Universitatea Politehnica Bucuresti

Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

Sisteme de Operare Avansate

Embedded Linux

Masterand: **Cocoru Vlad (**IISC)

2010

Cuprins

1. Introducere

Acest sistem de operare, derivat din distributiile de Linux larg raspandite, permite folosirea sistemului de operare (mare) Linux pe dispozitive embedded cum ar fi telefoanele mobile, asistentii digitali personali (PDAurile), playerele media[1], set-up boxurilor[2], precum si a altor dispozitive raspandite in electronica destinata largului consum, ca echipamentele de retea, dispozitive de automatizare, de navigatie sau intrumente medicale.

Spre deosebire de versiunile Linux destinate desktopurilor si serverelor, versiunile embedded sunt gandite pentru dispozitive cu resurse relativ limitate. Din considerente legate de costuri sau dimensiuni, aceste dispozitive embedded au in general mult mai putin RAM si spatiu de stocare secundar disponibil comparativ cu un sistem desktop, si foarte probabil folosesc memorie de tip flash in locul hard driskurilor.

Intrucat dispozitivele embedded deservesc sarcini mult mai specifice decat sistemele desktop, dezvoltatorii de obicei isi optimizeaza distributia embedded Linux pentru solutia particulara hardware pe baza careia dezvolta. Aceata optimizare inseamna de obicei reducerea numarului de drivere pentru dispozitive si a aplicatiilor software, precum si modificarea kernelului Linux pentru a-l face un sistem de operare real-time.

Ca dispozitive populare ce folosesc embedded Linux se numara telefoane mobile Motorola (bazate pe varianta MontaVista Linux [3]) sau Nokia (unele dispozitive din seria N), navigatoarele GPS TomTom, routere Cisco (seria Linksys) sau switchuri Cisco (seriile MDS si Nexus), acestea reprezentand numai cateva exemple populare, lista totala a lor fiind foarte lunga si in continua crestere.

Ca distributii cunoscute (si de utilizare mai larga) se pot aminti:

- **Qtopia**[4] (dezvoltata de Qt Software, subsidiara Nokia),
- **µClinux**[5] (dezvoltat pentru microcontrollere, preia kernelul obisnuit de Linux si elimina unitatea de management a memoriei),
- **Moblin**[6] (prescurtare de la „mobile Linux”, este un OS open source pentru dispozitive de internet mobile, cum ar fi *netbook*-uri sau *smart phone*-uri)

Exista, de asemenea, mai multe variante de embedded Linux, unele ce au fost create pentru dispozitive ce se fabrica pe o nisa, dar exista si cele create pentru un singur dispozitiv (cele foarte specializate, ce se regasesc doar la implementarea unui singur produs)

2. Nivelele Linux

Avand in vedere cele de mai sus, fiecare derivat Linux se bazeaza pe un kernel Linux general, pe care il voi caracteriza scurt in cele ce urmeaza.

Sistemele UNIX (deci, si Linux) prezinta o structura ierarhizata pe mai multe nivele (3):

A. **Nivelul hardware** – nivelul format din componentele harware CPU + Memorie (RAM) + Stocare pe suport de nivel secundar (HDD/Flash), Retea
B. **Nivelul sistem** – reprezentat de kernelul sistemului, are rolul de a oferi o interfata intre aplicatii si partea hardware, astfel incat aplicatiile sa fie protabile. Kernelul foloseste facilitatile multitasking ale procesoarelor i386 in modul protejat. Acest nucleu contine trei componente principale:
   - ✓ Sistemul de gestionare a proceselor (process manager)
   - ✓ Sistemul de gestionare a fisierelor (file system manager)
   - ✓ Componenta de comunicatie in retea (IP + TCP/UDP)
C. **Nivelul user** – nivelul in care se regasesc limbajele de comanda (shellurile) si diversele programe utilitare si aplicatiile utilizator.

Fiecare nivel se bazeaza pe serviciile/resursele oferite de nivelul imediat inferior.

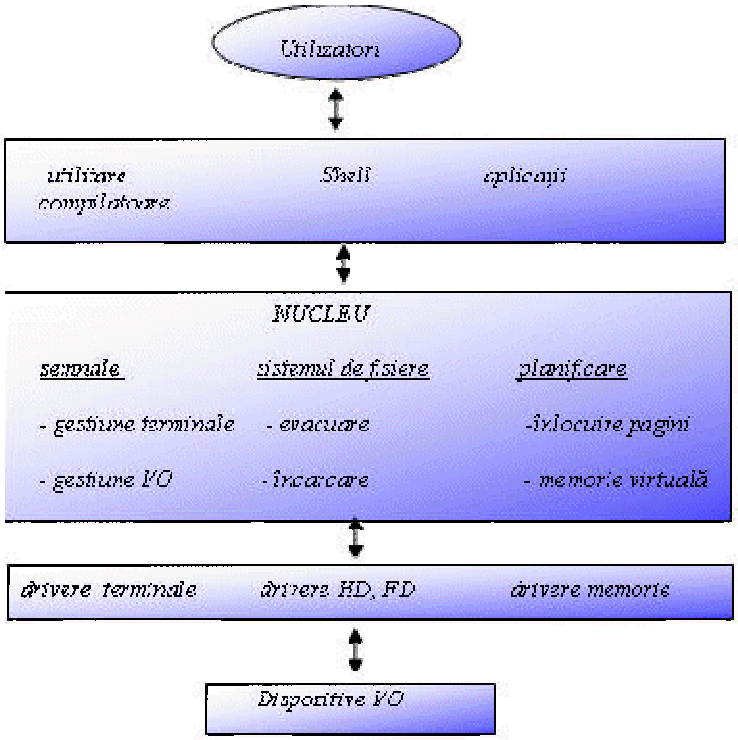

Fig. 1 – Structura sistemului UNIX

Din figura 1 se poate observa ca nucleul (kernelul) este intermediarul intre interfata furnizata de apelurile de sistem si echipamentele fizice.

Nucleul realizeaza gesiunea **fisierelor** si a **memoriei**, planificarea unitatii centrale intre procese (**gestiunea proceselor**), precum si **gestiunea operatiilor de I/O**. Aceste 4 operatii de gesiune vor fi detaliata in capitolele urmatoare.

3. Sistemul de gestiune a memoriei

Exista versiuni de linux ce nu au deloc gestiune a memoriei (cum ar fi µClinux), in care nu exista nicio diferentiere intre „user space" si „kernel space", si in care toate aplicatiile ruleaza la nivelul de privilegii 0, in care tot codul este executat in acelasi spatiu de memorie si de aceea nu este necesar un sistem de virtualizare a memoriei.

Exista, de asemenea, si versiuni de Linux ce suporta managementul memoriei, de la suportul clasic al Linuxului (care se indeparteaza de tema prezenta si nu va fi dezbatut in totalitate) pana la variante care nu permit existenta memoriei Swap. In aceste categorii, exista diferentiere intre „user space" si „kernel space" si exista de asemenea si niveluri de privilegii.

In Linux exista un sistem, numit kernel Out of Memory (OOM) ce este invocat atunci cand memoria RAM disponibila incepe sa se umple si devine insuficienta pentru cerintele sistemului. Daca in sistemele desktop/server acest scenariu este foarte rar intalnit, in dispozitivele embedded, care au memorie limitata, scenariul este mult mai des intalnit. Este adevarat ca in sistemele embedded nu sunt cereri de alocare a unor dimensiuni mari de memorie, dar si cererile mici si repetate pot conduce la declansarea sistemului OOM.

In functie de predictibilitatea comportamentului dispozitivului pe care se instaleaza embedded Linux, memoria ar putea fi dimensionata in asa fel incat nici in cazul cel mai defavorabil sa nu se umple (cazul ideal). Totusi, mult mai des se intalnesc situatii in care aceasta abordare nu este posibila.

Comportamentul pe care utilizatorul il percepe atunci cand sistemul se indreapta spre umplerea completa a memoriei este acela ca aplicatiile incep sa nu mai raspunda din cauza faptului ca sistemul incetineste semnificativ. Ideea gestiunii memoriei este tocmai prevenirea acestei stari (de incetinire pronuntata a sistemului) intrucat produce un mare discomfort utilizatorilor.

In momentul in care se ajunge la depasirea spatiului de memorie disponibil si se doreste incarcarea unei noi aplicatii, singura solutie este sa se inchida din aplicatiile ce ruleaza in mod

curent. Problema este ca procesul de alegere a aplicatiei care sa fie inchise poate conduce la rezultate nedorite (cum ar fi pierderea unor date pe care utilizatorul nu a apucat sa le salveze).

Din aceste motive, se doreste implementarea unor mecanisme care sa incerce invocarea acestui procedeu electiv.

Cel mai des intalnit algoritm este acela prin care pur si simplu se refuza alocarea de memorie aplicatiilor cand se trece de un prag de ocupare configurabil. Inaintea acestui refuz, se transmit mesaje dinspre kernel catre aplicatiile active cu somatia de a elibera spatiu (daca este posibil). Unele aplicatii isi pastreaza rezerve de memorie, si pot elibera o parte din aceasta la cerere (urmand ca la nevoie s-o ceara iarasi). Daca acest mecanism nu da rezultatele dorite (nu elibereaza suficient spatiu astfel incat sa incapa noua cerere sub pragul de semnalizare), se semnalizeaza utilizatorului nevoia de a elibera memorie (prin inchiderea aplicatiilor pe care acesta le considera nefolositoare la acel moment).

Exista si alti algoritmi (specializati pe un numar restrans de categorii de produse sau situatii intalnite), care implementeaza limitarile mai mult sau mai putin flexibil: pot sa nu permita un numar prea mare de aplicatii active (si sa inceapa sa le inchida pe cele mai vechi, cu sau fara confirmare din partea utilizatorului), care sa termine procesele in starile inactive (stopped) sau care sa aiba un numar de procese de care nu se atinge (un nucleu) si din restul sa aleaga prin procese elective.

4. Sistemul de gestiune a operatiilor I/O

Modul de gestionare a perifericelor cuprinde toate aspectele operatiilor de introducere si extragere a informatiei, pregatirea operatiei, lansarea cererilor de transfer de informatie, controlul transferului propriu zis, tratarea erorilor.

Principalele functii pe care trebuie sa le genereze un SO in scopul comunicarii cu perifericele sunt urmatoarele:

- Generarea comenzilor catre dispozitivele pericerice
- Tratarea intreruperilor specifice de I/O
- Tratarea eventualelor erori de I/O
- Furnizarea unei interfete utilizator cat mai standardizata si mai flexibila

Generarea comenzilor catre dispozitivele periferice se face de la nivelul kernel. In general, pe nivelul utilizator apare o cerere pentru accesul unui dispozitiv I/O. Se trece din nivelul user in nivelul kernel si automat se comanda perifericul (cu setarile impuse de utilizator).

Intreruperile specifice operatiilor de I/O sunt utile pentru managementul proceselor: cand un proces are nevoie de date de la un periferic, kernelul da comanda de acces si procesul este trecut in starea de asteptare (wait). Cand apare intreruperea ce semnalizeaza finalizarea procesului de transfer, procesul este readus in coada si urmeaza a-i fi permisa rularea (starea running).

Erorile de transfer pot fi de asemenea tratate de sistemul de operare, caz in care retransmiterile necesare sunt cerute de catre sistemul de operare fara interventia utilizatorului.

Sistemul Linux apeleaza perifericele intr-un mod uniform, si anume prin asocierea fiecarui dispozitiv a unui fisier, dispozitivele fiind pe urma apelate prin intermediul numelui fisierului. Aceasta asigura o interfatare standardizata si flexibila.

In versiunile embedded Linux uzuale, majoritatea driverelor din kernel pentru dispozitive sunt eliminate, pastrandu-se doar acelea care vor fi folosite de dispozitivul creat (in general numarul lor este mic). De asemenea, tratarea erorilor se poate face la nivelul kernel sau se poate transfera aceasta responsabilitate nivelului aplicatie (aplicatia va trebui construita astfel incat sa solicite retransferul in cazul in care se depaseste pragul de acceptare a erorii unei transmisiuni).

5. Sistemul de gestiune a fisierelor

Sistemul de gestiune a fisierelor (filesystem) este metoda prin care se stocheaza si organizeaza fisierele si datele pe care acestea le contin, cu scopul de a avea acces rapid la informatie si de a putea cauta prin ele.

Pentru Linux exista multe sisteme de gestiune a fisierelor, dar de obicei se alege intre cateva mai des intalnite, cum ar fi familia ext* (ext2, ext3 sau ext4), XFS, JFS, btrfs etc.

In sistemele embedded Linux, foarte rar exista un hard disk. De aceea, in majoritatea situatiilor se pune problema cum se poate crea un sistem de fisiere care sa nu se bazeze pe un hard disk. Exista mai multe tipuri de memorii Flash create special pentru stocarea non-volatila a datelor, cum ar fi memoriile flash NAND (negative AND – simuleaza comportamentul unei porti NAND la nivelul unui grup de tranzistori) sau NOR (negative OR) – simuleaza comportamentul unei porti NOR la nivelul unui grup de tranzistori) si dispozitivele „disk-on-chip”.

Memoriile NAND sunt mai ieftine, dar pot prezenta cateva erori hardware (se folosesc in general la dispozitive USB – memory stickuri – si memory carduri).

Memoriile NOR sunt mult mai rar predispuse la erori, de aceea se folosesc drept suportul de stocare a BIOSului in sistemele embedded.

Cum memoria Flash NOR este folosita de obicei pentru stocarea codului (mai este numita curent si „memorie de cod"), implementarea unui sistem de gestiune a fisierelor in aceasta memorie Flash NOR estesolutia cea mai eficienta din punct de vedere al costului si cea cu numarul minim de modificari pe placa de dezvoltare.

Funcțiile generale ale *sistemului de gestiune a fişierelor* sunt:

- ✓ *operatii cu fisiere (creare, citire, scriere, copiere, stergere, concatenare,etc.)*;
- ✓ *alocare de spatiu dinamic pentru fisiere pe suportul de stocare (flash memory si/sau HDD)*;
- ✓ *accesul la fisiere*;
- ✓ *administrarea spațiului liber pe suportul de stocare*;
- ✓ *schimbarea structurii sistemului de fişiere*.

Exista cateva probleme de adaptare de adaptare a distributiilor Linux clasice pentru sistemele embedded.

Prima ar fi ca sistemele de fisiere conventionale, cum ar fi clasica familie ext*, nu se pot folosi eficient pe memorii de tip Flash deoarece dimensiunea blocului unei memorii Flash este relativ mare (64K – 256K), comparativ cu dimensiunea de 4K tipica a sistemului ext2 sau 8K pentru sistemele ext3 si ext4. De asemenea, memoriile de tip Flash NOR au un numar limitat de stergeri per bloc, de obicei de ordinul a 100.000 de astfel de stergeri. Mai mult, costul memoriei Flash NOR este mare, de cateva ori mai scump decat memoria RAM. De aceea, se opteaza pentru un sistem de fisiere cu compresie.

O alta problema des dezbatuta des cand vine vorba de embedded Linux este folosirea principiului "executa la origine" (XIP – execute in place). Acest mod de operatie prevede ca procesorul sa mapeze pagini din memoria necesara unei aplicatii direct in memoria Flash, la adresa la care aceasta aplicatie se regaseste la nivel de cod, fara a mai incarca paginile in memoria RAM in prealabil. Acest procedeu permite reducerea memoriei RAM necesara sistemului. Problema cu acest procedeu este ca nu suporta compresia si este foarte complicat sa se faca un sistem de fisiere care sa permita scrierea.

Exista mai multe tehnologii Linux ce conlucreaza pentru a implementa sistemul de gestiune a fisierelor pentru un OS embedded Linux. Figura de mai jos ilustreaza legatura intre componentele standard din aceasta structura.

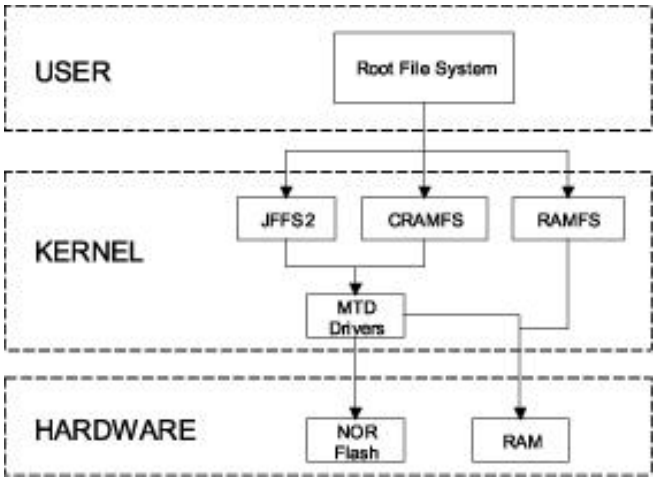

Fig. 2 – componentele sistemului de gestiune a fisierelor in embedded Linux (bazat pe memorie Flash)

5.1. initrd

In primele implementari ale embedded Linux, mecanismul initrd (initial disk ram) era des folosit pentru a memora o imagine comprimata a sistemului de fisiere in Flash. Acest mecanism a fost initial dezvoltat pentru ca un sistem Linux mic sa incapa pe un floppy disk de pe care sa se poata instala un Linux cu toate capabilitatile pe HDD. Secventa care se ruleaza la bootare intr-un sistem cu initrd este urmatoarea:

- Bootloaderul copiaza kernelul Linux comprimat si imaginea initrd din memoria Flash in RAM si trece la lucrul asupra kernelului
- Kernelul se decomprima in locatia corecta si incepe secventa de initializare
- Kernelul decomprima imaginea initrd in RAM si incepe sa foloseasca driverele ramdisk.

Comparativ cu alte abordari, aceasta idee are cateva dezavantaje. Prima este ca dimensiunea imaginii initrd este fixa. Nu foloseste eficient memoria RAM daca nu o umple (ce ramane liber nu se poate folosi) si dimensiunea nu se poate creste atunci cand este nevoie de spatiu de stocare suplimentar. In al doilea rand, modificarile facute asupra ei sunt pierdute la restart.

5.2. cramfs

Acesta este un sistem de fisiere comprimat, read-only, dezvoltat initial de Linus Torvalds (dezvoltatorul primului OS Linux) si inclus in recentele kerneluri de embedded Linux. In acest sistem de fisere, fiecare pagina este comprimata individual, permitand acces aleator la pagini.

O imagine cramfs este de obicei plasata in Flash, dar poate fi de asemenea plasata intr-un alt sistem de fisiere si sa se incarce cand este necesar. Acest sistem este folositor prin eficienta sa, si prin faptul ca uneor este necesar sa existe sisteme de fisiere ce sa suporte doar operatii de citire pe anumite partitii, pentru a evita coruperea fisierelor si pentru a imbunatati stabilitatea sistemului

O imagine cramfs se construieste cu utilitarul mkcramfs, ce creaza o imagine din continutul unui director specificat ca parametru de intrare. Acest utilitar se poate gasi in scripts/cramfs din arborele de resurse Linux.

5.3 ramfs

Acest sistem de fisiere pastreaza toate fisierele in RAM si este deseori folosit de sistemele ce ruleaza embedded Linux pentru a pastra datele temporare sau datele care se schimba des. Avantajul major al ramfs este ca poate creste sau se poate diminua pentru a se adapta fisierelor pe care le contine, spre deosebire de ramdisk (initrd) care are o dimensiune fixa. Acest sistem de fisiere a fost creat de asemenea de Linus Torvals si este inclus in distributiile Linux mai noi.

5.4. jffs2

Acest sistem de fisiere permite atat scrierea, cat si citirea si se bazeaza pe compresie. Este specific memoriilor Flash, si mai putin memoriilor RAM. jffs2 dezvolta jffs, adaugand capabilitatile de compresie.

Modul de operare al jffs2 este dinamic si functioneaza excelent pe memorii Flash. Sistemul de fisiere jffs2 este pur si simplu o lista de noduri (sau intrari intr-un fisier log) care contin informatii despre un fisier. Fiecare nod poate contine date ce pot fi adaugate intr-un fisier sau date ce vor fi sterse dintr-un fisier. Cand se monteaza un sistem de fisiere jffs2, intreaga lista de noduri este scanata pentru a determina cum se creaza fisierele. Nodurile sunt scrise in memoria

Flash secvential, pornind cu primul bloc. Daca scrieri aditionale sunt necesare, blocurile sunt scrise consecutiv pana se ajunge la limita Flashului, dupa care se reincepe de la primul bloc.

Sistemul jffs2 include un sistem de optimizare a folosirii memoriei disponibile, in sensul ca atunci cand exista blocuri ce nu sunt doar partial umplute, sistemul de fisiere le combina astfel incat sa redistribuie continutul cu scopul de a oferi cat mai mult spatiu in memoria Flash.

Acest sistem asigura uzarea memoriei Flash uniform, intrucat scrie in ea in toate blocurile, eliminand situatiile in care o memorie Flash e inutilizabila pentru ca doar cateva blocuri, pozitionate la inceputul ei, au devenit inutilizabile pentru ca de acolo s-au inceput mereu operatiile de scriere.

5.5. Driverul MTD

Sistemul MTD (memory technology device) este un subsistem Linux ce creaza o interfata generica pentru dispozitivele de memorie specifice embedded Linux, cum ar fi memoria Flash sau memoria RAM. Acesta ofera capabilitati simple de scriere, citire si stergere in cadrul memoriilor fizice. Sistemul MTD este compatibil cu cele 3 tehnologii prezentate mai sus.

Driverul MTD asigura suport complet pentru memoriile Flash NOR. Latinea busului pentru memoria Flash si numarul de chipuri Flash necesare pentru a implementa latimea busului pot fi configurate de utilizator sau pot fi detectate automat. Driverul MTD suporta de asemenea mai multe partitii Flash (adica un set de dispozitive Flash).

6. Sistemul de gestiune a proceselor

In cadrul oricarui sistem UNIX pot rula mai multe procese in regim concurent, regasite sub numele de procese. Procesele pot fi programele de utilizator, precum si o serie de procese speciale. Aceste procese speciale ruleaza in fundal (adica nu interactioneaza cu utilizatorul), cu rolul de a asigura diverse servicii (cum ar fi tiparirea la imprimanta, bazele de date, server Web s.a.). Aceste procese poarta denumirea de *daemoni.*

In cadrul sistemelor embedded Linux, date fiind limitarile de memorie (RAM), este de asteptat ca numarul proceselor concurente sa fie semnificativ mai mic.

Un proces se afla la un moment dat intr-o anumita stare, dupa cum se va vedea mai jos. In mod normal, fiecare proces va fi programat sa ruleze o perioada foarte scurta de timp, dupa care este trecut intr-o coada de asteptare, se ia urmatorul proces, se lasa si acesta sa ruleze o perioada foarte scurta de timp dupa care trece in coada. Algoritmul se repeta pentru toate procesele active pe sistem.

Funcțiile generale ale sistemului de gestiune a proceselor sunt:

- ✓ trecerea proceselor prin diverse stari (creare, asteptare, executie, terminare);
- ✓ planificarea proceselor pentru a avea acces la procesor;
- ✓ comunicarea intre procese;
- ✓ sincronizarea proceselor.

6.1. Starile posibile ale unui proces

Starile posibile ale unui proces sunt urmatoarele:

- **pregatit de lucru** – starea preliminara executiei sale efective
- **rulare** (running) – starea in care procesul primeste o cuanta de timp pentru a fi executat in cadrul procesorului (stare notata cu **R**)
- **asteptare** (sleep) – in vederea capatarii unei cuante de timp procesor (stare notata cu **S**)
- **asteptare** (wait) – in vederea realizarii unei operatii de I/O (aceste operatii find considerate mari consumatoare de timp, procesul va fi pus in starea de asteptare pana la terminarea respectivei operatiuni) (stare notata cu **D**)
- **oprit temporar** (stopped) – in care procesul nu va fi programat temporar pentru executie (stare notata cu **T**).
- **terminare** (terminate) – sistemul pregatind eliminarea procesului din memorie, urmand ca acesta sa dispara complet
- **zombie** – stare in care un proces trece atunci cand procesul sau parinte nu i-a determinat corect incercarea executiei sau zona de memorie pe care a solicitat-o nu a putut fi eliberata, ocupand astfel inutil loc in coada de asteprare (stare notata cu **Z**)

Fiecare proces este identificat printr-un identificator de proces (PID – **P**rocess **Id**entifier), un numar intreg mai mare decat 1. In mod normal, procesele sunt interactive, adica comunica cu utilizatorul prin intermediul terminalului asociat (in cazul embedded Linux, modul de interactiune este de obicei text, rareori fiind disponibile si modurile grafice X Window) fie pe dispozitivul in cauza, fie printr-o conexiune la distanta. Se va numi acest tip de procese ca fiind in prim-plan (*foreground*).

O alta categorie de procese sunt acelea care nu interactioneaza cu utilizatorul, fiind vorba in general de daemonii mentionati mai sus. Se spune ca aceste procese ruleaza in fundal (*background*).

Filosofia UNIX privind modul de viata al proceselor este ca orice proces este nascut de un alt proces, denumit procesul parinte (identificatorul acestuia este denumit PPID – **P**arent **PID**). La momentul pornirii sistemului, se creaza un pseudo-proces avand PID=0, care lanseaza in executie procesul *init*, acesta avand PID=1. Acesta va lansa alte procese, care la randul lor vor crea altele, astfel incat orice proces ce ruleaza pe masina are stramosul de rand maxim pe init.

Fiecare proces detine un set de drepturi si proprietati, acestea mostenindu-se de la parinte la copil. Exista ambele scenarii posibile: moartea procesului parinte implica moartea procesului copil, sau cele 2 procese functioneaza independent, moartea parintelui neimplicand si moartea copilului). Daca un proces isi pierde parintele, atunci PPID-ul sau va fi automat considerat ca fiind egal cu 1 (parintele sau devine *init*).

Procesele reprezinta imaginea dinamica (incarcata in memorie) a unui program, iar acel program este in fapt un fisier executabil detinut de un utilizator. Astfel, si procesul va avea un proprietar si va avea apartenenta la un grup. Drepturile de acces ale procesului si controlul sau depinde asadar de drepturile pe care le are proprietarul. Utilizatorii obisnuiti isi pot controla doar propriile procese. Utilizatorul root poate controla activitatea tuturor proceselor de pe masina.

6.2. Planificarea proceselor

Timul de calcul pe un sistem Linux este alocat in „jiffies”[7],[8] (perioade de timp foarte scurte, reprezentand o cuanta din timpul alocat pe microprocesor). Pe majoritatea distributiilor Linux, acest jiffie reprezinta 1/100 dintr-o secunda. In sistemele orientate spre real-time (ruland embedded Linux), acesta reprezinta 1/1024 dintr-o secunda.

Exista 3 tipuri de planificare a proceselor:

- **normal** – uneori denumita si „alta” (other), aceasta politica este specifica programelor normale
- **FIFO** – o politica specifica sistemelor real-time, presupune ca primul proces pornit (first in) va fi si primul proces care termina executia (first out). Procesul iese din executie daca intra in una din starile S, D, T, terminare sau zombie.
- **RR** (round robin) – o politica specifica de asemenea sistemelor real-tim, in care fiecare proces primeste o anumita cuanta de timp in care isi poate desfasura activitatea dupa care trebuie sa se opreasca si sa predea controlul unui alt proces (din coada).

Procesele Linux au urmatoarele caracteristici:

- **Politica** – normala sau real-time. Procesele real-time au o prioritate mai mare decat procesele normale
- **Prioritate** – reprezinta prioritatea procesului, fiind un numar intreg intre -20 si 19. Valoarea -20 reprezinta prioritatea maxima, valoarea 19 reprezinta prioritatea minima.

6.3. Comunicarea interprocese

Comunicarea interprocese se face in 6 moduri:

- **Semnale** – trimise de alte procese sau de catre kernel unui anume proces pentru a indica diverse conditii
- **Tuneluri** (sau conducte, „pipes") – cai neetichetate create in shell folosind de obicei caracterul „|" pentru a ruta rezultatele unui proces drept input pentru alt proces.
- **Linii FIFO** – cai etichetate ce opereaza pe baza principiului First data In, First data Out.
- **Cozi de mesaje** – mecanisme prin care se permite unuia sau mai multor procese sa scrie mesaje care pot fi citite de unul sau mai multe procese
- **Semafoare** – numaratoare care sunt folosite pentru a controla accesul la resurse partajate (shareuite). Aceste semafoare se folosesc pe post de dispozitive de blocare a accesului, pentru a evita situatia in care mai mult de un proces are permisiunea de a folosi aceeasi resursa la acelasi moment de timp
- **Memorie partajata** (shareuita) – permiterea accesului mai multor procese la aceeasi zona de memorie

Cozile de mesaje, semafoarele si memoria partajata pot fi accesate numai daca procesul are permisiunile de acces necesare.

In sistemele embedded Linux se pot elimina unele stari ale proceselor (de exemplu starea de „oprit temporar" sau starea de „zombie" – cand un proces nu are disponibile resursele, acesta va fi terminat). De asemenea, ca planificare a proceselor, in general se opteaza pentru RR (sisteme de timp real), desi exista si situatii in care planificarea normala deserveste mai bine natura aplicatiei. In final, se pot elimina si anumite modalitati de comunicare inter procese (de cele mai multe ori folosindu-se memoria partajata).

Bibliografie

[1] http://en.wikipedia.org/wiki/Media_center
[2] http://en.wikipedia.org/wiki/Set-top_boxes
[3] http://en.wikipedia.org/wiki/MontaVista
[4] http://en.wikipedia.org/wiki/Qtopia
[5] http://en.wikipedia.org/wiki/MClinux
[6] http://en.wikipedia.org/wiki/Moblin_project
[7] http://en.wikipedia.org/wiki/Jiffie
[8] http://www.comptechdoc.org/os/linux/howlinuxworks/linux_hlprocess.html
[9] http://www.linuxjournal.com/article/4678
[10] http://www.scribd.com/doc/22057048/Introducere-in-Linux
[11] Stefan Stancescu – Software de sistem – Note de curs