Proiect sisteme de operare
- Sistemul de fisiere -

Bondalici Razvan - Sistemul de fişiere Linux. Drivere pentru sisteme de fişiere în Windows.

Baluta Dan - Sistemul de fişiere Linux. Drivere pentru sisteme de fişiere în Windows.

Bastea Mihai - Introducere în problematica generala a stocării în siguranță a datelor pe un dispozitiv. Drivere pentru sisteme de fişiere în Windows. Functii Win32 API

Mantarau George - Introducere în problematica generala a stocării în siguranță a datelor pe un dispozitiv. Drivere pentru sisteme de fişiere în Windows. Functii Win32 API

Ionescu Andrei - sistemul de fisiere FAT, Functii Win32 API

Popa Ionut - sistemul de fisiere HFS, Noțiuni de criptografie

Gheorghisan Laurentiu - sistemul de fisiere NTFS

Stanescu Silviu (455A fosta grupa 445A) - Noțiuni de criptografie

Badescu Stefan(455A fosta grupa 445A) - Tehnologii RAID

Capitolul 1.
Introducere în problematica generala a stocării în siguranță a datelor pe un dispozitiv

" *Nimeni nu va fi supus la imixtiuni arbitrare în viata sa personala, în familia sa, în domiciliul lui sau în corespondenta sa, nici la atingeri aduse onoarei si reputației sale. Orice persoana are dreptul la protecția legii împotriva unor asemenea imixtiuni sau atingeri.*"
-- Articolul 12 din Declarația Universală a Drepturilor Omului

În era comunicațiilor prin Internet şi a stocării electronice a documentelor, intimitatea şi confidențialitatea sunt greu puse la încercare, mai mult ca niciodată. Ca să reuşeşti să spargi un seif de oțel este necesar un efort considerabil, însă interceptarea comunicațiilor electronice şi intrarea remote pe un calculator poate fi făcută fără prea mult efort şi fără prea multe piedici impuse de lege.

Datele importante trebuiesc protejate, mai ales pe sistemele împărțite de mai mulți utilizatori. Şi unde se găseşte majoritatea datelor daca nu pe dispozitivele de stocare (hard discuri, floppy, carduri de memorie, etc.). Pentru a interzice accesul la datele confidențiale este nevoie de un mecanism prin care aceste date sa fie protejate.

Protecția datelor de pe un disc se poate face prin trei modalități. Criptarea întregului disc, criptarea unei partiții sau criptarea unui fişier. Problemă cu ținerea ascunsă a datelor prin criptare este că acestea au nevoie „să se mişte". Imaginativă criptarea ca un gard în jurul datelor. Cât timp datele sunt în interior sunt în siguranță. Dar însă pentru a fi în acelaşi timp şi utile acestea trebuie să poată fi disponibile utilizatorului, de drept, de fiecare dată când are nevoie şi fără sa fie nevoit să parcurgă prea mulți paşi pentru realizarea acestui lucru. Ideal ar fi, pentru utilizator, ca procesul de criptarea, să fie invizibil şi să nu-i ocupe din timp prin necesitatea autentificării de fiecare dată când are nevoie să acceseze datele.

O altă problemă o reprezintă algoritmii criptografici folosiți, care nu trebuie să fie uşor de spart. Se ştie ca atunci când cheia secretă are o dimensiune convenabilă şi este suficient de frecvent schimbată, spargerea cifrului devine practic imposibilă, chiar dacă se cunoaşte algoritmul de cifrare. De aceea e indicat ca dimensiunea cheii să fie mare, dar chiar şi aşa experiența a arătat că orice schemă criptografică are o viață limitată şi că avansul tehnologic reduce, mai devreme sau mai târziu, securitatea furnizată de ea.

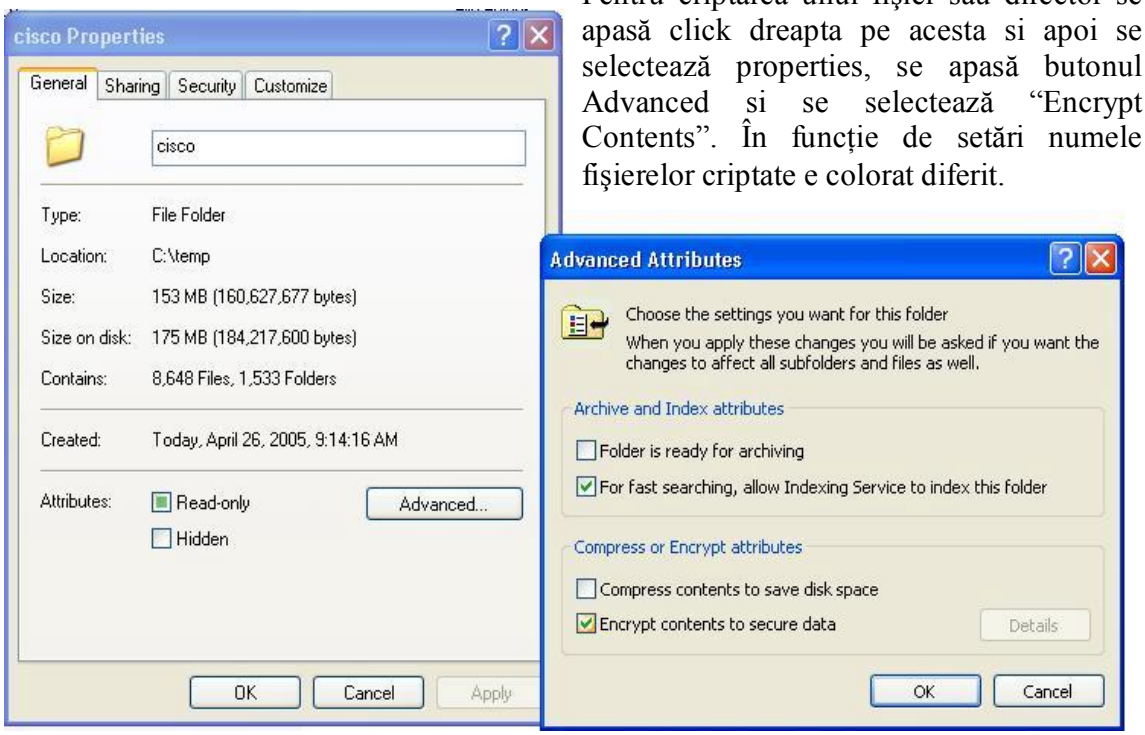
Ar fi bine de acoperit de un asemenea program şi aşa numitul „refuz plauzibil”. Aceasta reprezintă o vulnerabilitate tipic umană, prin care o persoană supusă la stres fizic sau de alta natură (de exemplu sub ameninţarea cu o armă), poate divulga parola. O soluţie la această problemă ar fi un mecanism care permite ca datele să fie decriptate cu mai mult de o parolă. Rezultatele fiind diferite, cu o parolă se pot decripta nişte date neimportante, dar care lasă impresia unei cooperări, iar cu cealaltă parolă să se ascundă datele cu adevărat sensibile.

Firma Microsoft pune la dispoziţie un mecanism, numit EFS, care se apropie de cerinţele pentru asigurarea securităţii unui disc.

Ce înseamnă caracteristica EFS (Encrypted File System – Sistem de Fişiere Criptat) a sistemului de fişiere NTFS pe Windows 2000/XP?

EFS permite să criptaţi datele de pe disc. Salvatul si accesul la fişiere se face normal, dar când se face scrierea pe disc, datele sunt criptate.

Pentru a putea folosi EFS trebuie folosita calitatea de „user” al 2000/XP. Fiecare utilizator ar trebui sa aibă un cont propriu protejat prin parolă. Când un utilizator criptează datele, acestea sunt accesibile doar din contul lui. Alţi utilizatori nu au acces la datele criptate. Conturile fără parola nu au nici un efect. Fişierele sistem nu pot fi criptate.



Pentru criptarea unui fişier sau director se apasă click dreapta pe acesta si apoi se selectează properties, se apasă butonul Advanced si se selectează “Encrypt Contents”. În funcţie de setări numele fişierelor criptate e colorat diferit.

Figura 1.1 Cum se poate cripta un fişier sau director cu EFS.

O problema, des întâlnită, apare la re-formatarea sistemului si la re-instalarea Windows-ului în timp ce exista date criptate pe alt disc şi nu există un back-up al “cheilor”. În acest caz nu se vor putea accesa datele criptate. Ce e de făcut? Răspunsul e ştergerea datelor, dacă nu pot fi recuperate folosind un Recover Agent sau dacă nu exista un backdoor (o metodă prin care pot fi sărite metodele obişnuite de autentificare).

Soluția: înainte de formatare trebuie întotdeauna făcut un backup al certificatului cu cheile publice, respectiv private. Cu toate ca nu este menționat în timpul procesului de criptare, XP creează pentru fiecare utilizator o pereche de chei privată şi publică. Acestea sunt necesare la folosirea EFS. Alți utilizatori nu au acces la chei, astfel neputâ nd accesa datele criptate. Când se re-instalează sistemul, noul certificat asociat contului, nu va funcționa cu datele criptate înainte. De aceea e foarte important ca aceste chei să fie salvate înainte.

Pentru aceasta:

1. Run şi comanda „mmc”.
2. În meniu click pe File->"Add/Remove Snap-in..." sau Ctrl+M
3. În tab-ul Standalone apăsați butonul Add
4. Selectați Certificates, apăsați add si apoi închideți şi OK
5. Selectați “Certificates - Current User” din „Console Root”, apoi Personal şi apoi Certificates
6. Se selectează certificatul care afişează cuvintele Encrypting File System în coloana Intended Purposes . Si apoi se exportă folosind click dreapta All tasks->Export

Folosind apoi Wizard-ul pentru export se salvează cheia privată într-un fişier .pfx. Se va cere şi protejarea cu parola a acestor chei. Acum avem cheia exportată şi în cazul în care vrem acces la datele criptate după formatare se dă un dublu click pe fişierul exportat şi apoi urmărind paşii Wizard-ului.

În afară de acest neajuns EFS mai are şi alte dezavantaje.

Unul din cele mai importante dezavantaje ale EFS este ca este dependent de sistemul de fişiere NTFS. Un disc formatat FAT sau FAT32 astfel nu poate fi protejat în nici un fel.

Un alt dezavantaj al EFS este că la realizarea criptării acesta creează o copie a fişierului original, criptează copia, şterge originalul şi apoi redenumeşte copia cu numele original. Aceasta face ca, în primul rând, la criptarea inițială a unui fişier să fie nevoie de spațiu liber pe disc şi în al doilea rând, prin copierea şi apoi ştergerea fişierului necriptat, folosind un program pentru recuperarea fişierelor şterse, se poate afla conținutul acestuia.

După cum se vede soluția EFS nu este întotdeauna cea mai bună. Astfel apare nevoia de un mecanism care să permită o bună protecție şi sa lucreze în fundal, rezident în memorie, care să se comporte ca un traductor între containerul datelor (care poate să fie un fişier de pe disc, o partiție a discului sau alte dispozitive de stocare precum un floppy disc) şi restul calculatorului, fără programul rezident în memorie şi fără parolă datele arătând ca un şir de numere aleatoare.

De asemenea în cazul în care programul rezident în memorie este oprit, hard drive-ul este deconectat subit sau sistemul este oprit, volumul să fie sigur în continuare, bineînțeles aceasta depinzând în mare parte de sistemul de fişiere pe care îl are la bază.

După cum se ştie operațiile de read/write de pe disc necesare aplicațiilor sunt realizate de sistemul de operare, în cazul nostru Windows 2000/Xp, cu ajutorul unui driver de disc (care de obicei face parte din sistemul de operare).

În figura 2.1 se poate vedea cum se fac în mod normal operațiile de read/write pe un disc.

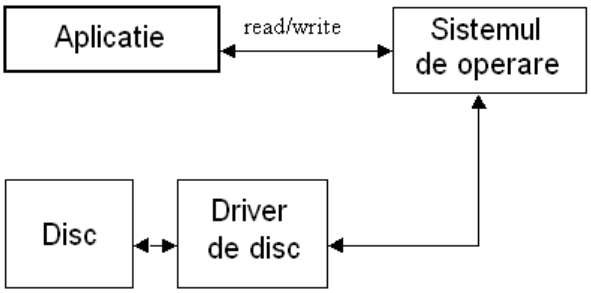


Figura 1.2 Operațiile de read/write pe un disc.

O modalitate prin care se poate interveni pentru criptarea/decriptarea datelor pe ciclurile de read/write este prin construirea unui driver filtru care să prindă datele din zbor (on-the-fly).

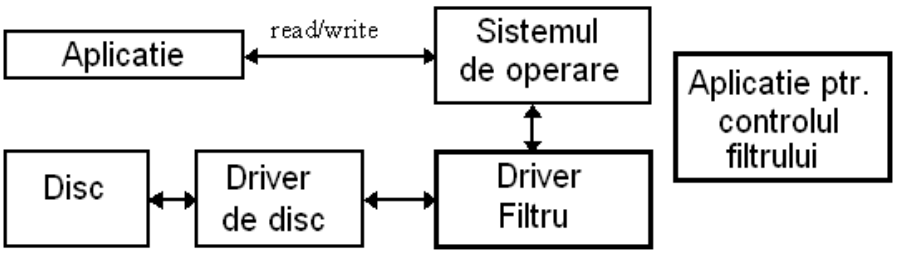


Figura 1.3 Operațiile de read/write pe un disc filtrate de un driver filtru.

Acest filtru nu trebuie să prelucreze toate cererile de I/O. Pentru ca procesul de criptare/decriptare să fie invizibil utilizatorului driverul filtru va crea propriile drive-uri virtuale. Doar operațiile pentru aceste drive-uri vor fi procesate de filtru. Aceste drive-uri virtuale pot fi văzute ca oricare alt disc având literele de drive corespunzătoare (de ex. E:, X: sau oricare litera care nu este deja folosită de un dispozitiv).

În continuare sunt descrise pe larg mecanismele folosite pentru construirea unei aplicații care să permită criptarea unui sistem de fişiere (FAT sau NTFS). În capitolul 2, informații despre cum se poate construi şi instala un driver filtru , în capitolul 3, informații despre algoritmii criptografici folosiți, precum şi alte detalii privind securiatatea.

In capitolul 4 este tratat sistemul de fidiere Linux, in capitolul 5 sistemul de fisiere FAT, in capitolul 6 sistemul NTFS iar in capitolul 7 sistemul HFS al MAC OS.

In capitolul 8 sunt prezentatate tehnologii RAID de securzire si stocare a datelor.

Capitolul 2.
Drivere pentru sisteme de fişiere în Windows

Judecând sistemele de operare prin prisma numărului de utilizatori şi a domeniilor de întrebuințare, sistemele de operare Windows 2000/ XP sunt considerate ca fiind unele dintre cele mai complete sisteme de operare. Prețul plătit pentru această calitate este complexitatea arhitecturii sistemului, aceasta fiind prezentată mai mult sau mai puțin detaliată în nenumărate lucrări.

O utilizare deosebită a acestui sistem este şi aceea de control a dispozitivelor hardware, facilitate pusă la dispoziţie cu ajutorul driverelor de dispozitiv, în cadrul acestui capitol se va încerca, în limitele admise, o abordare a tehnologiei de construcție a driverelor de dispozitiv sub sistemele de operare Windows 2000/XP, arhitectura internă a acestor două sisteme fiind asemănătoare.

2.1. Arhitectura Windows 2000/XP

O reprezentare din punct de vedere funcțional a sistemului de operare Windows 2000 este prezentată în figura 2.1. Pentru a înțelege mai bine relațiile dintre diferitele componente, reprezentarea este simplificată în mod deliberat.

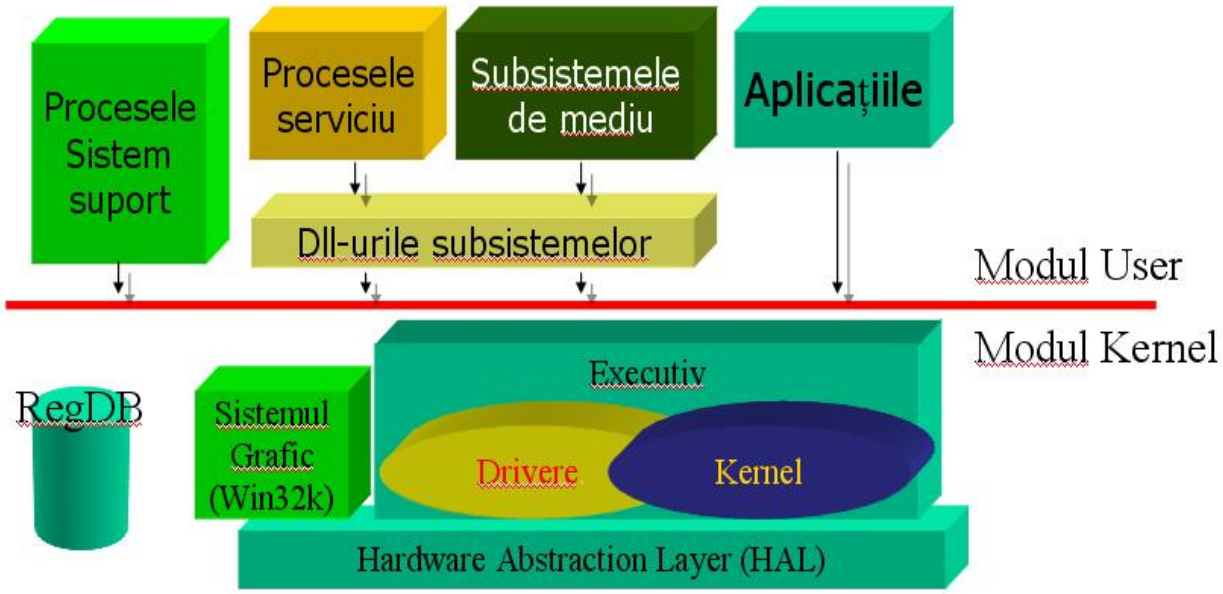


Figura 2.1 Arhitectura sistemului de operare Windows 2000

Înainte de a prezenta elementele care apar în figura 2.1, trebuie spus ce reprezintă modul user şi modul kernel, care sunt separate în figură prin linia punctată.

Pentru a îndeplini cerinţele impuse sistemului relativ la robusteţea şi siguranţa funcţionării, proiectanţii tehnologiei NT căreia îi aparţine şi sistemul de operare Windows 2000 au ales o arhitectură de tip client-server pentru implementarea nucleului sistemului. Astfel, o aplicaţie utilizator funcţionează în calitate de client a serviciilor puse la dispoziţie de sistemul de operare. Această arhitectură presupune că orice instrucţiune care se execută pe o maşină pe care este instalat sistemul de operare Windows 2000 se execută în modul user sau în modul kernel. Această împărţire a fost realizată în scopul protecţiei sistemului de operare considerat parte de încredere din punct de vedere al operaţiilor pe care le execută pe o maşină faţă de aplicaţiile utilizatorilor sistemului considerate parte de neîncredere. Având ca scop robusteţea şi siguranţa funcţionării, s-a considerat ca părţile de neîncredere să funcţioneze în modul user iar părţile de încredere în modul kernel. Diferenţa de funcţionare în cele 2 moduri este aceea că în modul user nu sunt permise o serie de operaţii precum accesul la tot spaţiul de memorie, accesul la porturi, execuţia anumitor instrucţiuni ale CPU considerate privilegiate, iar în modul kernel este permisă orice operaţie. Pentru implementarea acestei arhitecturi de lucru, în general, se apelează la suportul hardware oferit de CPU. De exemplu, procesoarele Intel implementează

hardware aceasta facilitate utilizând nivelele de privilegiu 0 şi 3 asociate modului kernel şi respectiv user.

Implementarea practică a celor 2 moduri de funcționare ale sistemului de operare Windows 2000 este rezumată în continuare. Prin utilizarea unor structuri de date aflate în memoria calculatorului sistemul de operare gestionează spațiile de memorie virtuală pe care aplicațiile care rulează la un moment dat îl văd(utilizează). Spațiul maxim vizibil este de 4GB. Din acest spațiu o aplicație poate avea acces la cel mult primii 2 GB. Maparea acestui spațiu virtual, care la sistemele de operare Windows este acelaşi cu cel liniar, la memoria fizică are loc prin utilizarea altor structuri de date gestionate de sistemul de operare(tabelele de pagini) realizând astfel protecția spațiilor de adrese între aplicații. Ceilalți 2 GB reprezintă spațiul virtual(liniar) unde se găseşte sistemul de operare, maparea în memoria fizică urmând acelaşi mecanism ca pentru aplicații. Deosebirea, totuşi, între cele 2 mapări este aceea că spațiul de maxim 2GB al fiecărei aplicații este mapat în altă zonă a memoriei fizice, pe când cel al sistemului de operare este mapat pe tot timpul funcționării în aceeaşi zonă astfel că la comutarea de task-un se schimbă doar maparea primilor 2 GB. acest lucru având o importanță deosebită pentru sistemul de operare(drivere) care întotdeauna trebuie să ştie spațiul cărei aplicații este vizibil în primii 2GB. Se consideră că. codul dispus în primii 2 GB(aplicație) se execută în modul user putând deci realiza operații restrânse, iar codul dispus în următorii 2 GB se execută în modul kernel putând realiza orice operație. Pentru implementarea optimă a mecanismelor enunțate mai sus Windows 2000 foloseşte facilitățile hardware oferite de diferitele arhitecturi(ex. Intel).

2.1.1. Procesele utilizator

Revenind la figura 2.1, cele 4 dreptunghiuri din partea superioară a figurii reprezintă cele 4 tipuri primare de procese utilizator care sunt:

1. *procesele sistem suport* - acestea sunt procese fixe care lucrează pentru sistemul de operare în scopul îndeplinirii unor sarcini care pot fi realizate din modul user; exemple de astfel de procese sunt: procesul logon, managerul de sesiune, etc. (a nu fi confundate cu procesele serviciu);
2. *procesele serviciu* - acestea găzduiesc serviciile Win32 precum Task Scheduler sau Spooler; multe aplicații server Windows 2000, precum Microsoft SQL Server, Microsoft Echange Server includ componente care funcționează ca servicii; o caracteristică a acestor procese este că nu au interfață grafică;
3. *aplicațiile utilizator* - pot fi una din cele 5 tipuri posibile:Win32, Windows 3.1, MS-DOS, POSIX sau OS/2 1.2;

4. *subsistemele de mediu* - sunt acele procese care fac publice aplicațiilor utilizator serviciile native ale sistemului de operare prin intermediul unor funcții apelabile, creând astfel un mediu de funcționare, o "personalitate"; Windows 2000 cuprinde 3 subsisteme de mediu: Win32, POSIX, şi OS/2;

În figura 2.1, sub procesele serviciu şi aplicațiile utilizator se găsesc dll-urile subsistemelor care au rolul de a intermedia apelul serviciilor sistemului de operare din modul user. Acestea sunt nişte librării dinamice care transformă o funcție documentată, apelabilă din modul user, in funcția serviciu corespunzătoare, internă sistemului de operare şi nedocumentată; exemple de astfel de librarii sunt: ntdll.dll, kernel32.dll, user32.dll, etc.

2.1.2. Componentele kernel ale sistemului de operare

Sub linia punctată din figura 2.1 sunt ilustrate componentele kernel ale sistemului de operare. Aceste componente trebuie înțelese mai degrabă ca nişte servicii (în sensul de deținere a capacității de realizare a unei sarcini cerute de un anumit client care poate fi o aplicație utilizator sau o altă componentă a sistemului de operare) pe care sistemul de operare le pune la dispoziția solicitanților. Ca observație, trebuie subliniat faptul că aceste componente nu trebuie înțelese nicidecum ca fiind nişte procese independente ca în cazul altor sisteme de operare.

Componentele sunt următoarele:

1. *executivul* - implementează serviciile de bază ale sistemului de operare precum managementul memoriei, al proceselor, al thread-urilor, securității, I/O. sau al comunicației interprocese:
2. *kernelul* - realizează funcții de nivel scăzut precum planificarea thread-urilor, tratarea întreruperilor şi a excepțiilor, sau sincronizarea interprocesor;
3. *drivere de dispozitiv* - includ atât driverele pentru dispozitive hardware care translatează cererile de operații I/O emise de aplicațiile utilizator în cereri I/O specifice dispozitivului hardware controlat, cât şi driverele sistemului de fişiere si de rețea;
4. *hardware abstraction layer (HAL)* - este un nivel de cod (subrutine) care izolează tot sistemul de operare de hardware-ul maşinii pe care acesta rulează în scopul evitării problemelor datorate din cauza diferențelor între platformele de lucru (de exemplu diferențele între plăcile de bază);

5. *sistemul grafic* - implementează funcţiile de realizare a interfeţei grafice(GUI) deosebite oferită de sistemele Windows;

Din punctul de vedere al fişierelor care alcătuiesc nucleul sistemului de operare Windows 2000, prezentat mai sus din perspectiva rolului în funcţionare, acesta se compune din fişierele din tabelul 2.1

Nume fişier	**Componente incluse**
ntoskrnl.exe	executivul şi kernelul
hal.dll	HAL (*hardware abstraction layer)*
Win32k.sys	Partea subsistemului Win32 care rulează în modul kernel
ntdll.dll	Conţine "puntea" de trecere din modul user în modul kernel
Kernel32.dll advapi32.dll user32.dll gdi32.dll	Dll-urile nucleu ale subsistemelor

Tabelul 2.1 Fişierele nucleului sistemului de operare Windows 2000

2.2. Driverele în arhitectura Windows 2000/XP

Relativ la drivere, în figura 2.1 este prezentat locul driverelor kernel în cadrul arhitecturii sistemului Windows 2000 însă aceste sunt de mai multe tipuri, în funcţie de criteriul de clasificare. În figura 2.2 sunt prezentate tipurile de drivere sub forma unui arbore, fiecare nivel reprezentând un criteriu iar nodurile reprezentând tipurile de drivere corespunzătoare unui anumit criteriu(nivel).

Pe primul nivel sunt situate driverele privite din perspectiva modului de execuţie a codului obiect al acestora:

- *drivere user-mode* - sunt acele drivere care funcţionează în modul user; la fel ca şi aplicaţiile utilizator, rolul lor este acela de a realiza operaţii care nu necesită nivel de privilegiu ridicat: exemple de astfel de drivere sunt driverele virtuale de dispozitiv(VDD), utilizate de maşina virtuală MS-DOS pentru asigurarea mediului de execuţie al aplicaţiilor MS-DOS prin translatarea operaţiilor privilegiate pe care aceste le execută în operaţii Win32 care folosesc driverele care rulează în modul kernel:
- *drivere kernel-mode* - sunt acele drivere ale căror funcţionare necesită privilegii sporite asupra resurselor hardware şi software, cerinţă

asigurată prin funcţionarea în modul kernel, alături de sistemul de operare după cum reiese şi din figura 2.1; aceste drivere sunt cele de interes şi în cazul acestei prezentări;

O altă clasificare este realizată din perspectiva tehnologiei în care sunt realizate driverele kernel. Unele sunt realizate utilizând o tehnologie moştenită de la versiunea anterioară de Windows NT 4.0 iar altele sunt construite conform modelului WDM. Cele 2 tipuri sunt destul de asemănătoare (modificarea unuia moştenit într-unul WDM este uşoară), însă cele WDM reprezintă ultima tehnologie promovată de firma Microsoft, incluzând un suport puternic pentru tehnologia Plug and Play (PNP), pentru controlul consumului de energie şi totodată fiind compatibile din punct de vedere al codului sursă (pentru a fi şi binar trebuie recompilate utilizând DDK-ul corespunzător) cu sistemul de operare Windows 98. Plug and Play reprezintă o tehnologie care implică atât hardware cât şi software care conlucra pentru implementarea operaţiilor de autodetecţie şi autoconfigurare specifice dispozitivelor moderne.

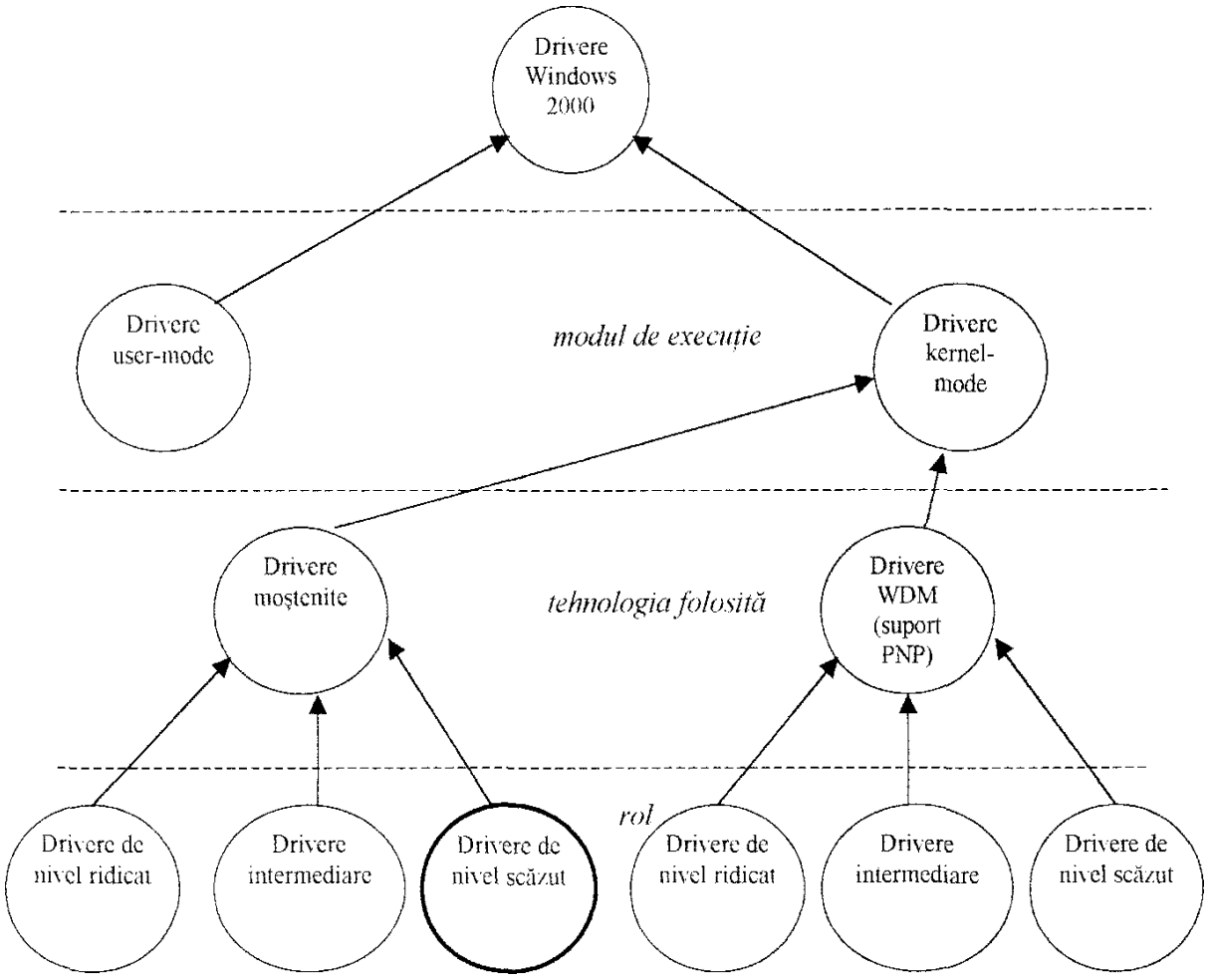


Figura 2.2 Clasificarea driverelor Windows 2000/XP

O caracterizare scurtă a driverelor kernel ar putea fi sintetizată astfel:
•au rolul de intermediar între aplicațiile utilizator sau alte componente ale sistemului de operare, şi dispozitivul fizic asociat; ele fac publice tuturor clienților permişi serviciile pe care respectivul dispozitiv le implementează (ex. transport de date, operații grafice, multimedia);
altfel zis se poate spune că gestionează funcționarea unui dispozitiv în contextul oferit de sistemul de operare gazdă;
•lucrează alături de sistemul de operare fiind considerate parte de încredere; reversul acestei facilități este că cerințele asupra corectitudinii codului sunt foarte ridicate, orice eroare care în cazul aplicațiilor ar fi prelucrată într-un mod riguros de sistemul de operare, în cazul driverelor poate duce la prăbuşirea întregului sistem; toate operațiile de genul alocare/dealocare de memorie, referențiere/dereferențiere handle-uri. etc. trebuie realizate explicit de driver;

•driverul beneficiază de facilitatea apelului direct al serviciilor sistemului de operare însă nu poate apela funcțiile subsistemului Win32 ca în cazul aplicațiilor utilizator (operație fără logică dealtfel);
•accesul la driver (apelul subrutinelor acestuia) are loc printr-o interfață standard pe care acesta şi-o defineşte;
•din punctul de vedere al statului său în cadrul sistemului, driverul poate fi văzut ca o bibliotecă (colecție) de funcții încărcată în spațiul de memorie virtual (liniar) al sistemului de operare, funcții care sunt apelate la inițiativa explicită a altcuiva (aplicație, sistemul de operare) şi nu precum thread-urile care la momente de timp bine definite li se dă controlul CPU; altfel zis "nu are viață"; ca observație trebuie spus că totuşi un driver poate crea thread-uri care au statutul celorlalte thread-uri ale sistemului de operare;
•contextele de execuție ale unui driver sunt:

- *contextul de trap sau excepție* - este cel mai comun şi apare atunci când o anumită aplicație cere o operație I/O care apelează funcțiile Win32 care fac trecerea execuției din modul user în modul kernel utilizând, în cazul arhitecturii Intel de exemplu, o poartă de întrerupere(int 2Eh), după care sunt apelate diferite servicii ale sistemului de operare care în final apelează o anumită subrutină din driver-ul care tratează tipul de cereri I/O emis de aplicație; rezultă că, contextul de execuție (memoria din primii 2GB, etc.) a driver-ului este cel al aplicației (thread-ului) care a cerut operația I/O; acest scenariu este valabil în condițiile în care între timp dintre momentul cererii I/O şi apelul codului driver-ului) nu este planificat pentru execuție alt thread, operație prin care se schimbă contextul (primiî 2 GB văzuți de driver, etc.), însă această operației planificare) este restricționată prin utilizarea unor mecanisme specifice, bazate pe priorități software de execuție(IRQL) care vor fi prezentate mai târziu;
- *contextul de întrerupere* - apare în cazul în care un anumit driver tratează o anumită întrerupere care poate apărea la orice moment de execuție, la care nu se cunoaşte contextul de lucru;
- *contextul thread-unlor kernel* - apare în cazul în care un driver creează un thread kernel care din punctul de vedere al planificării seamănă cu thread-urile user, astfel că şi în cazul acesta nu se poate face nici o bănuială asupra contextului de execuție iar thread-ul respectiv trebuie să țină cont de acest aspect.

2.3. Dezvoltarea unui Driver Filtru al Sistemului de Fişiere pentru Windows 2000/XP

Majoritatea aplicaţiilor moderne necesită o mare performanţă şi depind de rata de transfer a datelor pe şi de pe disc. Pentru aceasta sistemul de operare are nevoie de un driver de dispozitiv scris special pentru dispozitiv.

Pentru înţelegerea modului de construcţie a driverelor pentru sistemul de operare Windows 2000, în acest subcapitol se vor prezenta, paşii mai importanţi care trebuie urmaţi pentru dezvoltarea de drivere, respectiv a unui driver filtru pentru un sistem de fişiere.

2.3.1. Caracteristici generale

Resursele software minime necesare pentru dezvoltarea driverelor kernel constau din Device Development Kit (DDK) (indicată fiind versiunea pentru WindowsXP) şi produsul Visual C++ .NET din suita Visual Studio .NET 2003, însă se recomandă instalarea şi a suportului oferit de Service Development Kit (SDK). Pentru editarea fişierelor sursă se poate folosi orice editor text. Obţinerea driverelor (fişiere cu extensia .sys) din codul sursă poate fi realizată fie folosind utilitarul build din DDK (metodă recomandată versiunilor care vor intra în producţie) fie se poate utiliza mediul integrat Visual C++ pentru care trebuie setate corespunzător opţiunile referitor la formatul fişierului de ieşire (*.sys), metodă folosită la dezvoltare. Ambele metode fac apel la compilatorul de C şi celelalte utilitare necesare pentru obţinerea driverului, livrate de firma Microsoft.

Înainte de a prezenta structura driverului, este necesară prezentare generală unor concepte şi structuri la care se va face referinţă pe parcursul prezentării.

IRQL

Implementarea sistemului de operare Windows 2000 pe mai multe platforme hardware(Intel, PowerPC, SPARC) a dus la apariţia, printre altele, a problemelor legate de diferitele metode prin care fiecare tip de CPU implementează mecanismul de întreruperi în funcţie de priorităţi. Soluţia la această problemă a fost abstractizarea acestui mecanism prin software, definind *nivelele, cererilor de întrerupere* -- interrupt request level (IRQL). În figura 2.3 sunt prezentate toate nivelele cererilor de întrerupere definite de Windows 2000/XP, scopul acestora şi sursele care le generează.

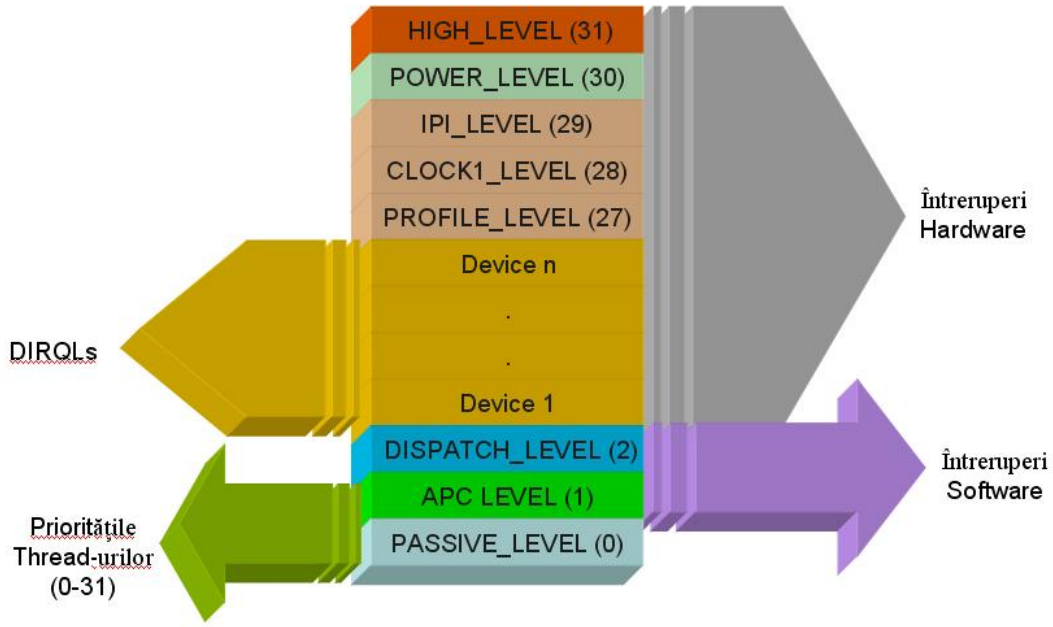


Figura 2.3 IRQL-urile.

Nivelele subliniate în tabelul 2.2 sunt cele mai utilizate în cazul driverelor. Acestea sunt folosite în cazul subrutinelor de tratare a întreruperilor (DIRQLs). În diferite subrutine care trebuie să se execute fără a fi întrerupte software (planificator sau alte thread-uri speciale - DISPATCH LEVEL), în thread-urile create de driver sau alte subrutine ale driverului a căror execuţie poate fi întreruptă de altă sursă software (PASSIVE LEVEL).

Numele IRQL	**Scopul IRQL**	**Sursa**
HIGHEST_LEVEL	folosit în cazul erorilor de magistrală şi a verificărilor hardware	hardware
POWER_LEVEL	folosit în cazul erorilor de alimentare	hardware
IPI_LEVEL	folosit în cazul sistemele multiprocesor	hardware
CLOCK2_ LEVEL	folosit pentru clock 2	hardware
CLOCK 1_ LEVEL	folosit pentru clock 1	hardware
PROFILE_LEVEL	profiling timer	hardware
DIROLs	folosite pentru maparea priorităţilor întreruperilor dispozitivelor externe	hardware

DISPATCH LEVEL	folosit la planificarea thread-urilor şi în cazul apelurilor procedurilor întârziate (DPC)	software
APC_LEVEL	folosit pentru apelul asincron al procedurilor	software
PASSIVE LEVEL	folosit în cazul execuţiei normale a thread-urilor	software

Tabelul 2.2 Nivelele cererilor de întrerupere

Trebuie subliniat faptul de a nu se crea confuzii între contextul de execuţie şi nivelele de prioritate ale CPU (IRQL), în contextul unei aplicaţii codul putându-se executa la diferite nivele de prioritate (IRQL). Modificarea IRQL curent este realizată explicit prin software din modul kernel utilizând serviciile KeRaiseIrql(...) şi KeLowerIrql(...), însă politica de gestiune a nivelului curent de execuţie al CPU permite ridicarea nivelului şi apoi coborârea acestuia şi nu invers, întreruperea execuţiei unui anumit cod poate fi realizată numai dacă codul care urmează a se executa este mai prioritar din perspectiva IRQL.

Acest mecanism software de organizare a priorităţilor de execuţie a codului pe un anumit procesor introduce anumite restricţii în ceea ce priveşte tipul codului şi nivelul de prioritate la care este executat. Motivul provine de la ideea că operaţii care nu sunt critice funcţionării sistemului nu trebuie executate la nivele de prioritate ridicată (şi pentru mult timp), care ar priva execuţia operaţiilor critice. De exemplu este incorect apelul funcţiei KeWaitForSingleObject la un nivel de prioritate mai mare decât PASSIVE _LEVEL ca de exemplu D1SPATCH__LEVEL, din moment ce această funcţie nu va face de obicei decât să aştepte apariţia unui eveniment. Acest lucru ar putea să priveze, de exemplu, Scheduler-ul (planificatorul) care rulează la nivelul de prioritate DISPATCH_LEVEL să preia controlul la momentul potrivii ducând astfel la disfuncţionalităţi în sistem.

IOCTL

Input/Output Control (IOCTL) reprezintă o structură pe 32 de biţi utilizată pentru transmiterea dintr-o aplicaţie (apelând funcţia DeviceIoControl) sau dintr-un driver (apelând IoCalldriver) unui alt driver a unor coduri de control. Prin acest mecanism se poate transporta şi date, în afară de codul de control.

I/O Request Packet (IRP)

Aproximativ tot ceea ce reprezintă operaţii I/O în Windows 2000/XP se desfăşoară utilizând ca formă de transfer a informaţiei aşa numitele pachete de cerere de intrare/ieşire (I/O Request Packets - IRP). Acestea sunt practic nişte structuri de date pe care I/O Manager (componentă a Executivului) le schimbă cu driverele şi acestea între ele în cadrul unei operaţii de intrare/ieşire cu un

dispozitiv. Din punct de vedere funcţional aceste structuri se comportă ca nişte *ordine de lucru* transmise de la o componentă la alta. Dintre elementele acestei structuri cele mai importante sunt pointerii la zona de memorie care conţine datele schimbate între componente, tipul operaţiei cerute (read, write, deviceiocontrol, etc.), codul de eroare, etc.

Obiecte driver

Implementarea sistemului de operare se bazează în mare măsură pe tehnologia orientată obiect. Fiecare entitate importantă din structura sistemului de operare este reprezentată prin obiecte (thread-uri, drivere, întreruperi, etc.) care reprezintă acea entitate în operaţiile în care este implicată.

Obiectele driver (driver objects) au rolul de a reprezenta un driver din punct de vedere al resurselor software cerute de acesta. Membri importanţi ai acestui obiect sunt pointerii către subrutinele pe care driverul le face publice şi care de altfel reprezintă interfaţa driverului către sistemul de operare (l/O Manager).

Singura subrutină exportată de un driver este subrutina cu numele (fixat) DriverEntry care este apelată la încărcarea driverului în memorie. Rolul principal al acesteia este de a crea interfaţa driverului prin iniţializarea pointerilor obiectului driver cu adresele subrutinelor driverului care devin astfel cunoscute I/O Managerului care până în acest moment nu cunoştea driverul decât prin prisma obiectului driver iniţializat la încărcarea driverului în memorie.

Obiecte dispozitiv

Obiectele dispozitiv (device objects) au rolul de a reprezenta din punct de vedere funcţional un anumit dispozitiv pe care driverul asociat îl gestionează, Există un obiect dispozitiv pentru fiecare dispozitiv fizic, virtual sau logic din sistem. Aceste obiecte sunt identificate prin nume unice în sistem (ex. “\\DosDevices\\VEFS"). Prin intermediul acestui nume, celelalte componente ale sistemului de operare (nu aplicaţii) pot interacţiona cu driverul asociat. Pentru a fi accesibil din aplicaţii (user mode) acestui obiect i se creează o legătură simbolică (symbolic link) care din punctul de vedere al aplicaţiilor este un nume asemănător cu cel cunoscut kernel-lului (ex. "\\??\\VEFS"). Prin această legătură o aplicaţie poate comunica cu driverul.

Un membru important al obiectului dispozitiv este legătura(pointerul) la lista coadă, pe care sistemul de operare (I/O Manager) o gestionează, prin care IRP-urile sunt semnalizate înainte de a fi prelucrate de driverul pe care acest obiect îl reprezintă. Aceasta este una din facilităţile pe care sistemul de operare le pune la dispoziţia dezvoltatorilor de drivere degrevându-i pe aceştia de această

grijă. Totuşi, dacă mecanismul gestionat de sistemul de operare nu convine, driverul îşi poate implementa propriile mecanisme de acces la dispozitiv.

Extensiile dispozitivului

Aceste extensii sunt practic nişte blocuri de memorie nepaginabilă alocată în zona sistem (protejată faţă de aplicaţii) care au rolul de a menţine informaţii despre un driver pe toată funcţionarea dispozitivului. Aceasta este soluţia la restricţia impusă driverelor, comparativ cu aplicaţiile, de a nu folosi variabile globale sau statice.

Spre exemplu structura folosită de driverul filtru.

```
typedef struct EXTENSION
{
BOOL bRootDevice;
/* Este acesta dispozitivul rădacina ? cu care aplicaţiile user-mode
comunică */

ULONG lMagicNumber;
/* Pentru a fi sigur că rutina de terminare nu trimite un IRP greşit */

int nDosDriveNo;
/* Numarul de Drive pe care această extensie o administrează */
BOOL bShuţingDown;                  /* Se opreşte driverul ? */
BOOL bThreadShouldQuit;             /* Instruieşte threadul de dispozitiv
                                       să renunţe */
PETHREAD peThread;                  /* Thread handle */
KEVENT keCreateEvent;   /*Evenimentul de creare a dispozitivului */
KSPIN_LOCK ListSpinLock;        /* IRP spinlock */
LIST_ENTRY ListEntry;           /* IRP lista de intrare */
KSEMAPHORE RequestSemaphore;    /* IRP lista cu cerere Semafor */

#ifdef USE_KERNEL_MUTEX
KMUTEX KernelMutex;         /* Mutex de Sync. pentru întregul thread */
#endif

HANDLE hDeviceFile;             /* Handle-ul de dispozitiv pentru
                                   dispozitiv */
PFILE_OBJECT pfoDeviceFile;  /* FileObject-ul dispozitivului */
PDEVICE_OBJECT pFsdDevice;      /* Handle-ul dispozitivului de cel
                                   mai de jos nivel */
```

```
	CRYPTO_INFO *cryptoInfo;	/* Informații Cryptografice */

	__int64 DiskLength;	/* Dimensiunea discului menționat de dispozitiv */
	__int64 NumberOfCylinders;	/* Informații partiție*/
	ULONG TracksPerCylinder;	/* Informații partiție*/
	ULONG SectorsPerTrack;	/* Informații partiție*/
	ULONG BytesPerSector;	/* Informații partiție*/
	UCHAR PartitionType;	/* Informații partiție*/

	KEVENT keVolumeEvent;	/* Structura eveniment folosită la
				instalarea unui dispozitiv */

	BOOL bReadOnly;	/* Este dispozitivul read-only ? */
	BOOL bRemovable;	/* Este dispozitivul removable media ? */
	BOOL bRawDevice;	/* Este aceasta o partiție-raw sau un
dispozitiv floppy-raw ? */
	BOOL bMountManager;	/* Dacă Mount manager-ul ştie despre
				volum */

	WCHAR wszVolume[64];	/* Pentru aplicați din user-mode, cum
				apare în tree view calea şi numele
				volumului montat, aici sunt stocate doar
				64 de caractere in loc de MAX_PATH =
				260 pentru a menține dimensiunea acestei
				structuri cât mai mică. A nu se schimba
				dimensiunea aceasta fără a se schimba şi
				dimensiunea din structura
				MOUNT_LIST_STRUCT! */

	long mountime;	/* Timpul când acest volum a fost montat ultima
			dată, pentru aplicația din user-mode */

	// Data/timpul fişierului container (folosite pentru a reseta data şi timpul
containerelor găzduite, după o încercare de montare nereuşită sau demontare).
	LARGE_INTEGER fileCreationTime;
	LARGE_INTEGER fileLastAcceşTime;
	LARGE_INTEGER fileLastWriteTime;
	LARGE_INTEGER fileLastChangeTime;

	} EXTENSION, *PEXTENSION;
```

Obiecte întrerupere

Rolul acestor obiecte este acela de a oferi kernel-ului o modalitate de localizare a subrutinei de tratare a întreruperii.

Între toate obiectele prezentate mai sus există legături foarte strânse prin intermediul pointerilor pentru a permite accesul la toate informațiile acestora pe diverse căi.

În afara de obiectele prezentate mai sus mai există şi alte obiecte care însă nu sunt utilizate la fel de des ca cele prezentate.

Înainte de a trece mai departe este necesar a se face o descriere sumară a subrutinelor pe care diferite componente kernel-mode ale sistemului de operare le oferă ca suport de dezvoltare driverelor, în condițiile în care acestea nu pot beneficia de accesul la subrutinele utilizate de aplicații (Win32) din modul user.

În tabelul 2.3 sunt prezentate componentele sistemului de operare, operațiile pentru care oferă suport şi nomenclatura subrutinelor pe care acestea le exportă.

Componetă	Oferă suport pentru...	Nomenclatura
Executiv	-alocarea memoriei -gestiunea cozilor -gestiunea listelor lookaside -thread-urile de lucru sistem	ExXxx()
HAL	-accesul la regiştrii didpozitivelor fizice accesul la magistrală	HalXxx()
I/O Manager	-suport general pentru drivere	ToXxx()
Kernel	-sincronizare -deffered procedure call(DPC)	KeXxx()
Memory Manager	-maparea spațiilor virtual şi fizic -alocarea memoriei	MmXxx()
Object Manager	-gestiunea handle-urilor	ObXxx()
Proceş Manager	-gestiunea thread-urilor sistem	PsXxx()
Rutime Library	-manipularea şirurilor -operații cu numere pe 8 octeți(large integer) -accesul la regiştrii sistemului de operare -funcții de securitate -funcții pentru gestiunea timpului şi a datei -suport pentru manipularea cozilor şi a listelor	RtlXxx() (marea majoritate)
Security Monitor	-verificarea privilegiilor -funcții pentru manipularea descriptorilor de securitate	SeXxx()

Amestecate	-servicii sistem interne	ZwXxx()

Tabelul 2.2 Subrutine suport pentru drivere

2.3.2. Driverul de sistem de fişiere.

Pentru a înțelege modul de lucru al driverelor filtru pentru sisteme de fişiere trebuie înțeles ce sunt driverele de sisteme de fişiere.

Un driver pentru un sistem de fişiere este o componentă a subsistemului de management al spațiului de stocare. Acesta furnizează mijloacele utilizatorilor pentru a stoca şi recupera informații de pe mediile nevolatile, precum hard-discurile. Aceste drivere sunt strâns legate cu subsistemele de management al memoriei si al cache-ului (Memory Manager şi Cache Manager)

Driverele pentru sisteme de fişiere diferă de celelalte drivere prin cateva caracteristici care le dau un rol special in structura sistemului de operare. Aceste caracteristici sunt:

- Driverele pentru Sisteme de Fişiere garantează sa fie apelate in contextual unui thread cerut.
- Driverele pentru Sisteme de Fişiere sunt strâns legate de subsistemele de management al memoriei si al cache-ului.
- Driverele pentru Sisteme de Fişiere sunt interconectate cu managerele de I/O şi de Obiecte (I/O şi Object Manager).
- Numai Driverele pentru Sisteme de Fişiere implementează puncte de lansare în execuție Fast I/O pentru operațiile de citire şi scriere.

Planul modelului Driver pentru Sisteme de Fişiere.

Driverele pentru Sisteme de Fişiere primesc cereri de deschidere, închidere, creare, citire şi scriere a fişierelor de pe disc. Aceste cereri de obicei încep de la procesul utilizator şi sunt expediate Sistemului de Fişiere prin intermediul subsistemului de management al I/O. Fig. 2.4 descrie cum un Driver pentru Sisteme de Fişiere furnizează servicii unui fir de execuție (thread) utilizator.

Când un fir de execuție (thread) utilizator emite un apel către o funcție de I/O, subsitemul Win32 invocă apelul serviciului corespunzător care cere operația în numele apelantului. Acum procesorul schimba nivelul de privilegiu la modul kernel. Manager-ul de I/O creează un pachet de cerere I/O (Irp) care descrie cererea de I/O şi care apelează Driver-ul pentru Sistemul de Fişiere la punctul de lansare în execuție (entry point) corespunzător. Driver-ul realizează procesarea corespunzătoare şi returnează rezultatul manager-ului de I/O, care îl returnează la rândul lui subsistemului Win32 (moment în care se schimbă din

nou nivelul de privilegiu la modul user), şi subsistemul Win32 furnizează eventual rezultatul procesului care a realizat cererea.

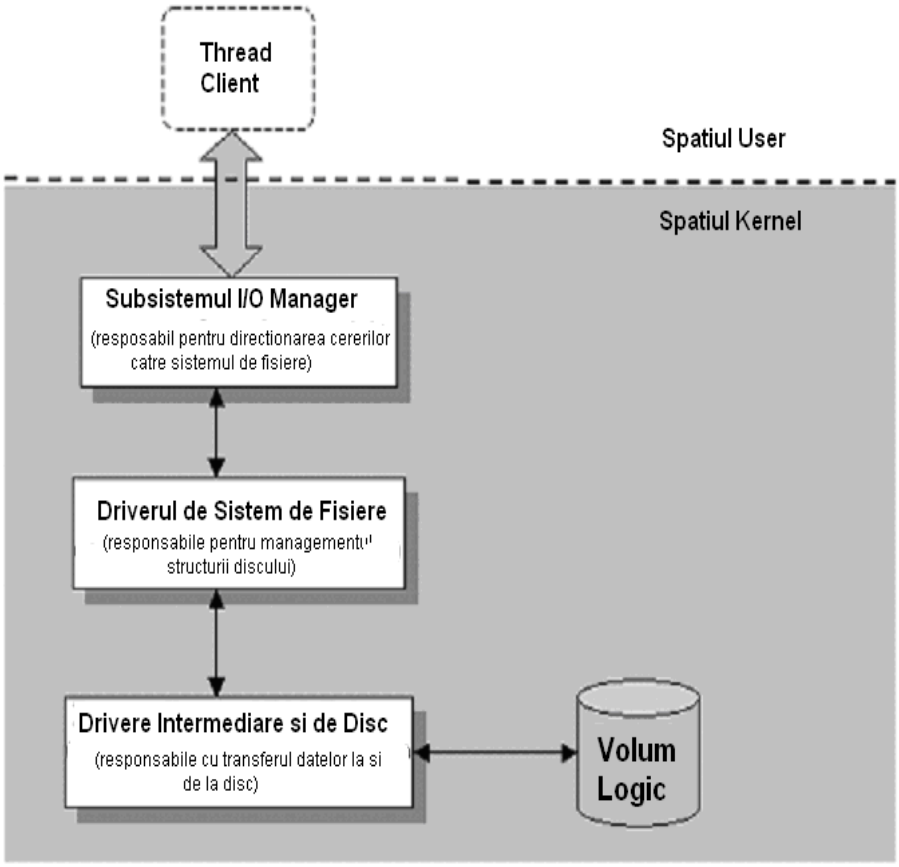


Figura 2.4 Planul modelului Driver pentru Sisteme de Fişiere

De obicei, o cerere de I/O pentru un fişier este transmisă de managerul de I/O driver-ului de sistem de fişiere folosind IRP-uri (Pachete de cereri de I/O). Oricum, cheltuielile asociate cu crearea, completarea şi distrugerea Irp-urilor este de obicei un inhibitor al bunei performanţe. De asemenea, dacă Managerul de Cache, depozitează datele, este posibil ca acele date să fie obţinute direct de la Cache-ul sistemului redirectând cererea către Managerul de Cache, în loc de a fi obţinută prin Irp-uri. I/O rapide (Fast I/O) sunt realizate dacă fişierul este găsit în Cache şi este întotdeauna o operaţie sincronă. Interesant este ca daca operaţia specifică cu fişierul nu poate fi realizată prin Fast I/O , atunci managerul de I/O foloseşte metoda Irp standard pentru realizarea operaţiei

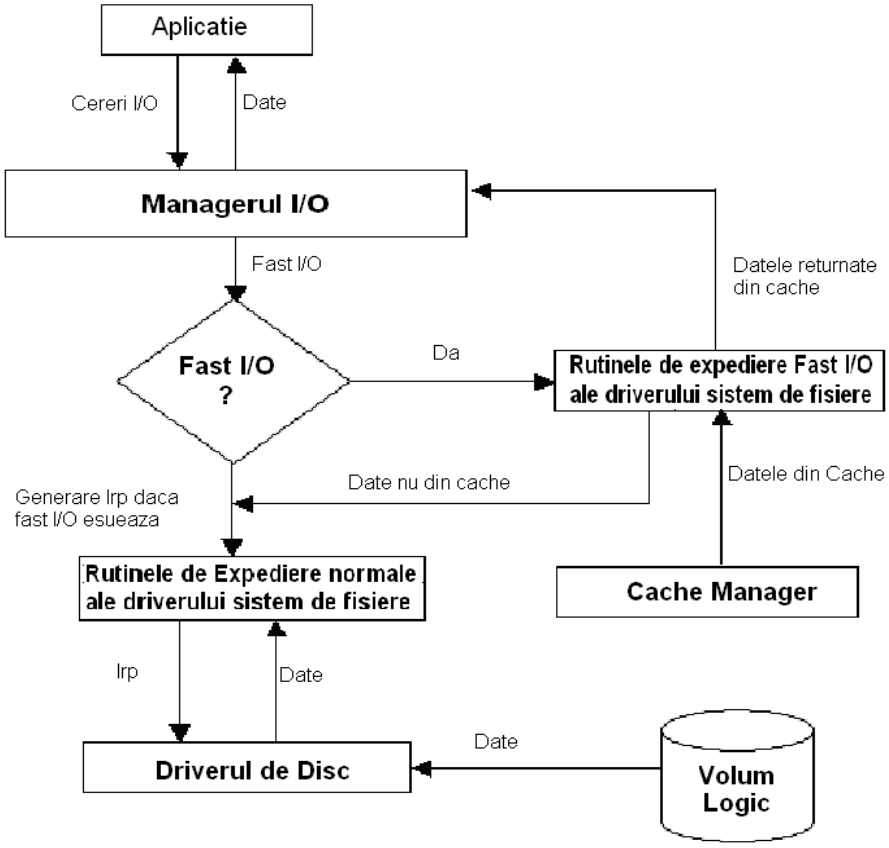


Figura 2.5 Controlul fluxului in modelul driver.

De fiecare data când managerul I/O primeşte o cerere utilizator pentru a accesa un fişier deschis, acesta invocă punctul de lansare în execuție al Fast I/O, care este de tip Boolean. Aceasta permite driverului să informeze managerul I/O daca se poate procesa cererea. Dacă da, atunci driver-ul de sistem de fişiere returnează TRUE ca stare a rutinei Fast I/O. Dacă nu, returnează FALSE ca stare a rutinei Fast I/O, în acest caz managerul de I/O creând un IRP care descrie tipul cererii I/O şi care apelează driver-ul de sistem de fişiere la entry point-ul de expediere (dispatch) corespunzător. Apoi driver-ul recepționează datele de la disk.

Managementul contextului unui thread

Driverul de sistem de fişiere este întotdeauna logic situat în capul stivei de drivere. Astfel driverul va fi întotdeauna apelat în contextul unui thread care face cererea. Aceasta garantează ca driverele de sistem de fişiere folosesc Neither I/O pentru descrierea cererilor. Implementarea Neither I/O permite unui driver de sistem de fişiere să manipuleze datele folosind adresa virtuală a procesului care face cererea. Astfel, cererile de I/O către driver trebuiesc pasate driverului în

contextul thread-ului care a inițiat cererea. Dacă un driver filtru schimbă contextul cererii, expediind cererea ca să fie procesată de un thread, driverul de sistem de fişiere va înceta să funcționeze deoarece adresa de referință din spațiul utilizator nu va mai fi validă în contextul noului thread.

Aceasta va duce la eroarea „page fault", deoarece driverul va referi o zonă de memorie inexistentă. Pentru evitarea acestei catastrofe, driverul filtur trebuie să se asigure că adresa virtuală pentru bufferul, celui care face cererea, este utilizabilă în contextul thread-ului.

Manager-ul I/O construieşte automat o MDL (Memory Descriptor List – Lista de Descriere a Memoriei) care descrie buffer-ul apelantului. MDL este o sturctură care urmăreşte paginile fizice asociate unui buffer virtual. O MDL constă dintr-un antet care descrie buffer-ul virtual, urmat de un array care listează paginile fizice asociate buffer-ului. Având, adresa virtuală în interiorul bufferului, este posibilă determinarea paginii fizice corespunzătoare.

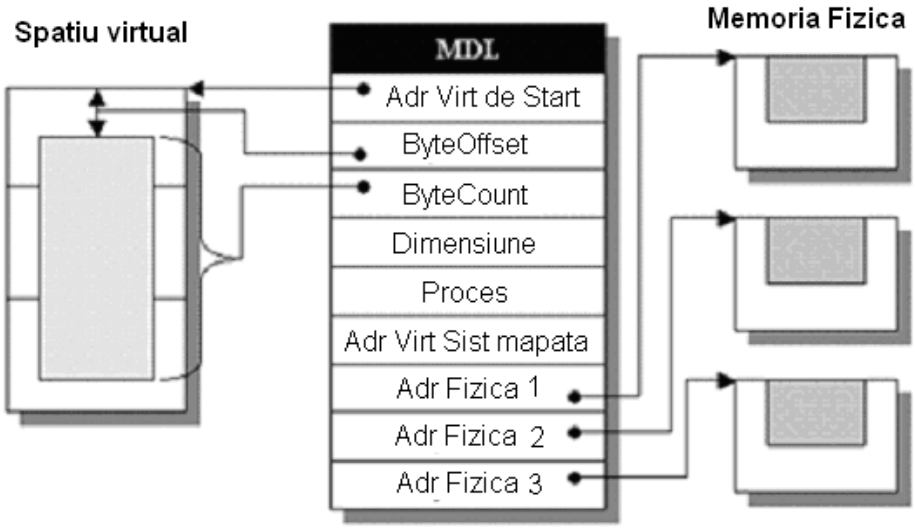


Figura 2.6 Structura unei MDL

De aceea dacă un filtru decide ca are nevoie ca buffer-ul procesului care a făcut cererea sa fie folosibil în contextul threadului, este necesară realizarea următorilor paşi:

-crearea unei MDL pentru descrierea bufferului procesului care a făcut cererea

-verificarea accesibilității paginilor care conțin bufferul şi blocarea lor în memoria fizică.

-maparea bufferului procesului care a făcut cererea descris de MDL în spațiul virtual de adrese al kernelului.

Un pointer la MDL este furnizat driver-ului într-un IRP (la IrpMdlAddres).

2.3.3. Driverul filtru de sistem de fişiere.

Mecanismul ideal de furnizare a unui I/O sigur este de a implementa un driver filtru deasupra unui driver de sistem de fişiere. Manger-ul I/O permite unui driver din modul kernel să ataşeze unul din Obiectele Dispozitiv la un Obiect dispozitiv creat de un driver diferit. Rezultatul este că IRP-ul destinat driverului asociat Obiectului Dispozitiv original, va fi trimis driver-ului asociat cu Obiectul Dispozitiv asociat. Acest driver asociat este Driverul Filtru. Acest driver filtru poate examina, modifica, completa şi pasa mai departe IRP-urile trimise driverului original.

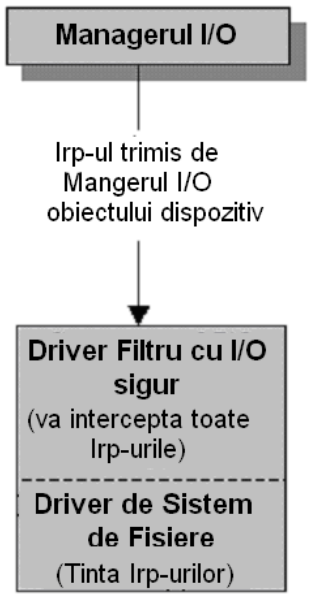


Figura 2.7 Driver filtru ataşat

De aceea un driver ataşat la un driver de sistem de fişiere va intercepta toate cererile I/O făcute către driverul de sistem de fişiere.

Driverele Filtru

Un driver filtru este un tip special la nivelul driverelor. Este diferit de celelalte drivere prin faptul că este „invisibil”. Acestea se ataşează la oricare alt driver şi interceptează cererile direcționate către obiectele dispozitiv ale driverelor de mai jos ca ierarhie. Principalul motiv pentru care sunt dezvoltate

este că aduce noi funcționalități față de cel disponibil. Driverul filtru poate folosi, de asemenea, serviciile țintei originale a cererilor de I/O sau chiar poate folosii serviciile altor drivere care lucrează in modul kernel.

Driverele filtru sunt folosite pentru a adăuga caracteristici unui dispozitiv fără sa-i modifice driverul de dispozitiv principal sau programele care folosesc acel dispozitiv. Filtrele permit modificarea unor aspecte ale unui driver fără sa-l rescrie. Necesitatea folosirii unui filtru driver poate fi uşor explicată.

De exemplu pentru dezvoltarea şi implementarea unei criptări/decriptări, directe şi în timp ce funcționează, a unui sistem de fişiere, este cea mai simplă soluție. În prezent sistemele de fişiere nu furnizează această capabilitate. Avantajul acestui filtru este că se vor putea folosi în continuare sistemele de fişiere native ale sistemului de operare Windows (de ex. FAT sau NTFS).

Cum preia controlul filtrul

De fiecare dată când este primită o cerere I/O utilizator, de către Manager-ul I/O pentru un fişier de pe un volum logic montat, Manager-ul I/O, în mod normal, înaintează cererea sistemului de fişiere care se ocupă de administrarea volumului montat.

Înainte de a înainta cererea, Managerul I/O verifică dacă alt obiect dispozitiv s-a suprapus peste obiectul reprezentând volumul logic montat şi redirectează cererea către acel obiect dispozitiv, care este în capul listei de obiecte dispozitiv. Astfel driver-ul filtru interceptează I/O înainte de a ajunge la sistemul de fişiere.

De aici, filtrul odată ataşat sistemului de fişiere, trebuie să fie garantat, filtrul interceptând toate IRP-urile trimise driverului de sistem de fişiere. Filtrul poate manipula IRP-urile apoi expediindu-le driverului driver-ului de pe nivelul inferior. Driverul de sistem de fişiere nu are nici o idee că un driver filtru este deasupra, el se comporta ca şi cum I/O Managerul trimite cererea utilizatorului direct la el.

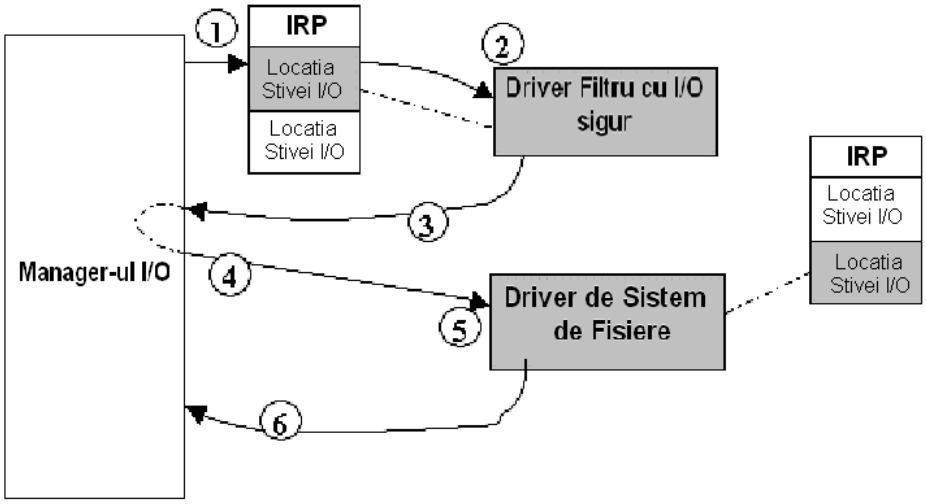


Figura 2.8 Fluxul Irp-urilor

Fig. 2.8 arată cum se desfăşoară fluxul IRP-urilor prin sistem după ce un driver filtru garantat a fost ataşat unui driver de sistem de fişiere.

Pasul 1. I/O Manager-ul creează IRP-ul pentru operația ce trebuie executată şi completează locația din stivă corespunzătoare driver-ului din nivelul cel mai de sus (în cazul nostru – filtrul).

Pasul 2. Filtrul la primirea IRP-ului face operațiile de procesare necesare.

Pasul 3. Filtrul returnează IRP-ul Manger-ului I/O copiindu-si stiva la cea mai de jos locație a stivelor driverelor.

Pasul 4. Managerul I/O pasează IRP-ul următorului driver din listă (în cazul nostru – driver-ul de sistem de fişiere).

Pasul 5. Driverul de la cel mai jos nivel procesează IRP-ul, folosind propria stiva pentru procesare.

Pasul 6. Driverul de sistem de fişiere returnează IRP-ul, I/O Managerului. Acesta eliberează atunci resursele alocate IRP-ului.

Gestionarea lățimii de bandă

Filtrul determină lățimea de bandă totală a discului, prin generarea unor cereri de citire şi estimând timpul luat până la servirea acestor IRP-uri.

Driverul filtru interceptează IRP-urile trimise driverului de sistem de fişiere, aceste IRP-uri fiind expediate conform lățimii de bandă necesară aplicațiilor. Lățimea de bandă necesară fiecărei aplicații este specificată printr-o interfață user. Lățimea de banda reprezintă numărul de Octeți care trebuie accesați, de aplicație, în număr de secunde.

2.3.4. Structura driverului filtru de sistem de fişiere.

Structura unui driver este compusă dintr-o serie de funcţii asemănător unei biblioteci. Absolut toate variabilele folosite sunt locale funcţiilor, alocarea lor fiind realizată pe stiva kernel, declararea de variabile globale sau statice fiind interzisă in cazul driverelor. O parte din funcţiile driverului sunt făcute publice componentei IO Manager prin intermediul obiectului driver care reprezintă driverul în relaţia sa cu I/O Manager-ul. Restul de funcţii care nu sunt publice sunt utilizate strict de către driver pentru realizarea anumitor sarcini care necesită o abordare structurată. Un caz special îl reprezintă driverele care îşi implementează mecanisme proprii prin care alte componente (drivere) pot accesa funcţiile sale. În continuare vor fi prezentate pe rând acele rutine pe care driverul filtru, le pune la dispoziţia I/O Manager-ului creând astfel interfaţa de acces la driver.

Valorile de retur ale funcţiilor apelate de I/O Manager trebuie să fie nişte constante definite într-un fişier header inclus în DDK, în funcţie de care Managerul I/O ia anumite decizii iar în cazul în care trebuie să dea un răspuns aplicaţiei, setează corespunzător codul de eroare Win32 care poate fi captat din aplicaţia din modul user cu funcţia GetLastError şi setează ca valoare returnată de funcţiile apelante (CreateFile, ReadFile, etc.) o valoare diferită de cea pentru succes. Intre codul de eroare captat din aplicaţie cu GetLastError şi membrul loStatus.Status al structurii IRP, există o mapare gestionată de I/O Manager. Modul de mapare, valorile codurilor de eroare Win32, valorile returnate de funcţiile driverului şi semnificaţia acestora sunt explicate în documentaţia MSDN livrată de firma Microsoft.

Rutina DriverEntry

Aceasta este singura subrutină exportată de driver la încărcarea sa în memorie şi în consecinţă numele său este fixat, toate celelalte rutine de acces la driver putând avea orice nume.

Această rutină este apelată automat de I/O Manager imediat după încărcarea driver-ului în memorie, parametrul principal transmis acesteia fiind un pointer la obiectul driver pe care I/O Managerul la instanţiat şi care reprezintă driverul.

Rolul principal al acestei rutine în cazul driverelor non-PNP este acela de a face cunoscute I/O Managerului celelalte rutine de care acesta are nevoie pentru a avea acces la serviciile oferite de driver. In cazul driverelor PNP această sarcină este lăsată în seama altei rutine, CreateDevice, pe care o face publică. Crearea interfeţei de acces la driver este realizată efectiv prin iniţializarea membrilor obiectului driver cu adresele subrutinelor pe care driverul le publică.

Pentru a realiza acest lucru este necesar ca semnătura acestor funcţii să urmeze un şablon fix.

O altă sarcină este aceea de instanţiere a obiectelor dispozitiv, fiecare dintre acestea reprezentând un dispozitiv fizic, logic sau virtual pe care driverul îşi propune să îl gestioneze. În cazul driverului filtru implementat, deoarece pot fi mai multe dispozitive referite, este necesară crearea unui management riguros al thread-urilor sistem pentru fiecare dispozitiv, astfel cererile care apar, sunt trimise spre tratare rutinei DispatchQueueIRP care pune într-o coadă orice IRP ca acesta să poată fi procesat mai târziu de către thread sau în unele cazuri imediat.

Această rutină rulează la IRQL = PASSIVE_LEVEL.

Rutina CreateRootDeviceObject

Această rutină este chemată imediat după ce rutina DriverEntry se termină şi iniţiază un Obiect Dispozitiv Radăcină. Aici se creează legătura simbolică prin care dispozitivele sunt montate din modul user.

Nivelul de privilegiu al acestei rutine precum şi a rutinei **CreateDeviceObject** este PASSIVE_LEVEL.

Rutina CreateDeviceObject

În mare iniţializează Obiectul Dispozitiv, returnând un pointer la Obiectul Dispozitiv, folosit si când nu mai este nevoie de el, pentru a putea fi şters, şi setează extensiile acestuia. Este setat câmpul *AlignmentRequirement* la FILE_WORD_ALIGNMENT ceea ce înseamnă că un buffer este aliniat în limitele unui cuvânt şi deoarece se va face un transfer cu cantităţi mari de date la un moment dat modul de transfer este setat la Direct I/O. Acest lucru îmbunătăţeşte performanţa driverului, reducând atât suprasolicitarea intreruperilor, cât şi prin eliminarea alocării memoriei şi operaţiilor de copiere inerente la Buffered I/O.

Această rutina este apelată de fiecare dată cand este primit un IRQ de montare a unui dispozitiv.

Rutina UnloadDriver

Este apelată de I/O Manager exact înainte de descărcarea din memorie a driverului. În linii mari face exact operaţiile contrare celor realizate în DriverEntry în ordinea inversă: înainte de toate demontează toate dispozitivele montate, apoi distruge thread-ul sistem, se "deconectează" de la alte drivere, şterge obiectul dispozitiv şi legătura simbolică către aplicaţii (devenind astfel "invizibil" modului user), reface starea iniţială a dispozitivului fizic controlat în caz că aceasta a fost alterată pe parcursul funcţionării, etc.

Este apelată la nivelul de privilegiu PASSIVE_LEVEL.

Rutina DeleteDeviceObject

Este rutina care şterge legătura simbolică dintre dispozitiv şi aplicație sau dacă Obiectul Dispozitiv nu este radacină este terminat thread-ul Obiectului Dispozitiv. Este apelată când apare vreo eroare de sistem la montarea unui dispozitiv sau când este demontat un dispozitv.

Rutinele StartThread, StopThread şi ThreadIRP

Prin intermediul acestora se realizează managementul thread-urilor (Crearea, Distrugerea şi Procesrea unui thread). Când este creat un Thread - acesta prin intermediul rutinei ThreadIRP face toată munca de procesare a IRP-urilor expediindu-le fie funcției **ReadWrite**, fie funcției **DeviceControl**.

Rutina ReadWrite

Rolul acestei rutine este acela de a efectua transferul de date cu dispozitivul cerut de funcția Win32 apelată (citire de date sau scriere de date la dispozitiv), criptarea şi decriptarea datelor pe funcțiile de write şi read.

Implementare acestei rutine este diferită de un ReadWrite normal, astfel pentru cererea făcută de I/O Manager (IRP) ca fiind în lucru (pending), la citire, controlul Irp-ului este redat prin apelul rutinei **Completion**.

IRP-ul construit de I/O Manager este trimis driverului ca ordin de lucru pentru realizarea unei operații de intrare, ieşire conține ca informație de bază un buffer, în care se află datele trimise din aplicație sau în care se vor depune datele pentru aplicație şi lungimea bufferului specificată în apelul Win32.

Pentru început face o verificare ca offset-ul de început plus lungimea să nu depaşească sfârşitul bufferului şi lungimea să fie multiplul mărimii unui sector. În cazul apariției unei erori este apelată rutina *IoCompleteRequest.* Această rutină este apelată de obicei când un driver termină procesarea unui IRP, Managerul I/O returnând starea operației apelantului original.

La cererea făcuta pentru scriere la dispozitiv se face criptarea buffer-ui folosind EncryptSectors (Buffer, Numarul Sectorului, Numărul de Sectoare, Cheia Secretă, Vectorul Inițial, Algoritmul de criptare).

DispatchReadWrite este apelată la nivelul de privilegiu PASSIVE_LEVEL

Rutina Completion

La întoarcearea din starea în lucru (pending) se realizează decriptarea pe funcția de read folosind DecryptSectors (Buffer, Numarul Sectorului, Numărul de Sectoare, Cheia Secretă, Vectorul Inițial, Algoritmul de criptare), după care se inițializează terminarea procesării IRP-ului.

Această rutină este apelată la un nivel de privilegiu mai mic sau egal cu DISPATCH_LEVEL.

Rutina DeviceIoControl

După ce este creată legătura între apelant ssi driverul de nivel mai jos este preluat pointerul obiectului dispozitiv, care îşi iniţializează obiectele dispozitiv, precum ssi un pointer către obiectul fişier corespunzător. Astfel driverul filtru se ataşează deasupra driverului de nivel mai jos.

Apoi este apelată funcţia *IoBuildDeviceIoControlRequest* care setează IRP-urile pentru cererile de control al dispozitivului ca să fie trimis sincronizat driverului de nivel mai jos, adică driverului de sistem de fişiere. IRP-urile sunt puse într-o coadă de IRP-uri, specifică thread-ului curent. Dacă thread-ul este nevoit să se termine, Managerul I/O anulează IRP-urile.

Parametrii primiţi de funcţia DeviceloControl sunt:

• handle-ul către driver

• codul de control (IOCTL) transmis de driverul filtru

• adresa buffer-ului care va fi transmis ca dată de intrare driverului; dacă nu sunt transmise date de intrare, este setat la NULL;

• lungimea buffer-ului de intrare; în caz că nu sunt date de intrare este setată la 0:

• adresa buffer-ului de ieşire în care vor fi depuse datele de ieşire ale driverului în cadzul în care nu există date de ieşire această adresă este NULL;

•lungimea buffer-ului de ieşire; în caz ca nu sunt date de ieşire este setată la 0;

Rutina DeviceControl

Această rutina manipulează anumite cereri venite de la sistemul de operare, necesara sistemului pentru a recunoaşte un drive, şi de asemenea se ocupă şi de funcţiile specifice dispozitivului nostru precum montarea şi demontarea.

Este apelată de I/O Manager atunci când o aplicaţie face apel la funcţia DeviceloControl. Aceasta reprezintă o modalitate prin care, o anumită aplicaţie care încearcă să schimbe date cu un driver de sistem fişiere, este interceptată, anumite cereri fiind manipulate sau chiar fiind adăugate noi funcţionalităţi.

2.3.5. Accesul aplicaţiilor la driverul filtru de sistem de fişiere.

Anterior a fost prezentată interfaţa pe care driverul o oferă sistemului de operare (I/O Manager-ului) pentru a avea acces la serviciile oferite de acesta. In continuare va ti prezentată interfaţa pe care sistemul de operare (I/O Manager-ul) o oferă aplicaţiilor pentru ca acestea sa poată beneficia de serviciile oferite de driverul filtru.

Accesul la driver este realizat utilizând o interfață standard pe care Windows 2000 o pune la dispoziție aplicațiilor user-mode pentru accesul la o gamă largă de surse de date: fişiere, pipe-uri, drivere, etc. Cele mai utilizate funcții ale acestei interfețe, sunt:

1.*CreateFile* - are rolul de livra aplicației un handle în sursa de date în funcție de numele pe care aceasta îl face vizibil aplicațiilor(cale fişier, legătură simbolică, etc); acest handle este utilizat de toate celelalte funcții Win32 ale interfeței;

2.*ReadFile, WriteFile* - au rolul de transfera date între aplicație şi sursa de date la care s-a obținut un handle cu CreateFile;

3.*DeviceloControl* - are rolul de a transmite coduri de control (IOCTL);

4.*CloseHandle* - şterge handle-ul obținut cu CreateFile din tabela de handle-uri a aplicației, făcând astfel inaccesibilă sursa de date până la un nou apel al funcției CreateFile;

În continuare sunt prezentate codurile de control (IOCTL) tratate de către driverul filtru. Acestea sunt împărțite în două categorii, cele din prima parte sunt cele care sunt manipulate, iar în a doua parte controalele private. Pentru fiecare IOCTL vor fi descrişi parametrii de intrare , cei de ieşire şi codul de eroare returnat de funcția Win32 GetLastError.

IOCTL_MOUNTDEV_QUERY_DEVICE_NAME

Este obligatoriu suportul pentru acest control pentru realizarea accesului clientului care administrează montarea. La primirea acestui IOCTL driverul trebuie sa furnizeze numele dispozitivului pentru volum, care va fi folosit de către aplicația client ca țintă pentru legătura simbolică. De exemplu numele dispozitivului ar putea fi "\Device\HarddiskVolume1".

Ca intrare primeşte Parameters.DeviceIoControl.OutputBufferLength aceasta indicând dimensiunea în Octeți a buffer-ului de ieşire. La ieşire clientul care administrează montarea returnează o structură de tipul MOUNTDEV_NAME, în partea de început a buffer-ului Irp->AssociatedIrp.SystemBuffer. Numele dispozitivului este introdus la adresa indicată de membrul Name al acestei structuri. În cazul în care buffer-ul de ieşire este prea mic ca să încapă numele dispozitivului va fi returnată starea de STATUS_BUFFER_OVERFLOW.

IOCTL_MOUNTDEV_QUERY_UNÎUE_ID

Este obligatoriu suportul pentru acest control pentru realizarea accesului al clientului care administrează montarea. La primirea acestui IOCTL driverul trebuie sa furnizeze un identificator unic pentru volum sau dispozitiv.

Ca intrare primeşte Parameters.DeviceIoControl.OutputBufferLength aceasta indicând dimensiunea în Octeți a buffer-ului de ieşire. La ieşire este returnată o structură de tipul MOUNTDEV_UNÎUE_ID, în partea de început a buffer-ului Irp->AssociatedIrp.SystemBuffer. În cazul în care buffer-ul de ieşire este prea mic ca să încapă numele dispozitivului va fi returnată starea de STATUS_BUFFER_OVERFLOW.

IOCTL_MOUNTDEV_QUERY_SUGGESTED_LINK_NAME

Uneori clientul care administrează montarea poate păstra litera de drive după ce sistemul a fost restartat, fără ajutorul managerului de mount. Ca răspuns la acest IOCTL este sugerată o literă de drive managerului de mount. Dacă nu există deja în baza de date a managerului de mount, atunci este folosită litera sugerată, altfel este ignorată. Litera de drive include calea completă a legăturii simbolice ssi are sintaxa clasică MS-DOS. Astfel, de exemplu, litera de drive „E” este reprezentată ca "\DosDevices\E:"

Ca intrare primeşte Parameters.DeviceIoControl.OutputBufferLength aceasta indicând dimensiunea în Octeți a buffer-ului de ieşire. La ieşire clientul care administrează montarea returnează o structură de tipul MOUNTDEV_SUGGESTED_LINK_NAME, în partea de început a buffer-ului Irp->AssociatedIrp.SystemBuffer. Numele dispozitivului este introdus la adresa indicată de membrul Name al acestei structuri. În cazul în care buffer-ul de ieşire este prea mic ca să încapă numele dispozitivului va fi returnată starea de STATUS_BUFFER_OVERFLOW.

IOCTL_DISK_GET_DRIVE_GEOMETRY

Returnează geometria drive-ului pentru disc. Valorile returnate sunt făcute să se potrivească dimensiunii discului. Acestea sunt tipul, numărul cilindrilor, numărul sectoarelor pe o pistă şi numărul Octeților dintr-un sector.

Ca intrare primeşte Parameters.DeviceIoControl.OutputBufferLength aceasta indicând dimensiunea în Octeți a buffer-ului de ieşire. La ieşire driverul returnează o structură de tipul DISK_GEOMETRY, în buffer-ul de la Irp->AssociatedIrp.SystemBuffer. Returnează STATUS_SUCCESS sau în cazul că apar erori următoarele stări

STATUS_UNRECOGNIZED_MEDIA,
STATUS_INVALID_PARAMETER,
STATUS_INVALID_DEVICE_REQUEST,
STATUS_INFO_LENGTH_MISMATCH,
STATUS_INSUFFICIENT_RESOURCES,
STATUS_BUFFER_TOO_SMALL.

IOCTL_DISK_GET_PARTITION_INFO

Returnează informații despre tipul, dimensiunea şi natura unei partiții.

Ca intrare primeşte Parameters.DeviceIoControl.OutputBufferLength aceasta indicând dimensiunea în Octeți a buffer-ului de ieşire. La ieşire driverul returnează o structură de tipul PARTITION_INFORMATION, în buffer-ul de la Irp->AssociatedIrp.SystemBuffer.

IOCTL_DISK_GET_DRIVE_LAYOUT

Returnează informații despre numărul de partiții, semnătura unui disc şi caracteristicile fiecărei partiții de pe disc. La ieşire driverul returnează o structură de tipul DRIVE_LAYOUT_INFORMATION.

IOCTL_DISK_IS_WRITABLE

Setează dacă tipul poate fi scris sau nu.

IOCTL-urile private.

OPEN_TEST:

Aceasta are un nume reprezentativ şi ceea ce face este deschiderea unui dispozitiv pentru a verifica buna funcționare a driverului.

WIPE_CACHE:

Şterge complet memoria cache a driverului. În aceasta sunt stocate maximum 4 parole.

CACHE_STATUS:

Returnează starea cache-ului driverului care poate fi STATUS_SUCCESS dacă există informații despre parole în memoria cache.

MOUNT_LIST:

Returneazăa o listă cu volumele criptate montate, în structura MOUNT_LIST_STRUCT. Aceasta are următoarele proprietăți:

```
unsigned long ulMountedDrives;      /* Zonă de biți cu literele tuturor
                                       drive-urilor montate */
short wszVolume[26][64];            /* Numele Volumelor montate */
unsigned __int64 diskLength[26];/* Dimensiune disc (drive montat) */
int ea[26];                         /* Algoritm criptografic */
BOOL hiddenVol[26];                 /* Dacă volumul este de tipul ascuns */
```

VOLUME_PROPERTIES:

Returnează proprietățile volumului montat, pentru aceasta se foloseşte structura VOLUME_PROPERTIES_STRUCT cu urmatoarele proprietăți:

```
int driveNo;                    /*numarul driveului*/
short wszVolume[64];            /* numele volumului*/
unsigned __int64 diskLength; /*dimesiunea volumului*/
int ea;                         /*o structură care reprezintă algoritmul de criptare */
int pkcs5;                      /*funcția de hash folosită */
BOOL hiddenVolume;              /*dacă volumul este ascuns */
BOOL readOnly;                  /*dacă volumul este deschis doar pentru
                                citire */
unsigned __int64 volumeCreationTime;    /*data creări volumului*/
unsigned __int64 headerCreationTime;    /*data creări antetului*/
```

RESOLVE_SYMLINK:

Realizează o legatură simbolică cu ținta. Structura folosită RESOLVE_SYMLINK_STRUCT are urmatoarele proprietăți:

```
WCHAR symLinkName[MAX_PATH]; /*numele legăturii simbolice*/
WCHAR targetName[MAX_PATH];     /*numele țintei*/
```

MOUNT:

Comanda pentru montarea unui volum. Aceasta foloseşte structura MOUNT_STRUCT cu proprietățile:

```
int nReturnCode;                /*Codul de reîntoarcere de la driver */
short wszVolume[MAX_PATH];      /* Numele volumului ce va fi
                                montat[max 260 caractere] */
char szPassword[MAX_PASSWORD + 1];  /* Parola [max 64+1] */
int nPasswordLen;               /* Lungimea parolei */
BOOL bCache;            /* Daca există parole în cache-ul driverului */
int nDosDriveNo;        /* Numărul de drive care va fi montat */
BOOL bMountReadOnly;    /* Daca montarea volumului va fi în
                        modul read-only */
BOOL bMountRemovable;   /* Daca montarea volumului va fi ca
                        removable media */
BOOL bExclusiveAccess;  /* Deschide fissierul/dispozitivul în modul
                        de acces  exclusive */
BOOL bMountManager;  /* Anunttă volumul managerului de mount */
long time;              /* Timpul când a fost montat volumul */
```

Când este cerut controlul mount este apelată rutina **MountDevice**, care face o verificare dacă este introdusă o literă de drive bună, creează Obiectul dispozitiv pe baza structurii mount, initializează Extensiile, crează un thread, apoi este anuntat Managerul de mount de venirea noului volum ssi, în plus, este creată o legatura simbolica cu Managerul de mount.

Pot apărea următoarele erori:

„ERR_BAD_DRIVE_LETTER" – când litera de drive nu este corectă

„CREATE_DEVICE_ERROR" – când nu poate fi creat Obiectul dispozitiv

„STATUS_INSUFFICIENT_RESOURCES" – canq nu pot fi alocate resursele pentru începerea unui thread.

UNMOUNT:

Demontează un volum. Aceasta foloseşte structura UNMOUNT_STRUCT cu proprietățile:

```
int nDosDriveNo;          /* Litera de drive care va fi demontată*/
BOOL ignoreOpenFiles;
int nReturnCode;          /*Codul de reîntoarcere de la driver */
```

Când este cerut controlul mount este apelată rutina **UnmountDevice**, după deschiderea şi validarea handle-ului volumului, întâi blochează volumul, iar apoi trece la demontarea acestuia şi la ştergea legăturii simbolice.

UNMOUNT_ALL:

Demontează toate volumele, montate prin intermediul acestui driver. Este folosită rutina **UnmountAllDevices,** care conform listei de drive+uri montate face unmount pentru fiecare în parte.

Capitolul 3.
Noțiuni de criptografie

3.1. Introducere în criptografie

Criptografia are o istorie lungă şi pitorească. În această secțiune, voi schița doar câteva dintre aspecte, ca informații de bază pentru ceea ce urmează.

Din punct de vedere istoric, la arta criptografiei au contribuit patru grupuri de oameni: armata, corpurile diplomatice, cei ce au ținut jurnale şi îndrăgostiții. Dintre acestea, armata a avut rolul cel mai important şi a structurat domeniul. În interiorul organizațiilor militare, mesajele ce trebuiau criptate erau de obicei date unor funcționari codori prost plătiți, pentru codare şi transmitere. Volumul de mesaje nu permitea ca această muncă să fie făcută doar de câțiva specialişti de elită.

Până la apariția calculatoarelor, una din marile constrângeri ale criptografiei a fost capacitatea funcționarilor codori de a realiza transformările necesare, adeseori pe câmpul de luptă, cu echipament redus. O constrângere suplimentară a fost dificultatea de comutare rapidă de la o metodă la alta, deoarece acesta implica reantrenarea unui număr mare de oameni. Totuşi pericolul ca un cod să fie capturat de către inamic a făcut să devină esențială posibilitatea de a schimba metoda criptografică imediat, în caz de nevoie.

Mesajele ce trebuie criptate, cunoscute sub numele de text clar (plain text), sunt transformate printr-o funcție parametrizată de o cheie (key). Ieşirea procesului de criptare , cunoscută sub numele de text cifrat (ciphertex), este apoi transmisă, adeseori prin curier sau radio. Presupunem că intrusul ascultă şi copiază cu acuratețe întreg textul cifrat. Totuşi, el nu ştie care este cheia de criptare şi astfel el nu poate decripta prea uşor textul cifrat. Uneori intrusul poate nu numai să asculte canalul de comunicație (intrus pasiv), ci şi să înregistreze mesajele şi să le retransmită mai târziu, să injecteze propriile sale mesaje sau să modifice mesajele legitime înainte ca ele să fie preluate de receptor (intrus activ). Arta de a sparge cifrul se numeşte criptanaliză (cryptanalysis). Arta de a concepe cifruri (criptografia) şi cea de a le sparge (criptanaliză) sunt cunoscute sub numele de colectiv de criptologie (cryptology).

Criptografia poate fi considerată ca studiul de cum să amesteci datele în aşa fel încât:

- oricine fură sau interceptează datele amestecate să nu poate să le refacă

- dar persoanele cărora le sunt destinate acele date să poată să le refacă cu uşurinţă

3.2. Găuri de securitate care pot fi exploatate.

Nici un sistem de securizare nu este complet sigur. Întotdeauna există metode de infiltrare a unui atacator în sistem. Se pot aminti câteva:

1. În primul rând este bine de avut grijă la aplicaţiile care au scurgeri de date. Pot exista bug-uri în program prin care anumite date importante să fie salvate pe disc.

2. Atenţie mare şi la datele care nu au fost şterse sigur. Orice fişier şters poate fi recuperat destul de uşor. Chiar şi fişierele care au fost şterse "sigur" este posibil de a fi recuperate prin scanarea suprafeţei discului. Fişierele şterse ar trebui securizate, de exemplu după ştergere zona unde era fişierul de şters să fie umplută cu date complet aleatoare.

3. Fişierele temporare şi cele de swap pot fi de asemenea o sursă de scurgere de date. Acestea trebuiesc de asemenea şterse sigur.

4. O metodă mai neobişnuită ar fi atacurile de tip "Vijelie". Ce înseamnă aceasta? Teoretic, emisiile electromagnetice ale monitorului, hard discului şi chiar ale tastaturii pot fi detectate şi înregistrate de la distanţă. Aceasta permite unui ascultător să vadă ce este pe ecran sau să detecteze parola în timp ce este tastată.

5. *Brute Force* (Forţa Brută). Există două metode de forţă brută (în mare înseamnă generarea tuturor sau a cât mai multor variante posibile până la reuşirea decriptării) : cea care se încearcă asupra parolei şi încercarea forţei brute asupra algoritmului. Pentru contrarea acestui atac este importantă folosirea unor parole care să fie greu de ghicit, de asemenea ajută daca sunt folosite atât caracterele mari cât şi cele mici, caracterele speciale şi cifrele. Este e muncă dificilă care va necesita multe operaţii.

6. Unii algoritmi pot fi sensibili la atacuri necunoscute publicului larg. Agenţiile de securitate **ar putea** deţine mecanisme mai rapide decât aplicarea forţei brute, sau care exploatează slăbiciunile algoritmilor.

7. Un atac credibil şi probabil eficient, este utilizarea forţei brute asupra persoanei care deţine parola, până ce aceasta este revelată. Factorul uman este, de obicei, cea mai mare gaură de securitate. În acelaşi scop pot fi folosite de asemenea şi aşa numitele „seruri ale adevărului”. Şi aceste metode pot fi contracarate printr-un mecanism care să asigure aşa numita “negare plauzibilă”.

3.3. Detalii tehnice necesare securizării

3.3.1. Notaţii

C *Blocul cifrat*
$D_k()$ Algoritmul de decriptare folosind cheia de *K*
$E_k()$ Algoritmul de criptare folosind cheia de *K*
H() Funcţia Hash (ex. RIPEMD-160)
i Index-ul blocului pentru blocuri de *n*-biţi; *n* este dependent de context
K Cheia de criptare/decriptare
m Index-ul blocului pentru blocuri de 64-biţi
n Index-ul blocului pentru blocuri de 128-biţi
P Bloc cu text în clar
R Random
W Valoare ptr XOR
^ Sau Exclusiv (XOR)
$\oplus$ Adunare modulo 2^n, unde *n* este mărimea în biţi a celui mai din stânga operand şi a valorii rezultate (ex., daca operandul stâng este o valoare pe 1-bit, iar cel drept este o valoare pe 2-biţi, atunci: $1 \oplus 0 = 1$; $1 \oplus 1 = 0$; $1 \oplus 2 = 1$; $1 \oplus 3 = 0$; $0 \oplus 0 = 0$; $0 \oplus 1 = 1$; $0 \oplus 2 = 0$; $0 \oplus 3 = 1$)
|| Concatenare

3.3.2. Schema de Criptare

La montarea unui volum se execută următorii paşi (aceasta dacă nu sunt parole în cache):

1. Primii 512 Octeţi (de ex. antetul unui volum standard) sunt citiţi în RAM, din care primii 64 Octeţi reprezintă valoarea de *salt* (Mai multe detalii la Specificaţii privind formatul unui volum).
2. 512 Octeţi la offset-ul 1536 de la sfârşitul unui volum sunt citiţi în RAM. Dacă acolo se află un volum ascuns, în acest punct se ştie antetul acestuia (indiferent dacă acolo este un volum ascuns trebuie determinat prin decriptarea acestor date).

3. Acum se încearcă decriptarea antetului volumului standard şi posibilul antet al unui volum ascuns. Toate datele folosite şi generate în timpul procesului de decriptare sunt ţinute în RAM. Următorii parametrii nu se cunosc şi pot fi determinaţi prin intermediul metodei aproximărilor succesive (adică să fie testate toate combinaţiile posibile ale următoarelor):
   a. Funcţia pseudo-aleatoare folosita pentru derivarea cheii de antet, care poate să fie HMAC-SHA-1 sau AC-RIPEMD-160. Parola introdusă de utilizator şi valoarea de *salt* aflată în primii paşi sunt pasate funcţiei de derivare a cheii de antet, care produce o secvenţă de valori din care sunt derivate cheia de criptare a antetului şi vectorul iniţial IV-ul (valori folosite pentru decriptarea antetului volumului).
   b. Algoritmii de criptare AES-256, Blowfish, CAST5, Serpent, Triple DES, Twofish.
   c. Tipul operaţiei: CBC sau outer-CBC
   d. Dimensiunea unui bloc.
   e. Dimensiunea cheilor.
4. Decriptarea este cu succes dacă primii 4 Octeţi din datele decriptate conţine şirul ASCII "TRUE" şi dacă suma de control CRC-32 a ultimilor 256 de Octeţi din datele decriptate ale antetului volumului este egală cu valoarea situată la în al 8-lea Octet din datele decriptate. Dacă aceste condiţii nu sunt îndeplinite montarea eşuează (principalele motive fiind o parolă incorectă, un volum corupt sau un tip de volum necorespunzător formatat).
5. Acum se presupune că avem parola corectă, algoritmul de criptare, modul, dimensiunea cheii şi a blocului, şi funcţia de derivare corectă a cheii de antet. De asemenea se ştie dacă se montează un volum ascuns sau nu. Mai exista şi verificarea versiunii.
6. Rutina de criptare este reiniţializată cu cheia principală şi cu vectorul iniţial obţinut din antetul volumului decriptat. Această cheie poate fi folosită pentru decriptarea oricărui sector al volumului, mai puţin zona cu antetul volumului(care a fost criptat folosind cheia de antet). Apoi volumul este montat.

3.3.3. Modul de cifrare

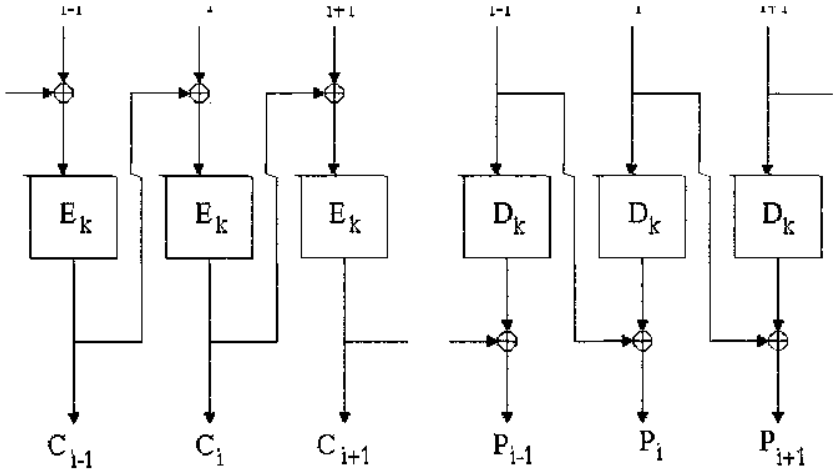
Există două moduri principale de utilizare în practică a algoritmilor simetrici; cifrarea bloc şi cifrarea şir de caractere (secvenţială). Cifrarea bloc operează cu blocuri de text clar şi cifrat - de regulă de 64 de biţi dar, uneori, şi

mai mari. Cifrarea secvențială operează cu secvențe de text clar şi cifrat de un bit sau octet (uneori chiar şi cu cuvinte de 32 de biți), în cazul cifrării bloc, aceiaşi bloc de text clar va fi cifrat de fiecare dată în acelaşi bloc de text cifrat, folosind aceeaşi cheie, în cazul cifrării secvențiale, secvențe similare de text clar vor fi cifrate diferit, în cazul unor cifrări repetate.

Pentru această aplicattie a fost aleasă cifrarea pe blocuri CBC (Cipher Block Chaining – Cifrare bloc cu Înlănțuire) deoarece aduce un surplus de securitate şi este perfect pentru criptarea datelor de mari dimensiune precum fişierele sau partițiile. De asemenea are o viteză ridicată.

Modul CBC adaugă mecanismului de criptare un bloc de reacție. Rezultatul rotarii unui bloc anterior revine prin buclă şi intervine în criptarea blocului curent. În felul acesta, textul cifrat nu mai depinde doar de textul clar, ci şi de modul de cifrare a blocului anterior.

Figura 3.1 prezintă modul de criptare CBC. După ce blocul de text clar este criptat, textul cifrat rezultat este stocat într-un registru al buclei de reacție, înainte ca următorul text clar să fie criptat, el este făcut XOR cu blocul din registrul de reacție şi devine următoarea intrare în rutina de criptare. După criptare, conținutul registrului este înlocuit cu blocul criptat. În acest fel criptarea blocului *i* depinde de toate cele i-1 blocuri anterioare.



(a) **Criptare CBC** **(b) Decriptare CBC**

Fig. 3.1 Modul *Cipher Block Chaining*

De asemenea este folosit şi un mod de cifrare multiplă, outer CBC.

În următorul tabel pot fi văzutti algoritmii de criptare şi modul de cifrare corespunzător fiecăruia.

Algoritmul de Criptare	Modul de Cifrare	Detalii despre Modul de Cifrare
AES-256	CBC	$C_i = E_k(P_i \wedge C_{i-1})$; C_0 este IV.
Blowfish	CBC	$C_i = E_k(P_i \wedge C_{i-1})$; C_0 este IV.
CAST5	CBC	$C_i = E_k(P_i \wedge C_{i-1})$; C_0 este IV.
Serpent	CBC	$C_i = E_k(P_i \wedge C_{i-1})$; C_0 este IV.
Triple DES	Outer-CBC	$C_i = E_{k3}(D_{k2}(E_{k1}(P_i \wedge C_{i-1})))$; C_0 este IV.
Twofish	CBC	$C_i = E_k(P_i \wedge C_{i-1})$; C_0 este IV.

Tabelul 3.1 Algoritmi şi modurile de cifrare

3.3.3. IV (Vectorul Inițial)

Este întotdeauna un număr aleator, unic pentru fiecare sector (dimensiunea unui sector este 512 Octeți) şi volum. Această valoare este generată astfel:

1. Octeții 256-263 (pentru un cifru pe bloc de 128 –biți este de la 256-271) din antetul decriptat al volumului sunt citiți. Dacă cifrurile în cascadă folosesc mai mult de un IV, următorii octeți reprezintă vectorii inițiali suplimentari (de exemplu pentru 3 cifruri pe 128 -biți în cascadă cu modul inner-CBC, primul IV primeşte Octeții 256-271, al doilea 272-287 şi 288-303 pentru al treilea IV).
2. Datele recuperate la pasul 1. sunt făcute XOR cu numărul de sector pe 64-biți (fiecare sector are 512 Octeți, şi sunt numerotate de la 0). În cazul unui cifru cu bloc de 128 –biți, valoarea recuperate pe 128 -biți este împărțita în două cuvinte de 64 –biți care sunt făcute XOR cu o valoare identică. Rezultatul pe 64, respective 128 –biți reprezintă IV.

De exemplu:

Ptr. un cifru pe bloc de 128 –biți
T1 = primii 64-biți ai valorii recuperate la pasul (1)
T2 = următorii 64-biți ai valorii recuperate la pasul
S = numărul sectorului (întreg pe 64-biți)
IV = (T1 ^ S) || (T2 ^ S)

Ptr. un cifru pe bloc de 64 –biți
T = valoarea pe 64-biți recuperată la pasul (1)

S = numărul sectorului (întreg pe 64-biți)
IV = T ^ S

Pasul (1) este realizat doar o data, imediat după montare, datele recuperate rămânând în RAM.

3.3.4. Whitening (înălbirea)

Pentru a fi şi mai dificilă obținerea din text cifrat , textul în clar, este folosită tehnica, care se mai numeşte şi "whitening". Fiecare 8 octeți ai unui sector(după ce acesta a fost criptat) sunt făcuți XOR cu o valoare pe 64 de biți, unica fiecărui sector şi volum. Valoarea este generată după cum urmează:

1. Octeții 264-271 (pentru un cifru pe 128 –biți octeții 272-279) sunt recuperați din antetul decriptat al volumului.
2. Octeții 272-279 (pentru un cifru pe 128 –biți octeții 280-287) sunt recuperați din antetul decriptat al volumului.
3. Datele recuperate la pasul (1) sunt făcute XOR cu numărul de sector pe 64 –biți.
4. Datele recuperate la pasul (2) sunt făcute XOR cu numărul de sector pe 64 –biți.
5. Este calculată valoarea pe CRC-32 pe 32 –biți a primilor 8 octeți ai valorii rezultate la pasul (3).
6. Este calculată valoarea pe CRC-32 pe 32 –biți a următorilor 8 octeți ai valorii rezultate la pasul (3).
7. Este calculată valoarea pe CRC-32 pe 32 –biți a primilor 8 octeți ai valorii rezultate la pasul (4).
8. Este calculată valoarea pe CRC-32 pe 32 –biți a următorilor 8 octeți ai valorii rezultate la pasul (4).
9. Valoarea calculată la pasul (5) este făcută XOR cu cea de la pasul (8).
10. Valoarea calculată la pasul (6) este făcută XOR cu cea de la pasul (7).
11. Valoarea pe 32 –biți calculată la pasul (9) reprezintă primii 32 de biți din valoarea de whitening, iar valoarea pe 32 –biți calculată la pasul (10) reprezintă următorii 32 de biți din valoarea de whitening.

Paşii 5-8 sunt parcurşi pentru a creşte distanța Hamming (numărul de biți prin care diferă două şiruri binare, se poate scrie ca fiind A şi B două şiruri binare distanța este $\sum$ | Ai - Bi |) dintre două valori de înălbire.

Pe scurt:

T1 = valoarea pe 64-biți recuperată la pasul (1)
T2 = valoarea pe 64-biți recuperată la pasul (2)
S = numărul sectorului (întreg pe 64-biți)
T1 = T1 ^ S
T2 = T2 ^ S
Q1 = primii 32-bițti ai T1
Q2 = următorii 32-bițti ai T1
Q3 = primii 32-bițti ai T2
Q4 = următorii 32-bițti ai T2
W = (CRC32(Q1) ^ CRC32(Q4)) || (CRC32(Q2) ^ CRC32(Q3))

Înălbirea este aplicată astfel:

Pentru un cifru pe 128-biți

T1 = primii 64-biți ai C
T2 = următorii 64-biți ai C
C' = (T1 ^ W) || (T2 ^ W)

Pentru un cifru pe 64-biți

C' = C ^ W

3.3.5. Algoritmii de criptare

Pentru criptarea datelor au fost aleşi un număr de 6 algoritmi simetrici.

AES

AES (Advanced Encryption Standard sau Standardul Evoluat de Criptare (numit şi Rijndael, după creatorii lui Joan Daemen şi Vincent Rijmen, care l-au publicat în 1998) specifică un algoritm criptografic aprobat de FIPS (Federal Information Processing Standards) şi poate fi folosit de către departamentele şi agențiile federale din SUA pentru protecția criptografică a informațiilor sensibile, dar nesecrete. Acest algoritm poate avea cheile ssi blocurile de dimensiune variabilă.

Folosit este AES cu cheia pe 256 –biți şi 14 runde. Acesta este anunttat în iunie 2003, după ce NSA a făcut revizuirea şi o analiză asupra acestuia, că este suficient de puternic pentru a proteja informațiile secrete catalogate ca TOP-SECRETE. Astfel produsele care incorporează AES încep sa fie considerate satisfăcătoare privind cerințele de Securitate a Informațiilor asociate cu protecția sistemelor de securitate şi/sau a informațiilor de securitate națională.

Blowfish

Este realizat de către Bruce Schneier în 1993. Nu este patentat, codul fiind liber pentru toți utilizatorii. Are cheia pe 448 –biți şi 16 runde. Acest cifru este cel mai rapid cifru dintre cei implementați.

CAST5

CAST5 sau CAST-128 a fost creat de către Carlisle Adams şi Stafford Tavares în 1997. Are cheia pe 128 –biți, iar blocul de 64 –biți. Acest cifru este patentat, dar este liber a fi folosit atât pentru aplicațiile necomerciale cât şi pentru cele comerciale. Este de asemenea algoritmul oficial folosit de Guvernul canadian pentru protecția datelor nesecrete.

Serpent

Creat de către Ross Anderson, Eli Biham şi Lars Knudsen. A fost publicat în 1998. Are cheia pe 256 –biți, iar blocul de 128 –biți. Serpent a fost unul dintre finaliştii pentru AES. Acesta nu a fost propus ca să fie noul algoritm standard cu toate că părea că are o margine de securitate mai mare decât al lui Rijndael, care a câştigat. Algoritmul Rijndael chiar a fost criticat, sugerându-se că, în viitor, structura matematică poate fi atacată.

Echipa care a creat algoritmul Twofish a făcut un tabel cu coeficienții de siguranță ai algoritmilor finalişti. Coeficientul de siguranță este definit ca numărul de runde a unui cifru de împărțit la cel mai mare număr de runde care au fost sparte. Astfel, dacă un cifru spart are cel mai mic coeficient de siguranță care este egal cu 1, Serpent a avut coeficientul cel mai mare, 3.56 pentru toate dimensiunile de chei suportate, iar Rijndael-256 a avut 1.56.

În ciuda acestui lucru, Rijndael a fost considerată alegerea cea mai bună avându-se ipn vedere combinațiile dintre performanță, eficiență, implementabilitate şi flexibilitate. La sfârşit Rijndael a primit 86 de voturi, Serpent 59, Twofish 31, RC6 23 şi MARS 13.

Triplu DES

Publicat în 1978. Este un algoritm cu un mod de criptare tripla (Outer-CBC), are 3 iterații a algoritmului DES (criptare-decriptare-criptare). Foloseşte 3 chei diferite de 56 –biți.

De remarcat că acest algoritm este mai încet.

Twofish

Creat de Bruce Schneier, David Wagner, John Kelsey, Niels Ferguson, Doug Whiting, şi Chris Hall a fost publicat în 1998. Are cheia pe 256 –biți şi blocul pe 128 –biți. Acest cifru foloseşte cutii S dependente de cheie. Acest

algoritm poate fi privit ca o colecție de 2^{128} de sisteme criptografice diferite, unde 128 de biți derivați dintr-o cheie de 256 –biți, controlează sistemul criptografic selectat. Acest lucru poate constitui o formă de securitate împotriva unor atacuri necunoscute.

Algoritm	Dezvoltați de către	Dimensiunea Cheii (Biți)	Dimensiunea Blocului (Biți)	Mod
AES	J. Daemen, V. Rijmen	256	128	CBC
Blowfish	B. Schneier	448	64	CBC
CAST5	C. Adams, S. Tavares	128	64	CBC
Serpent	R. Anderson, E. Biham, L. Knudsen	256	128	CBC
Triple DES	IBM, NSA	3*56	64	Outer-CBC
Twofish	B. Schneier, J. Kelsey, D. Whiting, Wagner, C. Hall, N. Ferguson	256	128	CBC

Tabelul 3.2 Algoritmii de cifrare

3.3.6. Algoritmii de hash

Aceşti algoritmi sun folosiți pentru derivarea cheii de antet.

SHA-1

NIST împreună cu NSA au proiectat Secure Hash Algorithm - Algoritm de Dispersie Sigură , standardul numindu-se SHS. Algoritmul procesează un mesaj de lungime maximă 2^{64}-1 biți şi produce un rezumat de 160 -biți.

Lucrează pe blocuri de 512 –biți, foloseşte 4 funcții matematice şi 4 constante aditive.

RIPEMD-160

Algoritmul RIPEMD-160 a fost elaborat de un grup de cercetători de la European RACE Integrity Primitives Evaluation. Algoritmul RIPEMD-160 primeşte la intrare un mesaj de lungime arbitrară şi produce la ieşire un rezumat de 160 de biţi.

Lucrează pe blocuri de 512 –biţi, foloseşte 5 funcţii matematice şi 9 constante aditive.

3.4. Derivarea cheii de antet, a valori Salt şi numărul de iteraţii folosit pentru generarea cheii.

Cheia de antet este folosită la criptarea şi decriptarea antetului unui volum. Pentru aceasta este folosită tehnica PBKDF2 specificată în PKCS #5 v2.0.

Este folosită o valoare Salt pe 512 –biţi (generată de Generatorul de numere aleatoare la crearea unui volum), ceea ce înseamnă că sunt 2^{512} chei pentru fiecare parolă. Aceasta scade signifiant vulnerabilitatea atacurilor de tip dicţionar (precalcularea tuturor cheilor pentru un dicţionar de parole este foarte greu de făcut când este folosită o valoare Salt). Cele 2000 de iteraţii ale funcţiei de derivare a cheii de antet fac ca timpul necesar pentru încercarea aflării cheii prin *Brute Force* (Forţă Brută), în special atacul de tip Dicţionar, să crească foarte mult. Derivarea cheii de antet se face pe baza unei funcţii de hash HMAC-SHA-1 sau HMAC-RIPEMD-160. Calitatea şi lungimea cheii derivate nu sunt afectate de dimensiunea rezultatului funcţiei de Hash (care este pentru ambele funcţii pe 160 –biţi).

3.5. Generatorul de Numere Aleatoare

Este folosit pentru generarea cheii de criptare principale, a Salt-ului şi pentru crearea valorilor IV şi de whitening.

Acest generator creează, în memoria RAM, o zonă de numere aleatoare de dimensiune 256 –biţi. Sursele din care sunt luate datele necesare generării valorilor sunt:

- Mişcările mouse-ului (este făcut un rezumat CRC32 al coordonatelor mouse-ului şi a timpului acestui eveniment):
  CRC32(*MouseCoordinates*) || CRC32(*EventDeltaTime* || *Evenţime*)
- Apăsatul butoanelor mouse-ului: (este făcut un rezumat CRC32 al ID-ului butonului şi a timpului acestui eveniment):
  CRC32(*MouseBuţonID*) || CRC32(*EventDeltaTime* || *Evenţime*)

- Apăsarea tastelor atâta timp cât mouse-ul este situat pe fereastra programului (este făcut un rezumat CRC32 al codului tastei şi a timpului acestui eveniment):
  CRC32(*KeyID*) || CRC32(*EventDeltaTime* || *AbsoluteEvenţime*)
- Statistici despre performanța discurilor.
- Librăria MS Windows CryptoAPI.
- Variabile de timp şi contoare ale sistemului de operare, colectate la intervale de 250 ms.

Înainte de scrierea unei valori obținute mai sus să fie scrisă în aceea zonă din RAM, valorarea este împărțită în octeți (de ex. rezultatul unui CRC-32 de 32 –biți este împărțita în patru octeți). Aceşti octeți sunt apoi scrişi în zonă, dar nu prin înlocuirea vechilor valori, ci prin adunarea modulo 2^8 cu ce se află acolo. La scrierea unui Octet poziția cursorului este crescută cu un Octet. Când cursorul ajunge la sfârşitul zonei, poziția sa este resetată la începutul zonei. În plus după ce o valoare este adusă în zonă, întreaga zonă este rezumată folosind o funcție de Hash (SHA-1 sau RIPEMD-160).

Implementarea şi modelul generatorului de numere aleatoare se bazează pe:

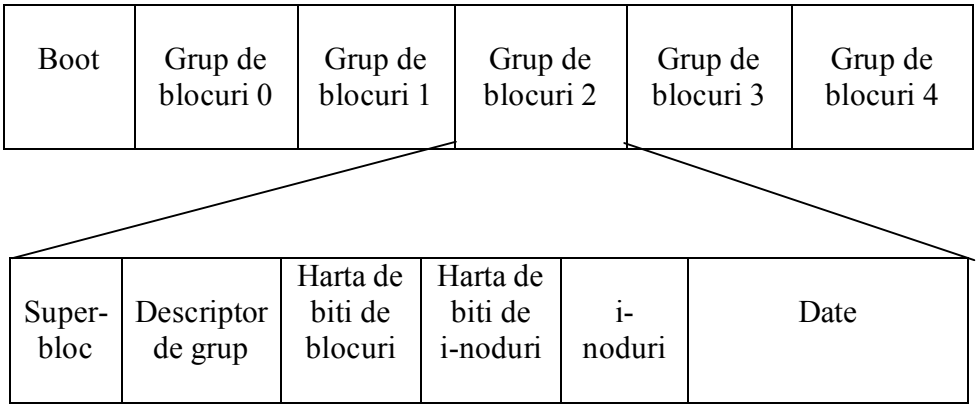
- *Software Generation of Practically Strong Random Numbers* de Peter Gutmann
- *Cryptographic Random Numbers* de Carl Ellison

Capitolul 4

Sistemul de fişiere Linux

Sistemul iniţial de fişiere al Linuxului a fost sistemul de fişiere MINIX. Totuşi datorită faptului că numele fişierelor erau limitate la 14 caractere şi că mărimea maximă a unui fişier era de 64 MB, aproape de la început a existat un interes pentru un sistem mai bun de fişiere. Prima îmbunătăţire a fost sistemul de fişiere Ext care permitea nume cu până la 255 de caractere şi fişiere de 2 GB, dar era mai lent decât sistemul de fişiere MINIX, deci căutarea a mai continuat un pic. Până la urmă, a fost inventat sistemul de fişiere Ext2 cu nume lungi de fişiere, fişiere mari şi o performanţă îmbunătăţită şi acesta a devenit principalul sistem de fişiere. Totuşi Linux-ul suportă în continuare o mulţime de sisteme de fişiere folosind sistemul de fişiere NFS. Atunci când Linux-ul este legat, se oferă o alegere de ce sisteme de fişiere să fie legate în nucleu. Altele pot fi încărcate dinamic ca module în timpul execuţiei, dacă este cazul.

Ext2 este foarte asemănător cu sistemul Berkeley Fast File cu mici diferenţe. Structura Ext2 este descrisă mai jos. În loc de a folosi grupuri de cilindre, care aproape nu mai înseamnă nimic cu geometriile virtuale de astăzi, el împarte discul în grupuri de blocuri fără a ţine cont de unde cad marginile cilindrelor. Fiecare grup de blocuri începe cu un superbloc, care specifică câte blocurişi i-noduri sunt, mărimea unui bloc, etc. Apoi urmează descriptorul de grup, care conţine informaţii despre poziţionarea hărţilor de biţi, numărul de blocuri şi i-noduri libere din grup şi numărul de cataloage din grup. Această informaţie este importanţă deoarece Ext2 încearcă să împrăştie cataloagele în mod egal pe disc.



Două hărți de biți sunt folosite pentru a ține evidența blocurilor libere şi a i-nodurilor libere, respectiv. Fiecare hartă are lungimea de un bloc. Cu blocuri de 1KB, această proiectare limitează un grup de blocuri la 8192 de blocuri şi 8192 de i-noduri. Primul număr este o limitare, dar al doilea nu este o limitare în practică. După aceea urmează chiar i-nodurile. Fiecare i-nod este de 128 de octeți de două ori mărimea unui i-nod unix standard. Spațiul suplimentar este folosit în modul următor. In loc de 10 adrese de bloc directe şi 3 adrese de bloc indirecte Linux-ul permite 12 directe şi 3 adrese indirecte. Mai mult adresele au fost extinse de la 3 la 4 octeți pentru a suporta partiții mai mare decât 224 blocuri (16 GB), ceea ce era deja o problemă în UNIX. În plus, sunt rezervate câmpuri pentru pointeri la liste de acces pentru un control mai fin al drepturilor, dar acestea sunt încă pe planşa de proiectare. Restul i-nodului este rezervat pentru utilizări ulterioare. Istoria a arătat ca biții nefolosiți nu rămân aşa pentru mult timp.

Operarea sistemului de fişiere este similară cu a sistemului de fişiere Berkeley rapid. Totuşi o diferență față de Berkeley este folosirea a 1KB standard peste tot. Sistemul rapid Berkeley foloseşte blocuri de 8KB şi apoi le împarte în bucăți de 1KB dacă este nevoie. Ext2 face asta în modul simplu. Ca şi sistemul Berkeley atunci când un fişier se măreşte, Ext2 încearcă să pună blocurile în acelaşi grup de blocuri ca şi restul fişierului, de preferat imediat după blocul precedent. De asemenea atunci când se adaugă un fişier, Ext2 încearcă să-l pună în acelaşi grup de blocuri ca şi catalogul său. Cataloagele noi sunt împrăştiate uniform pe disc.

Alt sistem de fişiere Linux este sistemul de fişiere /proc (proces), o ideea care a fost inventată inițial în a 8-a ediție a UNIX-ului de la Bell Labs şi mai apoi copiată în BSD 4.4 şi System V. Totuşi, Linux-l extinde ideea în mai multe feluri. Ideea de bază este că pentru fiecare proces din sistem, este creat un catalog în proc. Numele catalogului este PID-ul procesului exprimat ca număr zecimal. De exemplu, /proc/619 este catalogul corespunzător procesului cu PID-ul 619. În acest catalog există fişiere care par să conțină informații despre proces, cum ar fi linia de comandă şirurile de mediu şi măştile de semnale. De fapt, aceste fişiere nu există pe disc. Atunci când sunt citite, un apel de sistem obține, pe măsură ce este nevoie, informația de la proces şi o întoarce într-un format standard.

Multe dintre extensiile Linux-ului sunt legate de alte fişiere şi cataloage din proc. Ele conțin o gamă largă de informații despre UCP, partițiile discului,

dispozitive, vectori de întreruperi, contoare nucleu, sisteme de fişiere, module încărcate şi multe altele. Programele utilizator fără privilegii pot citi mare parte din aceste informaţii şi afla comportamentul sistemului într-un mod sigur. În unele dintre aceste fişiere se poate scrie pentru a modifica parametrii de sistem.

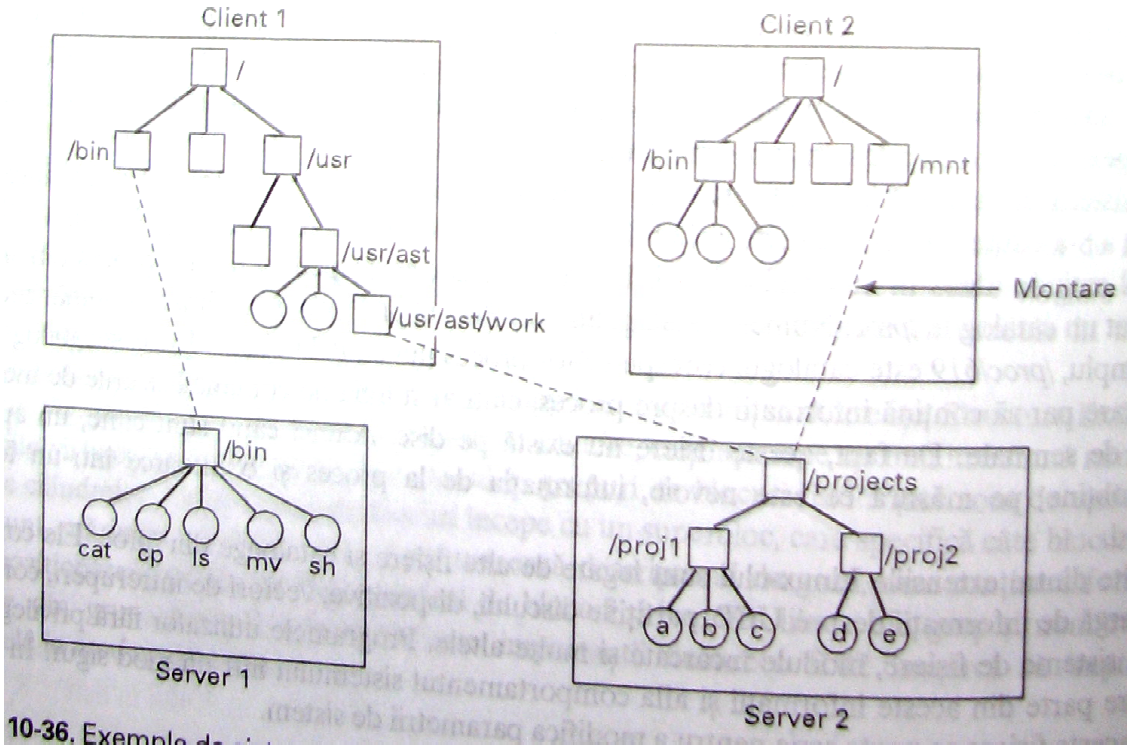
NFS: The Network File System (Sistemul de fişiere de reţea)

Reţelele au jucat un rol important în UNIX încă de la început (prima reţea UNIX a fost construită pentru a muta nucleele noi de pe PDP-11/70 pe Interdata 8/32 în timpul portării pe aceasta din urmă). În această secţiune vom studia sistemul NFS (Network File System) al Sun Microsistems care este folosit de către toate sistemele UNIX moderne (şi de către unele sisteme ne-UNIX) pentru a uni sisteme de fişiere de pe calculatoare distincte într-o entitate logică. Sunt interesante trei aspecte ale NFS-ului: arhitectura, protocolul şi implementarea.

Arhitectura NFS

Ideea de bază a NFS-ului este să permită unei mulţimi oarecare de clienţi şi servere să partajeze un sistem de fişiere comun. În multe cazuri toţi clienţi şi serverele se află în acelaşi LAN dar aceasta nu este necesar. Este de asemenea posibil să se ruleze NFS peste o reţea de mare întindere dacă serverul este departe de client. Pentru simplitate vom vorbi despre clienţi şi servere ca şi cum ar fi maşini distincte, deşi în realitate NFS permite fiecărei maşini să fie şi client şi server in acelaşi timp.

Fiecare server de NFS exportă unul sau mai multe din cataloagele sale pentru a fi accesate de către clienţi de la distanţă. Atunci când un catalog este făcut disponibil, sunt făcute disponibile şi toate subcataloagele, deci de fapt, un întreg arbore de cataloage este exportai unitar. Lista cataloagelor exportate este menţinută într-un fisier, deseori /etc/exports, astfel încât aceste cataloage să poată fi exportate automat în momentul în care serverul porneşte. Clienţii accesează cataloagele exportate montându-le. Atunci când un client montează un catalog (la distanţă), acesta devine parte a ierarhiei de cataloage, după cum se arată în fig.



10-36. Exemple de

Exemple de sisteme de fişiere de la distanţă montate. Cataloagele sunt reprezentate ca pătrate şi fişierele sunt reprezentate ca cercuri.

În acest exemplu, clientul 1 a montat catalogul bin de pe serverul 1 în propriul catalog bin, astfel că acum poate referi programul interpretorul ca /bin/sh şi obţine programul de pe serverul 1. Staţiile fără disk au deseori doar un sistem de fişiere schelet (în RAM) şi obţin toate fişierele în acest fel de la servere de la distanţă. Similar, clientul 1 a montat catalogul /projects al serverului 2 în catalogul său /usr/ast/work deci poate accesa acum fişierul a ca /usr/ast/work/proj/a. În final, clientul 2 a montat de asemenea catalogul projects şi poate de asemenea să acceseze fişierul a, ca mnt/proj1/a. După cum s-a văzut aici, acelaşi fişier poate avea nume diferite pe clienţi diferiţi datorită faptului că poate fi montat în locuri diferite în respectivii arbori. Punctul de montare este în întregime local pentru clienţi; serverul nu ştie locaţia sa de montare pe nici unul din clienţi.

Protocoalele NFS

Având în vedere faptul că unul din scopurile NFS este să suporte sisteme eterogene, cu clienţi şi servere rulând eventual sisteme de operare diferite pe hard diferit, este esenţial ca interfaţa dintre clienţi şi server să fie bine definită.

Numai atunci este posibil ca cineva să poată scrie o nouă implementare de client şi să se aştepte ca aceasta să funcţioneze cu serverele existente şi invers.

NFS realizează acest scop definind două protocoale client-server. Un protocol (protocol) este o mulţime de cereri trimise de la client sever-ului, împreună cu replicile corespunzătoare trimise de către server înapoi la client.

Primul protocol NFS tratează montarea. Un client poate trimite o cale la un server şi cere permisiunea de a monta catalogul undeva în ierarhia sa de cataloage. Locul unde va fi montat nu este conţinut în mesaj, deoarece pe server nu îl interesează unde va fi montat. Daca, calea este legală şi catalogul specificat a fost exportat, serverul trimite clientului un identificator sau manipulator de fişier (file handle). Identificatorul de fişier conţine câmpuri care identifică unic tipul sistemului de fişiere, discul, numărul i-nodului catalogului şi informaţii de securitate. Apelurile următoare de read şi write asupra fişierelor din catalogul montat sau orice alt subdirector al său folosesc identificatorul de fişier.

Atunci când UNIX-ul porneşte, se rulează fişierul cu comenzi /etc/rc înainte de a intra în modul multiutilizator. Comenzile pentru montarea sistemelor de fişiere de la distanţa se pot afla în acest fişier, montând astfel automat sistemele de fişiere de la distanţă înainte de a permite orice logare. Alternativ, mare parte din sistemele UNIX suportă automontarea (automounting). Această caracteristică permite ca o mulţime de cataloage de la distanţă să fie asociate cu un catalog local. Nici unul dintre aceste cataloage nu este montat (nici măcar serverele lor nu sunt contactate) atunci când clientul porneşte. În loc de aceasta, prima oară când este deschis un fişier la distanţă, sistemul de operare trimite un mesaj fiecărui server. Primul care răspunde câştigă şt catalogul sau este montat.

Automontarea are două avantaje principale asupra montării statice prin fişierul /etc/rc. În primul rând, dacă se întâmplă ca unul din serverele din /etc/rc să fie inactiv este imposibil să se pornească clientul, sau cel puţin nu fără dificultăţi, întârzieri şi destul de multe mesaje de eroare. Daca utilizatorul nici nu are nevoie de server în acel moment, toata munca este irosită. În al doilea rând permiţând clientului să încerce o mulţime de servere În paralel, se obţine un grad de toleranţă la defecte (pentru că este necesar să funcţioneze unul singur) şi performanţa poate fi îmbunătăţită (fiind ales prunul care răspunde - probabil cel mai puţin încărcat).

Pe de altă parte se presupune că toate sistemele de fişiere specificate ca alternative pentru automontare sunt identice. Deoarece NFS nu pune la dispoziţie nimic pentru replicarea fişierelor si a a cataloagelor, este treaba utilizatorului să

aranjeze lucrurile astfel încât toate sistemele de fişiere să fie identice. În consecinţă automontarea este folosită cel mai des pentru sisteme de fişiere numai cu citire care conţin fişierele binare ale sistemului si alte fişiere care se modifică rar.

Al doilea protocol NFS este pentru accesul la fişiere şi cataloage. Clienţii pot trimite mesaje la server pentru a manipula cataloage şi a citi şi scrie din fişiere. De asemenea ei pot accesa atributele fişierelor, cum ar fi modul fişierului, mărimea şi data ultimei modificări. Marea majoritate a apelurilor de sistem UNIX sunt suportate de NFS, poate cu excepţia surprinzătoare a lui open şi close.

Omiterea lui open şi close nu este un accident. Este intenţionată. Nu este necesar să se deschidă un fişier înainte de a fi citit, nici nu este necesar să se închidă când s-a terminat. In loc de aceasta, pentru a citi un fişier, un client trimite serverului un mesaj lookup care conţine numele fişierului cu o cerere de a-l căuta şi a întoarce identificatorul de fişier, care este o structură care identifica fişierul. Spre deosebire de un apel call acesta operaţie lookup nu copiează nici o informaţie în tabelele de sistem. Apelul read conţine identificatorul de fişier al fişierului care trebuie citit, offset-ul în fişierul care va fi citit şi numărul de octeţi doriţi. Fiecare mesaj de acest tip este independent. Avantajul acestei scheme este că serverul nu reţine nimic despre conexiunile deschise între apeluri. Asfel, dacă un server devine nefuncţional şi apoi îşi revine, nu se pierde nici o informaţie despre fişierele deschise, pentru că nu există nici o astfel de informaţie. Se spune despre un astfel de server care nu menţine informaţii despre starea fişierelor deschise că este fără stare (stateless).

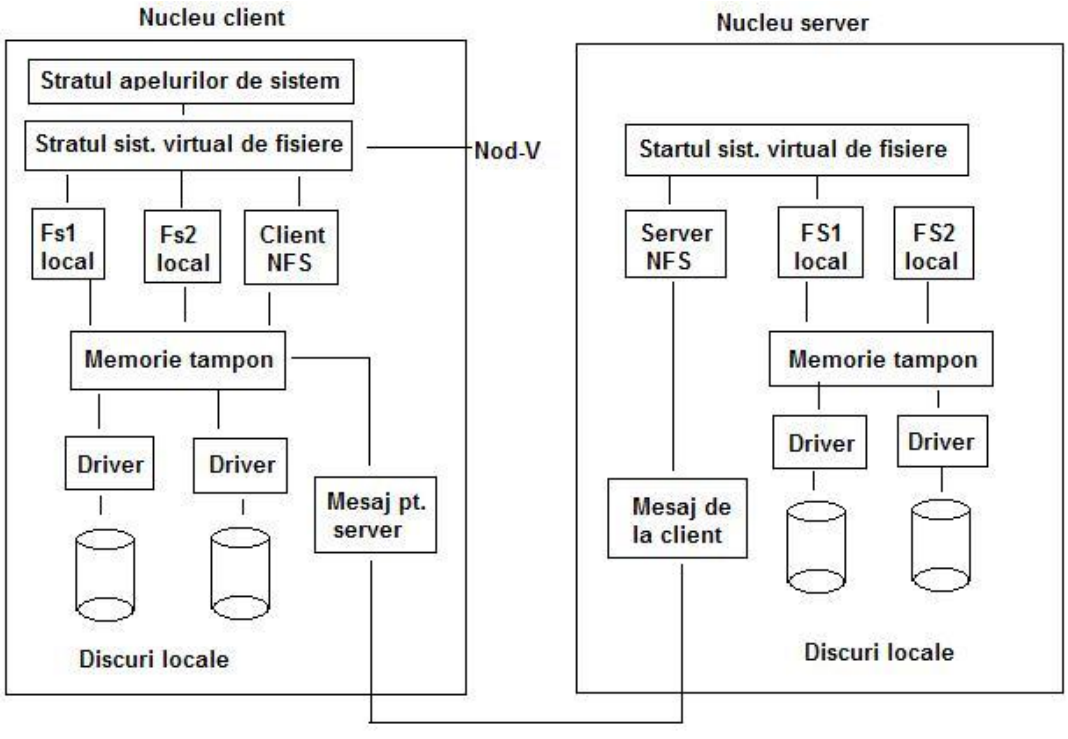
Din păcate, metoda NFS face dificilă obţinerea semanticii exacte a UNIX privind fişierele. De exemplu, un fişier UNIX poate fi deschis şi blocat astfel încât alte procese să nu îl poată accesa. Atunci când un fişier este închis, blocajele sunt eliberate. Într-un server fără stare ca NFS, blocajele nu pot fi asociate cu fişierele deschise, pentru că serverul nu ştie care fişiere sunt deschise. Deşi NFS necesită un mecanism adiţional separat pentru a trata blocarea fişierelor.

NFS foloseşte mecanismul de protecţie standard UNIX cu biţii rwx pentru proprietar, grup şi alţii. La început, fiecare mesaj de cerere conţinea id-urile de grup şi utilizator ale apelantului, care erau folosite de serverul de NFS pentru a valida accesul. In realitate avea încredere că acel client nu trişa. Experienţa mai multor ani a demonstrat din plin că o astfel de presupunerea era - cum să spunem - naivă. Acum, criptografia cu chei publice poate fi folosită pentru a stabili o

cheie sigură pentru a valida clientul şi serverul la fiecare cerere şi răspuns. Atunci când această facilitate este pornită, un client răuvoitor nu se poate da drept alt client pentru că nu cunoaşte cheia secretă a acelui client.

Implementarea NFS

Deşi implementarea clientului şi a serverului este independentă de protocoalele NFS, marea majoritate a sistemelor UNIX folosesc o implementare cu trei straturi. Primul strat este stratul apelurilor de sistem. Acesta tratează apeluri cum ar fi open, read şi close. După analizarea apelului şi verificarea parametrilor, invocă al doilea strat, stratul sistemului virtual de fişiere (VFS).
Treaba stratului VFS este să menţină un tablou cu o intrare pentru fiecare fişier deschis, analog cu tabloul de i-noduri pentru fişiere deschise din UNIX. În UNIX-ul obişnuit un i-nod este identificat unic printr-o pereche (dispozitiv, număr i-nod). În loc de aceasta stratul VFS are pentru fiecare fişier deschis o înregistrare numită v-nod (virtual i-node, i-nod virtual). V-nodurile sunt folosite pentru a determina dacă fişierul este local sau la distanţă. Pentru fişierele de la distanţă, este pusă la dispoziţie suficientă informaţie pentru a putea fi accesate. Pentru fişierele locale, sunt reţinute sistemul de fişiere si i-nodul deoarece sistemele UNIX moderne suportă sisteme de fişiere multiple (de ex V7, Berkelez FFS, ext2fs, -proc, FAT, etc.). Deşi VFS a fost inventat pentru a suporta NFS, marea majoritate a sistemelor UNIX moderne îl suportă ca o parte din system, chiar dacă NFS-ul nu este folosit.



Structura stratului NFS

Pentru a vedea cum sunt folosite v-nodurile, să urmărim o secvență de apeluri de sistem mount, open şi read. Pentru a monta un sistem de fişiere la distanța, administratorul de sistem (sau etc/rc) apelează programul mount specificând catalogul de la distanță, catalogul local în care va fi montat şi alte informații. Programul mount analizează numele catalogului de la distanță care trebuie montat si descoperă numele serverului de NFS pe care se află catalogul de la distanță. Apoi contactează acea
maşină cerând un identificator de fişier pentru catalogul de la distanță. Dacă catalogul exista şi este disponibil pentru montare la distanță, serverul returnează un identificator de fişier pentru catalog, in final, se realizează un apel de sistem mount, trimițându-se identificatorul nucleului.

Nucleul construieşte apoi v-nodul pentru catalogul de la distanță şi cere codului client NFS in figura de mai jos să creeze un r-node (remote i-node, i-nod la distanță) în tabelele sale interne pentru a stoca identificatorul de fişier. V-nodul indică către r-node. Fiecare v-node din stratul VFS va contine fie un pointer către un r-nod în codul client NFS, sau un pointer catre un i-nod intr-unul dintre sistemele locale de fişiere (reprezentați ca linii punctate în figura). Astfel dintr-un v-nod este posibil sa se vadă dacă un fişier sau catalog este local sau la distanță. Daca este local, pot fi localizate sistemul de fişiere corect şi i-nodul. Dacă este la distanță, pot fi localizate maşina de la distanta si
identificatorul de fişier.

Atunci când un fişier la distanța este deschis pe client, la un moment dat in timpul analizarii caii, nucleul se loveşte de catalogul in care este montat sistemul de fişiere de la distanță. Vede că acest catalog este la distanță şi în v-nodul catalogului găseşte un pointer către r-nod. Cere apoi codului client NFS să deschidă fişierul. Codul client NFS caută restul căii pe serverul de la distanță asociat cu catalogul montat şi obține un identificator de fişier. Creează un r-nod pentru fişierul de la distanță in tablourile sale şi raportează stratului VFS, care pune pentru fişier în tablourile sale un v-mode care punctează către r-node. Încă o dată vedem aici că fiecare fişier sau catalog deschis au un v-nod care punctează fie către un i-nod sau un r-nod.

Apelantului îi este dat un identificator de fişier pentru fişierul la distanță. Identificatorul de fişier este pus în corespondență pe v-nod de către tablourile din stratul VFS. Este de observat că nu se realizează nici o înregistrare în tabelele de la server. Deşi serverul este pregătit să pună la dispoziție identificatori de fişiere la cerere, el nu ține evidența a ce fişiere au identificatori de fişiere şi care nu. Atunci când îi este trimis un identificator de fişier pentru a fi accesat, el verifică iderrtificatorul şi daca este valid, îl foloseşte. Validarea poate include verificarea cheii de autentificare conținută în antetele RPC, dacă securitatea este activată.

Atunci când identificatorul de fişier este folosit pentru un alt apel de sistem, de exemplu, read, stratul VFS localizează v-nodul corespunzător şi din acesta determină dacă este local sau la distanţă şi de asemenea care i-nod sau r-nod îl descrie. Trimite apoi un mesaj serverului conținând indentificatorul, offsetul în fişier (care este reținut la client, nu la server) şi numărul de octeți. Din motive de eficienţă, transferurile dintre client şi server se realizează în bucăți mari, în mod normal 8192 octeți, chiar dacă sunt ceruți mai puțini octeți.

Atunci când cererea ajunge la server, este pasată stratului VFS, care determină care sistem local de fişiere are fişierul cerut. Stratul VFS face apoi un apel către acel sistem de fişiere pentru a citi şi a returna octeții. Aceste date sunt apoi trimise înapoi clientului După ce stratul VFS client a obținut bucata de 8-KB care a cerut-o, face automat o cerere pentru următoarea bucată, pentru a o avea dacă va fi cerută în scurt timp. Această facilitate, cunoscută ca citire înainte, imbunătățeşte performanța considerabil.

Pentru scrieri este urmată o cale asemănătoare de la client la server. De asemenea transferurile se realizează tot în bucăți de 8-KB. Dacă un apel de sistem write pune la dispoziție mai puțin de S-KB de date, datele sunt doar acumulate local. Numai când întreaga bucată de 8-KB este plină este trimisă serverului. Totuşi, atunci când un fişier este închis datele sale sunt trimise la server imediat

Altă tehnică folosită pentru îmbunătățirea performanțelor este folosirea memoriei intermediare ca şi în UNIX-ul obişnuit. Serverul foloseşte memoria intermediară pentru date pentru a evita accesele la disc, dar aceasta este invizibil clientului. Clienții mențin două memorii intermediare (zone tampon), una pentru atributele fişierelor (i-noduri) şi una pentru datele fişierelor. Atunci când este nevoie de un i-nod sau bloc de fişier, se face o verificare dacă qererea poate fi satisfăcută folosind zona tampon. Dacă da, se va evita traficul în rețea.

În timp ce zona tampon de la client ajută enorm la mărirea performanțelor, el şi introduce nişte probleme urâte. Să presupunem că doi clienți au în zona tampon acelaşi bloc dintr-un fişier şi că unul din ei îl modifică. Atunci când celălalt îl va citi, va citi versiunea veche. Zona tampon nu este coerentă.

Având în vedere potențiala gravitate a acestei probleme, implementarea NFS face mai multe lucruri pentru a o atenua. Întâi, fiecărui bloc din zona tampon îi este asociat un cronometru. Atunci când cronometrul expiră, înregistrarea este îndepărtată. In mod normal cronometrul este setat la 3 secunde pentru blocuri de date şi 30 de secunde pentru blocuri catalog. Realizând aceasta se reduce într-un fel riscul. În plus, ori de câte ori este deschis un fişier din yona tamponeste trimis un mesaj la server pentru a afla când a fost modificat ultima oară fişierul. Daca ultima modificare a survenit dupa ce copia locală a fost

intodusă in zona tampon, copia din zona tampon este îndepărtată şi noua copie este preluată de la server. În final a fiecare 30 de secunde expiră un cronometru al memoriei intermediare (zonei tampon) şi toate blocurile murdare (modificate) din zona tampon sunt trimise la server. Chiar dacă nu este perfect, aceste ajustări fac sistemul foarte utilizabil în marea majoritate a situațiilor practice.

Capitolul 5
Sistemul FAT

Fat e un sistem dezvoltat de Microsoft pentru MS DOS fiind primul sistem de fisiere al versiunilor de Microsoft Windows, pana la si incluzand Microsoft Vista. FAT-ul a fost standardizat ca ECMA-107 si iso/iec 9293 aplicandu-se la discuri floppy si discuri optice. Sistemul FAT nu este foarte complicat si este suportat din punct de vedere virtual de toate sistemele de operare pentru uz personal. Acest lucru il face un format ideal pentru floppy disc si carduri de memorie.

Cele mai commune implementari au un foarte mare neajuns, in asa fel incat cand fisierele sunt sterse si altele noi sunt create, fragmentele de directoare tind sa devina imprastiate , in acest fel citirea si scrierea devenind un proces incet. Defragmentarea e o solutie pentru a acest lucru, dar deseori este un process de durata si trebuie efectuat in mod regulat pentru a pastra fisierele sistemului FAT curate.

Sistemul FAT a fost creat pentru discurile Microsot Standalone Disc Basic. In august 1980, Tim Paterson a incorporat Fat in sistemul de operare 86-DOS pentru s-100 8086 CPU. Sistemul de fisiere era diferenta principala dintre 86-dos si predecesori.

Numele vine din folosirea unei tablete care centralizeaza informatia a caror zone apartin fisierelor, care sunt libere sau posibil neutilizabile si unde fiecare fisier este stocat pe disc. Pentru limitarea marimii acestei tablete si a numarului de biti necesari in intrarile directoarelor, spatial discului este alocat pentru fisiere in grupuri numite cluster. Numarul maxim posibil de clustere a crescut de-a lungul timpului in mod dramatic, si numarul de biti care este necesar pentru pentru a identifica un cluster e folosit pentru a denumi versiuni successive majore ale formatului. Standardul FAT a fost de asemenea extins si pe alte cai, in timp ce pastreaza in mod invers compatibilitatea cu software existente.

5.1 FAT 12

Versiunea initiala a FAT este acum cunoscuta ca FAT 12. construita ca un system de fisiere pentru floppy disk, si avea urmatoarele neajunsuri: adresele clusterelor aveau doar 12 biti, care nu limita doar numarul clusterelor pana la 4078, dar facea manipularea FAT putin mai inselatoare pentru un PC de 8 biti

sau 16 biti (registrii). Spatiul discului a fost stocat in sectoare de 16 biti care limiteaza marimea la 32 MB. FAT 12 era folosit de diferiti manufacturieri cu diferite formate fizice, dar un floppy disc tipic al momentului era 5.25-inch, 40 tracks cu 8 sectoare pe track, rezultand intr-o capacitate de numai 160 KB pentru ambele arii ale sistemului si fisierelor.

Prin conventie toate structurile de control au fost organizate sa se incadreze pe prima linie, evitandu-se miscarile importante in timp real si operatiunile de scriere., cu toate astea, aceasta varia depinzand de modul de fabricatie si formatul fizic al discului. De cand DOS ul a fost introdus pe FAT 12, nu avea support ierarhic al directoarelor; numarul maxim al fisierelor era limitat la cateva zeci.

O limitare care nu a fost adresata pana mai tarziu, a fost ca orice sector nefunctional in zona controlului structurii, track 0, ar putea preveni ca discheta sa fie folosita. Intstrumentul de formatare DOS a respins complet acest tip de dischete. Sectoarelor nefunctionale li s-a permis numai accesul in zona fisierelor, unde au facut intregul sustinator de cluster deasemenea neutilizabil.

5.2 FAT 16

In 1984 IBM a lansat pe piata de PC AT, care avea hard disk de 20 MB. Microsoft a introdus MS-DOS 3.0 in paralel. Adresele clusterelor erau marite la 16 biti, permitand un numar mai mare de clustere (pana la 65,517) si in cele din urma cu marimea fisierelor mai mare. Oricum, numarul maxim posibil de sectoare si marimea (partitie, decat pe disc) maxima de 32 MB nu a fost schimbata. Cu toate acestea, deja numitul FAT 16, acest format nu era ceea ce azi se intelege sub acest nume. Un disk e 20 MB format sub MS-DOS 3.0 nu era accesibil invechitului MS-DOS 2.0. bineinteles MS-Dos 3.0 poate inca accesa stilul

Partitii extinse si drive-uri logice

Pt a imbunatatii structura sistemului de fisiere Fat ,a fost dezvoltata o cercetare in paralel permitand o crestere a posibilitatilor maxime a spatiului de stocare FAT prin folosirea mai multor partitii FAT .Partitiile initiale trebuiau sa se foloseasca numai pentru impartirea discului intre sistemele de operare, DOS si Xenix pe vremea aceea, asa ca DOS a fost pregatit numai sa faca fata numai UNEI partitii FAT . Nu era posibil sa se creeze a partitie DOS multipla folosind

instrumentele Dos iar alte utilitare avertizau ca o astfel de operatie nu era compatibila cu DOS. Numai permitand mai multe partitii ideentice in DOS poate sa se ajunga la mai multe probleme: poate C: sa fie prima partitie FAT, pentru a fi mai simplu sau mai bine partittia marcata ca finnd activa in tabela partitilor, ca mai multe variante de DOS sa co-exixte? Si care partitie trebuie sa fi C:daca sistemul a boot-at dupa dischete ?
Pentru a permite folosirea mai multor partitii FAT intr-un mod compatibil , o noua partitie a fost introdusa (in MS-DOS 3.2 ianuarie 1986), se numea partitia extinsa; care era de fapt doar un container pentru partitii aditionale numit "logical drives". Initial numai 1 drive logic era posibil,permitand sa se foloseasca pana la 64 MB din discurile hardului. In MS-DOS 3.3 (din august 1987) aceasta limita a fost marita la 24 de drive..Au fost intotdeauna stocate pe disk ca un rand pe blocuri separate intr-o singura cutie; despre aceste blocuri se spune adesea ca sunt legate impreuna de linkurile din sectoarele EBR.Numai o partie extinsa a fost permisa. Drive-urile logice nu erau bootabile si partitile extinse putea numai sa fie create dupa prima partie FAT, care elimina toata ambiguitatea, dar de asemenea si posibilitatea de a boot-a mai multe versiuni DOS din acelasi harddisk.
Un advers folositor a schemei partitiei extinse a fost sa creasca semnificativ numarul maxim de partitii posibile pe hard discul PC-ului peste 4 care putea sa fie descrise un singur MBR.
Inaintea introducerii partitilor extinse niste hard discuri(care la timpul acela erau optionale, deoarece IDE exista inca) puteau face hard discuri mai mari care puteau aparea ca doua discuri separate.

Ultimul FAT16

In sfarsit in noiembrie 1987 Compaq DOS 3.31 a introdus ce noi azi numim formatul FAT16 cu o extindere pe sectorul discului de 16 biti pana la 32 biti.Rezultatul s-a numit initial DOS 3.31 Large File System(sistemul cu fisiere mari). Cu toate ca pe disc schimbarile erau aproape minore ,intregul cod de disc DOS a fost schimbat sa foloseasca 32 de biti pe sectorul de numere, o sarcina complicata prin faptul ca era scris pe 16 biti in limbajul de programare.

In 1988 imbunattirea a devenit mai gererala fiind disponibila si pe MS-DOS 4.0 si OS/2 1.1.Limita pe marimea partitiei era acum data pe 8 biti care a avea valoare maxima pe 64. Cu hardiscul obisnuit de 512 bitipe sector, care dadea 32k de clusters, a definit limita de 2 gigabiti pentru partitia de fat16. Pe media magneto-optica care poate avea 1 sau 2 kilobiti pe sector limita era proportional mai mare.

Mult mai tarziu WINDOWS NT a marit marimea maxim a clusterului pana la 64 KB . Oricum formatul final nu era compatibil cu nici un FAT implementat in timpul acela si genera massive fragmentary interne.Windows 98 suporta de asemea cititul si scrisul pe aceasta varianta dar utilitatile discului nu mergeau pe el.

Numarul directoarelor de intrare disponibile in timpul formatatului si stocate intr-un camp de 16 biti a obtinut o limita absoluta de 32767 intrari(32767 un multiplu de 32).Din motive istorice FAT12 si FAT16 foloseau 512 directoare de intrare de root pe mediile non-floppy si celelalte marimi erau necompatibile cu niste software sau dispozitive.

5.2 FAT32

FAT32 este o modalitate de formatare care permite o marime a fisierului de maxim 4 GB .Fisiere mai mari au nevoie de alt tip de formatare cum ar fi HFS+ sau NTFS.

Pentru a nu depasii volumul maxim de marime a lui FAT16, in timp ce inca ii permite DOS-ului in timp real sa faca fata formatului faca sa reduca neaparat memoria conventionala, Microsoft a decis sa implementeze a noua generatie de FAT32 cu numarul de clustere tinuta intr-un camp de 32 de biti, din care 28 de biti sunt folsiti pentru a tine numarul de clustere pentru un maxim de aproape 250 de milioane de clustere.

Aceasta poate permite ca marimea discului sa fie de 8 terabiti cu 32k de clustere, dar sectorul de boot foloseste un camp de 32 de biti pentru numaratoarea sectorului,limitand marimea la 2TB pe hrd disk cu pana la 512 biti pe sector.

Pe Windows 95/98 , datorita versiunii Microsoft ScanDisk care includea aceste sisteme de operare fiind o aplicatie 16-bit,structuraFAT ne permitea cresterea pana la 4 milioane de clustere, determinand o limita a volumului de 127.53 gibabiti. O limitare a Fdisc-ului in versiunile orginale Windows 98/98 SE cauza o raportare incorecta a volumului diskurilor de peste 64 GB. O corecta versiune este disponibila la Microsoft. Aceste limitary nu se aplica pe Windows 2000/XP cu exceptia Setup-ului in care este o limita de 32 de GB.Windows ME suporta sistemele FAT32 fara nici o limitare.

FAT32 a fost introdus cu Windows 95 OSR2, cu toate ca reformatarea necesita folosirea acesteia, si DriveSpace 3 nu a suportat-o niciodata. Windows 98 a introdus o utilitate de convertire a hard-disk-urilor din FAT16 in FAT32 fara nici o pierdere de date. In linia NT, suportul pt FAT32 a aparut in Windows 2000. Un driver gratuit FAT32 pentru Windows NT4.0 a fos pentru Winternals,

o companie achizitionata mai tarziu de catre Microsoft. De la achizitia driverului nu mai este official valabila.

Windows 2000 si XP pot citi sau scrie fisiere de sistem FAT32 de orice marime, dar programul de formatare inclus in Windows 2000 sau mai nou nu pot crea decat sisteme FAT32 de 32 de GB sau mai putini. Aceasta limitare a fost impusa prin design si datorita faptului ca multiplele sarcini deveneau incite si ineficiente pentru un system foarte mare ca FAT32.Acesta limitare poate fi evitata cu utilitati de formatare third-party.

Maximul posibil a marimi fisierelor FAT32 este 4GB minus 1 byte. Captura video si editarea aplicatiilor si alte software'uri pot cu usurinta depasii aceasta limita. Pana la mid-2006,aceaia care foloseau sisteme boot duale sau cine muta dirvere externe de date intre computere cu diferite sisteme de operare nu au avut de ales decat sa ramana la FAT32. De atunci, suportul complet pentru NTFS a devenit disponibil in Linux si multe alte sisteme de operare, instaland FUSE library impreuna cu aplicatia NTFS-3G. Schimbul de date este de asemenea posibil intre Windows si Linux folosind fishiere system ale Linux-native, ext2 sau ext3, prin intermediul folosirii driverelor externe pentru Windows cum ar fi ext2 IFS, cu toate aceastea Windows nu poate boota din partitiile ext2 sau ext3.

Capitolul 6
Sistemul NTFS

NTFS este sistemul standard de fisiere Windows NT, incluzand versiunile ulterioare WIndows 2000, Windows XP, Windows Server 2003, Windows Server 2008 si Windows Vista.

NTFS inlocuieste sistemul de fisiere FAT ca sistem preferat pentru Windows. NTFS prezinta aventaje fata de FAT si HPFS (High Performance File System) cum ar fi suportul imbunatatit pentru metadata, folosirea structurilor avansate pentru a imbunatati performatia, fiabilitatea si ultilizarea spatiului pe disc si alte extensii cum ar fi cele de securitate: access control lists si file system journaling. Specificatiile exacte ale sistemului de securitate sunt confidentiale desi sistemul poate fi licentiat de Microsoft prin programul de licentiere a proprietatii intelectuale.

Versiuni

NTFS are 5 versiuni:
- v1.0
- v1.1
- v1.2 intalnita in NT 3.51 si NT 4
- v3.0 intalnita in WIndows 2000
- v3.1 intalnita in Windows XP, Windows Server 2003 si Windows Vista

Versiunile NTFS instalate pe sistemele de operare amintite mai sus sunt uneori denumite dupa sistemul de operare cu care sunt distribuite. Versiunea v1.2 este cunoscuta si ca 4.0 iar 3.1 este uneori cunoscuta(incorect) ca 5.0 si 5.1. Fiecare noua versiune adauga imbunatatiri ca de exemplu: Windows 2000 a introdus diviziuni (disk quotas) iar Windows Vista a introdus Transactional NTFS, legaturi simbolice NTFS si functionalitatea "self-healing".

Caracteristici

In NTFS toate datele - nume de fisiere, data creatiei, permisii de acces si continut - sunt stocate ca meta-date (date despre date). Aceasta apropiere abstracta a permis adaugarea unor imbunatatiri ulterioare in timpul dezvoltarii

Windows NT - un exemplu interesant este adaugarea campurilor de indexare folosite de ultilitarul "Active dosary".

NTFS permite orice secventa de valori pe 16 biti pentru codarea numelor (nume de fisiere, fluxuri de date, index-uri) ceea ce inseamna ca sistemul suporta codarea UTF-16 dar nu verifica daca o secventa respecta sau nu UTF-16 (permite orice secventa de valori scurte, nerestrictionate de standardul UNICODE).

Sistemul foloseste structuri de tipul "B+ trees" pentru a indexa datele de sistem. Desi aceasta structura presupune o implementare complexa ea permite o cautare mai rapida a fisierelor in majoritatea cazurilor. Un fisier tip "jurnal" este folosit pentru a garanta integritatea sistemului de fisiere - dar nu si integritatea continutului fisierelor. Sistemele care folosesc NTFS sunt cunoscute pentru fiabilitatea imbunatatita fata de sistemele de fisiere FAT.

Tabelul Master File (MFT) contine meta-date despre aproximativ fiecare fisiere, dosar sau meta-fisiere pe un volum NTFS - incluzand nume de fisiere, locatii, dimensiuni si permisii. Structura sa suporta algoritmi care minimizeaza fragmetarea discului. O inregistrare de tip dosar consta intr-un nume de fisiere si o cheie de identificare constand in numarul inregistrarii reprezentante in MFT. Identificatorul unui fisiere contine deasemenea si un contor de reutilizare pentru a detecta vechimea acestuia.

Metafisiere

In Windows NT 5.0 (Windows 2000) si versiunile mai vechi comanda utilitara DIR ar lista informatii despre meta-fisiere daca ar fi cerute explicit: dir /a c:\$MFT, altfel acestea s-ar comporta ca fisiere invizibile, fiind mai bine ascunse decat fiserele "hidden" sau "system". Aceast comportament a fost eliminat incepand ci Windows NT 5.1 (XP).

Fid	Nume fisier	Scop
0	$MFT	Descrie toate fisierele de pe acest volum
1	$MFTMirr	Dubleaza primele inregistrari foarte importante ale $MFT, de obicei 8 (4 kb)
2	$LogFile	Jurnal al schimbarilor efectuate asupra fisierelor de sistem
3	$Volume	Contine identificatorul volumului, eticheta sa, versiunea fisierului de system si modificatoare ale

		volumului: montat, verificare chkdsk ceruta, redimensionare $LogFile ceruta, montat pe NT4, numarul de serie dupa actualizare, actualizare a structurii ceruta. (numarul de serie al volumului se afla in $Boot)
4	$AttrDef	Descrie tipurile de inregistrari folosite in intrarile $MFT
5	.	dosar radacina
6	$Bitmap	harta alocarii gruparilor volumului
7	$Boot	contains a Volume boot record including level 2 bootloader, a BIOS parameter block including volume serial number. This file is always placed at the beginning of the volume. It also contains the clusters where $MFT and $MFTMirr begin.
8	$BadClus	Acest fisier contine toate gruparile volumului marcate ca deteriorate (“bad sectors)
9	$Secure	Baza de date a listei de control a aceesului
10	$UpCase	Se speculeaza a fi o harta a folosirii literelor mari
11	$Extend	Un dosar a fisirelor de system continand fisierele 24, 25, 26
12..23	Rezervat pentru extensii $MFT	
24	$Extend\$Quota	administrarea a spatilui
25	$Extend\$ObjId	Identificator contextual de securitate
26	$Extend\$Reparse	Baza de date a legaturilor simbolice
27..	pagefile.sys	Inceputul intrarilor ce contin fisiere

Intrarile pana in 24 sunt tratate special de NTFS si sunt nume de fisiere sau continuturi dificile: unelte special create pentru scop.

Fisiere permanente vs. fisiere temporare

Pentru a optimiza depozitarea cazului usual de fisiere de date mici, NTFS tinde sa plaseze continutul fisierului in tabelul fisierului master - daca se potriveste, in loc sa foloseasca acelasi spatiu MFT pentru a lista gramada ce contine datele. Prima precizata este numita "data permanenta" de programatorii juridici . Data actuala care se potriveste este foarte dependenta de caracteristicile fisierului, dar 700 caractere sunt obisnuite pentru nume de fisiere scurte fara ACL-uri.

Interoperabilitate

Detalii despre internii implementarii sunt inchisi, fapt ce este dificil pentru vanzatorii partii terte sa furnizeze ustensile pentru a manevra NTFS.

Linux

Citirea/scrierea totala si sigura este furnizata de driver-ul NTFS-3G. Este inclus in majoritatea distributiilor de Linux.

Alte solutii majoritar read-only si out-datate exista deasemenea:

- esenta Linux 2.2:Partitiile NTFS pot fi citite de esenta de la versiunea 2.2.0.
- esenta Linux 2.6: contine un driver scris de Anton Altaparmakov (Universitatea Cambridge) si Richard Russon. Suporta citire de fisiere , scriere peste si modificarea marimii,in cateva cazuri.
- NTFSMount: Un driver cu suport limitat de scriere/citire al fisierului si al dosarului este disponibil folosind NTFSMount.
- NTFS pentru Linux : Un driver comercial cu suport total de citire/scriere disponibil de la Paragon.
- NTFS captiv: Un driver "de ambalare" care foloseste propriul driver the Windows, ntfs.sys.

Este notabil faptul ca toate cele 3 drivere de spatiu ale utilizatorului, in principal NTFSMount, NTFS-3G si NTFS captiv, sunt construite pe sistemul de fisiere in spatiul pentru user (FUSE), o esenta a modulului de Linux insarcinat cu arcuirea spatiului de user si al codului de esenta pentru a salva si a recupera datele.

Aproape toate driverele listate mai sus (cu exceptia Paragonului NTFS pentru Linux) sunt surse deschise (GPL). Datorita complexitatii structurilor interne NTFS, atat driverul esenta built-in 2.6.14 cat si driverele FUSE nu permit schimbari ale volumului care nu este considerat sigur, pentru a evita coruptia.

Windows

In timp ce versiunile diferite de NTFS au un mare grad atat de compatibilitate de inaintare cat si de inapoiere, exista consideratii tehnice pentru a monta volume noi de NTFS in variante mai vechi de Windows. Acest lucru afecteaza butarea dubla, cat si hard disk-urile portabile externe.

De exemplu, "Versiunile anterioare" (Copia Umbra a Volumului) sunt pierdute deoarece OS-ul mai vechi nu stie cum sa pastreze noile caracteristici ale datelor updatate.

Altele

Driverul NTFS-3G asigura acces citire/scriere sigur si total pe FreeBSD, Mac OS X, NetBSD, Linux, Haiku si FreeDOS.

Versiunile FreeBSD, eComStation si Mac OS X 10.3 si urmatoarele ofera suport NTFS read-only (exista un driver NTFS beta care permite scrierea/stergerea pentru eComStation, dar in general acest procedeu este considerat nesigur). Un instrument gratis tert pentru BeOS, care e bazat pe NTFS-3G, permite scrierea si citirea totala a NTFS.

Exista deasemenea un driver scriere/citire comercial pentru DOS numit "NTFS4DOS".

Compatibilitatea cu FAT

Momentan Microsoft furnizeaza un instrument (convert.exe) pentru a converti HPFS (doar pe Windows NT 3), FAT16 si, pe Windows 2000 si variante mai curente, FAT32 in NTFS, dar nu invers. Instrumente terte variate sunt toate capabile sa redimensioneze partitiile NTFS. Microsoft a adaugat abilitatea de a restrange sau a extinde o partitile cu Windows Vista .

Din motive istorice, versiunile de Windows care nu suporta NTFS ,toate tin timpul intern ca timp zonal local, si astfel asta fac toate celelalte sisteme de fisiere in afara de NTFS care sunt suportate de versiuni curente de Windows.Totusi, Windows NT si descendentii sai inregistreaza timpi de tip UTC si fac conversiile cele mai potrivite pentru scopuri de display. Astfel, timpii NTFS sunt in UTC. Asta inseamna ca atunic cand fisierele sunt copiate sau mutate intre partitiile NTFS si non-NTFS, OS-ul trebuie sa converteasca timpii

pe loc. Dar daca cateva fisiere sunt mutate cand timpul de salvare al zilei (DST) este activ, si alte fisiere sunt mutate cand timpul standard este activ, pot aparea cateva ambiguitati in conversii. Ca rezultat, in mod special la scurt timp dupa una din ziele in care timpul zonal local se schimba, userii pot observa ca unele fisiere au timpi incorecti cu o ora. Datorita diferentelor in implementarea DST intre emisferele nordice si sudice, asta poate rezulta intr-o potentiala eroare de timp mai sus de 4 ore in oricare 12 luni.

Article I.Caracteristici

NTFS v3.0, a 3-a versiune a NTFS care este introdusa, include diverse noi caracteristici deasupra predecesorilor ei: partea folosirii discului, suportul fisierului din loc in loc, cautarea linkului si criptarea nivelului de fisier, cunoscut ca si Fisierul de Sistem Criptat (EFS)

- **Fluxul alternat de date (ADS)**

  Fluxul alternat de date permite fisierelor sa fie asociate cu mai mult de un flux de date. De exemplu, un fisier ca text.txt poate avea un ADS cu numele de text.txt : secret.txt (a formei filename:ads) care poate fi accesata cunoscand numele de ADS sau navigand programe in dosare specializate. Fluxurile alternate nu sunt detectabile in marimea originala a fisierului, dar sunt pierdute cand fisierul original (ex: text.txt) este sters cu un InlocuireFisier sau cu un apel la InlocuireaFisier (sau un apel care foloseste acele apeluri) , sau cand fisierul este copiat sau mutat intr-o partitie care nu suporta ADS (ex: o partitie FAT, un floppy, sau o partajare de retea). In timp ce ADS este o caracteristica folositoare, poate usor ocupa spatiul hard discului daca a fost uitat sau n-a fost detectat.

- **Cote**

  Cotele discului au fost introduse in NTFS v3. Ele permit administratorului computerului care ruleaza o versiune a Windowsului care suporta NTFS sa seteze o limita a spatiului de disc pe care userii o pot utiliza Permite de asemenea administratorilor sa tina urma a cat spatiu al discului foloseste fiecare utilizator. Un administrator poate specifica un nivel concret de spatiu pe care un user il poate folosi inainte sa ajunga la limita de spatiu. Cotele discului nu tin cont de compresia fisierului transparent NTFS , daca acesta este ativat.

- **Fisierele risipite**

  Fisierele risipite sunt fisiere ce contin seturi de date risipite, date in principal umplute cu zerouri. Multe aplicatii stiintifice pot genera mari seturi de date risipite. Din aceasta cauza, Microsoft a implementat suport pentru fisierele risipite permitand o aplicatie sa specifice regiuni de date goale(zerouri). O aplicatie ce citeste un fisier risipit il citeste in mod normal cu systemul de fisier , calculand ce date trebuie returnate ,bazandu-se pe ofsetul fisierului. O data cu fisierele compresate, marimea initiala a fisierelor risipite nu sunt luate in considerare cand se determina limitele cotei.

- **Puncte de montare pentru volumelor**

  Asemanator punctelor de montare Unix, unde radacina altui fisier de sistem este atasata unui dosar. In NTFS acesta permite fisiere aditionale sa fie montate fara a necesita o litera separata (ex C: sau D:) pentru fiecare.

- **Jonctiuni ale dosarelor**

  Asemanator punctelor de montare pt volume, oricum jonctiunile dosarelor leaga alte dosare din sistemul de fisiere in loc de alte volume. De exemplu, dosarul C:\exampledir cu un atribut junctiune dosar care contine o legatura catre D:\linkeddir va trimite automat spre dosarul D:\linkeddir cand este accesat de o aplicatie in mod utilizator. Acesta functionalitate este similara coceptual cu o legatura simbolica spre un dosar in UNIX exceptand faptul ca tinta in NTFS trebuie sa fie intotdeauna alt dosar. (Fisierele Unix permit in general legaturi simbolice spre orice tip de fisiere.)

- **Legaturi de tip „Hard”**

  Incluse initial pentru a suporta subsistemul POSIX in Windows NT, legaturile tip „Hard” sunt similare cu jonctiunile dosar dar folosite pentru fisiere. Acestea pot fi aplicate numai pentru fisierele care apartin aceluiasi volum.

- **Managmentul Ierarhic al stocarii (HSM)**

Managmentul ierarhic al stocarii reprezinta o cale de a transfera fisiere care nu sunt folosite pentru o perioada de time spre medii mai ieftine.

- **Stocare struturata nativ (NSS)**

  NSS a fost o tehnologie ActiveX de stocare a documentelor care de la inceput a fost oprita de Microsoft. Aceasta permitea documentelor ActiveX sa fie stocate in acelasi formate multi-flux pe care ActiveX in folosea intern. Un filtru fisier de sistem NSS a fost incarcat si folosit pentru a procesa fluxurile multiple independent de aplicatie si cand fisierul a fost copiat catre un disk formatat Non-NTFS va permite transferul mai multor fluxuri de date intr-unul singur.

- **Compresia fisierelor**

  NTFS poate compresa fisiere folosind o varianta a algoritmului LZ77 – folosit si de celebrul utilitar WinZip. Desi accesul citire-scriere catre fisiere compresate este transparent, Microsoft recomanda compresia pe sisteme server deoarece aceasta incarca in mod considerabil procesorul.

- **Stocarea unei singure instante (SIS)**

  Atunci cand mai multe dosare au unele fisiere identice SIS permite ca stocarea unui singur fisier si creeaza referinte catre acel fisier. SIS consta intr-un fisier de sistem tip filtru care se ocupa de copierea, modificarea si concatenarea fisierelor, si un serviciu al spatilui ultizatorului care cauta fisierele identice.

- **Criptarea fisierelor de sistem (EFS)**

  EFS ofera modalitati puternice si transparente de criptare a oricarui fisier sau dosar al unui volum NTFS. EFS lucreaza in conjunctura cu serviciile EFS, CryotoAPI al Microsoft si libraria EFS File System Run-Time Library.

- **Legaturi simbolice**

  Legaturile simbolice au fost introduse in Widows Vista. Legaturile simbolice (sau legaturi „Soft”) sunt rezolvate de catre client. Deci cand o

legatura simbolica este partajata, tinta este subiectul restrictiilor de acces ale clientului, nu ale serverului.

- **Transactional NTFS**

Ca si Windows Vista, aplicatiile pot folosi Transactional NTFS pentru a grupa schimbarile fisierelor impreuna intr-o sesiune. Sesiunea va garanta ca toate schimbarile se aplica, sau nici una dintre ele, si ca aplicatiile din afara sesiunii nu vor vedea schimbarile pana exact in secunda cand ele vor fi gata.

Article II.Limitari

Sistemul NTFS are urmatoarele limitari:

- **Nume de fisiere rezervate**

Desi sistemul de fisiere suporta cai de pana la 32,000 caractere Unicode cu fiecare component ai caii (dosar sau fisier) de pana la 255 caractere, unele nume sunt neutilizabile deoarece NTFS isi stocheaza meta-data in fisiere normale (desigur ascunse si in cea mai mare parte inaccesibile), deci utilizatorii nu pot folosi aceste nume. Aceste fisiere se afla toate in dosarul radacina al volumului (si sunt rezervate doar pentru acel dosar). Aceste nume sunt: $MFT, $MFTMirr, $LogFile, $Volume, $AttrDef, . (punct), $Bitmap, $Boot, $BadClus, $Secure, $Upcase, si $Extend; . (punct) si $Extend sunt ambele dosare, celelalte sunt fisiere.

- **Maximum Volume Size**

Teoretic, dimensiunea maxima a unui volum NTFS este2^{64}-1 grupari dar practic aceasta dimensiune a fost implementata in Windows XP ca 2^{32}-1 grupari. De exemplu, folosing grupari de 64kb, dimensiunea maxima a unui volum NTFS este de 256 TB minus 64 KB. Folosind marimea implicita a unei grupari de 4 kb dimeansinea maxima a unui volum NTFS este de 16 Tb minus 4 Kb. Deoarece tabela de partitii de pe discul master boot record (MBR) suporta partitii de pana la 2 Tb, volumele dinamice sau de timp GPT (Globally Unique Identifier Partition Table) trebuiesc folosite pentru a creea partitii bootabile NTFS de peste 2 TB

- **Marimea maxima a unui fisier**

  Teoretic aceasta este de 16 Eb minus 1 Kb ($2^{64} - 2^{10}$ bytes). Implementare: 16 Tb minus 64 kb ($2^{44} - 2^{16}$ bytes).

- **Fluxuri alternative de date**

  Sistemul Windows poate – sau nu poate – folosi fluxuri alternative de date. Depinzand de sistemul de operare, fisiere de sistem utilitare si rezervate, un transfer de fisiere poate extrage fluxuri de date neobservat. O modalitate sigura de a copia sau muta fisiere este folosind functiile de sistem BackupRead si BackupWrite, care permit programelor sa enumere fluxurile, sa verifice daca fiecare trebuie scris pe volumul destinatie si sa sara fluxurile amenintatoare.

- **Lungimea maxima a unui cai**

  O cale absoluta poate contine pana la 32767 caractere, iar una relativa pana la 255 caractere.

Capitolul 7
Sistemul HFS

Hierarchical File System (HFS) e un system de fisiere dezvoltat de Apple Inc. pentru a fi folosit in computerele ce ruleaza Mac OS. Conceput initial pentru folosirea pe hard diskuri sau pe floppy diskuri poate fi gasit si pe medii read-only cum sunt CD Rom-urile.

HFS a fost introdus de Apple in Septembrie 1985 ca inlocuitor pentru Macintosh File System (MFS), primul sistem de fisiere introdus un an mai devreme cu calculatorul Macintosh. Dezvoltat de Patrick Dirks si Bill Bruffey HFS impartaseste cateva caracteristici cu MFS, care nu erau disponibile la vremea respectiva in alte sisteme de fisiere cum era de exemplu FAT.

Fisierele puteau avea mai multe forkuri de informatie (de obicei de date si de resurse), cu ajutorul carora codul uinui program putea fi stocat separat de resursele programului cum ar fi icoanele. Fisierele erau apelate dupa un ID unic decat dua nume, iar numele unui fisier putea contine 255 caractere.

Din cauza ca MFS era optimizat pentru a fi folosit pe unitati media mici si incete, anume floppy disk-uri HFS a fost introdus pentru a rezolva problemele de performanta aparaute cu introducerea unitatilor mai mari anume a hard disk-urilor. Principala problema era timpul necesar pentru aratarea continutului unui folder. La MFS toate informatiile despre foldere si fisiere era stocate intr-un singur fisier in care sistemul trebuia sa caute pentru a construi o lista cu fisierele dintr-un anumit director. Acest procedeu dadea rezultate pe un sistem de cateva sute de kilobytes de stocare sau cateva sute de fisiere dar pe masura ce capacitatea de stocare a crescut la megabytes performantele au scazut vizibil.

Solutia a fost inlocuirea structurii de directoare a MFS cu una potrivita pentru sisteme de fisiere mai mari. HFS a inlocuit structura tabelara cu Catalog File care foloseste o structura de tip arbore B*, o strucutra in care se pot face cautari foarte rapide idiferent de dimensiuni. HFS are refacute diferite structuri pentru a putea stoca numere mai mari, integerii pe 16 biti fiind inlocuiti in totalitate de cei pe 32 de biti. Ciudat singurul loc unde aceasta marire nu a avut loc este directorul de fisiere, unde HFS este limitat la maxim 64k fisiere.

Desi HFS este un sistem de fisiere de sine statator, este bine documentat si sunt multe modalitati de a accesa diskurile formatate HFS din majoritatea sistemelor de operare moderne.

In 1998, Apple introduce HFS Plus pentru a adresa neajunsuri legate de alocarea ineficienta a spatiului pe disk in HFS si pentru a adauga alte imbunatatiri. HFS este compatibil cu versiunile curente de Mac OS, dar incepand cu Mac OS X un volum HFS nu poate fi folosit pentru bootare.

Design

Sistemul de fisiere ierarhic (HFS) imparte un volum in blocuri logice de 512 bytes. Aceste blocuri sunt grupate in blocuri de alocare care pot contine unul sau mai multe blocuri logice, in functie de dimensiunile volumului. HFS foloseste o valoare pe 16 biti pntru a apela blocurile de alocare, limitand numarul acestora la 65 536.

Sunt 5 structuri care alcatuiesc un volum HFS:

1. Blocurile logice 0 si 1 ale volumului sunt blocuri de boot care contin informatiile pentru start-up-ul sistemului.

2. Blocul logic 2 contine Master Directory Block (MDB). In acesta sunt definite o mare parte de date referitoare la volum in sine, de exemplu data si ora cand volumul a fost creat, locatia altor structuri ale volumului sau marimea structurilor logice cum ar fi blocurile de alocare. Mai exista un duplicat al MDB numit Alternative Master Directory Block localizat la capatul celalalt al volumului. Acesta este folosit in mare parte de utlitarele de disc.

3. Blocul logic 3 este blocul de inceput pentru Volume Bitmap, care tine cont care blocuri de alocare sunt folosite si care sunt libere. Fiecare bloc de alocare din volum este reprezentat printr-un bit in Volume Bitmap. Daca bitul este setat blocul este folosit, daca bitul este liber blocul este deasemenea liber.

4. Extetent Overflow File e un arbore B* care contine spatiu suplimentar unde se inregistreaza ce blocuri de alocare sunt folosite pentru care fisiere, odata ce spatiul din Catalog File este folosit. Versiunile mai noi au adaugat abilitatea ca in Extent Overflow File sa poate stoca informatii despre blocuri bad, pentru a impiedica sistemul sa aloce un bloc bad unui fisier.

5. Catalog File este un alt arbore B* care contine registri despre toate fisiserle si directoarele stocate in volum. Stocheaza 4 tipuri de informatii. Fiecare fisier contine un File Thread Record si un File Record in timp ce fiecare director contine un Directory Thread Record si un Directory Record. Fisierele si directoarele din Catalog File sunt apelate dupa un unic Catalog Node ID sau CNID.

Probleme specifice HFS

Catalog File care stocheaza toate informatiile despre fisiere si directoare intr-o singura structura de date sufera de probleme legate de performanta cand sistemul adminte multitasking, pentru ca un singur program pate scrie/citi din aceasta structura intr-un moment, astfel ingreunand sistemul.
Mai apare si o problema de siguranta pentru ca o eroare in acest fisier poate distruge intregul sistem. Acest lucru vine in contrast cu alte sisteme de fisiere care tin informatiile legate de fisiere si directoare in structuri separate (cum ar fi FAT sau Unix File System) unde stocarea structurile pe tot discul inseamna ca stricarea unui singur director nu e fatala si datele pot fi recuperate folosind datele retinute in portiunile stricate.

Capitolul 8
Tehnologii RAID de stocare si securitate a datelor

Definite

RAID (Redundant Array of Inexpensive (or Independent) Drives (or Disks)) este o tehnica implementata hardware sau software folosind doua sau mai multe unitati HardDisk pentru a crea un singur Disk logic cu performante inbunatatite.

Istorie

Prima decriere a tehnicilor RAID a fost patentata in anul 1987 cu titlul "System for recovering data stored in failed memory unit." de catre Norman Ken Ouchi (IBM). Revendicarile facute in acest patent descriu ce vom numi in continuarea acestei prezentari termenul de RAID 5. De asemenea mentioneaza tehnici de „disk mirroring” si „disk duplexing” (RAID 1) si protectie cu paritate dedicata.

Abia in anul 1987 termenul RAID a fost definit de catre David A. Patterson, Garth A. Gibson si Randy Katz (Berkeley University of California) prin lucrarea "A Case for Redundant Arrays of Inexpensive drives (RAID)". Cu toate ca pana in prezent implementarile tehnicilor RAID difera substantial fata de cele din anul 1987 numele numerotate ale tehnicilor au ramas (RAID 1,2,3,4...).

Avantaje si Aplicabilitate

La baza orcarei inovatii tehnologice sta necesitatea. RAID a aparut si a fost dezvoltat ca urmare a incapacitatii unui simplu HDD de a face fata cerintelor tehnice din ce in ce mai mari. Crearea unor dispozitive cu viteza de acces si spatiul de stocare cat mai mari au pus in dificultate (sau imposibilitate) producatorii de HDD. De aceea tehnica RAID aduce urmatoarele avantaje:

Accesul in paralel a datelor ceea ce mareste gradul de acces la date decat pe un singur disc si in acelasi timp se mareste si spatiul de stocare. Pretul pentru o unitate (in cazul in care ar putea fi tehnologic realizabila) cu spatiu de stocare 10 TB cu o viteza de aces de 700MB pe secunda ari fi enorm.

Redundanta datelor pe mai multe discuri ofera protectie in cazul defectarilor. Spre exemplu, se pot folosi configuratii care, in cazul defectarii si inlocuirii unui HDD, datele sa fie recuperate fara erori.

Domeniul de folosire al acestuia este cel al serverelor, in care nevoia pentru viteza de acces si spatiul de stocare este foarte mare. Parasind domeniul enterprise se pot totusi intalni tehnici RAID si in statii de lucru, insa nivelul RAID folosit este mic (deobicei intalnit RAID 0).

Moduri de implementare a tehnicii RAID

Distributia de date intre multiplele unitati poate fi realizate atat pe cale software cat si pe cale hardware. Aditional exista si un mod hibrid, partial hardware, partial software.

Implementare software

Implementarile software sunt furnizate de majoritatea sistemelor de operare. Acest layer este pus deasupra driverelor discurilor si ofera un strat de abstractizare discurile logice si discurile fizice. In mod tipic RAID-ul software este limiat la RAID 0, 1 si 5. In sisteme de operare muti-thread (Linux, Mac OS X, Novell,Windows NT, 2000..) sistemul de operare poate face mai multe procese I/O (permite multiple scrieri si citiri). Aceasta capabilitate permite, in plus, realizarea RAID 0/1 in implementare software cu toate ca nimeni nu foloseste aceasta practica. Motivul este cererea ridicata de putere de procesare pentru calcularea paritatii (datele redundante). Implementarile software necesita foarte putine resurse si timp de prelucrare mici. In anul 2007 calculatoarele personale, obijnuite, ofera puterea de calcul necesara crescand folosirea procesorului cu doar un procent in plus.Ele sunt eficiente ca si cost (pana intr-un punct) insa nu ofera acelasi grad de performanta ca si RAID-ul bazat pe hardware. Fiind software, el trebuie sa foloseasca resursele unui server gazda legat de spatiul de stocare, CPU-ul gazdei fiind o parte ocupat cu procesarea datelor. In cazul unei defectiuni a gazdei datele din spatiul de stocare nu pot fi accesate atat cat defectiunea persista. Inhardware matricea de HDD-uri poate fi atasata usor altei gazde astfel timpul de recuperare se reduce simtitor .Totusi acest sistem este la fel de eficient ca si cel software in cazul in care nu exista un sistem de back-up al hostului pe care sa se faca trecerea rapid.

O alta facilitate adusa de implementarea software este ca permite crearea de matrici RAID si din partitii ale unui HDD.

Implementare hardware

RAIDul hardware are ca cerinta minima un controler RAID. Intr-un calculator personal acesta poate fi un card PCI sau poate fi inclus in cip-set-ul placii de baza. In aplicatiile industriale controlerul sicreate ca si unitate separata. HDD-urile pot fi IDE/ATA, SATA, SCSI, SSA, canal optic si combinatii intre acestea.Unitatea ce foloseste acest sistem de stocare poate fi legat direct la controler sau, mai des intalnit, printr-o retea SAN (Storage Area Netork - o retea ce foloseste un protocol de comunicare asemanator cu SCSI ce, spre exemplu, se poate suprapune peste o retea EthernetControlerul se ocupa de managementul discurilor si calculeaza datele redundante caracteristice nivelului RAID ales. Majoritatea acestor sisteme au implementate o memorie cache nevolatila, ce inbunatatesc performantele. Implementarea hardware prezinta performante garantate, fara utilizare a procesorului gazdei, controlerul prezentand sistemului de operare un simplu disc logic. De asemenea, sistemele industriale prezinta suport pentru "hot swapping" (schimbarea HDD-urilor defecte fara a fi necesara oprirea sistemului).

Implementare hibrida

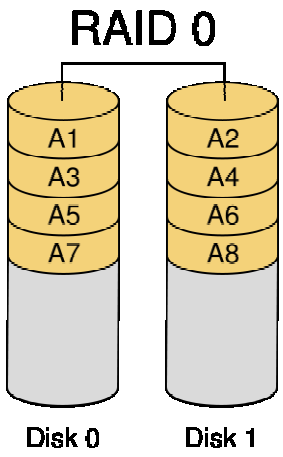
Echipamentele RAID hibride au aparut odata cu introducerea controlerelor RAID ieftine, implementate in controlere HDD si ca extensii de BIOS reprezentate de divere de sistem. Din nefericire aceste controlere fac toate calculele necesare in regim software, ele aduna majoritatea dezavantajelor implementarii software si hardware. Ca si implementarile HW. ele sunt proprietare unui fabricant de controlere RAID si nu se pot face combinari intre mai multe controlere. Avantajele constau in abilitatea de a boot-a de pe discul logic si integrarea mai stransa cu driverul device-ului ce ofera o manipulare mai buna a erorilor.

Nivele RAID

Nivele standard

Nivelele RAID standard sunt un set de configuratii de baza ce se gasesc in orice implementare ce implica tehnicile : striping (partajarea unui bloc de date in mai multe "fasii" - stripes), mirroring (oglindirea, copierea datelor) sau parity (tehnica resopnsabila pentru generarea datelor de corectie si control a erorilor).

RAID 0

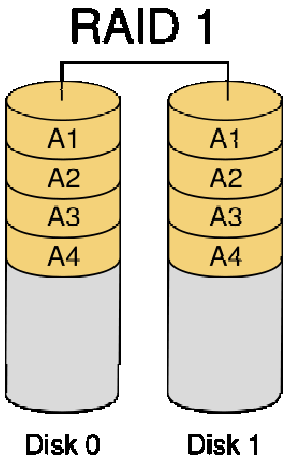


RAID 0 (sau "stripe set") inparte blocurile de date in mod egal pentru fiecare doua sau mai multe discuri fara date ce asigura redundanta. De mentionat este ca RAID 0 nu este inclus in primele documente ce definesc tehnicile RAID si nu ofera reundanta a datelor. RAID 0 este folosit pentru a inbunatati performantele si pentru a crea un numar mai mic de discuri virtuale.

Timpul de defectare a acestei metode este n ori mai mic decat timpul de defectare a unui singur disc ceea ce maresa vulnerabilitatea in timp al acestui sistem. Pierderea unui singur HDD duce la o pierdere masiva de date. Cu toate ca exista posibilitatea de recuperare a datelor pierdute prin programe speciale (in cazul in care discul nu este in totalitate stricat), cel mai probabil ele vor fi incomplete si corupte. Astfel recuperarea de date in acest caz este foarte scumpa si ne este garantata.

Timpul de acces este inbunatatit in special daca sistemul este folosit in aplicatii ce necesita citiri de date mai mici decat blocurile de date scrise (de obicei dimensiuni de 512B).Un exemplu bun ar fi bazele de date. La fisierele de dimensiuni mari timpul de acces este la fel ca al unui singur disc.Avantajul apare atunci cand datele sunt dispersate in mod egal pe discuri si cand zonele de interes sunt mai mici ca dimensiuni decat unitatea de stripe.Putin mai norocosi au fost cei ce au incercat performantele in divertisment (jocuri) inregistrandu-se performante minimale sau chiar negative.

RAID 1



Functionarea este simpla, RAID 1 creaza copii a discului, imagini in oglinda. Acesta este folositor cand este nevoie de perfomante bune la citire, fata de performante asupra capacitatii maxime, teoretic ajungandu-se la viteze duble de acces la citire. Astfel de matrice poate avea marimea maxima cat al celui mai mic disc din sistem. Pentru ca fiecare HDD contine o copie a informatiei creste (exponential) redundanta sistemului.

RAID 1 are multe avantaje administrative. Spre exemplu se poate crea un backup al sistemului fara intreruperea serviciilor prin simpla deconectare a discului si apoi copierea lui pe un sistem de backup. Apoi se introduce discul la loc in sistemul RAID si sistemul isi "reconditioneaza" imaginea dupa ultimele modificari.

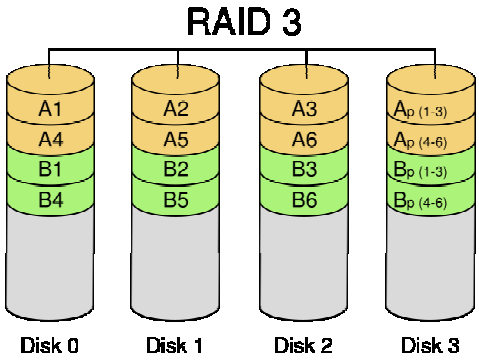
RAID 2

Cu toate ca este o metoda neintalnita in practica ea este singura ce ofera detectie si corectie a erorilor cu acuratete. Celelalte metode pot detecta erori, pot corecta o parte din ele dar in cazuri mai complicate nu se pot descurca fara interventie umana. Principiul este simplu. Datele sunt impartie in Bytes pentru care sunt calculate datele redundante conform codului Hamming pt detectie si corectie a erorilor. Discurile sunt impartite in proportie 4 la 3 (4 discuri date , 3 discuri biti de control).

Dezavantajul este ca discurile trebuie sa se roteasca exact la aceiasi viteza, necesitand un acces sincron asupra lor ceea ce duce la un cost ridicat de productie.

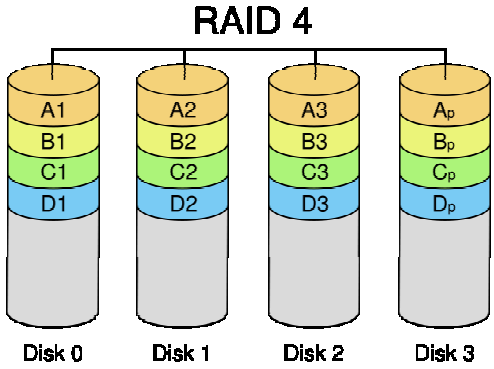
RAID 3

La fel ca si RAID 2 si aceasta metoda este mai putin intalnita. Foloseste imparte blocul de date la nivel de byte si foloseste un disc separat pentru calcularea bitilor de paritate (prin XOR logic). Efectul negativ este ca nu poate face fata la mai multe cereri de date simultan, datele unui pachet fiind imprastiat pe toate discurile.



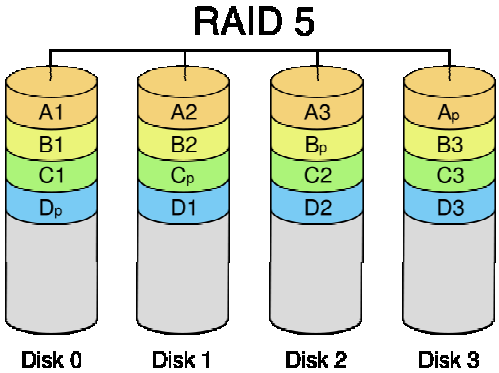
RAID 4

RAID 4 foloseste, spre deosebire de 3, stripe-uri la nivel de bloc de date. Asta permite fiecarui disc sa se comporte independent, accesul fiind mult inbunatatit.Foloseste n discuri pentru date si unul pentru calculul paritatii. De obicei RAID 4 este implementat hardware cu suport pentru calcularea paritatii.



RAID 5

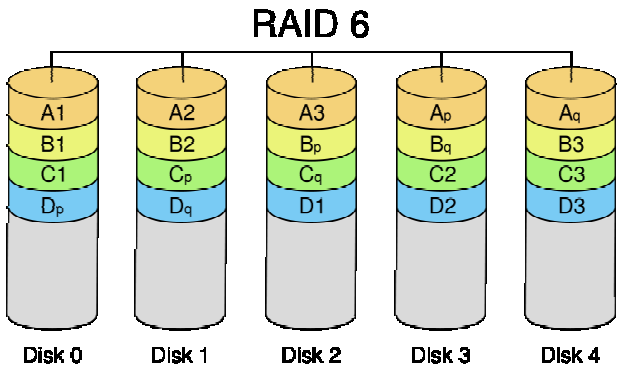
Asemanator cu RAID 4 foloseste stripe-uri la nivel de bloc de date, cu conditia ca pachetele de paritate sint impatrite uniform in toate discurile. El pastreaza avantajul accesului simultan asupra discurilor.Diferenta fata de RAID 4 este ca, atunci cand un disc cedeaza RAID 4 nu trebuie sa calculeze datele de paritate (in exemplul de mai sus, in cazul defectului 1/4 din disc reprezinta date redundante ce nu trebuiesc calculate).Performantele sunt asemanatoare cu RAID 0 pentru acelasi numar de discuri.In cazul unui defect in timpul scrierii datelor blocul de paritate poate sa devina eronat si va duce la o



reconstituire defecta. Aici intervin controlorele RAID cu memorie cache nevolatila pentru a reduce acest risc.

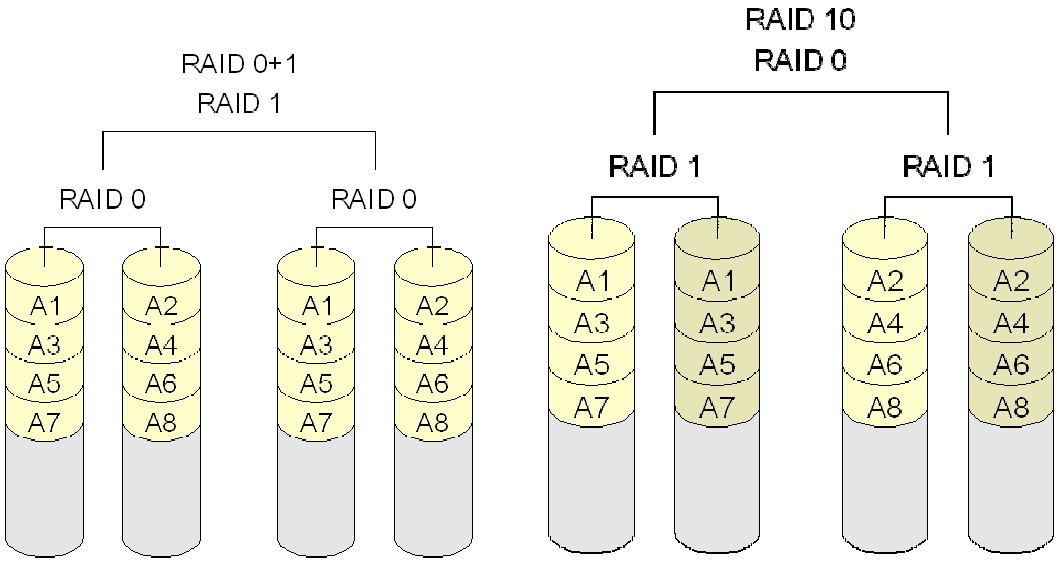
RAID 6

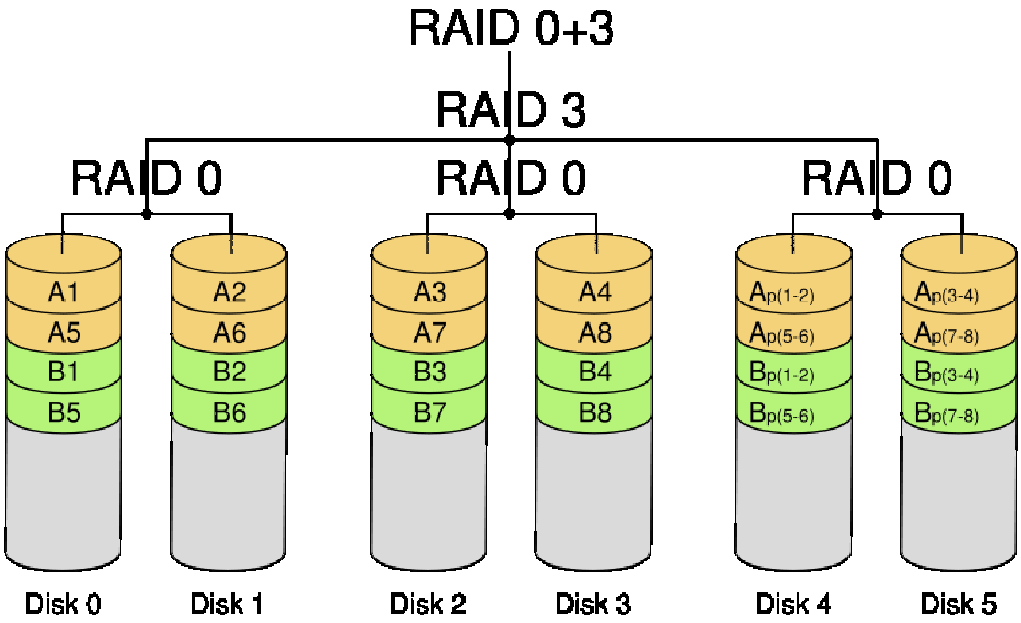
Noutatea adusa fata de RAID 5 este introducerea unui bloc de date de control al paritatii in plus. Aceasta metoda poate fi vazuta ca si un caz mai special al codarii Reed-Solomon. In principiu aceasta tehnica permite (conform definitiei SNIA - Storage Networking Industry Association) continuarea operatiilor de scriere si citire cu doua discuri defecte in sistem.



Nivelurile RAID Hibride (Nested - cuibarit)

Principiul de functionare a nivelelor hibride este de a folosi in locul discurilor fizice alte nivele RAID. Exemple:




RAID 0+3
RAID 3
RAID 0
RAID 0
RAID 0
A1
A5
B1
B5
A2
A6
B2
B6
A3
A7
B3
B7
A4
A8
B4
B8
Ap(1-2)
Ap(5-6)
Bp(1-2)
Bp(5-6)
Ap(3-4)
Ap(7-8)
Bp(3-4)
Bp(7-8)
Disk 0
Disk 1
Disk 2
Disk 3
Disk 4
Disk 5

Capitolul 9
Functii Win32 API

O interfata API este un cod sursa oferit de catre sistemul de operare sau o librarie pentru a permite apeluri la serviciile care pot fi generate din API-uri respective de catre un program.
Un program care ofera functionalitatea descrisa de interfata API este implementarea interfetei API. Interfata API in sine este abstracta, in sensul ca specifica instanta dar nu se implica in detalii de implementare.
Termenul API este folosit in 2 sensuri:

- O interfata coerenta care consta din cateva clase sau cateva seturi de functii sau proceduri interconectate.
- Un singur punct de intrare, cum ar fi o metoda, o functie sau o procedura.

Doua Interfete API foarte cunoscute sunt Single UNIX Specification si Microsoft Windows API.
Interfete API sunt deseori incorporate in Software Development Kit (SDK) [1].

Modelul de design a Interfetelor API

Exista o multime de modele de design a Interfetelor API. Cele prevazute pentru executie rapida deseori consta din functii, proceduri, variabile si structuri de date. Exista si alte modele cum ar fi interpretatori (emulatori)[2] care evalueaza expresii in ECMAScript (cunoscut sub nume JavaScript) sau alt layer abstract, oferind programatorului posibilitatea de a evita implicarea in relatiile functiilor cu nivelul inferior al abstractiei.
Unele Interfete API, cum sunt cele standard pentru un sistem de operare, sunt implementate ca librarii de cod separate care sunt distribuite impreuna cu sistemul de operare. Altele au integrata functionalitatea interfetei API direct in aplicatie. Microsoft Windows API este distribuita cu sistemul de operare. Interfetele API pentru sisteme embedded, cum sunt console de jocuri video, in general intra in categoria API-urilor integrate direct in aplicatie.

[1] SDK este un set de unelte de dezvoltare care ofera posibilitate de creare a aplicatiilor pentru un anumit pachet software, software framework, platforma hardware, sistem de operare, console de jocuri video, etc.
[2] Interpretatori compileaza codul sursa atunci cand acesta este intalnit in executia programului, spre deosebire de compilatori care separa faza de compilare de cea de executie

O interfata API care nu necesita drepturi mari de acces sunt numite "open" (OpenGL ar fi un exemplu).
Doua linii generale ale diplomatiei de publicare a Interfetelor API:

- Unele companii protejeaza informatiile despre Interfete API. De exemplu, Sony a facut Interfata API oficiala pentru PlayStation 2 disponibila doar dezvoltatorilor cu drepturi speciale. Asta a dat posibilitatea companiei Sony de a controla cine scrie jocuri pentru PlayStation 2.
- Unele companii fac Interfete API disponibile in public. De exemplu, Microsoft are o mare parte a informatiilor despre Interfete API disponibile freeware, ce ofera posibilitatea de dezvoltare a programelor pentru platforma Windows

Cateva exemple de Interfete API:

- Single UNIX Specification (SUS)
- Microsoft Win32 API
- Java Platform, Enterprise Edition API's
- OpenGL cross-platform API
- DirectX for Microsoft Windows
- Google Maps API
- Wikipedia API
- Simple DirectMedia Layer (SDL)
- svgalib pentru Linux si FreeBSD
- Carbon si Cocoa pentru Macintosh OS

Microsoft Windows API's

Windows API, neoficial WinAPI, este numele dat de catre Microsoft pentru un set de Interfete API disponibile in sisteme de operare Microsoft Windows. Aceste interfete au fost construite pentru a fi folosite de catre programatori C/C++ si sunt cel mai direct mod de a interactiona cu sistemul Windows pentru aplicatii software. Accesul la nivel inferior la sistemul Windows, in general necesar pentru drivere, este oferit de catre Windows Driver Foundation in versiunea curenta a Windows-ului.
In sisteme de operare Windows este disponibil un Software Development Kit (SDK), care ofera documentatia si unelte pentru a permite dezvoltatorilor crearea aplicatiilor folosind Interfete API si tehnologii Windows asociate.

Istoria

Interfetele API Windows au oferit dintotdeauna acces la structura sistemelor Windows. Din acest motiv ele sunt construite in mare parte pentru programatori. Programatorilor li s-a oferit multa flexibilitate si putere in dezvoltarea aplicatiilor. In aceasi timp aplicatiilor Windows li s-a impus mare responsabilitate in manipularea nivelelor inferioare.
Pe parcurs au fost facute multe modificari la sistemul de operare Windows si Interfetele API Windows au fost de asemenea schimbate pentru a tine pasul cu sistemul de operare. Interfetele API pentru Windows 1.0 au avut mai putin de 450 de functii, in comparatie cu API-uri moderne care au mii de functii. Avand asta in vedere, Microsoft a pus mare accent pentru compatibilitatea inversa, adica compatibilitatea API-urilor noi cu API-uri din urma.
Una dintre cele mai mari schimbari facute de Microsoft era schimbarea de la Sisteme de Operare pe 16 biti la Sisteme de Operare pe 32 biti. Pentru a oferi compatibilitate, Microsoft a scris, pentru noile versiuni pe 32 biti, o schema complexa de Interfete API care permiteau codului scris pe 32 biti sa apeleze cod scris pe 16 biti (si invers in unele cazuri limitate).
Aproape fiecare noua versiune a sistemului de operare Windows a introdus schimbari in Interfete API Windows. Numele a ramas consistent intre diferite versiuni. In cele din urma Microsoft a schimbat numele din Win32 API in Windows API.
Desi Microsoft detine drepturile de autor asupra Interfetelor API Windows, in general este acceptat ca alti producatori sa emuleze Windows oferind Interfete API identice. Proiectul Wine este cea mai cunoscuta incercare de a oferi compatibilitate Interfetelor API Windows pe sisteme de operare Unix-like. *ReactOS* a mers cu un pas mai departe si a oferit emulare intregului sistem de operare Windows. *HX DOS-Extender* este alt proiect care emuleaza Interfetele API Windows , pentru a permite rularea programelor Windows mai simple din linia de comanda DOS.

Compilatoare suportate

Pentru a dezvolta software care foloseste Interfetele API Windows, este nevoie de compilator care poate importa si manipula fisierele DLL si obiectele COM caracteristice Microsoft-ului. Compilatorul trebuie sa accepte dialectul limbajelor C sau C++ si sa manipuleze IDL (interface definition language) fisiere si fisiere header care expun denumirile functiilor interioare ale Interfetelor API. Aceste compilatoare, unelte, librarii si fisiere header sunt impreunate in Microsoft Platform SDK (Software Development Kit). Pentru mult timp familia

de compilatoare si unelte Microsoft Visual Studio si compilatoare Borland, au fost singurele care puteau la cerintele sus mentionate. Acuma exista *MinGW* si *Cygwin* care ofera interfete bazate pe *GNU Compiler Collection. LCC-Win32* este disponibil pentru utilizare non-comerciala, continand compilator C si intretinut de catre Jacob Navia. *Pelles C* este compilator C gratuit oferit de catre Pelle Orinius.

Componentele Interfetelor API in Windows

Functionalitatea oferita de Interfete API Windows poate fi grupata in sapte categorii:

- Base Services (Servicii de baza)
  Ofera acces resurselor fundamentale disponibile in Windows. Sunt incluse sisteme de fisiere, dispozitive, procese, fire de executie, acces la registri Windows si tratarea erorilor. Aceste functii se afla in fisierele kernel32.dll si advapi32.dll pe sisteme de operare pe 32 biti.
- Graphics Devise Interface (Interfata Dispozitivelor Graice)
  Ofera functionalitate pentru afisarea continutului grafic pe monitoare, imprimante si alte dispozitive de iesire. Se afla in gdi32.dll pe sistemele de operare pe 32 biti.
- User Interface (Interfata utilizator)
  Ofera functionalitati pentru a crea si manipula ferestre si majoritatea controlaelor de baza, cum ar fi butoane si scrollbar-uri, pentru a recepta intrare de la mouse si tastatura, impreuna cu alte functionalitati asociate cu GUI (Graphical User Interface). Aceste functii se afla in user32.dll. De la Windows XP, controlele de baza se afla in comctl.dll, impreuna cu controalele generale (Common Control Library).
- Common Dialog Box Library (Biblioteca Ferestrerol de Dialog Generale)
  Ofera aplicatii si ferestre de dialog standard pentru deschiderea si salvarea fisierelor, alegerea culuorii si fontului, etc. Linraria se afla in fisier numit comdlg32.dll si mai tarziu de la Windows 95 in fisier shlwapi.dll. Este grupat sub categoria *User Interface* a Interfetelor API.
- Common Control Library (Libraria de Control Generala)
  Ofera acces aplicatiilor la unele controale avansate ale sistemului de operare. Acestea includ status bars, progress bars, toolbars si tabs. Libraria se afla intr-un fisier DLL numit comctl32.dll. Este grupat sub categoria *User Interface* a Interfetelor API.
- Windows Shell

Componenta Interfetelor API Windows care care permite aplicatiilor acces la functionalitatile shell ale sistemului de operare, la fel ca si schimbarea lui in vederea inbunatatirii. Acesta componenta se afla in fisierul shell32.dll si mai tarziu incepand cu Windows 95 in shlwapi.dll. Este grupat sub categoria *User Interface* a Interfetelor API.

- Network Services (Servicii de retea)

  Ofera acces la posibilitatie sistemului de operare legate de retea. Subcomponentele lui includ NetBIOS, Winsock, NetDDE, RPC si multe altele.

Interfete API asociate Web

Internet Explorer contine multe Interfete API care sunt la randul sau folosite de alte aplicatii. Aceste Interfete API pot fi considerate ca parte a Interfetelor API Windows. Internet Explorer ofera:

- Controlul browser-ului web incorporat, aflat in shdocvw.dll si mshtml.dll.
- Servicii URL, aflate in urlmon.dll. Aplicatiile pot deasemenea oferi serviciile proprii URL.
- Librarii pentru suport multilingvisit si international (mlang.dll)
- DirectX Transforms, un set de filtre pentru imaigni
- Suport XML (MSXML componente)
- Acces la Windows Adress Book

Interfete API pentru interactionare intre programe

In mare parte Interfetele API Windows se ocupa de interactionare intre Sistemul de Operare si aplicatii care ruleaza pe el. Pentru a face posibila comunicarea intre diferiate aplicatii, Microsoft a dezvoltat o serie de tehnologii incorporate in Interfetele API Windows. Incepand cu Dynamic Data Exchange (DDE), care a fost inlocuit cu Object Linking and Embedding (OLE) si mai tarziu cu Component Object Model (COM)[3].

Interfete API asociate multimedia

[3] COM este o platforma Microsoft introdusa in 1993. Este folosita pentru a permite comunicare intre procese si crearea dinamica de obiecte in orice limbaj care suporta tehnologia respectiva. Implementarea nu depinde de limbaj. O inlocuire pentru COM este Microsoft .NET framework, iar suport pentru servicii web fiind Windows Communication Foundation (WCF). Windows Vista se bazeaza in special pe aceste elemente.

Microsoft a oferit un set de interfete API numit DirectX ca fiind incorporat in toate kiturile de instalare inca de la Windows 95 OSR2. DirectX ofera un set de servicii pentru jocuri si multimedia.

Microsoft DirectX

Microsoft DirectX este o colectie de Interfete API folosita pentru manipularea taskurilor legate de multimedia, in special programarea jocurilor si video, pe platforme Microsoft.
DirectX este de asemenea folosit si de alti producatori software, in mare parte in sectorul de inginerie, din cauza abilitatii de redare rapida a obiectelor 3D de inalta calitate.
Atat DirectX runtime cat si software development kit (kitul de dezvoltare soft) sunt disponibilie gratuit, dar sunt proprietate Microsoft si sunt closed-source (fara posibilitate de schimbare, rescriere, suprascriere). DirectX a fost initial redistribuit de catre dezvoltatori de jocuri impreuna cu kiturile de instalare, dar in ultim timp DirectX a fost inclus in kit de instalare a sistemului de operare (sau in Service Packs). Unii dezvoltatori de jocuri inca mai includ DirectX in kitul de instalare si ofera posibilitate de a-l instala (sau de a face update) dupa instalarea jocului.
Cel mai recente versiuni ale DirectX-ului sunt DirectX 10 si DirectX 9.0L, disponibile exclusiv pentru Windows Vista (motivul fiind, dupa cum sustine Microsoft, faptul ca exista schimbari in arhitectura grafica a Windowsului si din cauza introducerii Windows Display Driver Model).

Interfetele API din DirectX

Majoritatea Interfetelor API din DirectX sunt in forma de obiecte COM.
Componentele care le contine DirectX sunt:

- DirectX Graphics, alcatuit din doua Interfete API (DirectX 8.0 sau mai mult):
    - DirectDraw: pentru generarea de obiecte grafice 2D (acum dezaprobat, desi este inca folosit de multi programatori)
    - Direct3D (D3D): pentru generarea obiectelor grafice in 3D
- DirectInput: pentru tastaura, mouse, joustick sau alte controlere pentru jocuri (nu mai este folosit decat pentru Xinput, in controlere la Xbox 360)

- DirectPlay: pentru comunicare jocurilor in retea (Impreuna cu DirectInput a fost folosit ultima data in DirectX 8. Acuma este dezaprobat)
- DirectSound: pentru redarea sunetului si inregistrarea sunetului in forma wave
    - DirectSound3D (DS3D): pentru redarea sunetelor 3D
- DirectMusic: pentru redarea sunetelor autorizate de catre DirectMusic Producer
- DirectX Media: contine DirectAnimation pentru animatii web 2D, DirectShow pentru redare multimedia, DirectX Transform pentru interactionare web si Direct3D Retained Mode pentru obiecte grafice 3D de nivel inalt. DirectShow mai contine DirectX Plugins pentru procesarea semnalului audio si DirectX Video Acceseration pentru a accelera redarea video.
- DirectX Media Objects: suport pentru pentru obiecte cu streamuri cum sunt encodere, decodere si efecte
- DirectSetup: pentru instalarea componentelor DirectX (care de fapt nu prea este API)

Istoria

Spre sfarsitul anilor 1994 Microsoft era pe punct de a lansa urmatorul sistem de operare - Windows 95. Factorul care era sa determine succesul sistemului de operare era ce, de fapt, va putea fi rulat pe acest sistem de operare. Trei angajati la Microsoft (Craig Eisler, Alex John si Eric Engstorm) erau ingrijorati de faptul ca programatorii considerau sistemul de operare DOS (lansat tot de Microsoft) o platforma mai buna pentru dezvoltarea jocurilor. Asta insemna ca putine jocuri vor fi dezvoltate pentru Windows 95 si asta va influenta succesul acestui sistem de operare.
DOS oferea acces direct la placa video, tastatura, mouse-ul, dispozitivele de sunet si alte componenete ale sistemului, in timp ce Windows 95, avand memoria protejata, restrictiona acces la toate acestea. Microsoft avea nevoie de o modalitate de a oferi programatorilor ceea ce doresc. Eisler, John si Engstorm au dezvoltat o sulutie care in cele din urma era sa fie numita DirectX.
Prima versiune a DirectX-ului a fost lansata in Septembrie 1995 avand nume Windows Games SDK. Ea era, de fapt, inlocuire pentru Interfetele API din sistemele pe 16 biti, care aveau un design foarte slab. In mare, prima versiune de DirectX ofera posibilitate de a incorpora elemente multimedia de inalta performanta.

Inainte de existenta DirectX-ului, Microsoft avea deja inclus setul de API-uri OpenGL pe platformele Windows NT. In acel timp OpenGL necesita hardware de inalta performanta si era limitat pe inginerie si utilizatori CAD[4]. Direct3D (introdus de persoane care au dezvoltat DirectX ca o alternativa pentru OpenGL) urma sa fie o concurenta pentru (atunci) mai costisitor (din punct de vedere hardware) OpenGL pentru dezvoltarea jocurilor. Cu timpul, dupa ce puterea placilor video au crescut, OpenGL a devenit standard si conducatorul pietei. In acel punct a inceput "batalia" intre adepti ai cross-platform OpenGL si Windows-only Direct3D, care dupa cum multi sustineau era inca un exemplu de adoptare, extindere si eliminare (embrace, extend and extinguish)[5] de la Microsoft. Alte Interfete API din DirectX sunt deseori combinate cu OpenGL in crearea jocurilor video datorita faptului ca OpenGL nu include toate functionalitatile care le are DirectX (cum ar fi suport pentru sunet sau joystick). Totusi, combinatia intre OpenGL si OpenAL (Open Audio Library) pentru acest scop a devenit populara.
DirectX este folosit ca si baza pentru Interfetele API de la console Xbox si Xbox 360, oferite de catre Microsoft. Interfetele API au fost dezvoltate de catre Microsoft si NVIDIA (care a dezvoltat hardware folosit de consola). Intefetele API din Xbox sunt similare cu cele din DirectX 8.1, dar fara posibilitate de update. Xbox a fost numit DirectXbox, dar numele a fost prescurtat pentru motive comerciale.
In 2002 Microsoft a lansat DirectX 9 cu suport pentru programe *shader* [6] mult mai lungi cu *pixel* si *vertex shader*, versiunea 2.0. Microsoft a continuat cu update, si in August 2004 a introdus *shader* model 3.0 in DirectX 9.0c.

Compatibilitate

Producatori hardware sunt nevoiti sa scrie drivere si sa testeze fiecare bucata aparte, pentru a le face compatibile cu DirectX. Unele dispozitive hardware au doar drivere compatibile cu DirectX (ceea ce inseamna ca utilizatorul trebuie sa instaleze DirectX pentru a folosi acele dispozitive). Versiunile mai vechi de DirectX includeau si o librarie care continea toate driverele cunoscute si disponibile la momentul lansarii versiunii respective. La aceasta practica s-a

[4] Computer - aided design (CAD) este un set de unelte folosite pe calculator pentru a oferi asistenta pentru ingineri, arhitecti si alti designeri profesionisti pentru dezvoltarile lor.
[5] Este o fraza pentru a descrie strategia de la Microsoft care presupune acceptarea standardelor folosite, extinderea lor cu capabilitati proprii si apoi folosirea diferentelor obtinute pentru a distruge concurenta. Prima data aparut in New York Times in 1996 in articolul "Microsoft Trying to Dominate the Internet".
[6] Shader, in grafica calculatoarelor, este un set de instructiuni software, care este folosit de resurse grafice pentru efecte de redare.

renuntat in favoarea sistemului de reactualizare, bazat pe web, Windows Update care permite utilizatorilor sa descarce doar drivere relevante pentru dispozitivul lor.
Inainte de DirectX 10, DirectX era considerat ca fiind compatibil cu versiuni precedente (versiunile noi erau compatibile cu cele vechi). Asta este consecinta pozitiva a modelului COM folosit pentru API-ul DirectX.
Cu Windows Vista si Direct3D 10 cu schimbari radicale, compatibilitate nu mai este posibila. Din acest motiv DirectX 10 ofera Interfata API Direct3D 9, astfel incat sa fie posibila rularea jocurilor si aplicatiilor mai vechi.

DirectX 10 si 9.0L

Windows Vista este lansata cu DirectX 10 (si 9.0L pentru compatibilitate cu versiuni mai vechi) si este singurul sistem de operare din familia Windows care il contine. Schimbari radicale au fost facute: *DirectInput* este defavorizat si inlocuit cu Xinput, din Xbox. De asemenea, *DirectSound* este defavorizat si inlocuit cu XACT[7]. DirectX 10 a renuntat la suport pentru accelerarea dispozitivelor audio, iar sunetul este redat in softearul pe CPU. *DirectPlay* este defavorizat si inlocuit cu *Games for Windows - LIVE*[8], in timp ce *DirectShow* este defavorizat si inlocuit cu Media Foundation, un set diferit de Interfete API lansate impreuna cu Windows Vista, pentru manipularea de secvente audio si video. *DirectMusic* va fi probabil singurul ramas intact.

Direct3D

O proprietate noua majora in DirectX 10 este Direct3D (initial numit Windows Graphics Foundation). Folosind noul Windows Display Driver Model, Shader Model 4.0 si noile, mai stricte, cerinte pentru producatori de GPU (Graphical Processing Unit) care trebuie satisfacute pentru a sustine compatibilitate cu Direct3D, versiunea 10 de Direct3D reprezinta indepartarea de la practicile versiunilor mai vechi. Pentru a oferi compatibilitate cu versiunile precedente de Direct3D, DirectX 10 contine de fapt trei versiuni de Direct3D:

- Direct3D 9: aceasta Interfata API emuleaza comportamentele ale Direct3D 9 pe Windows XP pentru a oferi compatibilitate cu aplicatii

[7] XACT este libraria audio lansata de catre Microsoft ca fiind o parte din DirectX SDK. Initial a fost dezvoltata pentru Xbox, iar mai tarziu a fost modificata sa fie compatibila si cu Microsoft Windows.
[8] Games for Windows - LIVE este serviciul online de jocuri pentru Windows, care permite calculatoarelor, ruland Windows Vista, sa se conecteze la LIVE service (acest serviciu a fost initial dezvoltat pentru Xbox)

mai vechi. Toate detaliile si avantajele ale Windows Display Driver Model (WDDM) din Vista, sunt ascunse de aplicatie (in caz ca WDDM drivere sunt instalate). Aceasta este singura Interfata API disponibila in caz ca exista doar drivere grafice XP.

- Direct3D 9Ex (initial numit 9.0L): permite acces total la noile capabilitati ale WDDM, in timp ce pastreaza compatibilitate pentru aplicatii Direct3D existente folosind o interfata API separata. Efectul de transparenta ("glass effect") din *Windows Aero* se bazeaza pe codul din Direct3D 0Ex. Cand 9Ex era inca sub numele de cod 9.0L, existau zvonuri ca acesta va fi Direct3D 10 pentru Windows XP. Pana la urma s-a demonstrat ca nu este asa, in mare parte din cauza incompatibilitati al WDDM pe Windows XP.
- Direct3D 10: Temporar, singurul GPU compatibil cu Direct3D este NVIDIA GeForce 8 Series, care de fapt profita foarte putin de capabilitatile ale Direct3D 10.

Alternative

Exista alternative pentru aceasta arhitectura, unele mai complete decat altele. Nu exista o solutie unica care sa ofere toate functionalitatile care le ofera DirectX, dar prin combinare de librarii - OpenGL, OpenAL, SDL, OpenML, FMOD, etc - utilizatorul pote sa implementeze o solutie comparabila cu DirectX si in aceasi timp gratuita si cross-platform (poate fi folosite pe diferite platforme).

Bibliografie

[1] „Microsoft Windows Internals 4th Edition” - by Mark Russinovich
[2] „Windows NT/2000 Native API Reference” - by Gary Nebbett
[3] „The Windows NT Device Driver development” - Peter G. Viscarola, W. Anthony Mason
[4] „Windows NT File System Internals: A developer's Guide” - R. Nagar - capitolul despre driverele filtru
[5] „Microsoft Extensible Firmware Initiative FAT32 File System Specification” - FAT32 Whitepaper al firmei Microsoft.
[6] „Microsoft Developer Network” - pe scurt MSDN - http://msdn.microsoft.com/
[7] „Platform SDK Documentation”
[8] „Windows 2000/XP Device Driver Kit Documentation”
[9] NIST, "Advanced Encryption Standard (AES)”, Federal Information Processing Standards Publication 197, November 26, 2001, - http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.
[10] C. Adams, „The CAST-128 Encryption Algorithm”, Request for Comments 2144, May 1997, - http://www.rfc-editor.org/rfc/rfc2144.txt.
[11] RSA Laboratories, PKCS #5 v2.0: „Password-Based Cryptography Standard”, RSA Data Security, Inc. Public-Key Cryptography Standards (PKCS), March 25, 1999 - ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs5v2-0.pdf
[12] H. Krawczyk, IBM, M. Bellare, UCSD, R. Canetti, IBM, HMAC: Keyed-Hashing for Message Authentication, Request for Comments 2104, February 1997, - http://www.rfc-editor.org/rfc/rfc2104.txt.
[13] Peter Gutmann, „Software Generation of Practically Strong Random Numbers”, 1998 Usenix Security Symposium, - http://www.cs.auckland.ac.nz/~pgut001/pubs/usenix98.pdf.
[14] Victor Valeriu Patriciu, Monica Ene Pietroseanu, Ion Bica, Costel Cristea - "SECURITATEA INFORMATICA ÎN UNIX si INTERNET" - Editura Tehnica, 1998
[15] Dr. Brian Gladman – „Rutine AES, Serpent, şi Twofish”.
[16] Eric Young – Librăriile libdes, libcast şi blowfish.
[17] Multe alte resurse electronice de pe INTERNET.
[18] www.Wikipedia.org