Sistemul de fisiere

Zglimbea Alexandru (cap. 5) - coordonator

Manole Alexandru (cap.1, 2)

Paslaru Cristian (cap 3, 4)

Cuprins

Capitolul 1

Concepte fundamentale: exemple si comparatii Linux si Windows

1.1 Definitii. Concepte

Sistemul de fișiere (în engleză: *file system* sau și *filesystem*) desemnează organizarea internă a unui purtător de date, care de obicei este cuplat la un calculator. Pe parcursul vieții lor, fișierele sunt supuse multor operații: ele trebuiesc create, denumite, regăsite, renumite, citite, modificate, duplicate, deplasate, reorganizate, defragmentate, șterse ș.a. În tot acest timp este nevoie și de o corespondență clară între numele unui fișier, util în primul rând omului, și adresa sa de pe purtătorul de date, necesară calculatorului. Regăsirea rapidă a unui fișier este și ea importantă, în condițiile în care un singur purtător de date actual poate găzdui chiar și sute de mii de fișiere. Pentru a asigura viteze de funcționare mulțumitoare, organizarea internă a stocării și accesării fișierelor unui purtător de date trebuie să țină cont și să profite de caracteristicile acestuia. Programele necesare pentru operațiile cu fișiere fac de obicei parte din sistemul de operare al calculatorului la care sunt cuplate.

1.2 Reprezentarea partitiilor

Sistemul de fisiere adoptat de SO Linux este diferit fata de SO Windows. In Linux exista o singura structura de directoare. Totul incepe din directorul radacina(root), reprezentat de simbolul "/", dupa care se extinde in sub-directoare. In Windows acest lucru este tratat diferit : utilizatorul poate avea mai multe resurse tip disc , notate diferit A:-Z („drive letter"), A,B pentru FD:, fiecare fiind radacina a unui arbore separat. Linux plaseaza toate partitiile sub directorul radacina (root) in care se „mounteaza" sub denumirea unor directoare. Cel mai aproape de „radacina" in Windows este partitia „c:". Partitia reprezinta o fractiune din HDD-ul unui calculator a carei marime o selecteaza utilizatorul, aceasta fiind cuprinsa intre 1-100% din marimea hard-diskului. Un calculator poate avea una sau mai multe partitii in functie de necesitatile utilizatorului. Sub Windows, diversele partitii existente sunt detectate la bootare si li se ofera o litera din alabet (C-Z). Sub Linux, daca nu se mounteaza o partitie sau un dispozitiv, sistemul nu stie de existenta acestora.Nu este o metoda usoara de acces pentru incepatori, dar ofera o mare flexibilitate in sensul ca se pot ascunde anumite portiuni din HDD sau diverse componente in functie de necesitatile utilizatorului, e un plus si pentru securitate si ergonomie.
Acest mod de abordare gen sistem de fisiere unificat al Linuxului mai are si alte avantaje. De exemplu directorul „/usr" care contine majoritatea fisierelor executabile. Linux ne permite sa mountam acest director pe alta partitie sau chiar si pe alt calculator din retea. Sistemul nu va detecta nici o incompabilitate, deoarece /usr va aparea ca un director local. In Windows acest lucru nu este posibil, de exemplu mutarea directorului „Program Files" pe alta partitie nu pastreaza o functionalitate totala. Un lucru minor la prima vedere, dar piesa importanta in functionabilitate este faptul ca Linux foleste slash „/" pentru despartirea cailor catre diverse fisiere spre deosebire de Windows care foloseste back-salsh „\". Acesta reacomodare poate fi destul de dificila pentru un obisnuit cu Windows, durand ceva vreme pana la acomodare. Mai mult, Linux este un sistem „case sensitive" adica face diferenta intre litere mari si litere mici

spre deosebire de Windows la care nu conteaza daca denumire fisierului este introdusa cu litere mici sau litere mari. Sub Linux fisierele sunt identificate prin i-number(index dintr-un sir de i-noduri). Fiecare fişier are un singur ***i-node*** care conţine :

1. identificatorul utilizatorului ce este proprietarul fişierului;
2. tipul fişierului (obişnuit, director, pipe sau special);
3. drepturile de acces;
4. timpul ultimului acces şi al ultimei modificări, data şi ora ultimei modificări efectuate asupra i-node-ului;
5. numărul de legături (a se vedea comanda unlink);
6. adresele sectoarelor de pe HDD ce conţin datele fişierului;
7. lungimea fişierului în octeţi.

Andrew S. Tanenbaum, Sisteme de operare moderne, Byblos, 2004

1.3 Sisteme de fisiere Windows (FAT&NTFS)

Un sistem de fișiere este structura subiacentă utilizată de un computer pentru organizarea datelor pe hard disk. Dacă se instalează un hard disk nou, este nevoie de partiționarea și formatarea sa utilizând un sistem de fișiere, înainte de a începe stocarea de date sau programe. În Windows, cele trei opțiuni cu privire la sistemele de fișiere din care se alege sunt NTFS, FAT32 și FAT (cunoscut și ca FAT16), un sistem mai vechi și mai rar utilizat.

NTFS

NTFS (New Technology File System) este un sistem de fişiere dezvoltat special pentru Windows NT şi îmbunătăţit pentru Windows 2000. NTFS4 este folosit la Windows NT, în timp ce sistemul de fişiere pentru Windows 2000 este NTFS5. Windows XP foloseşte o versiune uşor îmbunătăţită a NTFS5. Facilităţile principale oferite de acest sistem de fişiere sunt următoarele: • foloseşte adrese de disc de 64 de biţi şi poate suporta partiţii de
până la 264 bytes ;

- permite folosirea caracterelor Unicode în numele de fişiere;
- permite folosirea numelor de fişiere de până la 255 de caractere, inclusiv spaţii şi puncte;
- permite indexarea fişierelor;
- oferă posibilitatea managementului dinamic al sectoarelor ;
- datorită compatibilităţii POSIX, permite crearea *hard-link*-uri, face distincţie între litere mari şi mici în cadrul numelor de fişiere şi păstrează informaţii de timp referitoare la fişier;
- permite utilizarea fişierelor cu seturi multiple de date.

FAT32

FAT32, și varianta mai rar utilizată FAT, au fost utilizate în versiuni anterioare ale sistemelor de operare Windows, incluzând Windows 95, Windows 98 și Windows Millennium Edition. FAT32 nu oferă securitatea furnizată de NTFS, astfel că, dacă dețineți o partiție sau un volum FAT32 pe computer, orice utilizator care are acces la computer poate citi orice fișier din acesta. FAT32 are, de asemenea, limitări de dimensiune. Nu aveți posibilitatea să creați o partiție FAT32 mai mare de 32 GO în această versiune de Windows, nici să stocați un fișier mai mare de 4 GO pe o partiție FAT32.

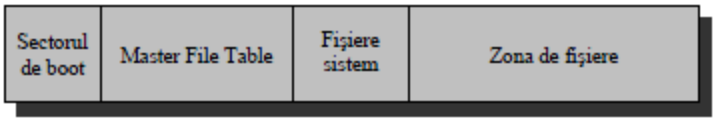
Principalul motiv de utilizare a FAT32 este deținerea unui computer care execută uneori Windows 95, Windows 98 sau Windows Millennium Edition, iar alteori execută această versiune de

Windows, configurație denumită și multi-încărcare. În acest caz, va fi necesar să instalați sistemul de operare mai vechi pe o partiție FAT32 sau FAT și să vă asigurați că este o partiție primară (una care poate găzdui un sistem de operare). Orice partiții suplimentare pe care va trebui să le accesați când se utilizează versiuni anterioare de Windows trebuie să fie, de asemenea, formatate utilizând FAT32. Aceste versiuni anterioare de Windows pot accesa partițiile NTFS sau volumele printr-o rețea, dar nu pe computerul personal.

1.4 Structura unei partiţii NTFS

La formatarea unei partiţii (volum) conform NTFS se creează o serie de fişiere sistem, dintre care cel mai important este fişierul *Master File Table* (*MFT*), care conţine informaţii despre toate fişierele şi directoarele de pe volumul NTFS, fiind un fel de baza de date a sistemului. Prima locaţie pe o partiţie NTFS este *sectorul de boot*, care este sectorul 0 al partiţiei şi conţine un program (cod executabil) de pornire a sistemului.

Alte informaţii necesare programului de boot-are (de exemplu informaţii necesare accesării volumului) pot fi înscrise în sectoarele de la 1 la 16, care sunt rezervate în acest scop. In figura este ilustratata structura unui volum NTFS la terminarea formatării. Un volum NTFS există cel puţin o intrare în MFT, inclusiv pentru MFT. Toate informaţiile despre un fişier, incluzând numele, dimensiunea, informaţii de timp referitoare la fişier, permisiuni şi datele efective sunt păstrate în MFT sau în spaţiul situat în exteriorul MFT-ului care descrie intrări în MFT. Toate aceste informaţii sunt considerate atribute ale fişierului, acesta fiind tratat ca o
colecţie de atribute. Un atribut este o secvenţă de octeţi organizaţi în două componente: componenta de descriere a atributului (header) şi conţinutul său. Atributele de fişier sunt păstrate în MFT, atunci când dimensiunea lor permite să fie memorate în intrarea corespunzătoare din MFT sau în zone auxiliare de pe HDD, exterioare fişierului MFT şi asociate intrării din MFT a fişierului.



Structura unui volum NTFS

1.5 Tipuri de fişiere şi drepturi de acces în NTFS

În NTFS putem identifica următoarele tipuri de fişiere:

• *fişiere sistem*: sunt fişierele descrise în tabelul de mai sus şi conţin informaţii ce sunt folosite numai de către sistemul de operare (metadata).

• *fişiere cu seturi multiple de date (Alternate Data Streams - ADS)*: sunt fişiere care pe lângă setul de date principal (implicit), mai conţin şi alte seturi distincte de date. Toate aceste
seturi de date sunt reprezentate prin atribute de tip *Data*. Modul de creare şi utilizare, pentru un fişier, a seturilor de date auxiliare celui principal, este descris mai jos.

• *fişiere arhivate*: NTFS poate arhiva şi dezarhiva fişierele „*onthe-fly*", adică în momentul efectuării operaţiilor de scriere şi respectiv, citire a datelor din ele. Acest mecanism este invizibil aplicaţiilor ce utilizează astfel de fişiere.

• *fişiere criptate:* EFS (*Encrypted File System*) oferă support pentru a stoca fişiere criptate pe un volum NTFS. Criptarea este transparentă pentru utilizatorii care au cerut criptarea fişierului.Accesul celorlalţi utilizatori nu este permis la aceste fişiere.

• *fişiere „rare” (sparse files):* sunt fişiere în care informaţia scrisă nu se găseşte într-o singură zonă contiguă, ci zonele în care s-au scris date alternează cu zone mari în care nu s-au scris („găuri”). NTFS permite setarea unui atribut special al acestor fişiere, prin care se indică sistemului de I/E să aloce spaţiu pe disc numai pentru zonele efectiv scrise din fişier.

• *fişiere de tip „hard-link”:* sunt fişiere speciale introduse de NTFS5. Aceste fişiere permit ca un fişier să poate fi accesat prin mai multe căi fără ca datele efective să fie duplicate. Dacă ştergem un fişier la care există şi o altă legătură, datele nu vor fi şterse de pe disc până când nu se şterg toate legăturile. Un fişier de tip *hardlink* poate fi creat folosind funcţia *CreateHardLink* sau comanda "fsutil hardlink create" (în Windows XP).

În ceea ce priveşte drepturile de acces în NTFS, ele sunt gestionate prin *liste de control al accesului (ACL).* Aceste ACL-uri conţin informaţii care definesc pentru fiecare utilizator sau grup de utilizatori drepturile pe care le are asupra unui fişier. Drepturile de acces se numesc *permisiuni.* Pentru a avea un control mai fin şi mai uşor asupra drepturilor de acces, s-au introdus (începând cu Windows 2000) nişte grupuri de permisiuni, denumite *componente de permisiuni.* Fiecare dintre ele grupează una sau mai multe permisiuni speciale, după cum urmează:

Traverse Folder / *Execute File* setată pentru permisiunea X
List Folder / *Read Data* setată pentru permisiunea R
Read Attributes setată pentru permisiunea R + X
Read Extended Attributes setată pentru permisiunea R
Create Files / *Write Data* setată pentru permisiunea W
Create Folders / *Append Data* setată pentru permisiunea W
Write Attributes setată pentru permisiunea W
Write Extended Attributes setată pentru permisiunea W
Delete Subfolders and Files setată pentru permisiunea D
Delete setată pentru permisiunea D
Read Permissions setată pentru permisiunea R + W + X
Change Permissions setată pentru permisiunea P
Take Ownership setată pentru permisiunea O

Setarea acestor permisiuni poate fi făcută şi din interfaţa grafică în secţiunea *Security* (*Advanced...*) din fereastra de proprietăţi (*Properties*) ale unui fişier.

http://www.ntfs.com/ntfs-mft.htm
Andrew S. Tanenbaum, Sisteme de operare moderne, Byblos, 2004
Razvan Rughinis, Razvan Deaconescu, George Milescu, Mircea Bardac, Utilizarea Sistemelor de Operare, Printech 2008

Capitolul 2

Funcţii API Win32 pentru sistemul de fişiere NTFS

Acronimul API este o abreviere a Application Programming Interface. Aşadar Windows API (sau Win32 API) este un set de funcţii oferite de sistemul de operare Windows pentru manipularea resurselor calculatorului. Orice sistem de operare oferă (sau exportă) un set de astfel de functii, pentrua fi utilizate de programatori în dezvoltarea de aplicaţii specifice acelui
sistem de operare. Denumirea de Win32 API mai este folosită uneori pentru a marca diferenta dintre sistemele de operare Windows pe 16 biti (Windows 3.X) şi sistemele de operare Windows pe 32 de biti (Windows 9X, Windows NT, Windows 2000, Windows XP). 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.

Datorită diferenţelor dintre programele DOS şi Windows, Win32 API va utiliza o serie de tipuri de date noi. Câteva din acestea sunt:

- **HANDLE** – este un tip generic (**identificator**) , utilizat pentru manipularea obiectelor folosite în program (fişiere, ferestre, etc.). Pentru a manipula un astfel de obiect, este necesară întâi obţinerea unui asemenea HANDLE, în urma apelării unei funcţii care îl returnează. Toate operaţiile ulterioare cu obiectul respectiv se vor face prin intermediul mărimii HANDLE şi nu prin intermediul numelui obiectului respectiv. Uzual, dacă execuţia funcţiei care returnează identificatorul eşuează, acesta va avea valoarea NULL;
- **HWND** - este definit ca typedef HANDLE HWND; şi este folosit pentru manipularea ferestrelor;
- **DWORD** – (**D**ouble **Word**) este un întreg fără semn pe 32 de biţi. Un DWORD este compus din două mărimi WORD. Uzual, marea majoritate a compilatoarelor permit extragerea celor două componente WORD prin funcţii de tipul high() şi low();
- **LPVOID** – (**L**ong **P**ointer **Void**) este definit ca typedef void* LPVOID, fiind deci un pointer void reprezentat pe 32 de biţi;
- **LPCSTR** – (**L**ong **P**ointer **C**onstant **Str**ing) – reprezintă un pointer pe 32 de biţi spre un şir de caractere constant. Este utilizat de obicei atunci când şirul este utilizat ca parametru al unei funcţii şi funcţia nu îl modifică;
- **LPCTSTR** – (**L**ong **P**ointer **C**onstant **T**o **Str**ing) – reprezintă un pointer pe 32 de biţi spre un şir de caractere constant **Unicode**; Unicode este un cod de caractere pe 16 biţi, capabil să reprezinte caracterele tuturor limbilor. Este utilizat de platformele Windows NT (2000, XP). Windows 95, 98 şi Milenium nu îl utilizează. Pentru a defini un pointer similar, dar spre un şir ASCII, vom declara tipul LPCSTR;
- **LPTSTR** – (**L**ong **P**ointer **T**o **Str**ing) – reprezintă un pointer pe 32 de biţi spre un şir de caractere în format **Unicode**;
- **LPSTR** – (**L**ong **P**ointer **Str**ing) – reprezintă un pointer pe 32 de biţi spre un şir de caractere în format ASCII;
- **WPARAM** şi **LPARAM** – cuvinte cu lungimea de 32 de biţi, utilizate în general pentru a transmite parametri asociaţi unui mesaj Windows;
- **LRESULT** – o valoare pe 32 de biţi, returnată de o funcţie;

Exemple de functii:

Funcţia *CreateFile*

Funcţia este folosită pentru a crea un fişier sau pentru a deschide un fişier existent. Sintaxa funcţiei este următoarea:

HANDLE CreateFile(
LPCTSTR lpFileName**, DWORD** dwDesiredAccess,
DWORD dwShareMode,
LPSECURITY_ATTRIBUTES lpSecurityAttributes,
DWORD dwCreationDisposition,
DWORD dwFlagsAndAttributes,
HANDLE hTemplateFile);

Funcţia *DeleteFile*

Funcţia şterge un fişier existent şi are următoarea sintaxă:
BOOL DeleteFile(
LPCTSTR lpFileName); // numele fişierului
Returnează o valoare nenulă în caz de succes şi 0 altfel.

Funcţia *CloseHandle*

Funcţia închide un handler de fişier obţinut anterior cu funcţia *CreateFile*.
BOOL CloseHandle(
HANDLE hObject**);** //handler catre obiect
Returnează o valoare nenulă în caz de succes, 0 altfel.

Funcţia *ReadFile*

Funcţia citeşte date dintr-un fişier, începând de la poziţia indicată de către pointerul fişierului. După ce operaţia de citire a fost finalizată, pointerul de fişier este ajustat cu numărul de octeţi citiţi efectiv, mai puţin în cazul în care handler-ul de fişier este creat cu atributul FILE_FLAG_OVERLAPPED. Dacă handler-ul de fişier este creat pentru intrare-ieşire suprapusă (I/O), aplicaţia trebuie să ajusteze poziţia pointerului de fişier după operaţia de citire.

BOOL ReadFile(
HANDLE hFile, // handler către fisier
LPVOID lpBuffer, // buffer de date
DWORD nNumberOfBytesToRead, // nr octeti de citit
LPDWORD lpNumberOfBytesRead, // nr octeti cititi

Funcţia *WriteFile*

Această funcţie scrie date într-un fişier şi este destinată atât pentru operaţii sincrone cât şi pentru operaţii asincrone. Funcţia începe să scrie datele în fişier la poziţia indicată de pointerul de fişier. După ce operaţia de scriere a fost terminată, pointerul de fişier este ajustat cu numărul de octeţi scrişi efectiv, cu excepţia cazului în care fişierul este deschis cu FILE_FLAG_OVERLAPPED.

BOOL WriteFile(
HANDLE hFile,
LPCVOID lpBuffer,
DWORD nNumberOfBytesToWrite,
LPDWORD lpNumberOfBytesWritten,
LPOVERLAPPED lpOverlapped);

Dacă funcţia se termină cu succes, valoarea returnată va fi nenulă.
Dacă funcţia eşuează, valoarea returnată este 0.

Funcţia *SetFilePointer*

Funcţia **SetFilePointer** deplasează pointerul unui fişier deschis.
DWORD SetFilePointer(
HANDLE hFile,
LONG lDistanceToMove,
PLONG lpDistanceToMoveHigh,
DWORD dwMoveMethod);

Funcţia *CreateDirectory*

Această funcţie creează un nou director. Dacă sistemul de fişiere existent suportă opţiuni de securitate pentru directoare şi fişiere, funcţia va aplica descriptorul de securitate specificat pentru noul director.

BOOL CreateDirectory(
LPCTSTR *lpPathName*,
LPSECURITY_ATTRIBUTES *lpSecurityAttributes*);
Dacă funcţia se termină cu succes, valoarea returnată este nenulă, altfel este 0.

http://en.wikipedia.org/wiki/Windows_API
http://msdn2.microsoft.com/en-us/library/default.aspx

Capitolul 3

Apelurile pentru sistemul de fişiere in Linux

3.1. Sistemul de fişiere Linux – considerente generale

Sistemul de fişiere din Unix este caracterizat prin:
• o structură ierarhică
• tratare consistentă a fişierelor de date
• protejarea fişierelor

În cazul sistemului de fişiere din Linux se urmăresc aceleaşi principii de bază ca şi în cazul Unix, adică se încearcă ca intrarea şi ieşirea pentru divese dispozitive ca discuri, terminale, imprimate să se realizeze la fel ca şi lucrul cu fişiere obişnuite.

Fiecare fişier este reprezentat de o structură numită *inode*. Fiecare inode conţine descrirea fişierului: tipul fişierului, drepturile de acces, proprietarul, informaţii referitoare la dată, dimensiune, referinţe către blocurile de date. Adresele blocurilor de date alocate fişierului sunt de asemenea stocate în cadrul *inode*-ului. Când un utilizator solicită o operaţie de intrare/ieşiere asupra fişierului, *kernel*-ul sistemului de operare converteşte indexul curent într-un număr de bloc şi utilizează acest număr ca şi index în tabela adreselor de bloc şi scrie sau citşte un bloc.

3.2. Accesul la fişiere

În Linux fişierele sunt acesate prin intermediul unor descriptori de fişiere. Fiecare proces putând avea deschise un anumit număr maxim de fişiere la acelaşi moment de timp, implicit acest număr este 256. Prin convenţie descriptorii 0, 1 şi 2 sunt alocaţi de către sistem la pornire:
• descriptorul 0 este utilizat ca şi intrare standard
• descritorul 1 este ieşire standard
• descriptorul 2 este ieşirea standard de eroare.

Fiecare descriptor de fişiere alocat este asociat cu un descriptor de fişiere deschi s. Un descriptor de fişiere conţine informaţii referitoare la un anumit fişier, un index care precizează unde va avea loc următorul acces în fişier, modul de acces la fişier, şi alte informaţii legate de acesta. Este posibil ca mai mulţi descriptori de fişiere, chiar şi aparţinând unor procese separate să indice în acelaşi timp spre acelaşi descripor de fişier deschis, adică informaţia dintr-o structură de decriptor deschis va fi utilizată simultan de mai mulţi descriptori de fişiere. Există cinci apeluri care generează descriptori de fişiere: *creat, open, fcntl, dup* şi *pipe.*

Apelul sistem *open*:
```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

Returnează*: descriptorul de fisier sau -1 in caz de eroare.*

Apelul sistem *creat:*
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
*int creat(const char *path, mode_t mod);*
Returnează*: descriptorul de fişier sau -1 in caz de eroare.*

Apelul sistem *read:*
Pentru a citi un număr de octeţi dintr-un fişier, de la poziţia curentă, se foloseşte apelul **read**. Sintaxa este:
#include <unistd.h>
*ssize_t read(int fd, void *buf, size_t noct);*
Returnează: *numarul de octeţi citiţi efectiv, 0 la EOF, -1 in caz de eroare.*

Apelul sistem *write*
Pentru a scrie un număr de octeţi într-un fişier, de la poziţia curentă, se foloseste apelul *write*. Sintaxa este:
#include <unistd.h>
*ssize_t write(int fd, const void *buf, size_t noct);*
Returnează: *numărul de octeţi scrişi şi -1 în caz de eroare.*

Apelul sistem *opendir*
#include <sys/types.h>
#include <dirent.h>
*DIR *opendir(const char *pathname);*
Returnează *pointer dacă este OK, NULL în caz de eroare.*
*struct dirent *readdir(DIR *dp);*
Returneaza pointer daca este OK, NULL in caz de eroare.

Apelul sistem *link*
Pentru a adăuga o nouă legătura la un director se foloseste apelul:
#include <unistd.h>
*int link(const char *oldpath, const char newpath);*
Returnează: *0 in caz de reuşită si -1 in caz contrar*.

Apelul sistem *unlink*
Pentru a şterge o legătură (cale) dintr-un director se foloseşte apelul:
#include <unistd.h>
*int unlink(const char *path);*
Returneaza: *0 în caz de reuşită şi -1 în caz contrar*.

4.3. Atributele fişierelor

Pentru o gestionare eficientă a fişierelor este necesară nu numai cunoaşterea unui n umăr cât mai mare de date referitoare la acestea dar şi posibilitatea modificării acestor date. În paragraful următor vor fi prezentate o serie de funcţii folosite pentru modificarea atributelor fişierelor. Una dintre caracteristicile esenţiale ale sistemului de operare Linux este securitatea. Implementarea acesteia la nivel de structură de fisiere se face prin precizarea pentru fiecare fişier în parte a drepturilor de citire, scriere şi execuţie pentru proprietarul fişierului, pentru grupul proprietarului şi pentru toate celelalte clase de utilizatori. Aceste drepturi pot fi modificate cu ajutorul apelurilor sistem *chmod* şi *fchmod*:

#include <sys/types.h>
#include <sys/stat.h>
*int chmod(const char *path, mode_t mode);*
int fchmod(int fildes, mode_t mode);
Returnează: *0 in caz de succes şi -1 în caz contrar.*

www.linuxiso.org
www.whatis.com

Capitolul 4

Ext 2 si arhitectura NFS la Linux

In LINUX primul sistem de fisiere era bazat pe sistemul de fisiere MINIX dar pe masura ce sistemul de operare LINUX s-a maturizat a aparut Extended Filesystem care a adus imbunatiri sistemului de fisiere dar din punct de vedere al performantei era nesatisfacator.Ext 2 sau Second Exended Filesystem a aparut in 1994 si odata cu noile modificari acesta a devenit cel mai folosit sistem de fisiere pe LINUX:

O parte din modificarile aduse de Ext 2 LINUX sunt:

- Atunci cand se creaza un sistem de fisiere , administratorulpoate sa aleaga marimea optima a blocului (intre 1024 sau 4096 bytes) depinzand de marimea fisierelor care se asteapta a fi stocate.In acest sens este de preferat sa alegi un bloc de marime 1024 atunci cand majoritatea fisierelor au marime mai mici de 1024 bytes pentru ca asta duce la mai putine fragmentari interne.Pe de alta parte se aleg blocuri mai mari de cateva mii de bytes pentru fisierele care depasesc 1024 de bytes pentru ca in acest caz vor avea loc mai putine transferuri intre discuri;

- De altfel administratorul mai are posibilitatea de a alege cate inoduri sa permita unei partitii de o anumita marime depinzand de dumarul de fisiere care se vor stoca acolo.Acest lucru va duce la marirea eficientei spatiului liber de pe disc;

- Sistemul de fisiere partitioneaza blocurile de disc in grupuri si fiecare grup contine blocuri de date si inoduri stocate in track-uri adiacente.Cu acest tip de structura fisierele stocate intr-un singur grup de blocuri pot fi accesate cu un timp de cautare mai mic;

- In sistemul de fisiere Ext 2 sunt suportate symbolic link-urile rapide.Calea unui symbolic link se gaseste in inod daca aceasta are mai putin de 60 de bytes;

- Exista un support pentru verificarea starii sistemului de fisiere in momentul boot-arii.Verificarile sunt facute de /sbin/e2fsck;

- De altfel Ext 2 are un suport pentru fisierele imuabile (care nu se schimba) si pentru fisierele de tip append-only (unde se pot adauga date doar la sfarsitul fisierului).Aceste protectii nu pot fi dezasamblate nici de catre superuser;

Alte aparitii sau imbunatatiri la sistemul de fisiere Ext 2 :

-Fragmentarea blocurilor

Dat fiind faptul ca de obicei administratorii aleg blocuri de marime mare pentru a accesa discurile recente rezultatul este ca fisierele mici care sunt stocate in blocuri mari fac o risipa de spatiu.Aceasta problema se poate rezolva daca li se permite fisierelor sa fie stocate in diferite fragmente din acelasi bloc.

-Lista de control al accesului

In loc sa clasificam userii unui fisier in cele 3 grupuri(proprietar,grup si altii) – lista de control al accesului(ACL) este asociata fiecarui fisier pentru a putea specifica drepturile pentru oricare dintre useri.

- Fiserele comprimate si fisierele criptate

Aceasta noua optiune care trebuie specificata in momentul crearii unui fisier le va permite utilizatorilor sa stocheze si sau sa stocheze fisiere comprimate sau criptate pe disc.

-Stergerea logica

Optunea de undelete le va permite userilor sa recupereze cu usurinta continutul fiserului anterior sters.

Sistemele de fisiere al Ext 2 sunt create de catre utilitarul /sbin/mke2fs.

Acest program face urmatoarele actiuni:
-initializeaza superblocul si grupul descriptor;
-verifica daca partitia contine blocuri defective iar daca gaseste acest tip de blocuri creaza o lista cu acestea;
-pentru fiecare grup de blocuri ,se rezerva toate blocurile de disc de care este nevoie pentru stocarea superblocului.;
-initializeaza tabelul de inoduri ale fiecarui grup de blocuri
-creaza directorul /root;
-creaza directorul lost+found care este folosit de catre /sbin/e2fsck pentru a tine o legatura cu blocurile pierdute sau defective gasite;
-grupeaza blocurile defective (daca sunt) in directorul lost+found

Pentru a fi cat mai eficient ,o partitie a sistemului de fisiere Ext 2 care stocheaza majoritatea informatiei pe disc o si copiaza pe RAM atunci cand sistemul de fisiere este montat.

www.linuxiso.org

Capitolul 5

Compresia si criptarea fisierelor NTFS

5.1 Compresia fisierelor NTFS

Compresia se foloseste pentru a reduce dimensiunile fisierului,fiind importanta pentru a economisi timp,spatiu ,dar si pentru datele redundante.Aceasta se mai foloseste si pentru a elibera un anumit spatiu pe disk,atat pentru fisiere text,imagini,video,audio etc.
Exemple de aplicatii pentru compresia datelor:

- Fisiere:GZIP,BZIP,BOA
- Arhive:PKZIP
- Sistem de fisiere:NTFS
- Imagini:GIF,JPEG
- Sunet:MP3
- Video:MPEG,HDTV
- Baza de date:GOOGLE
- Comunicatii:ITU-T T4 Group 3FAX

Partitiile sistemului de fisiere NTFS suporta compresia fisierelor intr-un fisier de baza individual.Algoritmul de compresie al fisierelor de catre sistemul de fisiere NTFS se numeste compresia Lempel –Ziv.Acesta este un algoritm de compresie « fara pierdere »,prin care se intelege ca datele nu se pierd atunci cand are loc compresia fisierului ,in opozitie cu algoritmii de compresie « cu pierdere » ca in cazul JPEG ,cand de fiecare data cand se
face compresia datelor au loc pierderi.

Prin compresia datelor se reduce marimea fisierului minimizand datele. Intr-un fisier text,datele redundante pot fi adesea caractere intamplatoare,de exemplu caracterul spatiu,sau vocale precum literele e sau a pot fi de asemenea si caractere de tip string. Fiecare algoritm de compresie a datelor minimizeaza date redundante intr-un anumit mod.De exemplu, « Algoritmul de codare Huffman » atribuie un cod caracterelor dintr-un fisier si este bazat aparitia acelor caractere.Un alt algoritm de compresie,numit codarea RUN-LENGTH
imparte in doua categorii caracterele care se repeta :prima parte specifica timpul de repetare al caracterului(perioada),iar cea de-a doua parte indica ce caracter s-a repetat.Un alt algoritm de compresie,cunoscut sub numele de algoritmul LEMPEL-ZIV,converteste variabilele de tip string in coduri fixe care ocupa mai putin spatiu decat stringul original.

http://msdn2.microsoft.com/en-us/library/aa364219.aspx

5.2 Compresia fisierelor sistemului de fisiere NTFS

In sistemul de fisiere NTFS, compresia are loc in mod transparent.Acest lucru implica folosirea fara schimb de cereri pentru modificarile aplicatiilor.Compresia bitilor fisierului nu este accesibil pentru aplicatii; se pot vedea doar datele necomprimate.Prin urmare,aplicatiile care deschid fisierele comprimate pot opera in sistemul NTFS cu datele care nu au fost
comprimate.Totusi,aceste fisiere nu pot fi copiate de un alt sistem de fisiere.

Daca dorim sa facem compresia unor date mai mari de 30Gb,aceasta nu se
va finaliza cu succes.
http://msdn2.microsoft.com/en-us/library/aa364219.aspx

Atributele compresiei

In sistemul de partitii ale fisierelor NTFS ,fiecare fisier si director au cate un atribut de compresie.Celelalte sisteme de fisiere pot implementa cate un atribut de compresie pentru fisiere individuale. Se poate determina daca un sistem de fisiere suporta un atribut de compresie pentru fisier si director apeland functia GetVolumeInformation si examinand fanionul (« bit flag ») FS_FILE_COMPRESSION.

Utilizand GetFileAttributes si GetFileAttributesEx se poate determina atributul de compresie al fisierului sau directorului. Daca atributul de compresie al unui fisier este FILE_ATTRIBUTE_COMPRESSED,atunci toate datele din fisier sunt comprimate.Daca atributul de compresie nu are nicio valoare, atunci nicio data din fisier nu a fost comprimata.

Atributul de compresie este un indicator simplu BOOLEAN al starii comprimarii.

Atributul de compresie al unui director de fisiere prezinta attribute de compresie nesigure pentru fisierele si subdirectoarele nou create.Cand se apeleaza **CreateFile** sau **CreateDirectory** pentru a creea un fisier nou sau un director,fisierul cel nou sau directorul mostenesc atributul compresiei din directorul parinte.

http://msdn2.microsoft.com/en-us/library/aa363849(VS.85).aspx

5.3 Starea compresiei

Fiecare fisier sau director dintr-o partitie care suporta compresie au o anumita stare.
Pe cand atributul compresiei unui fisier sau director indica pur si simplu daca un fisier sau director a fost comprimat sau nu,starea compresiei specifica formatul datelor de comprimat.
Folosind codul de control FSCTL GET COMPRESSION se poate determina starea compresiei unui fisier sau director.Starea compresiei este o valoare pe 16 biti.Aceasta operatie seteaza atributul compresiei fisierului sau directorului. COMPRESSION_FORMAT_NONE arata ca fisierul nu este comprimat,iar valoarea COMPRESSION_FORMAT_DEFAULT arata ca fisierul a fost comprimat .

Folosind codul de control FSCTL SET COMPRESSION se seteaza starea compresiei a fisierului sau directorului. Prin aceasta operatie se seteaza si atributul de compresie al fisierului sau directorului. Setand starea compresiei la o valoare diferita de zero,se face compresia fisierului,daca starea este setata pe zero are loc decompresia fisierului.Acestea sunt operatii sincrone.Fisierul este comprimat sau extins imediat ce se seteaza aceste stari.
Setand starea de compresie a directorului aceasta nu implica imediat compresia sau extinderea imediata(ca in cazul fisierului).

http://msdn2.microsoft.com/en-us/library/aa363849(VS.85).aspx

Obtinerea dimensiunii unui fisier comprimat

Folosind functia **GetCompressedFileSize** se obtine dimensiunea unui fisier comprimat.Daca un fisier a fost comrpimat dimensiunea sa va fi mai mica decat atunci cand era necomprimat.Folosind functia **GetFileSize** se poate determina dimensiunea fisierului necomprimat.

5.4 Efecte ale compresiei prin mutarea sau copierea fisierelor

Mutand sau copiind fisiere intre partitii se schimba starea compresiei.Starea compresiei unui fisier NTFS se controloleaza prin atributul sau. De exemplu,daca se muta un fisier necomprimat intr-un folder comprimat,fisierul ramane necomprimat dupa mutare.

Daca se copiaza un fisier comprimat intr-un folder necomprimat,fisierul devine automat necomprimat ,deci capata aceeasi stare ca si folderul in care este copiat.

http://msdn2.microsoft.com/en-us/library/aa364223(VS.85).aspx

In cazul compresiei pot aparea de asemenea erori.Un exemplu de eroare poate fi „Sistemul de fisiere nu poate suporta compresia.”Cauza acestei erori este datorita dimensiunii partitiei.Compresia fisierelor NTFS nu este suportata pentru o locatie mai mare de 4Kb.Pentru a rezolva aceasta problema,se poate face reformatarea partitiei NTFS utilizand clustere de dimensiuni mai mici sau egale cu 4Kb.

In concluzie,exista doua cai pentru a masura performantele compresiei de date NTFS : dimensiunea si viteza.Se poate spune cat de bine lucreaza compresia comparand dimensiunile fisierelor sau datelor necomprimate cu ale celor comprimate.

5.5 Criptarea fisierului

Sistemul de fisiere criptate sau EFS(Encrypting File System) ,a fost introdus in NTFS 5.0 fiind in plus un nivel de securitate pentru fisier si director.El ofera o protectie criptografica fisierelor individuale din partitiile sistemului de fisiere NTFS folosind o cheie publica pentru system. Controlul accesului la obiectele fisierului sau directorului oferit de modelul de securitate al WINDOWS-ului este suficient pentru a proteja informatia “sensibila” de accesul neautorizat.Totusi,daca un laptop care contine date importante este pierdut sau furat , protectia datelor din punct de vedere al securitatii este compromisa.Criptand fisierele marim securitatea. Pentru a determina daca un sistem de fisiere suporta criptarea fisierului sau directorului ,putem apela functia **GetVolumeInformation** si examinand fanionul FS_FILE_ENCRYPTION.Urmatorii itemi nu pot fi criptati:

- Fisierele comprimate

Sistemul de fisiere

- Sistemul de directoare
- Tranzactiile
- Registrele de baza

Putine fisiere pot fi criptate.
TxF nu poate suporta o mare parte din operatiile de criptare cu fisierele EFS .Singura operatie pe care o suporta TxF este citirea operatiilor ,ca de exemplu **ReadEncryptedFileRaw**.

http://msdn2.microsoft.com/en-us/library/aa364223(VS.85).aspx

5.6 Lucrul cu fisierele si directoarele criptate

Un programator sau utilizator poate semnala un fisier ca fiind criptat . Un fisier gasit criptat a fost criptat de sistemul de fisiere NTFS utilizand driverul curent de criptare.Daca mai tarziu fisierul a fost gasit necriptat,acesta a fost decriptat si plasat intr-un fisier text nesecurizat. Pentru a cripta un nou fisier se foloseste functia **CreateFile** impreuna cu flagul FILE_ATTRIBUTE_ENCRYPTED.Pentru a cripta un fisier existent se foloseste functia **EncryptFile** cu ajutorul careia se cripteaza toate datele.Daca fisierul a fost deja criptat ,**EncryptFile** va intoarce o valoare diferita de zero , care indica succesul operatiei.Daca fisierul a fost deja comprimat , EncryptFile face decomprimarea fisierului inainte de a-l cripta.

Pentru a decripta un fisier criptat se foloseste functia **DecryptFile**.Daca fisierului nu era criptat ,DecryptFile nu face nimic,dar returneaza o valoare diferita de zero.

Functia **EncryptionDisable** activeaza sau dezactiveaza criptarea directorului indicat si a fisierelor din director.Aceasta nu afecteaza subdirectoarele criptate anterior in directorul indicat.

http://www.microsoft.com/technet/security/guidance/cryptographyetc/efs.mspx

Caracteristici importante ale EFS:

- Criptarea EFS poate avea loc la nivel de fisier al sistemului,nu la nivel de aplicatie;criptarea si decriptarea fisierelor sunt transparente pentru utilizator si pentru aplicatie;daca un folder este criptat , fiecare fisier din acesta care este creat sau mutat intrun alt folder poate fi criptat;daca un utilizator doreste sa acceseze un fisier criptat va introduce o parola;daca parola nu este corecta,li se va afisa un mesaj de roare care contine „ accesul nepermis”.
- Parolele(cheile) pot fi arhivate si tinute intr-un loc sigur pentru a nu fi descoperite.
- Aceste chei sunt protejate de parola utilizatorului;orice utilizator care afla parola poate avea acces la fisierele criptate si sa le decripteze.
- Inainte de criptarea fisierelor trebuie sa ne asiguram ca partitia apartine unui fisier NTFS,fisierul nu a fost comprimat,s-a scris accesul catre fisier.

5.7 Beneficiile securitatii NTFS

NTFS asigura securitatea fisierelor si folderelor cu ajutorul ACLurilor (LISTE DE CONTROL AL ACCESULUI).ACL-urile sunt descriptori de securitate atasati fisierelor si directoarelor din sistemul de fisiere NTFS.Niciun fisier sau director nu poate avea mai multe nivele de permisiune al accesului.Inainte ca unui proces sa i se permita accesul catre un fisier,sistemul de securitate verifica daca procesul este autorizat sa faca acest lucru.

NTFS suporta directoare active.Directoarele active permit sistemului sa acceseze un domeniu,si utilizand autorizarea din artea serverului de baza stabileste permisiunile fisierelor.Sistemul de fisiere FAT nu implementeaza securitatea,si toti utilizatorii au drepturi egale de acces la fisierele si directoarele din sistem.