Structura sistemelor de operare Windows şi Linux

1. **Structurile de bază ale fiecărui sistem de operare în parte: concepte generale, structura nucleului**

2. **Nivelul de abstractizare al hard-ului**

3. **Interpretorele de comenzi din Linux**

4. **Comparaţie interfeţe**

    4.1 GNOME

    4.2 KDE

    4.3 Interfaţa în mod text

5. **Biblioteca apelurilor de sistem pentru Linux**

6. **Win32 API şi registru de informaţii pentru Windows**

    6.1 Win32 API
    6.2 Interactia intre programe

7. **Subsistemul POSIX şi interfaţa de proces Win32**
    7.1 POSIX
    7.2 Interfaţa de proces Win32

1. Structurile de bază ale fiecărui sistem de operare în parte: concepte generale, structura nucleului

Majoritate distribuitorilor de Linux (Craftworks, Debian, Slackware, Red Hat) includ şi sursele kernel. De obicei kernel-ul Linux care a fost instalat în sistemul tău Linux a fost construit din aceste surse. Prin natura lor aceste surse tind să fie puţin depăşite aşa că puteţi să luaţi ultimele tipuri de surse de pe unul din site-urile web.

Linux este un system de operare multitasking (multiple procese care au loc concomitant), multiutilizator, ceea ce înseamnă că mai multe persoane pot folosi diferite aplicaţii pe acelaşi computer în acelaşi timp. Acesta diferă de MS-DOS, în care doar o singură persoană poate să utilizeze sistemul în acel moment. Lucrând în Linux, pentru a te identifica în sistem trebuie să te **înregistrezi**, care constă în introducerea **numelui de utilizator** (numele pe care sistemul îl foloseşte ca să te identifice) şi introducerea **parolei**, care este un cuvânt-cheie pentru înregistrarea în contul tău. Pentru că tu eşti singurul care ştie parola, nicio altă persoană nu se poate înregistra în system cu numele tău de utilizator.

În sistemele tradiţionale UNIX, administratorul de sistem îţi repartizează un nume de utilizator şi o parolă iniţială când ţi se dă un cont în sistem. Oricum, pentru că în Linux tu eşti administratorul de sistem, trebuie să-ţi stabileşti contul personal înainte să te înregistrezi. În următoarele discuţii se va folosi numele de utilizator “larry”.

În adăugire, fiecare sitem are un **nume gazdă** reparizat. Acest nume gazdă îi dă aparatului tău un nume, îi dă personalitate şi şarm. Numele gazdă este folosit pentru a indetifica individual aparatele în reţea, dar chiar dacă aparatul tău nu face parte dintr-o reţea trebuie să aibă un nume gazdă. Pentru exemplele de mai jos, numele gazda este ``mousehouse''.

Sursele kernel din Linux au un sistem de numerotare foarte simplu. Orice numar kernel întreg (de exemplu 2.0.30) este un adapost, o declanşare, kernel şi orice numar kernel împar (de exemplu 2.1.42) este o expunere kernel.

Expunerile kernel au toate variantele noi şi suportă orice tip de sistem. Deşi pot fi cam instabile, ceea ce nu se doreşte, este important pentru comunitatea Linux să încerce ultima variantă de kernel. În acest fel sunt testate pentru întreaga comunitate. Este de reţinut că întotdeauna se merită arhivarea sistemului în cazul în care se foloseşte un kernel neproductiv.

Schimbarile produse la sursele kernel sunt distribuite ca fişiere **patch.** Utilitatea patch este folosită pentru a aplica seriile editate pe un set de fişiere sursă. De exemplu, dacă se vrea schimbare unei unei surse kernel 3 varianta 2.0.29 în sursa kernel 3 varianta 2.0.30 trebuie obţinut fişierul patch 2.0.20 şi aplicate la acea sursă 3.

Un program atât de mare şi de complex precum Linux kernel poate fi cam încurcat la început. Este mai degrabă ca un ghem mare de sfori fără sfârşit. Privind dintr-o singură parte a kernel-ului de cele mai multe ori conduce la a privi mai multe alte fişiere înrudite şi nu după mult timp se uită ceea ce se caută. Următoarele subsecţiuni dau un indiciu unde ar trebui să cauţi, în cel mai bun loc, fişierul dat.

Într-un sistem bazat pe Intel, sursele kernel pornesc când loadlin.exe sau LILO a încărcat kernel-ul şi are control asupra acestuia. Caută în arch/i386/kernel/head.S pentru această parte. Head.S face nişte setări specific arhitecturale apoi sare la rutina main() în init/main.c.

Semafoarele sunt utilizate pentru a proteja regiuni critice bazate pe structuri de date şi coduri. De reţinut că fiecare accesare a unei piste critice de date precum VSF inode care descrie un director este făcută de kernel pe baza unui program de generare de coduri. Este foarte periculos să se permită unui proces distrugerea unei structuri critice de date care este folosită de către alt proces. O cale în a obţine acest lucru este folosirea unui buzz lock în acea zonă critică de date care este accesată, dar acest lucru este o abordare simplistă care nu va permite o bună funcţionare a sistemului. În schimb, Linux foloseşte semafoare pentru a permite doar unui singur proces de a accesa regiuni critice de date şi coduri; toate celelalte procese care vor să acceseze această sursă vor fi făcute să aştepte până când aceasta se va elibera. Procesele de aştepatre vor fi suspendate, iar alte procese vor putea continua rularea în mod normal.

Linux are noţiune destul de simplă asupra timpului; măsoară timpul în bătăi de ceas din momentul în care sistemul porneşte. Toate sistemele de timp sunt bazate pe această măsurare, care este cunoscută ca **jiffies** după disponibilitate globală variabilă a aceluiaşi nume.

În Windows kernel convenţia folosită la apelul funcţiilor pentru a indica succes este următoarea: se întoarce `STATUS_SUCCESS` pentru succes şi o valoare diferită de `STATUS_SUCCESS` pentru insuccess.

În Windows kernel pentru a realiza operaţii atomice se folosesc funcţiile Interlocked. În continuare sunt prezentate câteva dintre acestea:

- `InterlockedCompareExchange` compară două valori de 32 biţi şi în funcţie de rezultatul comparaţiei schimbă cu o a treia valoare de 32 biţi
- `InterlockedDecrement` decrementează cu o unitate o variabilă întreagă
- `InterlockedExchange` schimbă două valori
- `InterlockedExchangeAdd` adună două valori şi întoarce suma
- `InterlockedIncrement` incrementează cu o unitate o variabilă întreagă

www.tldp.org

2. Nivelul de abstractizare al hard-ului

Unul dintre elementele cruciale ale designului Windows îl reprezintă portabilitatea pe o varietate de platforme hardware. Nivelul de abstractizare al hard-ului (HAL - Hardware Abstraction Layer) este un element cheie al realizării acestei portabilităţi. HAL este un modul kernel încărcabil (Hal.dll) care oferă interfaţă de nivel scăzut la platforma hardware pe care rulează Windows. El ascunde detaliile dependente de hardware cum ar fi interfeţele I/O, controllerele de întreruperi şi mecanismele de comunicaţie multiprocesor.

Deci mai degrabă decât să acceseze hardware-ul direct, componentele interne ale Windows ca şi driverii scrişi de utilizatori îşi menţin portabilitatea prin apelarea rutinelor HAL atunci când au nevoie de informaţii dependente de platformă. Din acest motiv rutinele HAL sunt documentate in Windows DDK.

Driverii din Windows nu manipulează hardware-ul direct, ci apelează funcţii din HAL pentru a interfaţa cu hardwareul. Driverii sunt scrişi în C(uneori C++) şi prin urmare prin utilizarea bună a rutinelor HAL, pot fi portabili în cod sursă pe arhitecturile CPU suportate de Windows şi portabile binar într-o familie de arhitecturi.

Există mai multe tipuri de driveri:

- Driveri de dispozitive hardware manipulează hardware-ul (folosind HAL) pentru a scrie sau a citi de pe un dispozitiv fizic sau reţea. Sunt multe tipuri de driveri hardware cum ar fi driveri

de bus, driveri de interfaţă umană, driveri pentru medii de stocare şi aşa mai departe.

- Driveri pentru sisteme de fişiere sunt driveri Windows care acceptă cereri I/O orientate pe fişiere şi le traduc în cereri I/O orientate pe un anumit dispozitiv.
- Driveri filtru de sistem de fişiere cum ar fi aceia care fac mirroring pentru diskuri (o copie a unui disk ce se găseşte pe un disk separat şi se foloseşte pentru backup) şi criptare pentu diskuri, interceptează cererile I/O şi fac procesări de valoare adaugată înainte de a trece cererea la nivelul următor.
- Network redirectors and servers sunt driveri de sisteme de fişiere care transmit cereri I/O de sisteme de fişiere unei maşini de pe reţea şi deasemenea primesc astfel de cereri.
- Driveri de protocol implementează un protocol de reţea cum ar fi TCP/IP, NetBEUI si IPX/SPX
- Driveri de filtrare ai fluxului kernel sunt înlănţuiţi pentru a realiza procesare de semnal pe fluxurile de date ca înregistrarea şi afişarea audio şi video

Vizualizarea HAL-urilor de baza incluse în Windows

Pentru a vizualiza HAL-urile din Windows se deschide fişierul Driver.cab în foderul specific sub \Windows\Driver Cache. (de exemplu, pentru sisteme x86 numele fişierului este \Windows\Driver Cache\i386\Driver.cab). Se cauta fişierele care încep cu "Hal" şi o să se observe fişierele din tabel descrise în Windows DDK.

Deşi mai multe HAL-uri sunt incluse în Windows, numai unul dintre ele este ales în momentul instalării şi copiat pe diskul de sistem cu numele de fişier Hal.dll. Prin urmare nu se poate presupune că un disk de sistem ce aparţine unei instalări x86 va boota pe un procesor diferit dacă HAL-ul care suportă acel procesor este diferit.

Nume Fişier HAL	Sisteme Suportate
Hal.dll	PC-uri Standard
Halacpi.dll	PC-uri ACPI(Advanced Configuration and Power Interface)
Halapic.dll	PC-uri APIC(Advanced Programmable Interrupt Controller)
Halaacpi.dll	PC-uri APIC ACPI
Halmps.dll	PC-uri Multiprocesor
Halmacpi.dll	PC-uri ACPI Multiprocesor
Halborg.dll	Staţie de lucru Silicon Graphics(numai Windows 2000)

Halsp.dll	Compaq SystemPro (numai Window XP)

Determinarea HAL-ului pe care îl rulaţi de mai sus.

Există 2 metode de a determina HAL-ul pe care îl rulaţi:

1. Se deschide fişierul \Windows\Repair\Setup.log, se caută Hal.dll şi se vizualizeaza numele fişierului de după semnul egal. Acesta este numele HAl-ului extras din Driver.cab.
2. În Device Manager se caută numele driverului în Computer device type.
3. Microsoft Windows Internals autor Mark E. Russinovich David Solomon

3. Interpretorele de comenzi din Linux

Interpretoarele de comenzi din Linux poartă numele generic de shell. Calculatorul poate înţelege decât limbajul binar format din 1 şi 0. Datorită complexităţii ridicate a limbajului binar, în sistemele de operare au fost introduse interpretoare de comenzi, care realizează conversia instrucţiunii introduse de utilizator într-o limbă general cunoscută (de obicei engleza) în cod binar recunoscut de calculator. Shell-ul este un interpretor care execută comenzi introduse de la un dispozitiv de intrare standard cum ar fi tastatura şi mouse-ul sau citite dintr-un fişier. Dacă introducem comenzile de la tastatură vorbim despre interfaţă în linie de comandă (CLIs - command line interfaces).

Odată validată comanda, shell-ul o trimite către nucleu (kernel), care are rolul de punte de legătură între hardware-ul calculatorului şi diferite programe/shell-uri sau aplicaţii. Shell-ul nu este o parte a nucleului de sistem, dar îl foloseşte pentru a executa programe, crea fişiere şi altele.

În sistemul de operare Linux se găsesc mai multe shell-uri, pe care le putem afla cu ajutorul comenzii **$ cat /etc/shells.** Fiecare shell înţelege o anumită sinataxă de comandă şi accesează diferite funcţii predefinite.

În cadrul ultimelor versiuni de sistem de operare Linux introducerea comezilor, chiar şi în limbaj cunoscut, a fost înlocuită cu interfeţe grafice (GUIs - graphical user interfaces) uşor de folosit de către utilizator (user friendly). Scăzând astfel şi mai mult complexitatea operării sistemului de operare, care necesita cunoştinte aprofundate a comezilor şi instrucţiunilor.

Pe cele mai multe sisteme de operare Linux există un program numit *bash* (Bourne Again Shell - o versiune mai complexă a programului Bourne shell) care acţionează ca program shell. Pe lângă bash mai găsim şi shell-urile *ksh*,*tcsh* şi *zsh*.

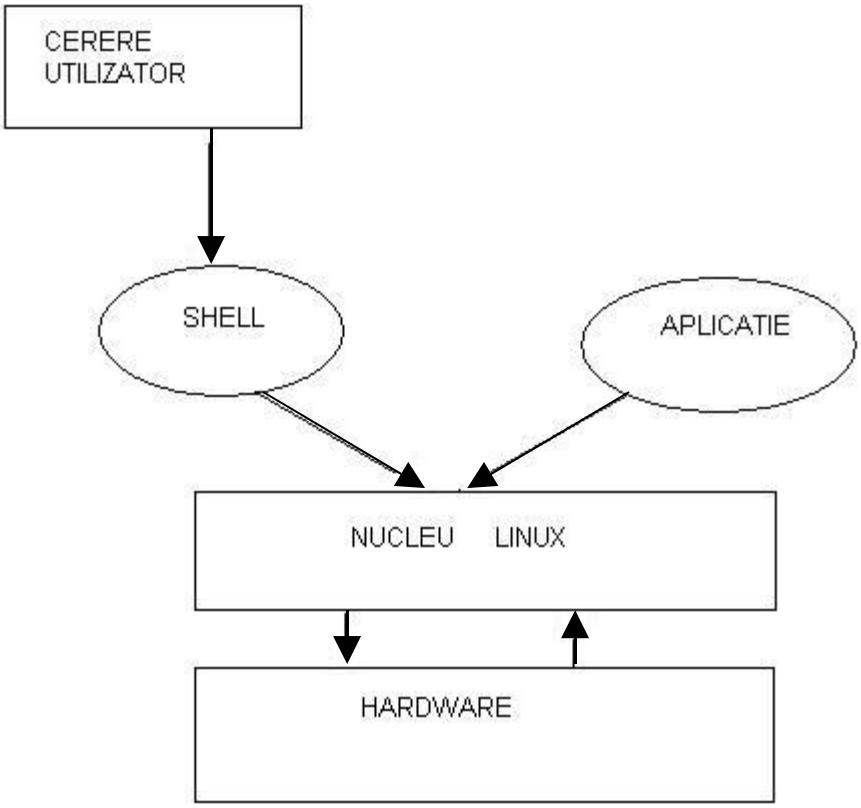
KSH - (Korn shell) este o comandă şi un limbaj de programare care execută comenzile citite de pe un terminal sau dintr-un fişier. De asemenea mai întâlnim *Rksh* care este o verisune restricţionată a interpretorului de comenzi ksh, folosit la setarea numelui de logare. *Rpfksh* este o veriune profil a shelului ksh, care execută comenzi cu atributele specificate de profilul utilizatorului.

TCSH - un shell C care se ocupă de completarea numelui fişierelor şi editarea liniilor de comandă. Este o versiune îmbunătăţită dar complet compatibilă cu versiunea Berkeley Unix C shell, *csh(1)*. Interpretorul de comandă tcsh, este pe de o parte un shell interactiv de logare, pe de altă parte un shell care procesează comenzile dintr-un script.

ZSH - shell-ul Z, care reprezintă o versiune îmbunătăţită a interpretorului de comandă tcsh.

Alte shell-uri întâlnite în sistemele Linux:

- *jsh* Job control shell
- *rc* Plan 9 shell
- *rsh* Remote Shell



1.Diagrama funcţionare shell

Linux Shell Scripting Tutorial - A Beginner's handbook

4. Comparaţie interfeţe

Linux şi Windows au amândouă o interfaţă grafică GUI (Graphical User Interface) şi o interfaţă în linie de comandă CLI (Command Line interface). Interfaţa grafică a Windows s-a schimbat drastic de la Windows 3.1 la Windows 95 apoi puţin la trecerea la Windows 2000 şi destul de mult odată cu apariţia Windows XP şi se va schimba din nou odată cu noua versiune a Windows, cea care va înlocui Windows XP. Windows XP are o funcţie de teme care ofera unele posibilităţi de customizare a aspectului şi a felului cum lucrează interfaţa grafică.

Linux oferă în general două interfeţe grafice KDE şi Gnome. Una dintre cele mai distribuite versiuni de Linux, Lindows are interfaţa grafică realizată în aşa fel încât să arate mai mult ca interfaţă de Windows decât

celelalte. Există şi XP de pentru Linux care într-adevăr face Linux să arate ca Windows.

Diferenţa este că interfaţa grafică Linux este opţională în timp ce interfaţa Windows este parte intergrantă a sistemului de operare. Eficienţa şi stabilitatea sunt mărite prin rularea unui server Linux fără interfaţă grafică, ceea ce serverele de Windows nu pot face. De asemea faptul că Linux nu depinde de interfaţa grafică face controlul şi administrarea de la distanţă a unui computer Linux să fie mai simplă şi mai naturală decât a unui computer Windows.

Această flexibilitate a interfeţei Linux are şi avantaje şi dezavantaje: utilizatorii experimentaţi pot să îşi customizeze totul după cum doresc însă este mai dificil pentru utilizatorii noi, pentru care fiecare computer Linux pe care îl întâlnesc ar putea să arate şi să se comporte diferit.

Michael Horowitz - A comparison of Linux and Windows

Exemple de interfeţe Linux:

4.1 GNOME

GNOME este o încercare internaţională de a elabora un mediu desktop complet, interfaţa grafică de utilizator (GUI) prin care se interacţionează cu sistemul de operare. Aceasta este elaborată ca software gratuit.

GNOME face parte din proiectul GNU şi poate fi folosit cu diferite sisteme de operare bazate pe UNIX, cel mai important fiind LINUX.
Proiectul GNOME pune accent pe simplicitate, utilizare uşoară, şi a face lucrurile funcţionale. Celelate scopuri ale proiectului sunt:
- Libertatea de a creea un mediu desktop care va avea întodeauna codul sursă disponibil pentru refolosire sub licenţă gratuită.

- Accesibilitatea : asigurând faptul că desktopul poate fi folosit de oricine, indiferent de abilitatile tehnice.
-Internaţionalizarea şi localizarea: să facă desktopul disponibil în multe limbi. În momentul actual Gnome este tradus în peste 100 de limbi.
-Să fie prietenos cu dezvoltatorii : asigură faptul că o să fie uşor să fie scrise programe care să se integreze uşor cu desktopul, şi să permita dezvoltatorilor libera alegere a limbajului de programare.
-Suportul : să asigure suportul altor instituţii pe lângă comunitatea GNOME.

GNOME este construit în jurul modelului clasic de desktop. Felul în care manevrează ferestre, aplicaţii şi fişiere este similar cu cel al sistemelor de operare contemporane. În configurarea de baza desktopul are un meniu de lansare pentru acces rapid la programele instalate şi locaţiile de fişiere. Toate opţiunile pot fi mutate aproape oriunde doreşte utilizatorul.

Aspectul GNOME poate fi modificat prin folosirea temelor. Teme populare includ Bluecurve şi Clearlooks. GNOME pune accentul pe a fi uşor de utilizat de către oricine.

4.2 KDE

KDE(K Desktop Enviroment) este un soft gratuit care încearcă să fie un sistem puternic pentru un mediu desktop uşor de folosit. Scopul proiectului este de a oferi funcţii desktop de bază şi aplicaţii pentru nevoile zilnice ca şi unelte şi documentaţie pentru ca dezvoltatorii să poată scrie aplicaţii pentru sistem.

KDE este construit cu toolkitul Trolltech Qt care rulează pe majoritatea sistemelor de tip UNIX, Mac OS X şi Microsoft Windows.

KDE încearcă să realizeze programe uşor de folosit fără a sacrifica din funcţii. Pentru a îmbunătăţi interfaţa cu utilizatorul s-a lucrat la reducerea complexităţii vizuale la versiunile 3.2 până la 3.5. Unul dintre scopurile principale ale KDE 4.0 este de a identifica zonele în care utilizarea este prea complexă şi să rezolve aceste probleme.

KDE încearcă să facă acţiunile dificile mai simple, cum ar fi adăugarea de imprimante (locale sau pe reţea) configurarea securităţii pentru sistemele Wireless 802.11 (cum ar fi WEP), şi instalarea de fonturi noi şi decoraţii pentru ferestre.

Interfaţa KDE a fost criticată pentru că ar fi prea complexă şi ar include prea multe opţiuni configurabile. Totuşi un raport al gradului de dificultate al utilizării care evalua o versiune customizată a KDE 3.1 a arătăt în 2003 că utilizatorii de Windows au reuşit să înveţe foarte rapid KDE, le-a placut şi au reuşit să realizeze acţiunile pe care şi le-au propus la fel de repede ca şi în Windows XP.

4.3 Interfaţa în mod text

Interfaţa în linie de comandă este cunoscută şi ca interpretorul de comenzi. Utilizatorii Windows îl mai numesc şi DOS promt. Utilizatorii Linux îl numesc shell. Fiecare versiune Windows are un singur interpretor de comenzi. În general interpretoarele de comenzi ale seriei Windows 9x sunt foarte similare şi clasa NT de versiuni Windows (NT,2000,XP) au deasemena interpretoare de comenzi similare. Există totuşi diferenţe între un interpretor de comenzi Windows 9x şi unul ce face parte din clasa NT a Windows. Linux ca toate versiunile de Unix suportă multiple interpretoare de comenzi, dar de obicei foloseşte BASH (Bourne Again Shell). Alte interpretoare mai sunt Korn shell, Bourne shell, ash şi C shell.

Michael Horowitz - A comparison of Linux and Windows

5. Biblioteca apelurilor de sistem pentru Linux

Apelurile de sistem sunt modalitatea prin care utilizatorul poate cere servicii nucleului. Aceste servicii sunt specifice sistemului de operare ca de exemplu: reţea, spaţiu de memorare, memorie şi altele. De exemplu dacă utilizatorul doreşte să citească un fişier atunci el trebuie să facă apelurile de sistem „deschide” şi „citeşte”. În general apelurile de sistem nu sunt chemate direct de procese. Există o bibliotecă C care oferă o interfaţă către toate apelurile de sistem.

Apelul de sistem *fork()* găsit în cadrul sistemului de operare Linux are rolul de a crea procese. De menţionat este faptul că noul proces se va numi proces copil al procesului chemat anterior. Va putea fi recunoscut prin ID-ul trimis de către sistemul de apel *fork()*. Putem avea urmatoarele situaţii:

- să se returneze valoarea -1, caz în care apelul de sistem nu a reuşit

- se returnează id-ul procesului copil în cadru procesului parinte şi valoarea 0 în procesul copil.

Apelul de sistem *exec()* are rolul de a schimba procesul actual cu un program nou fără ca PID-ul să se schimbe. În cazul în care apelul de sistem nu reuşeşte apare o eroare în cadrul procesului chemat.

Apelul de sistem *wait()* este folosint în situaţiile în care se doreşte o sincronizare între procesul părinte şi procesul copil. Se va returna valoarea -1 în caz de nereuşită.

Apelul de sistem *dup* este folosit atunci când se doreşte copierea unui descriptor de fişier.

Mai întâlnim şi apelurile de sistem *getenv*, *setenv* şi *unsetenv* care sunt folosite în cazul în care se doreşte citirea, modificarea sau ştergerea unei variabile din cadrul tabelei de variabile environ.

În cadrul unui apel de sistem un cod de nucleu este rulat la cererea unui proces al utilizatorului. Acest cod are nivelul de privilegiu cel mai mare şi anume nivelul 0 (CPL-0). Cum toate procesele utilizatorului rulează în cadrul nivelului de privilegiu 3, apare problema implementării mecanismului apelului de sistem, întrucât trebuie să apelam un cod de nivel 0 de la nivelul 3. Linux obişnuia să implementeze apelurile de sistem pe toate platformele x86 cu ajutorul întreruperilor software. Pentru a executa un apel de sistem, procesul va copia numărul apelului de sistem dorit în registrul %eax şi va executa comanda „int 0x80”. Se va genera astfel întreruperea 0x80 şi va fi apelată o rutină de serviciu al întreruperei respectiv, care va chema apelul de sistem dorit.

Odată cu apariţia procesoarelor mai evoluate care au instrucţiunile SYSENTER/SYSEXIT (instrucţiuni optimizate să ofere performanţă maximă pentru tranziţii către nivelul de privilegiu 0) Linux a implementat un nou mecanism al apelurilor de sistem. Acest mecanism funcţionează în felul

următor: fiecărui apel de sistem îi este alocat un număr între 0 şi 30000. Nucleul crează o pagină în memorie şi o ataşează la adresa fiecărui proces încărcat în memorie. Această pagina conţine codul de implementare al mecanismului apelului de sistem intrare/ieşire. Nucleul cunoaşte această pagină sub denumirea de *virtual dinamic shared object* (vdso), ea nu există fizic. Cu ajutorul funcţiilor SYSENTER/SYSEXIT apelarea nivelului 0 se face foarte uşor, rezultând o performanţă mărită faţă de vechiul mecanism de apelare.

Fiecare apel de sistem are o funcţie asociată în biblioteca apelurilor de sistem.

De menţionat este faptul că noile sisteme de operare linux pot funcţiona şi pe procesoare care nu suportă funcţiile SYSENTER/SYSEXIT. În momentul iniţializării nucleului sistemului este chemată rutina sysenter_setup(), care creează o pagină de memorie în care este codul instructiunii sysenter, dacă procesorul suportă aceasta, iar dacă nu se apeleaza întreruperea 0x80. Biblicoteca C poate folosi cel mai rapid mod de apel de sistem sărind la o adresă fixă din pagina virtuală.

Despre un apel de sistem trebuie să ştim tipul acestuia, parametrii, tipurile parametrilor şi ce tip de rezultat întoarce. Există 6 macrouri care uşurează execuţia unui apel de sistem şi au forma:

_syscallX(tip,nume,tip1,arg1,tip2,arg2,...)

Unde *X* este între 0 şi 5 şi reprezintă numărul de argumente aşteptat de apelul de sistem, *tip* este tipul de rezultat întors, *nume* este numelele apelului de sistem. Aceste macrouri creează o funcţie cu numele *nume* şi cu argumentele specificate, putând astfel executa apelul de sistem ca *nume*. De obicei nu este necesară execuţia directă a unui apelde sistem, dar sunt situaţii când biblioteca C standard nu implementează o execuţie elegantă a funcţiei. În funcţie de cerinţe se poate modifica macroul _syscall(). Macrourile vor returna rezultatul *r* al apelului de sistem când *r* este nenegativ, dar va returna -1 şi va seta variabila *err no* la *–r* când *r* este negativ.
Unele apeluri de sistem precum *mmap* au nevoie de mai mult de cinci argumente. O asemenea situaţie cere ca argumentele să fie puse pe stivă şi să fie pasat un pointer la blocul de argumente.

www.gnu.org/software/libc

6. Win32 API si registru de informatii pentru Windows

6.1 Win32 API

Windows Application Programming Interface(API) este interfaţa de programare de sistem a familiei de sisteme de operare Microsoft Windows, incluzând Windows 2000, Windows XP, Windows Server 2003, Windows 95, Windows 98, Windows Millenium Edition(Me). Fiecare sistem de operare implementează un subset diferit al Windows API.

Înainte de introducerea versiunilor pe 64 de biţi ale Windows XP şi Windows Server 2003 interfaţa de programare a versiunilor pe 32 de biţi a sistemelor de operare Windows se numea Win32 API pentru a se diferenţia de versiunea originală pe 16 biţi a Windows API care era interfaţa de programare a versiunilor pe 16 biţi ale Windows.

Windows API este compus din mii de funcţii apelabile care sunt împărţite în următoarele categorii:

1. **Servicii de bază**

   Oferă acces la resursele fundamentale disponibile unui sistem Windows. Sunt incluse sisteme de fişiere, dispozitive, procese şi thread-uri, acces la registru Windows şi controlul erorilor. Funcţiile se regăsesc în fişierele kernel.exe, krnl286.exe sau krnl386.exe pe Windows pe 16 biţi şi în kernel32.dll şi advapi32.dll pe Windows pe 32 de biţi.

2. **Interfaţa dispozitive grafice**

   Oferă funcţionalitatea necesară afişării conţinutului grafic pe monitore, imprimante şi alte dispozitive de afişare. Se regăseşte în gdi.exe pe Windows pe 16 biţi şi în gdi32.dll pe Windows pe 32 de biţi.

3. **Interfaţa utilizator**

   Oferă funcţionalitatea pentru creerea şi administrarea ferestrelor şi a majorităţii controalelor de bază cum ar fi

butoanele şi barele de scroll, primeşte informaţia de la mouse şi tastatură şi are alte funcţionalităţi asociate cu partea de interfaţă grafică de utilizator (GUI) a Windows. Această unitate funcţională se regăseşte în user.exe pe Windows pe 16 biţi şi în user32.dll pe Windows pe 32 biţi. Începând cu versiunile Windows XP controalele de bază se găsesc în comctl32.dll alături de controalele uzuale.

4. **Biblioteca de ferestre de dialog uzuală**

Oferă aplicaţiilor ferestrele de dialog pentru deschiderea şi salvarea de fişiere, alegerea culorii şi fontului etc. Biblioteca se regăseşte într-un fişier numit commdlg.dll pe Windows pe 16 biţi şi în comdlg32.dll pe Windows pe 32 biţi.

5. **Biblioteca de control uzuală**

Oferă aplicaţiilor acces la unele controale avansate oferite de sistemul de operare. Acestea includ bare de stare, bare de progres, toolbar-uri şi tab-uri. Biblioteca se regăseşte într-un fişier dll denumit commctrl.dll pe Windows pe 16 biţi şi în comctl32.dll pe Windows pe 32 biţi.

6. **Windows shell**

Componentă a Windows API ce permite aplicaţiilor acces la funcţionalităţile oferite de shell-ul sistemului de operare ca şi posibilitatea de a-l schimba şi îmbunătăţii.

Componenta se regăseşte în shell.dll pe Windows pe 16 biţi şi în shell32.dll pe Windows pe 32 biţi .

7. **Servicii reţea**

Oferă acces la diversele elemente de reţea ala sistemului de operare. Subcomponentele sale includ NetBIOS, Winsock, NetDDE, RPC şi multe altele.

Win32 API nu a fost destinat să fie interfaţă de programare originală pentru Microsoft Windows NT. Deoarece proiectul Windows NT a început ca un înlocuitor pentru OS/2 versiunea 2, interfata de programare principală a fost APi-ul pe 32 de biţi OS/2 Presentation Manager. După un an de lucru la proiect s-a lansat Microsoft Windows 3.0 şi a avut un mare succes. Ca urmare Microsoft a modificat direcţia de dezvoltare şi a făcut Windows NT înlocuitorul familiei de produse Windows şi nu a lui OS/2. La acest punct de turnură a apărut necesitatea specificării Windows API înainte de asta Windows API există numai ca o intefaţă pe 16 biţi.

Chiar dacă Windows API va introduce multe funcţionalităţi noi care nu erau disponibile pe Windows 3.1 Microsoft a decis să facă noul API compatibil cu numele pe 16 biţi ale funcţiilor Windows API şi să folosească tipurile de date oricând era posibil pentru a uşura portabilitatea aplicaţiilor Windows pe 16 biţi pe Windows NT.

Microsoft Windows Internals autor Mark E. Russinovich David Solomon

6.2 Interacţia între programe

Windows API se ocupă în principal cu interacţia dintre sistemul de operare şi o aplicaţie. Pentru comunicaţia între diversele aplicaţii Windows Microsoft a dezvoltat o serie de tehnologii pe lângă Windows API. Au început cu Dynamic Data Exchange (DDE) care a fost urmat de Object Linking and Embedding (OLE) şi mai tarziu de Component Object Model (COM).

Microsoft Windows Internals autor Mark E. Russinovich David Solomon

7. Subsistemul POSIX şi interfaţa de proces Win32

7.1 Posix

Posix (Portable Operating System Interface) se referă la o colecţie de standarde internaţionale pentru interfeţele sistemelor de operare în stil UNIX. Standardele POSIX încurajează producătorii să implementeze interfeţe de tip UNIX pentru a le face compatibile, astfel încât programatorii să-şi poată muta cu uşurinţă aplicaţiile de pe un sistem pe altul.

Windows implementează numai unul din standardele POSIX, POSIX.1, cunoscut înainte ca ISO/IEC 9945-1:1990 sau IEEE POSIX 1003.1-1990. Acest standard a fost introdus pentru a îndeplinii cerinţele introduse de guvernul SUA în anii 1980 care cerea respectarea standardului POSIX.1.

Deoarece respectrea POSIX.1 era un scop obligatoriu pentru Windows, sistemul de operare a fost proiectat pentru a asigura că suportul de baza al sistemului este prezent pentru a implementa un subsistem POSIX.1 (cum ar fi funcţia fork care este implementată în executivul Windows şi suportul pentru legături de fişiere hard în sistemul de fişiere Windows). Însă deoarece POSIX.1 defineşte un set limitat de servicii (cum ar controlul proceselor, comunicaţia între procese ş.a.m.d.), subsistemul POSIX al Windows 2000 nu este un mediu de programare complet. Şi deoarece aplicaţiile nu pot să amestece implicit cereri între subsisteme în Windows, aplicaţiile POSIX sunt limitate strict la setul de servicii definit în POSIX.1. Această restricţie înseamnă că un executabil POSIX în Windows nu poate creea un thread sau o fereastră sau să folosească RPC(Remote Procedure Calls) sau socket-uri.

Pentru a rezolva această limitare Microsoft oferă un produs numit Windows Services for UNIX care include de la versiunea 3.5 un mediu de subsisteme POSIX îmbunătăţit care oferă aproape 2000 de funcţii UNIX şi 300 de utilitare asemănătoare UNIX.

Acest subsistem POSIX îmbunătăţit foloseşte la mutarea aplicaţiilor UNIX pe Windows. Deoarece programele sunt conectate ca executabile POSIX ele nu pot apela funcţii Windows. Pentru a face ca aplicaţiile UNIX să fie compatibile cu Windows şi a permite folosirea funcţiilor Windows se pot folosii pachete de compatibilizare UNIX-Windows. Astfel o aplicaţie UNIX poate fi recompilată ca un executabil Windows şi poate începe să integreze apeluri la funcţii native Windows.

Pentru a compila şi conecta o aplicaţie POSIX în Windows este nevoie de colectoarele şi bibliotecile POSIX din platforma SDK. Executabilele POSIX sunt conectate la biblioteca de subsistem POSIX, Psxdll.dll. Deoarece implicit Windows este configurat să pornească subsistemul POSIX la cerere, prima dată când se rulează o aplicaţie POSIX, procesul de subsistem POSIX (Psxss.exe) trebuie să fie pornit. El rămâne pornit până ce sistemul se reboooteză. (dacă se închide procesul de subsistem POSIX nu se vor mai putea rula alte aplicaţii POSIX până la rebootare). Imaginea POSIX în sine nu este rulată direct în schimb o imagine specială de suport Posix.exe este rulată, care creează un proces fiu care rulează aplicaţia POSIX.

Microsoft Windows Internals autor Mark E. Russinovich David Solomon

7.2 Interfaţa de proces Win32

Aproape fiecare versiune nouă a Microsoft Windows a introdus propriile modificări Windows API. Numele API a fost păstrat în diferitele versiuni Windows şi schimbările de nume au fost limitate numai la schimbările arhitecturale şi de platformă majore. Microsoft a schimbat în cele din urmă numele familiei Win32 API în Windows API şi a inclus in el versiunile trecute şi viitoare ale API.

- Win16 este API-ul pentru primele versiuni pe 16 biţi ale Windows. Acestea erau denumite iniţial simplu Windows API. Funcţiile Win16 API se regăsesc în principiu în fişierele nucleu ale sistemului de operare: kernel.exe (sau krnl286.exe sau krnl386.exe), user.exe şi gdi.exe. În ciuda extensiei exe acestea sunt biblioteci legate dinamic(DLL).

- Win32 este API-ul pe 32 de biţi al versiunilor moderne ale Windows. API-ul este compus din funcţii implementate ca şi la Win16 în DLL-uri sistem. DLL-urile nucleu ale Win32 sunt kernel32.dll, user32.dll şi gdi32.dll. Win32 a fost introdus odată cu Windows NT. Versiunea Win32 care a fost lansată cu Windows 95 se denumea Win32c, "c" venea de la "compatibilitate", dar acest termen a fost abandonat în favoarea Win32. În Windows NT 4.0 şi succesorii lui (inclzând toate versiunile moderne ale Windows), apelurile Win32 sunt executate de două module, crss.exe în mod utilizator şi win32k.exe în mod kernel.
- Win32s este o extensie a familiei Windows 3.1x a Microsoft Windows care implementeza un subset al Win32 API pentru aceste sisteme. "S" vine de la "subset".
- Win32 pentru Windows pe 64 pe biţi, cunoscut ca Win64, este versiunea API ce se axează pe versiunile pe 64 de biţi ale Windows şi anume Windos XP Professional x64 Edition şi Windows Server 2003 x64 Edition (pentru procesoare x86-64) şi Windows XP 64-bit Edition şi Windows Server 2003 pentru seriile Itanium. Versiunile pe 64 de biţi sunt doar încă două platforme suportate în arhitectura Windows NT deci ambele versiuni pe 32 şi 64 de biţi ale unei aplicaţii pot fi compilate dintr-un singur cod. Toţi pointerii de memorie sunt pe 64 de biţi aşa că codul sursă trebuie să fie verificat pentu compatibilitate cu aritmetica pointerilor pe 64 de biţi şi sunt rescrise după cum este necesar. Nu există nici o funcţie nouă specifică versiunilor pe 64 de biţi ale Windows.

en.wikipedia.org

Concluzii:

Structura sistemelor de operare Linux şi Windows este asemănătoare, amândouă au un kernel care are rolul de a oferi o interfaţă între aplicaţii şi partea de hardware şi conţine sistemul de gestionare al fişierelor sistemul de

gestionare al proceselor şi componenta de comunicaţie în reţea. Ambele sunt multitasking, multiuser şi pot rula pe mai multe platforme. Ambele sunt structurate pe nivele.

Nivelul de abstactizare al hardului oferă sistemelor de operare portabilitate, ele necomunicând direct cu harware-ul ci prin intermediul acestui nivel.

În Linux interpretorul de comenzi este numit şi shell (asemănător cu command.com din MS-DOS). Există mai multe interpretoare de comenzi: bash, tcsh, pdksh,ksh,zsh.

Interfeţele grafice ale Windows şi Linux diferă destul de mult. În timp ce Windows are o singură interfaţă grafică ce este parte integrantă a sistemului de operare, Linux are mai multe interfeţe grafice ce pot fi instalate şi prezintă un grad mare customizare. Intrefeţele în mod text(interpretoarele de comenzi) sunt destul de asemănătoare.

Apelurile de sistem sunt modalitatea prin care utilizatorul poate cere servicii nucleului. Aceste servicii sunt specifice sistemului de operare ca de exemplu: reţea, spaţiu de memorare, memorie şi altele. Cele mai cunoscute apeluri sunt fork şi exec.

Windows Application Programming Interface(API) este interfaţa de programare de sistem a familiei de sisteme de operare Microsoft Windows. Fiecare sistem de operare implementează un subset diferit al Windows API.

Posix (Portable Operating System Interface) se referă la o colecţie de standarde internaţionale pentru interfeţele sistemelor de operare în stil UNIX. Standardele POSIX încurajează producătorii să implementeze interfeţe de tip UNIX pentru a le face compatibile, astfel încât programatorii să-şi poată muta cu uşurinţă aplicaţiile de pe un sistem pe altul.

Bibliografie :

Microsoft Windows Internals autor Mark E. Russinovich David Solomon www.tldp.org

Linux Shell Scripting Tutorial - A Beginner's handbook
Michael Horowitz - A comparison of Linux and Windows

ro.wikipedia.org

en.wikipedia.org

www.gnu.org/software/libc

1. **Structurile de bază ale fiecărui sistem de operare în parte: concepte generale, structura nucleului (Ududek Marian)**

2. **Nivelul de abstractizare al hard-ului (Frîncu Alexandru)**

3. **Interpretorele de comenzi din Linux (Tudoroiu Adrian)**
4. **Comparaţie interfeţe (Neagu Cristina)**

    4.1 GNOME

    4.2 KDE

    4.3 Interfaţa în mod text

5. **Biblioteca apelurilor de sistem pentru Linux (Tudoroiu Adrian)**

6. **Win32 API şi registru de informaţii pentru Windows (Neagu Cristina)**