dispozitivelor in directorul */dev*. Acesta ofera, de asemenea, un */dev* dinamic, la fel ca cel oferit de vechiul, si acum eliminat, sistem de fisiere *devfs*. Aproape toate distributiile Linux utiliza *udev* pentru a gestiona directorul */dev*, de aceea este necesar pentru a porni in mod corespunzator masina.

Din pacate, *udev* se bazeaza pe structura */sys*, care este cunoscuta ca se schimba din timp in timp, cu noile versiuni de kernel. In trecut unele dintre aceste schimbari au fost cunoscute pentru faptul ca au stricat buna functionare a *udev*, astfel incat masina nu pornea in mod corespunzator.

Este recomandat sa fie utilizata versiunea de *udev*, care vine cu distributia de Linux, deoarece este strans legata de procesul de bootare specific fiecarei distributii. Pentru a accesa ultima versiune de *udev* aceasta poate fi gasita la adresa <http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev.html>.

Pentru a determina versiune de *udev* instalata pe sistem rulati urmatoarea comanda:

***$ udevinfo –V***

#### Unelte de process

Pachetul ***procps*** include instrumentele utilizate frecvent, *ps* si *top*, precum si multe alte instrumente pentru gestionarea si monitorizarea proceselor care ruleaza pe sistem.

Pentru a instala acest pachet puteti accesa pachetul la adresa http://procps.sourceforge.net.

Pentru a determina versiunea pachetului *procps* instalata rulati in terminal comanda:

***$ ps -- version***

#### Uneltele **PCMCIA**

Pentru a utiliza in mod corespunzator dispozitivele ***PCMCIA*** cu Linux, un program de ajutor adaugat de utilizator trebuie sa fie utilizat pentru a configura dispozitivele. Pentru versiuni mai vechi de kernel, acest program a fost numit ***pcmcia-cs***, dar care a fost inlocuit cu un sistem mult mai simplu, numit ***pcmciautils***. Pentru a utiliza dispozitivele ***PCMCIA***, trebuie instalat acest pachet pentru o functionare corecta.

Pachetul poate fi gasit la adresa ftp:// [ftp.kernel.org/pub/linux/utils/kernel/pcmcia](ftp://ftp.kernel.org/pub/linux/utils/kernel/pcmcia).

Este recomandat folosirea celei mai recente versiuni, in scopul de a profita de caracteristicile noi din subsistemul ***PCMCIA***, cum ar fi incarcarea automata de drivere atunci cand sunt gasite noi dispozitive.

Pentru a determina versiunea pachetului ***pcmciautils*** instalat pe sistem folosim comanda:

***$ pccardctl -V*** [3]

# Structurile de date ale Kernelului

Acest capitol prezinta mai multe structuri de date utilizate in codul kernel-ului de Linux.Ca oricare proiect software mare, kernel-ul de Linux ofera aceste structuri de date generice si primitive pentru a incuraja reutilizarea codului. Dezvoltatorii de kernel ar trebui sa utilizeze aceste structuri de date ori de cate ori este posibil. In urmatoarele sectiuni, vom acoperi cele mai utile dintre aceste structuri de date generice, care sunt urmatoarele:

* Liste legate (Linked lists)
* Cozi (Queues)
* Harti (Maps)
* Arbori binary (Binary trees)

## Liste inlantuite

*Lista inlantuita* este cea mai simpla si cea mai comuna structura de date in kernel-ul de Linux .O *lista inlantuita* este o structura de date care permite stocarea si manipularea unui numar variabil de elemente, numite nodurile listei. Spre deosebire de o matrice statica, elementele dintr-o lista legata sunt create si introduse dinamic in lista. Acest lucru permite gestionarea unui numar variabil de elemente necunoscute la momentul compilarii. Deoarece elementele sunt create la momente diferite, ele nu ocupa neaparat regiuni invecinate din memorie. Deci, elementele trebuie sa fie legate intre ele, astfel fiecare element din lista contine un pointer la urmatoarul element. Cand elemente sunt adaugate sau eliminate din lista, indicatorul sau pointerul la nodul urmator este pur si simplu ajustat.

### Liste inlantuite individual si de doua ori

Cea mai simpla structura de date, lista legata poate arata similar cu urmatoarea reprezentare:

*struct list\_element{*

*void \*data; // data efectiva*

*struct list\_element \*next; // pointer la urmatorul element*

*};*

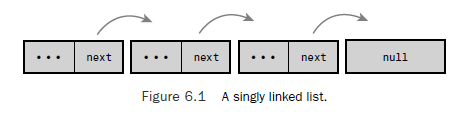


Figura 3.1 Lista simplu inlantuita

In unele liste *inlantuite*, fiecare element contine, de asemenea, un pointer la elementul precedent. Aceste liste sunt numite liste legate de doua ori, deoarece acestea sunt *inlantuite*atat inainte si inapoi. Listele legate, cum ar fi lista din figura 3.1, care nu au un pointer la elementul anterior sunt numite liste *inlantuite* individual.

O structura de date care reprezinta o lista intalntuita de doua ori ar arata similar cu acesta:

*struct list\_element{*

*void \*data; // data efectiva*

*struct list\_element \*next; //pointer la elementul urmator*

*struct list\_element \*prev; // pointer la elementul precedent*

*};*

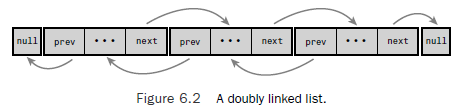


Figura 3.2 Lista dublu inlatuita

### Liste inlantuite circular

In mod normal, pentru ca ultimul element dintr-o lista intalntuita nu are nici un element care sa il urmeaze, este setat pentru a indica o valoare speciala, cum ar fi *NULL*, pentru a indica ca este ultimul element din lista. In unele liste intalntuite, ultimul element nu indica o valoare speciala. In schimb, indica inapoi la prima valoare a listei. Aceasta se numeste lista intalntuita circulara deoarece lista este ciclica. Listele intalntuite circular poat veni in ambele versiuni intalntuite de doua ori si individual. Intr-o lista circulara dublu intalntuita, indicatorul "anterioar" este primul nod de la ultimul nod. Figurile 3.3 si 3.4 sunt liste intalntuite circular individual si dublu circular.

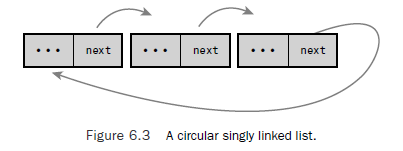


Figura 3.3 Lista inlantuita circular

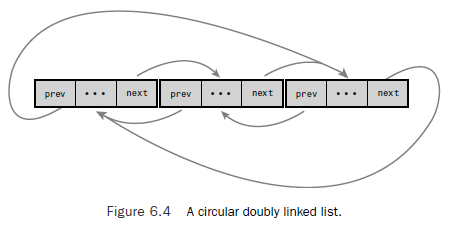


Figura 3.4 Lista circulara dublu inlantuita

Desi implementarea listei legate din kernel-ul Linux-ului este unica, acesta este in mod fundamental o lista circular de doua ori legata. Folosind acest tip de liste legate ofera cea mai mare flexibilitate.

### Parcurgerea unei liste inlantuita

Parcurgerea unei liste inlantuite este liniara. Vizitati un element, urmati indicatorul urmator, si vizitati si elementul urmator. Acest proces se repeta pana la terminarea listei. Aceasta este cea mai simpla metoda de a parcurgere a unei liste inlantuite. Listele inlantuita nu sunt foarte performante in cazurile de utilizare care implica accesul aleator la elementele listei . In schimb, se utilizeaza liste inlantuite, atunci cand iterarea peste toata lista este importanta si este necesara adaugarea si eliminarea dinamica de elemente.

In implementarea listelor inlantuite, primul element este adesea reprezentat de un pointer special numit head (cap), care permite accesul usor la inceputul listei. Intr-o lista inlantuite noncirculara, ultimul element este delimitat de urmatorul pointer care este NULL. Intr-o lista inlantuite circular, ultimul element este delimitat, deoarece aceasta indica head-ul. Parcurgand lista, prin urmare, apare parcurgere liniara la fiecare element de la primul la ultimul. Intr-o lista de doua ori inlantuita, parcurgerea poate aparea, de asemenea inapoi, liniar de la ultimul element la primul element. Desigur, fiind dat un element specific al listei, se poate itera lista in ambele sensuri pornind de la acel element, fara a mai fi nevoie de parcurgerea intregii liste.

## Cozi

Un model de programare comun in kernel-ul oricarui sistem de operare este producatorul si consumatorul. In acest model, un producator creeaza date,spre exemplu mesaje de eroare care trebuiesc sa fie citite sau pachete de rutare pentru a fi procesate, in timp ce un consumator,citeste procese, sau consuma date. De multe ori cel mai simplu mod de a pune in aplicare acest model este cu ajutorul unei cozi.Producatorul impinge date in coadasi consumatorul extrage datele introduse de consumator in coada.Consumatorule preia datele in ordinea in care a fost introduse. Primele date introduse in coada sunt primele extrase. Din acest motiv, cozile sunt de asemenea numite FIFO-uri(PIPI), prescurtarea de la primul intrat, primul iesit. A se vedea Figura 6.5 pentru un exemplu.

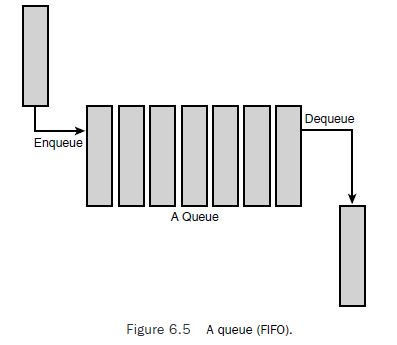


Figura 3.5 Coada

Impementarea cozii in kernel-ul de Linux se numeste *kfifo* si este implementata in *kernel/kfifo.c* si este declarata *<linux/kfifo.h>.*

### Kfifo

Kfifo Linux functioneaza ca cele mai multe coada, oferind doua operatiuni principale: punearea in coada (numit *in*) si scoatearea (*out*) Obiectul kfifo mentine doua offseturi in coada de asteptare: un offset *in* si un offset *out*. Offset-ul de *in* este locatia din coada la care se va face urmatoarea extragere. The de offset este locul in coada de asteptare din care dequeue urmatoare va occur. Offset-ul de *out* este intodeauna mai mic sau egal cu offsetul de *in*. Nu ar face sens ca offset-ul sa fie mai mare deoarece nu ar putea extrage date care nu au fost introduse.

Operatia de introducere de date in coada *in* copiaza datele in lista incepand cu offsetul *in*. Cand operatia a fost incheiata, acest offset este incrementat cu numarul de date introduse in lista. Operatia de extragere (out) copiaza datele din lista incepand cu offset-ul *out.* Cand operatia a fost incheiata, offsetul este incrementat cu numarul de date extrase. Cand offset-ul *out* este egal cu offset-ul *in* atunci lista este goala si nu mai pot fi extrase date pana cand nu sunt introduse noi date. Cand offset-ul *in* este egal cu lungimea cozii, nu se mai pot efectua introduceri de date in lista pana cand lista nu este golita.

## Harti (maps)

O harta, cunoscuta si sub numele de *vector asociativ*, este o colectie de chei unice, unde, fiecarei chei ii este asociata o anumita valoare. Relatia dintre o cheie si valoarea acesteia se numeste mapare. Hartile suporta cel putin 3 operatii fundamentale:

* Add (cheie,valoare)
* Sterge(cheie)
* Valoare = cauta(cheie)

Chiar daca un tabel hash-uit este un tip de map, nu toate map-rile sunt implementate prin intermediul hash-uiri. In locul unui tabel hash-uit, map-urile pot folosi o cautare binara de tip arbore pentru asi stoca datele. Un arbore cu cautare binara permite pastrarea ordinii, permitandu-I utilizatorului sa parcurga colectia ordonata in mod eficient.

Desi termenul general pentru toate colectiile este cartografierea unei cheie la o valoare, numele de harti de multe ori se refera, in mod special, la un vector asociativ care implementeazaun arbore binar de cautare, spre deosebire de un tabel hash.

Kernel-ul de Linux ofera o structura de harta de date simpla si eficienta, dar aceasta nu este o harta de uz general. In schimb, acesta este conceputa pentru un caz de utilizare specific: cartografierea unui numar unic de identificare (UID) la un pointer. In plus fata de furnizarea celor trei operatii principale, implementarea Linux-ului furnizeaza o noua operatie numita *allocate* care este pusa peste operatia de *add*. Aceasta operatie *allocate* nu doar adauga o pereche UID / valoare la harta, dar, de asemenea, genereaza UID-ul.

## Arbori binari

Un arbore este o structura de date care furnizeaza ca structura ierarhica a datelor asemanatoare cu cea a arborilor. Matematic, aceasta este un graf orientat aciclic, conectat, in care fiecare varf (numit nod) are zero sau mai multe elemente de iesire si zero sau un singur element de intrare. Un arbore binar este un arbore in care nodurile au cel mult doua margini de iesire, care este, un copac in care nodurile au zero, unu, sau doi copii. Vezi Figura 6.6 pentru un arbore binar.

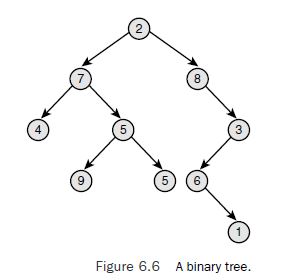


Figure 3.6 Arbori binari

### Arbori binari de cautare

Un arbore binar de cautare este un arbore binar cu o ordonare specifica a elementelor impusa pe nodurile sale. Ordonarea este definita prin urmatoarele reguli:

* Sub-arborele din stanga radacinii contine numai noduri cu valori mai mici de radacina.
* Sub-arborele din drepta radacini contine numai noduri cu valori mai mari decat radacina.
* Toate subarbori sunt, de asemenea, arbori cu cautare binara.

Un arbore binar de cautare este un arbore binar in care toate nodurile sunt ordonate astfel incat copiii din stanga au o valoare mai mica ca parintele si copii din dreapta au o valoare mai mare ca cea a parintelui. Prin urmare, cautarea pentru un nod dat sau traversarea arborelui sunt actiuni eficiente si rapide . Vezi Figura 6.7 pentru un arbore binar de cautare.

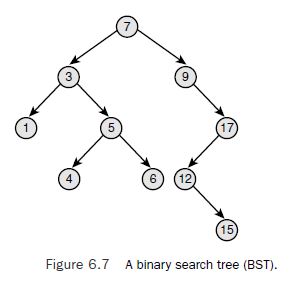


Figura 3.7 Arbori binari de cautare

### Echilibrarea arborilor binari de cautare

Adancimea unui nod se masoara prin cat de multe noduri parinte sunt de la radacina. Nodurile de la baza arborelui, cei care nu au copii, sunt numite *frunze*. Inaltimea arborelui este adancimea la cel mai la cel mai indepartat nod de radacina. Un arbore binar de cautare echilibrat este un arbore in care adancimea tuturor frunzelor difera cu cel mult unu (a se vedea figura 6.8). Un arbore binar de cautare cu auto-echilibrare este un arbore binar de cautare care incearca, sa ramana (semi) echilibrat.[4]

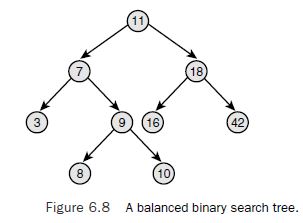


Figura 3.8 Arbori binari de cautare echilibrat

# Sincronizarea kernel-ului

Kernel-ului poate fi privit ca un server care raspunde cererilor, aceste cereri pot veni fie de la un proces care ruleaza pe un procesor sau de la un dispozitiv extern care emite o cerere de intrerupere. Facem aceasta analogie pentru a sublinia ca anumite parti ale kernel-ului nu sunt exploatate in serie, ci intr-un mod intercalat. Astfel, ei pot da nastere la conditii de intreceri, care trebuie sa fie controlate prin tehnici de sincronizare corespunzatoare.

## Contexte de executie si locking

Relativ la modul de executie in kernel, distingem doua contexte: context **proces** si context **intrerupere**. Ne aflam in context proces atunci cand rulam cod ca urmare a unui apel de sistem sau cand rulam in contextul unui kernel thread. Atunci cand rulam in rutina de tratare a unei intreruperi sau a unei actiuni amanabile, rulam in context intrerupere.

Unele dintre apelurile din API-ul kernel pot duce la blocarea procesului curent. Exemple comune sunt folosirea unui semafor sau asteptarea unei conditii. In acest caz, procesul este trecut in starea WAITING si alt proces este rulat. O situatie interesanta apare in momentul in care o functie ce poate duce la suspendarea procesului curent este chemata din context intrerupere. In acest caz, nu exista un proces curent, si din aceasta cauza rezultatele sunt impredictibile. De cate ori sistemul de operare detecteaza aceasta conditie va genera o conditie de eroare care va duce la oprirea sistemului de operare.

Una dintre cele mai importante caracteristici ale programarii in kernel este paralelismul. Atat Linux cat si Windows suporta sisteme SMP, cu mai multe procesoare, dar si preemptivitate in kernel. Acest lucru face programarea kernel mai dificila, deoarece accesul la variabilele globale trebuie sincronizat, fie cu primitive de spinlock, fie cu primitive blocante. Desi este recomandat sa se foloseasca primitive blocante, acestea nu pot fi folosite in context intrerupere, asa ca singura solutie de locking in context intrerupere sunt spinlock-urile.

Spinlock-urile sunt folosite pentru realizarea excluderii mutuale. Atunci cand nu pot obtine accesul la regiunea critica nu suspenda procesul curent, ci folosesc mecanismul de busy-waiting (asteapta intr-un ciclu while eliberarea lock-ului). Codul care se executa in regiunea critica protejata de un spinlock nu are voie sa suspende procesul curent (trebuie sa respecte conditiile executiei in context intrerupere). Mai mult, nu se va ceda procesorul decat pentru servirea intreruperilor. Datorita mecanismului folosit, este important ca un spinlock sa fie detinut cat mai putin timp posibil.

## Preemptivitate

Atat Linux cat si Windows folosesc nuclee preemptive. Nu trebuie confundata notiunea de multitasking preemptiv cu notiunea de kernel preemptiv. Notiunea de multitasking preemptiv se refera la faptul ca sistemul de operare intrerupe rularea unui proces in mod fortat, atunci cand acestuia i-a expirat cuanta de timp si ruleaza in user-space, pentru a rula alt proces. Un kernel este preemptiv daca un proces ce ruleaza in kernel-mode (ca urmare a unui apel de sistem) poate fi intrerupt pentru a rula un alt proces.

Datorita preemptivitatii, atunci cand partajam resurse intre doua portiuni de cod ce pot rula din contexte proces diferite, trebuie sa ne protejam cu primitive de sincronizare, chiar si in cazul uni-procesor.

## Cand sincronizarea este necesara si cand nu este necesara

O conditie pentru aparitia unei competitii poate aparea in cazul in care rezultatul unui calcul depinde de modul in care sunt imbricate doua sau mai multe cai intercalate de control ale kernel-ului. O regiune critica este o sectiune de cod care trebuie sa fie complet executata de catre calea de control a kernel-ului, inainte de o alta cale de control al kernel-ului sa poata intra.

Intercalarea cai de control a kernel-ului complica viata dezvoltatorilor de kernel: ei trebuie sa se aiba o atentie deosebita, pentru a identifica zonele critice in manipularea (handler) exceptiilor, handler-le de intrerupere, si fire de executie ale kernel-ului. O data ce o regiune a fost identificata trebuie sa fie protejata corespunzator pentru a se asigura ca in orice moment cel mult o cale de control a kernel-ului este in interiorul acestei regiuni.

De exemplu sa presupunem, ca doua handlere de intrerupere diferite trebuie sa acceseze aceeasi structura de date care contine mai multe variabile interconectate, un buffer si un numar intreg care indica lungimea sa. Toate declaratiile care afecteaza structura de date trebuie sa fie puse intr-o singura regiune critica. In cazul in care sistemul include un singur procesor, regiunea critica este implementata prin dezactivarea intreruperilor in timp ce se acceseaza structura de date partajata.

Pe de alta parte, in cazul in care aceeasi structura de date este accesata numai de catre serviciile de rutina ale apelurilor de sistem, si in cazul in care sistemul include un singur procesor, regiunea critica poate fi implementata destul de simplu prin dezactivarea preemptiunii kernel-ului in timp ce se acceseaza structura de date partajata.

Lucrurile sunt mult mai complicate in sistemele multiprocesor. Mai multe procesoare ar putea executa cod kernel in acelasi timp, astfel incat dezvoltatorii de kernel nu poat presupune ca o structura de date pot fi accesat in conditii de siguranta doar pentru ca preemptiunea de kernel este dezactivata si ca structura de date nu este adresata de o intrerupere sau exceptie.

Sincronizarea cailor de control a kernelului poate fi explicata pe scurt prin urmatoarele afirmati:

* Toate handler-ele de intrerupe recunosc intrerupere pe PIC sidezactiveaza linia de IRQ. Aparitia ulterioara a aceeleasi intreruperi nu poate avea loc pana cand nu se termina handler-ul.
* Handler-ele de intrerupere, si softirqs sunt atat nonpreemptable si neblocabile, astfel incat acestea nu pot fi suspendate pentru un interval de timp lung. In cel mai rau caz, executarea lor va fi usor intarziata, deoarece alte intreruperi apar in timpul executarii.
* O cale de control al kernel-ului care executa o tratare de intrerupere nu poate fi intrerupta de o cale de control a kernel-ului care executa o functie de amanare sau un serviciu de a unui apel de sistem.

Fiecare dintre optiunile de proiectare de mai sus poate fi privita ca o constrangere, care poate fi exploatata pentru a codifica unele functii de kernel mai usor. Iata cateva exemple de posibile simplificari:

* Handlerele de intrerupere si tasklets trebuie sa nu fie codate ca functii de reintrare.
* Variabile pe CPU accesate de doar softirqs si tasklets nu necesita sincronizare.
* O structura de date accesate de catre un singur fel de tasklet nu are nevoie de sincronizare.[1]

# Concluzii

  Kernel-ul scris aprope in intregime in limbajul de programare C si cu citeva linii de cod scrise in [limbajul de asamblare](http://ro.wikipedia.org/w/index.php?title=Limbajul_de_asamblare&action=edit&redlink=1).

Dezvoltat sub licenta [GNU GPL](http://ro.wikipedia.org/wiki/GPL_(licen%C8%9Ba)) (GNU General Public License), [codul sursa](http://ro.wikipedia.org/wiki/Cod_surs%C4%83) al nucleului Linux este [software liber](http://ro.wikipedia.org/wiki/Software_liber). Acest lucru constituie probabil cel mai mare avantaj al Linuxului, posibilitatea de a putea fi imbunatatit de orice dezvoltator care doreste sa isi aduca aportul la acest proiect.

Nucleul este cel mai bine cunoscut ca principala componenta a sistemului de operare [GNU/Linux](http://ro.wikipedia.org/wiki/GNU/Linux). Distributiile de [software](http://ro.wikipedia.org/wiki/Software) bazate pe nucleul Linux se numesc [Distributii GNU/Linux](http://ro.wikipedia.org/wiki/Distribu%C8%9Bii_GNU/Linux).

Kernel-ul este inima Linux-ului. Asta inseamna ca nu exista Linux fara Kernel. Kernel este mediatorul intre programe si hardware, locul unde sunt depozitate driverele. Kernel-ul ofera suport pentru filtrarea pachetelor ce trec prin retea si gestioneaza procesele ce ruleaza in memorie.

Kernel-ul Linux-ului este modular. Asta inseamna ca unele drivere pot fi compilate sub forma de module. Avantajul obtinut astfel consta in faptul ca driverul va sta in memorie numai cand este necesar. Asta inseamna o functionare mai rapida a sistemului si o boot-are mai rapida.

Modularitatea si capacitatea de modificare a kernel-ului in functie de arhitectura proprie a calculatorului personal reprezinta un alt avantaj important al kernel-ului de Linux. Asadar, in functie de tipul procesorului, al placii grafice sau al altor componente noi ne putem compila propriul kernel specific configuratiei care o detinem, optimizand sistemul.

# Bibliografie

[1] Understanding the Linux Kernel , Daniel P.Bovet and Marco Cesati

[2] <http://en.wikipedia.org/wiki/Linux_kernel>

[3]Linux Kernel in a nutshell, Greg Kroab-Hartman

[4]Linux Kernel Development, Third Edition, Robert Love