Zarządzanie plikami we współczesnych systemach operacyjnych

Wojciech Kazana
Mariusz Sidorowski
Piotr Truszkowski

grupa 2

Wprowadzenie

Co to jest system plików, czym jest plik.

Wiek XX wprowadził świat w erę techniki, w erę komputerów. Każdy rok stanowi nieustający rozwój w tej dziedzinie. Gordon Moore w 1965 roku wysunął tezę, żę ekonomicznie optymalna liczba tranzystorów w układzie scalonym podwaja się co 18 miesięcy. Co ciekawe teza znakomicie się sprawdza, nie tylko wobec mocy obliczeniowej komputerów, także odnosi się do przepustowości sieci, rozmiarów pamięci RAM czy pojemności dysków twardych. Widząc naocznie jak prężnie przemysł komputerowy się rozwija nie ma wątpliwości, że równie prężnie musi się rozwijać oprogramowanie wspierające coraz to nowsze, lepsze, szybsze, pojemniejsze komputery.

Ludzie konstruując komputery, musieli zdawać sobie sprawę, że będą musieli przygotować pewną abstrakcyjny byt do przechowywania danych. Początkowo nie wymagało to wyrafinowanych metod poczynając od taśm magnetycznych kończąc na dzisiejszych dyskach twardych, płytach CD, pamięciach flash. Wszystkie te materiały są przygotowane na trwały zapis informacji. O to aby zapis czy odczyt był możliwie najefektywniejszy mają dbać właśnie systemy plików. To system plików wraz z pojęciem pliku, katalogu przedstawia pewną użytkownikowi pewną abstrakcję. Dla niego plik „miś.avi” to jest jedynie bezwartościowy ciąg zer i jedynek. To wspaniały film! I tak go postrzega, jako pewien atomowy, obiekt abstrakcyjny. Użytkownik dostaje pewien interfejs poleceń dzięki któremu różne pliki postrzega właśnie jako dokumenty tekstowe, czy filmy. Może plik otworzyć, zapisać, utworzyć katalog, itd. To jest właśnie główne zadanie systemy plików oraz systemów operacyjnych – wprowadzać pewną abstrakcję danych, która umożliwi użytkownikowi intuicyjne poruszanie się po systemie. Nie należy jednak zapominać o tym co sprawia, że owe systemy plików są przyjazne użytkownikowi, że operacje na plikach, katalogach są efektywne. Czy co też istotne nasze

dane są bezpieczne, odporne na nieoczekiwaną awarię. Czy nasze dane na dyskach są dobrze strzeżone, nie zawsze chcemy żeby ktoś oglądał nasze dane. To właśnie o tym traktować będą następne rozdziały.

Historia

Wbrew pozorom historia systemów plików sięga daleko dalej niż pierwsze skonstruowane komputery. Warto sobie uświadomić, że system plików to tak naprawdę sposób przechowywania informacji, taki aby był efektywny. Gromadzenie danych jest w ludzkości głęboko zakorzenione, już starożytni szukali sposobów aby w uporządkowany sposób przechowywać swoją wiedzę. Za przekład może służyć biblioteka aleksandryjska z III w. p.n.e. W czasie największej świetności zgromadzono tam około 490tys zwojów. Przez kolejne wieki szukano coraz to lepszych sposobów jak gromadzić przeróżne materiały. I wystarczy odwiedzić współczesną bibliotekę aby zobaczyć jak znaleźć poszukiwaną książkę. Tak samo w hipermarkecie – wszystkie produkty są ułożone według pewnego porządku. Czy choćby wspomniana ksiązka – w niej również znajdują się różne informacje, np. encyklopedię cechuje ścisły porządek alfabetyczny na hasłach. Na kartach historii jako ważny zwrot przywoływany jest wynalazek Jana Gutenberga – druk – to on wprowadził Europę w renesans. I tak aż do XX wieku to na papierze gromadzono całą wiedzę. Rozwój techniki i nauki pokazał nowy sposób magazynowania wiedzy – na taśmach magnetycznych, czy potem na dyskach twardych. Rozwój informatyki pozwalał na tworzenie oprogramowania nadzorującego nad zapisem, odczytem danych z tychże nośników. Z tą chwilą rodziły się się pierwsze systemy plików. Zapewniały, wcześniej wspomniany interfejs, operacji na nośniku. Nieustający rozwój informatyki dał początek systemom operacyjnym a te – coraz lepszym systemom plików. I to właśnie systemy operacyjne Microsoft Windows jak i Linux „wychowały" swoje systemy plików. Windows: FAT12, FAT16, FAT32, NTFS. Linux: ext, ext2, ext3, reiserFS, reiser4.
Dzisiejszy świat to z pewnością epoka informacji, wobec tego sposób w jaki przechowujemy dane jest istotny. Dlatego też ciągle pojawiają się coraz to lepsze rozwiązania zarówno starych jak i nowych problemów. O każdym z wymienionych systemów plików w kolejnych rozdziałach.

Współczesne systemy plików

Dla współczesnych plików, nie tylko istotne jest jak najefektywniej wykorzystać istniejące struktury danych. Ważna jest także bezpieczeństwo danych, ich odzyskiwanie po awarii, czy też możliwość szyfrowania danych. Stąd nowoczesne systemy jak NTFS czy reiserFS(reiser4) udostępniają możliwość księgowania, szyfrowania czy kompresji.

Księgowanie(ang. journalling) to w informatyce termin związany z konstrukcją baz danych oraz systemów plików. Przy użyciu księgowania dane nie są od razu zapisywane na dysk, tylko zapisywane w swoistym dzienniku/kronice. Dzięki takiemu mechanizmowi działania zmniejsza się prawdopodobieństwo utraty danych: jeśli utrata zasilania nastąpiła w trakcie zapisu - zapis zostanie dokończony po przywróceniu zasilania, jeśli przed - stracimy tylko ostatnio naniesione poprawki, a oryginalny plik pozostanie nietknięty.

Szyfrowanie zapewnia, że nikt niepowołany nie będzie przeglądał naszych danych – ani inny użytkownik ani nawet administrator.

Windows

Microsoft jest na rynku oprogramowania prawdziwym gigantem. Windows – sztandarowy produkt tej firmy – do systemu operacyjnego dostarcza także swoje systemy plików. Poczynając od FAT12, FAT16, FAT32 aż do nowoczesnego NTFS. Niewątpliwie popularność tego systemu promuje systemy plików projektowane przez giganta z Redmond.

FAT - File Allocation Table

Wprowadzenie

Można powiedzieć, że każdy system operacyjny dysponuje "swoim" systemem plików. Jest to naturalne podejście gdyż wtedy to może być optymalnie wykorzystane struktury danych użyte do implementacji samego systemu operacyjnego jak i systemu plików, uzupełniają się wzajemnie, system operacyjny może wiedzieć wtedy lepiej na jakie operacje może sobie częściej pozwalać a na jakie nie. Microsoft w latach 80 zajmuje się MS-DOS oraz początkami systemu Windows(1985 – wersja 1.0). Wtedy to już Microsoft dysponował własnym systemem plików – FAT.

Organizacja

Obecnie istnieją trzy rodzaje FAT: FAT12, FAT16, FAT32. FAT był projektowany jako prosty system plików, początkowo z myślą o małych dyskach i prostych strukturach katalogów. Podstawowa różnica między kolejnymi wersjami to liczba bitów, na których koduje się numery jednostek alokacji plików zwanych klastrami.

Partycja FAT, poza początkowymi sektorami, jest podzielona na klastry (jednostki alokacji pliku). Każdy klaster składa się z jednego lub kilku sektorów(sektor jest zawsze w całości odczytywany i zapisywany, przeważnie wielkości 512bajtów), klastry są numerowane. System operacyjny na podstawie numeru klastra oblicza numer logiczny sektora (numer sektora od początku partycji) a na tej podstawie numer ścieżki, głowicy i sektora na ścieżce (dawniej fizyczne położenie na dysku), identyfikując jednoznacznie sektor i dokonując odczytu lub zapisu wybranego sektora.

Klaster w całości jest przydzielony jednemu plikowi. Plik w katalogu zawiera numeru pierwszego klastra pliku, gdzie znajdują się dalsze części pliku opisuje wpis w FAT. W tablicy FAT pod numerem odpowiadającym numerowi pierwszej części pliku jest umieszczony numer kolejnego klastra przydzielonego plikowi lub liczba z zakresu FFF8h-FFFFh jeśli to jest ostatni klaster pliku. Jeżeli dany klaster jest wolny, to w FAT odpowiada mu wpis 0000h, a FFF7h oznacza uszkodzony klaster, dokładniej w tabelce poniżej:

Klaster	FAT12	FAT16	FAT32
Wolny	0x000	0x0000	0x?0000000
Zarezerwowany	0x001	0x0001	0x?0000001
Używany	0x002-0xfef	0x0002-0xffef	0x?0000002-0x?fffffef
Zarezerwowane wartości	0xff0-0xff6	0xfff0-0xfff6	0x?ffffff0-0x?ffffff6
Uszkodzony	0xff7	0xfff7	0x?ffffff7
Ostatni	0xff8-0xfff	0xfff8-0xffff	0x?ffffff8-0x?fffffff

Wielkość klastra zależy od maksymalnego numeru klastra zależnego od wersji FAT, czyli 12, 16 albo 32, jednak ta nie określa bezpośrednio ilości dostępnych klastrów. Np. w FAT12 będzie ich mniej niż 4096 (2^{12}), ponieważ niektóre z nich mają znaczenie wyłącznie systemowe i nie są dostępne dla użytkownika. Ilość dostępnych klastrów jest jednocześnie maksymalną liczbą możliwych do zapisania plików na partycji. Jeżeli w FAT12 użyjemy dla dysku o pojemności 20MB – dysk ten ma 40960(= 20MB/512B) sektorów, więc klaster musi zawierać 10 sektorów, co odpowiada 5KB.

Szczegóły limitów na konkretnych wersjach FAT:

Limity	FAT12	FAT16	FAT32
rozmiar pliku	32MB	2GB	4GB
ilość plików	4,077	65,517	268,435,437 (4,177,920 praktycznie)
długość nazwy pliku	8+3 lub 255 gdy korzystamy z długich nazw		
rozmiar partycji	32MB	4GB	2TB (124,55GB praktycznie)

Jaki wpływ ma inna liczba bitów na alokację klastrów:

System FAT	Liczba bajtów na klaster w tablicy alokacji	Limit klastrów
FAT12	1.5 (12 bitów)	Poniżej 4087
FAT16	2 (16 bitów)	Pomiędzy 4087 a 65526
FAT32	4 (32 bity)	Pomiędzy 65526 a 268,435,456 (4,177,920 praktycznie)

Organizacja danych na partycjach FAT:

Boot sektor	**Tablica alokacji**	**Duplikat**	**Główny katalog**	**Reszta danych**
Tablica BPB (Bios Parameter Block) oraz program ładujący system operacyjny (boot sector) dla partycji systemowej. Blok BPB zawiera informacje potrzebne do wyliczenia położenia i rozmiaru pozostałych regionów.	Zajmuje kilka sektorów. Zawiera informacje dla systemu operacyjnego na temat klastrów. Każda pozycja w tablicy FAT odpowiada jednemu klastrowi. Na partycji może być kilka kopii tablicy FAT, zazwyczaj dwie.		Główny katalog i jego podkatalogi zawierają nazwę pliku, atrybuty, informacje o czasie utworzenie i modyfikacji, wskaźnik na pierwszy klaster z danymi. (w FAT32 nie istnieje w tej formie)	zajmowany przez podkatalogi i wszystkie pliki; podzielony na logiczne bloki – klastry

Zawartość pól boot sektora:

Początek bajtów	**Długość pola**	**Przykładowa wartość**	**znaczenie**
00	3 bajty	EB 3C 90	Pierwsza instrukcja procedury ładowania
03	8 bajty	MSDOS5.0	Nazwa OEM
0B	25 bajty		Tablica BPB
24	26 bajty		Rozszerzona tablica BPB
3E	448 bajty		Procedura ładowania i informacje o partycji
1FE	2 bajty	0x55AA	Znacznik końca

Historia FAT12, FAT16, FAT32...

Powstały w 80 latach FAT12 był systemem przeznaczonym dla dyskietek. Posiadał wiele ograniczeń, brakowało wsparcia dla katalogów. Adresy klastrów jedynie 12 bitowe przez co rozmiar partycji ograniczony był do około 2MB(wtedy dyskietki miały 360KB).

W 1983 roku wprowadzony zostaje FAT16. Charakterystycznym elementem było nazewnictwo plików składające się z 8-literowej nazwy i 3-literowego rozszerzenia(np. `plik_123.txt`)

Prawdą jest (doświadczalnie udowodnione), że Windows najszybciej działa na dobrze zdefragmentowanym systemie plików... FAT16 (ze względu na duży rozmiar klastra dane wczytują się szybciej). Jednak marnuje się dużo przestrzeni - jest to tzw. slack space. Wynika to stąd, że w jednym klastrze może znajdować się mniej (dużo mniej) niż 32KB np. 1K, a i tak cały będzie zajęty tylko przez jeden fragment/plik. To daje nam zmarnowane 31KB razy np. 50000 klastrów (partycja ok. 1,7GB) i już jest ponad 150MB straconego miejsca.

W roku 1997 Microsoft wprowadza FAT32. Adresowanie klastrów jest, mimo nazwy systemu, 28-bitowe. Zwiększono ilość bitów na adresację klastrów gdyż rozmiar partycji FAT16 stał się zbyt mały. Klaster nie mógł być większy niż 32KiB, co dawało partycje o pojemności jedynie do 2GiB. W FAT32 można zaadresować do 268,435,438(2^{28}) klastrów. Partycje tego typu mogłyby mieć rozmiar wielu TB, jednak ograniczenia napisanego przez Microsoft programu ScanDisk spowodowały wprowadzenie ograniczenia do 4,177,920(2^{24}) klastrów, co dawało partycje o rozmiarze ok. 124,55GB.

Struktury danych w FAT

Każdy folder w systemie plików FAT zawiera 32-bajtowe wpisy dla każdego pliku bądź folderu zawartego w nim.

```
struct katalog {
  nazwa pliku (8 bajtów);
  rozszerzenie (3 bajty);
  atrybuty (1 bajt);
  czas utworzenia (3 bajty);
  data utworzenia (2 bajty);
  ostatni dostęp (2 bajty);
  czas ostatniej modyfikacji (2 bajty);
  data ostatniej modyfikacji (2 bajty);
  numer początkowego klastra (2 bajty);
  rozmiar pliku (4 bajty);
  wielkość liter (1 bajt);
  zastrzeżone (2 bajty);
};
```

Wspomniane charakterystyczne nazewnictwo do 8 liter nazwy pliku oraz 3 litery na rozszerzenie. W nazwie pliku znakami specjalnymi mogą być:

- `00h` – pusty wpis, nie używany wcześniej,
- `E5h` – stary wpis został wymazany,
- `2Eh` – wpis specjalny występujący we wszystkich katalogach oprócz katalogu głównego oznaczający że pierwszy znak jest kropką. Ponieważ znak „.” - wskazuje na bieżący katalog, „..” - wskazuje na katalog macierzysty lub na `NULL`, jeśli katalog macierzysty jest katalogiem głównym.

Dodatkowo nazwy `CON`, `AUX`, `COM1`, `COM2`, `COM3`, `COM4`, `LPT1`, `LPT2`, `LPT3`, `PRN`, `NUL` są zarezerwowanymi nazwami.

Można wymienić następujące atrybuty:

- `0x01` tylko do odczytu,
- `0x02` ukryty,
- `0x04` systemowy,
- `0x08` etykieta dysku,
- `0x10` katalog,
- `0x20` archiwum,
- `0x40` urządzenie,
- `0x80` nieużywane,
- `0x0F` używane dla długich nazw.

Ponadto dla każdego wpisu znajduje się pole zastrzeżone, w FAT32 zawiera 16 starszych bitów numeru początkowego klastra. W innych wersjach pole jest ignorowane.

Rozmiar pliku w przypadku katalogów jest równy 0.

Warto powiedzieć, że w Windows95 wprowadzono nakładkę na system plików nazwaną VFAT(Virtual FAT). Umożliwia ona nadawanie plikom dłuższych nazw oraz korzystanie ze znaków narodowych. Później nakładkę wprowadzona na Windows NT. Jak tego dokonano. Można byłoby zmodyfikować istniejącą strukturę odpowiadającą katalogom, ale wtedy nie byłoby możliwe dostanie się do zasobów dyskowych po uruchomieniu systemu DOS, a tej niedogodności chciano uniknąć. Więc przechowuje się takie same informacje o pliku oraz początkową część nazwy, do dalszej części nazwy użyto drugiej struktury:

```
struct długie_nazwy {
  id fragmentu (1 bajt);
  pierwszych 5 liter nazwy (10 bajtów);
  kolejnych 6 liter nazwy (12 bajtów);
  ostatnie 2 litery nazwy (4 bajty);
  numer początkowego klastra (2 bajty, zawsze 0);
  atrybut (1 bajt);
  suma kontrolna krótkiej nazwy (1 bajt);
  zarezerwowane (1 bajt);
};
```

Pierwsze 6 bitów id fragmentu oznacza numer fragmentu(63 możliwe fragmenty), w praktyce całkowita liczba znaków to 250.
Następnie bit 7 wskazuje czy dany wpis jest ostatnim fragmentem. Bit 8 jest ustawiony jeśli wpis został usunięty(oznaczać to może usunięcie pliku bądź jedynie skrócenie nazwy).

Atrybut przyjmuje wartość `0x0f`(standardowo interpretowanej jako błąd) Stąd pliki takie są niedostępne dla starszego oprogramowania(żaden z programów nie obsługuje pliku, który ma flagi: tylko do czytania, ukryty, systemowy, etykieta dysku…).

Podsumowanie FAT

Obecnie systemy FAT są zastępowane przez nowszy produkt Microsoftu NTFS. Jest on dużo wydajniejszy i stabilny. Pomimo tego obsługę dla systemów FAT zawierają inne systemy operacyjne niż Windows: OS/2, Linux, FreeBSD, BeOS także Mac OS X na komputerach Apple. FAT jest nadal jest powszechnie używany, sprawia to jest dostępność i prostota. Często używa się go na dyskietkach, dyskach małej pojemności czy też na kartach pamięci flash używanych w kamerach i napędach USB oraz wszędzie tam gdzie z powodów technicznych nie można zastosować bardziej zaawansowanych systemów plików. Ważnym aspektem dla niektórych użytkowników jest też kompatybilność, stosunkowo młody system plików NTFS nie jest wspierany przez starsze wersje Windows.

NTFS

Wprowadzenie

System NTFS(ang. New Technology File System, w wolnym tłumaczeniu "system plików nowej generacji") pochodzi od systemu HPFS, który był opracowywany wspólnie przez Microsoft i IBM dla systemy OS/2. Sam HPFS posiada kilka ulepszeń w stosunku do FAT. Wsparcie dla metadanych, użycie zaawansowanych struktur danych(już nie tylko tablica alokacji) w celu polepszenia wydajności(szybkość i pojemność dyskowa). NTFS dodatkowo zawiera listy kontroli dostępu(ACL) i dziennik operacji dyskowych(usługa księgowania). NTFS jest standardowym systemem plików dla nowszych wersji Windows: Windows2000, Windows XP i Windows Server 2003. Warto powiedzieć, że NTFS powstał w głównej mierze z myślą o systemie operacyjnym Windows NT. Ponieważ FAT nie nadawał się do zastosowania go do systemu wieloużytkownikowego(brak obsługi praw dostępu do plików) należało zbudować nowy system plików od zera.

Jak na nowoczesny system plików przystało, NTFS zawiera usługę księgowania(od wersji 5), szyfrowania plików i katalogów(EFS – encode file system, nie jest jednak możliwe zaszyfrowanie partycji systemowej) czy kompresji danych w locie.

Organizacja i struktury danych w NTFS

W systemie plików NTFS, każda struktura jest plikiem, wliczając w to struktury do zarządzania partycją i statystyk. Informacje kontrolne o partycji przechowywane są w zestawie specjalnych plików, które są tworzone wraz z partycją. Zawierają one informacje o plikach na partycji, rozmiarze partycji, rozmieszczeniu klastrów itp. Jedynym wyjątkiem od zasady „wszystko jest plikiem" jest boot sektor, który poprzedza na partycji pliki specjalne. Odpowiedzialny jest za najbardziej podstawowe funkcje, np. ładowanie systemu operacyjnego.

Każdy plik na partycji NTFS stanowi zbiór atrybutów. Dotyczy to nawet zawartości pliku, która traktowana jest jako jeden z atrybutów. Inne atrybuty to nazwa pliku i jego rozmiar. Dzięki takiemu rozwiązaniu system operacyjny traktuje pliki jako obiekty o różnych charakterystykach, pozwala to na łatwe zarządzanie plikami i dodawanie atrybutów w przyszłości.

Boot sektor	**Główna tablica plików (MFT – Master File Table)**	**Pliki systemowe**	**Dane**
Skok do kodu ładującego system operacyjny, opis systemu plików, ponadto przechowuje wskaźnik na MFT.	Tablica z wpisami dla każdego z plików	Pierwsze 15 wpisów w MFT jest zarezerwowane dla specjalnych plików wykorzystywanych do trzymania dodatkowych informacji o systemie plików.	Pliki użytkownika

Boot sektor NTFS pomimo nazwy może zajmować nawet 16 sektorów na dysku. Składa się z dwóch struktur. Pierwsza zawiera podstawowe informacje o partycji jak nazwa, rozmiar. Druga zawiera dod małego programu, który instruuje system plików jak załadować system operacyjny.

Początek bajtów	**Długość pola**	**Przykładowa wartość**	**znaczenie**
0x0B	WORD	0x0002	Ile bajtów na sektor
0x0D	BYTE	0x08	Ile sektorów na klaster
0x0E	WORD	0x0000	Zarezerwowany sector
0x10	3 BYTES	0x000000	zawsze 0
0x13	WORD	0x0000	Nie używane przez NTFS
0x15	BYTE	0xF8	Media Descriptor
0x16	WORD	0x0000	zawsze 0
0x18	WORD	0x3F00	Ile sektorów na ścieżce
0x1A	WORD	0xFF00	Liczba nagłówkó
0x1C	DWORD	0x3F000000	Ukryte sektory

0x20	DWORD	0x00000000	Nie używane przez NTFS
0x24	DWORD	0x80008000	Nie używane przez NTFS
0x28	LONGLONG	0x4AF57F0000000000	Wszystkie sectory
0x30	LONGLONG	0x0400000000000000	Logical Cluster Number for the file $MFT
0x38	LONGLONG	0x54FF070000000000	Logical Cluster Number for the file $MFTMirr
0x40	DWORD	0xF6000000	Clusters Per File Record Segment
0x44	DWORD	0x01000000	Clusters Per Index Block
0x48	LONGLONG	0x14A51B74C91B741C	Volume Serial Number
0x50	DWORD	0x00000000	Suma kontrolna

NTFS podobnie jak FAT zarządza nie pojedynczymi sektorami lecz całymi klastrami. Tutaj jednak podobieństwa się kończą. NTFS wybiera rozmiar klastrów w zależności od rozmiaru partycji. Poniżej tabela.

Rozmiar partycji	**Liczba sektorów**	**Rozmiar klastra**
Poniżej 0.5GB	1	0.5
0.5GB – 1GB	2	1
1GB – 2GB	4	2
2GB – 4GB	8	4
4GB – 8GB	16	8
8GB – 16GB	32	16
16GB – 32GB	64	32
Powyżej 32GB	128	64

Informacja o wszystkich plikach zapisanych na partycji NTFS przechowywana jest w głównej tablicy plików – MFT. W odróżnieniu od FAT, który ma odrębną tablicę alokacji na początku dysku, NTFS umieszcza swój kluczowy element – Master File Table (MFT) - w ukrytych plikach. MFT zarządza wszystkimi plikami woluminu i tzw. metadanymi w relacyjnej strukturze danych. Informacja na temat plików jest uszeregowana w liniach; ich atrybuty (ukryty, zaszyfrowany, spakowany, systemowy, itd.) w kolumnach. Metadane zawierające informację o samym MFT, są przechowywane w 16 pierwszych zapisach, mających łącznie 16 KB.

Tablica MFT poniżej pokazuje pierwszych kilka rekordów. Następne rekordy danych MFT zawierają informację o pozycji w MFT, zawartości i nieużywanej powierzchni. Nawiasem mówiąc pliki do 900 bajtów mogą się znaleźć w całości w jednym rekordzie. Co do większych plików, MFT zawiera wskazówki, gdzie odnaleźć je w pamięci. To samo dotyczy folderów: jeśli są dostatecznie małe, są całkowicie zawarte w MFT. NTFS zarządza obszerniejszymi folderami, których struktura danych wskazuje na zewnętrzne klastry w tzw. strukturze B-drzewa. Zaletą struktury B-drzewa jest to, że NTFS indeksuje podobne pliki lub ich nazwy, co przyśpiesza wyszukiwanie właściwych plików.

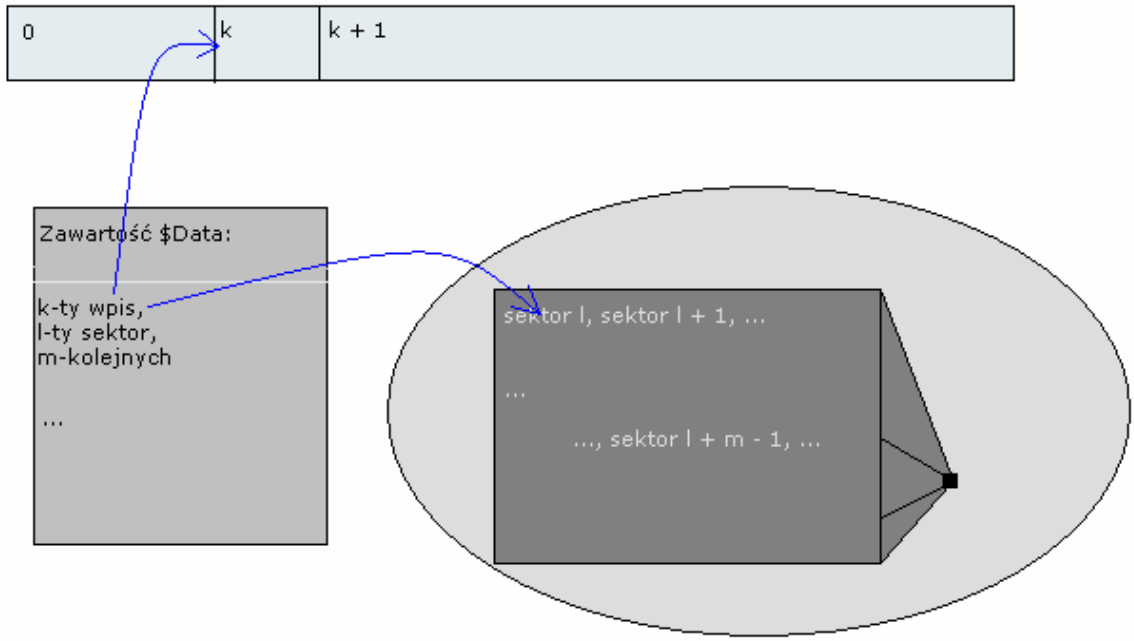
Rekord	Nazwa pliku	Zawartość	Opis
0	$Mft	Informacja MFT	Podstawowy zapis danych wszystkich folderów i plików
1	$MftMirr	Informacja MFT	Kopia pierwszego zapisu
2	$LogFile	Plik protokołu	Służy do przywrócenia spójności NTFS w następstwie błędów systemu
3	$Volume	Informacja o woluminie	Rozmaite informacje, takie jak nazwa i wersja woluminu
4	$AttrDef	Definicje atrybutów	Nazwy, liczby i opisy atrybutów pliku
5	.	Indeks katalogu głównego	Dane katalogu głównego
6	$Bitmap	Bitmapa klastra	Pokazanie wolnych klastrów
7	$Boot	Sektor startowy	Zawiera program startujący, pomiędzy innymi elementami
8	$BadClus	Uszkodzone sektory	Informacja o uszkodzonych sektorach
9	$Secure	Plik kopii zapasowej	Opis kopii zapasowej dla wszystkich plików
10	$Upcase	Konwerter	Tłumaczy małe litery na Unicode
11	$Extend	Rozszerzenia NTFS	Różne funkcje dodatkowe
12 do 15		Wolna przestrzeń	Na przyszły użytek

Jednego pliku może dotyczyć jeden lub więcej wpisów w MFT. Każdy wpis (rekord) w MFT składa się z małego nagłówka zawierającego podstawowe informacje o rekordzie oraz atrybutów, z których każdy ma nagłówek i ewentualnie dane. Lista jest uporządkowana zgodnie z numerami ID atrybutów. Dane atrybutu mogą być przechowywane wewnątrz rekordu MFT (wtedy mówimy o atrybutach rezydentnych) lub poza rekordem (wtedy mówimy o atrybutach nie rezydentnych). Jeśli lista atrybutów nie mieści się w jednym rekordzie MFT wtedy na potrzeby pliku rezerwowany jest kolejny rekord. Poniżej znajduje się tabela z możliwymi atrybutami. Użytkownik może dodawać plikom nowe atrybuty. Można dodać kilka atrybutów tego samego typu (np. strumień danych) pod różnymi nazwami. Windows 2000 używa tego mechanizmu aby z plikiem kojarzyć takie informacje jak podsumowanie czy informacje o autorze.

ID	Nazwa atrybutu	Opis
0x10	$Standart_information	Czas dostępu, modyfikacji, …
0x20	$Attribute_list	Lista atrybutów w innym wpisie MFT
0x30	$File_name	Atrybut dla długich(255 w Unicode). i skróconych nazw plików(8.3).
0x40	$Volume_version	Nie używane – usunięte w Windows 2000
0x40	$Object_id	Dodane w Windows 2000. GUID(Global Unique ID) przypisany wpisowi
0x50	$Security_description	Identyfikuje własciciela pliku oraz uprawnionych użytkowników
0x60	$Volume_name	Używane jedynie dla $Volume_version
0x70	$Volume_information	j.w. Nie używane – usunięte w Windows 2000
0x80	$Data	Dane pliku(zezwala na występowanie wielu atrybutów danych dla jednego pliku)
0x90	$Index_root	W przypadku katalogu korzeń B-drzewa wpisów
0xa0	$Index_allocation	W przypadku katalogu węzły B-drzewa
0xb0	$Bitmap	W przypadku katalogu bitmapa wolnych miejsc w katalogu
0xc0	$Symbolic_link	Nie używane w Windows 2000
0xc0	$Reparse_point	Dodane w Windows 2000, Używany przez punkty instalacji woluminów, a także przez sterowniki filtrów IFS (Installable File System) w celu oznaczenia niektórych plików jako specjalnych dla danego sterownika.
0xd0	$Ea_information	Dla zgodności z HPFS
0xe0	$Ea	j.w.
0xf0	$Property_set	
0x100	$Logged_utility_stream	Dodane w Windows 2000, Zbliżony do strumienia danych, z rejestracją operacji w pliku dziennika NTFS podobnie, jak zmiany metadanych NTFS, używany przez szyfrowanie EFS

Pojawia się pytanie jak dotrzeć do pliku w NTFS. Otóż jeśli do opisania pliku zostały użyte tylko atrybuty rezydentne, operacja ogranicza się do sięgnięcia wprost do odpowiedniego wpisu w MFT.

Sprawa jest nieco trudniejsza w przypadku atrybutów nierezydentnych, albowiem informacja o miejscu faktycznego przechowywania danych na dysku podaje przedziały. Idea polega na tym, że zapamiętywane są wskaźniki do spójnych bloków i ich rozmiar, z których czerpiemy informację, jak z nich posklejać cały plik. Istotne jest także to, że nie są przechowywane bezwzględne wskaźniki do danych na dysku, lecz względne, do poprzedniego wpisu. Odczytujemy kolejno wpisy do sektorów(na rysunku poniżej) dla k-tego wpisu pobieramy sektory od l-tego do (l + m - 1) sektora i podobnie postępujemy z następnymi wpisami.



Struktura katalogów kryje się w atrybutach `$Index_root`, `$Index_allocation`, `$Bitmap`. Ponieważ jak wcześniej wspomnieliśmy dane katalogowe są przechowywane w B-drzewie. Wpis katalogowy dla zwiększenia wydajności zawiera całą nazwę pliku i kopię jego atrybutu `$Standart_information`.

Węzły drzewa mają wielkość 4KB a wskaźniki do nich są przechowywane w atrybucie `$index_allocation`(w postaci przedziałowych wskazań do bloków na dysku). Korzeń drzewa trzymany jest w atrybucie `$Index_root`. Pliki w drzewie posortowane są leksykograficznie po nazwie.

Księgowanie w NTFS

Dziennik zmian jest nową funkcją NTFS w systemie Windows 2000, udostępniającą trwały rejestr zmian dokonywanych w plikach w woluminie. NTFS korzysta z dziennika zmian w celu śledzenia informacji o dodawanych, usuwanych i modyfikowanych plikach w każdym woluminie. Gdy dowolny plik jest tworzony, modyfikowany lub usuwany, system NTFS dodaje odpowiedni rekord do dziennika zmian w danym woluminie.

W momencie startu sterownika sprawdzany jest plik dziennika i to, czy wszystkie operacje zostały zakończone. Jeśli nie to są powtarzane/kończone. Przez to NTFS v.5.0 jest zdecydowane bezpieczniejszy od wcześniejszych systemów używanych w Microsoft Windows.

Dziennik zmian umożliwia znaczną skalowalność aplikacji, które bez jego wykorzystywania musiałyby przeszukiwać cały wolumin w poszukiwaniu zmian. Projektanci aplikacji indeksujących system plików, tworzących przyrostowe kopie zapasowe, menedżerów replikacji lub skanerów antywirusowych mogą korzystać z dziennika zmian w celu zwiększenia ich wydajności.

Dziennik zmian jest znacznie bardziej efektywnym sposobem rozpoznawania zmian w danym woluminie niż znaczniki czasu lub powiadomienia plików. Aplikacje, które do tej pory musiały skanować cały wolumin w celu odnalezienia zmian, mogą teraz wykonać ten proces jednorazowo, a następnie odwoływać się do dziennika zmian. Koszt operacji wejścia/wyjścia w tych aplikacjach staje się zależny od ilości zmienionych plików, a nie od ilości plików w woluminie.
Każdy rekord w dzienniku zmian zajmuje w przybliżeniu 80-100 bajtów. Dziennik zmian posiada określony limit rozmiaru, który nigdy nie jest przekraczany. Po jego osiągnięciu usuwana jest odpowiednia część najstarszych rekordów.

API związane z dziennikiem zmian są w pełni udokumentowane. Mogą z nich korzystać wszyscy dostawcy oprogramowania. Dostawcy oprogramowania planują wykorzystać tą funkcję do zwiększenia skalowalności i wydajności wielu różnorodnych produktów, takich jak oprogramowanie służące do tworzenia kopii zapasowych oraz antywirusowe.

Ponadto księgowanie jest cechą nowoczesnych systemów plików. Mają go np. BeFS (system plików BeOS-a), ext3 i reiserFS. Dzięki księgowaniu w Windows 2000/XP po restarcie, podczas startu systemu nie pojawi się chkdsk - co stałoby się w przypadku pracy na FAT16/32.

Kompresja w NTFS

System Windows 2000 umożliwia kompresję poszczególnych plików, folderów oraz całych woluminów NTFS. Pliki skompresowane w woluminie NTFS mogą być odczytywane i zapisywane przez dowolną aplikację systemu Windows bez dokonywania uprzedniej dekompresji za pomocą oddzielnego programu. Dekompresja pliku następuje automatycznie podczas jego odczytu, natomiast kompresja jest wykonywana po jego zamknięciu lub zapisaniu. Skompresowane pliki i foldery posiadają atrybut C, widoczny podczas przeglądania ich w Eksploratorze Windows.
Jedynie system plików NTFS może odczytywać dane w formie skompresowanej. Przed udostępnieniem pliku aplikacji, takiej jak Microsoft Word, lub poleceniu systemu operacyjnego, takiemu jak copy, sterownik

kompresji dokonuje jego dekompresji. Na przykład, gdy plik taki jest kopiowany pomiędzy dyskami twardymi komputerów używających systemu Windows 2000, to jest on w pierwszej kolejności dekompresowany, następnie kopiowany, a w końcu kompresowany podczas zapisywania.

Algorytm kompresji w systemie Windows 2000 jest zbliżony do używanego przez aplikację `DriveSpace 3` w systemie Windows 98, z jedną istotną różnicą – aplikacja ta potrafiła kompresować jedynie wolumin podstawowy lub logiczny. System NTFS pozwala na kompresję całego woluminu, jednego lub kilku folderów w woluminie lub nawet poszczególnych plików w folderze znajdującym się w woluminie NTFS.
Algorytm kompresji w systemie NTFS został zaprojektowany do obsługi klastrów o rozmiarze do 4 KB. Jeżeli w woluminie NTFS rozmiar klastra jest większy od 4 KB, to żadna z funkcji kompresji NTFS nie jest dostępna.

Woluminy, foldery i pliki w systemie NTFS mogą być skompresowane lub zdekompresowane. Stan kompresji folderu nie musi odzwierciedlać stanu kompresji znajdujących się w nim plików. Na przykład folder może być skompresowany, natomiast niektóre znajdujące się w nim pliki nie – jeżeli użytkownik selektywnie je zdekompresuje. Stan kompresji folderów i plików można zmieniać za pomocą Eksploratora Windows lub narzędzia `Compact` dostępnego z poziomu wiersza poleceń.

Szyfrowanie w NTFS

Kolejnym wygodnym usprawnieniem NTFS użytej w Windows 2000 jest szyfrowanie danych. Można szyfrować na dowolnym poziomie, pliki lub katalogi. Wszystko dzieje się na dość niskim poziomie – do użytkownika należy tylko decyzja, które partie danych mają zostać szyfrowane – odpowiada za do moduł związany ze Systemem Szyfrowania Plików (EFS - Encrypting File System), połączony ze sterownikiem urządzania.

EFS korzysta z szyfrowania za pomocą klucza symetrycznego w połączeniu z technologią kluczy publicznych. Pozwala to zabezpieczyć pliki i zagwarantować, że jedynie ich właściciel będzie miał do nich dostęp. Użytkownicy EFS otrzymują certyfikat cyfrowy oraz parę klucza publicznego i prywatnego. EFS korzysta z zestawu kluczy użytkownika zalogowanego w lokalnym komputerze, w którym przechowywany jest klucz prywatny.

Użytkownicy pracują z zaszyfrowanymi plikami i folderami w taki sam sposób, jak z innymi plikami. Szyfrowanie jest dla nich przezroczyste, system automatycznie odszyfrowuje plik lub folder, gdy użytkownik z niego korzysta. Podczas zapisania pliku wykonywany jest proces jego szyfrowania. Intruzi usiłujący uzyskać dostęp do zaszyfrowanego pliku lub folderu (próbując go otworzyć, skopiować, przenieść lub zmienić jego nazwę) otrzymują komunikat „Odmowa dostępu”.

Aby zaszyfrować lub odszyfrować folder lub plik, należy ustawić dla niego atrybut Zaszyfrowany, w taki sam sposób, jak ustawia się inne atrybuty. Po zaszyfrowaniu folderu wszystkie znajdujące się w nim pliki i podfoldery są automatycznie szyfrowane. Zalecane jest szyfrowanie na poziomie folderu.

Quota w NTFS

W NTFS można wprowadzić tzw. quotę (ograniczenie przestrzeni dyskowej dla użytkownika – możliwość znana z systemów linuksowych) oraz zabezpieczenia dostępu, np. można dokonać takich ustawień, aby nikt nie mógł otworzyć wybranych katalogów. Można tu również dokonać szyfrowania plików i folderów, co ograniczy do nich dostęp zostawiając możliwość odczytu/zapisu tylko szyfrującemu (przeważnie Administratorowi). Dla zwykłych użytkowników domowych (Home Users) może to jednak nie mieć znaczenia.

Wersje NTFS

- **NTFS v4 -** informacje o strukturze danych przechowywane są w rejestrze. Stąd problemy z przenośnością danych. Stosowany był w systemach NT do wersji 4.
- **NTFS v5 -** informacje o strukturze danych przechowywane są w specjalnym pliku. Plik ten nadal przechowywany jest w pamięci operacyjnej komputera. Stosowany jest w systemach Windows 2000, Windows XP, Windows 2003.
- **NTFS v6 -** informacje o strukturze danych przechowywane są bezpośrednio w systemie plików na dysku. Stosowany jest w systemach Windows XP, Windows 2003 (i prawdopodobnie w następcy XP - MS Vista).

Podsumowanie NTFS

NTFS wprowadził wiele ulepszeń w stosunku do systemów FAT. Najważniejszymi są: lepsze użycie pamięci w dużych woluminach, korekcję błędów w następstwie zawieszeń systemu, ochronę przed nieautoryzowanym dostępem, usługę katalogową i szyfrowanie danych. Charakterystyki odzyskiwania danych przez NTFS wymagają specjalnej wzmianki: Windows zapisuje, w czasie rzeczywistym, wszystkie modyfikacje w systemie plików w punktach kontrolnych. Punkty używane następnie są do naprawy w tle błędów systemu, powstałych przez wymuszony restart. NTFS może zarządzać partycjami o pojemności setek terabajtów (jeden terabajt to milion megabajtów). Jeśli chodzi o bezpieczeństwo, administratorzy korzystają ze zintegrowanych funkcji

ochronnych, takich jak uprawnienia dostępu użytkowników do plików i folderów, oraz EFS (ang. Encryption File System - System kodowania plików).

Pomimo swoich zalet, system NTFS u poprzedników Windows 2000, nie był w stanie sprostać wszystkim potrzebom ówczesnych komputerów. System NTFS rozprowadzany z Windows NT ogranicza zakres automatycznie nadawanych oznaczeń partycji do 26 liter (dyski od A do Z). Co więcej, zmiany partycji zawsze wymagają restartu. Faktem jest również, że informacja o woluminie NTFS jest przechowywana w rejestrze, co przysparza wiele kłopotów, jeśli używacie dysku z innym systemem.

W Windows 2000 problem ten rozwiązuje Logical Disk Manager (LDM), który nie wymaga już liter dysków. System NTFS jest również w stanie przechowywać informację o dysku na nim samym, w ten sposób rozwiązując problem wymiany dysków. Ulepszenia NTFS w Windows XP są niewielkie w porównaniu do Windows 2000. Poprawiono transfer danych i zastąpiono stały rozmiar 512 bajtowego klastra, możliwością określenia jego wielkości. Funkcje administracyjne, na przykład: indeksowanie folderów, ograniczanie ewentualnych strat danych i dostępu do danych, również zostały udoskonalone.

Porównanie windowsowych systemów plików

FAT32 jest szybszy, bo cache'owana jest mała tablica alokacji plików, w NTFS-ie zaś trzeba cache'ować dużą MFT (Meta File Table) nawet o rozmiarze 32-64 MB (w zależności od wielkości partycji i klastrów). Windows 2000/XP w tej samej konfiguracji chodzi szybciej na FAT-cie. Rozmiar klastra w NTFS-ie domyślnie wynosi od 512 lub 1K. Dostęp do dużych danych teoretycznie jest wolniejszy (w FAT32 standardowo 4K), (czyli np. do nagrywanie CD z ISO lepszy może być FAT32, a najlepszy będzie FAT16 z klastrem 16K/32K).

Pojedynczy plik na partycji FAT nie może przekroczyć 2GB. Partycja FAT16 nie może być standardowo większa niż 2GB, a FAT32 niż 8,4 GB (nie starczyłoby klastrów). Istnieją jeszcze inne wersje FAT'a: FAT16X i FAT32X (w `fdisk` włączona opcja obsługi dużych dysków), które mogą nie stosować się do tych ograniczeń. Jest to jednak standard czysto Microsoft'owy.

Jeśli nieistotne dla nas jest marnowanie miejsca, a zależy nam na szybkości(pomijając bezpieczeństwo) to najlepszy jest FAT16, jeśli zależałoby nam trochę bardziej na miejscu to wybrałbym FAT32. W innych wypadkach (bezpieczeństwo i ochrona danych) zdecydowanie NTFS.

Wszelkie kryteria zebrane do tabeli poniżej:

Kryteria	NTFS v5	NTFS	FAT32	FAT16
Systemy operacyjne	Windows 2000, Windows XP	Windows NT, Windows 2000, Windows XP	Windows 98, Windows ME, Windows 2000, Windows XP	DOS, wszystkie wersje Microsoft Windows
Limity				
Maks rozmiar partycji	2TB	2TB	2TB	2GB
Maks liczba plików	Praktycznie nieograniczone	Praktycznie nieograniczone	Praktycznie nieograniczone	~65,000
Maks rozmiar pliku	Zależny tylko od rozmiaru partycji	Zależny tylko od rozmiaru partycji	4GB	2GB
Maks liczba klastrów	Praktycznie nieograniczone	Praktycznie nieograniczone	268,435,456 (4,177,920 praktycznie)	65,535
Maks długość pliku	Powyżej 255	Powyżej 255	powyżej 255	Standard - 8.3 (możliwe ponad 255)
Cechy systemów				
Nazwy plików	Standard Unicode	Standard Unicode	Systemowy zbiór znaków	Systemowy zbiór znaków
Kopia zapasowa	Kopia MFT	Kopia MFT	kopia FAT	kopia FAT
Lokalizacja boot sektora	Pierwszy i ostatni sektor	Pierwszy i ostatni sektor	Pierwszy sektor	Pierwszy sektor
Atrybuty pliku	Standardowe i do wyboru	Standardowe i do wyboru	standardowe	Standardowe
kompresja	Tak	Tak	Nie	Nie
szyfrowanie	Tak	Nie	Nie	Nie
Prawa dostępu	Tak	Tak	Nie	Nie
Quota	Tak	Nie	Nie	Nie
Punkt montowania partycji	Tak	Nie	Nie	Nie
Ogólne przeznaczenie				
bezpieczeństwo	Tak	Tak	Nie	Nie
Odzyskiwanie danych	Tak	Tak	Nie	Nie
przeznaczenie	Mała na małych partycjach, duża na dużych.	Mała na małych partycjach, duża na dużych.	Duża na małych partycjach i mała na dużych	Duża na małych partycjach i mała na dużych
Przestrzeń dyskowa	Maks	Maks	średnia	Minimalna na dużych partycjach
Tolerancja błędów	Maks	Maks	min	średnia

Linux

EXT2

Ext2, czyli *Second Extended File System* wciąż chyba jeszcze jest podstawowym i najszerzej używanym systemem plików dla Linuxa. Jest bardzo efektywny w typowych zastosowaniach, a równocześnie stosunkowo prosty. Jest uważany za jeden z najlepszych "standardowych" systemów plików.

Ext2 powstał jako rozwinięcie systemu plików Ext, który z kolei zastąpił używany we wczesnych wersjach Linuxa system Minix. Minix narzucał dość poważne ograniczenia na nazwy plików (do 14 znaków) jak i na rozmiar samej partycji (maksymalnie 64MB). Ext usunął te ograniczenia, ale nie był wystarczająco wydajny (powodował dużą fragmentację plików), a także nie zapamiętywał daty dostępu do pliku ani daty zmiany metadanych pliku.

Główne cechy systemu Ext2

- Zapewnia wszystkie elementy systemu plików Unix (dowiązania symboliczne, pliki specjalne, prawa dostępu...)
- Wysoka wydajność dzięki przeciwdziałaniu fragmentacji (poprzez przydzielanie bliskich bloków oraz prealokację).
- Wydajny mechanizm dowiązań symbolicznych
- Stabilny i dobrze przetestowany (sam system plików, jak również program naprawiający *e2fsck*)
- Dobrze zdefiniowany sposób dodawania rozszerzeń
- Niezależny od tworzącego systemu operacyjnego (wszystkie pola wielobajtowe zapisane w standardzie little-endian)
- Maksymalny rozmiar partycji to 4TB, a pojedynczego pliku 2GB. Maksymalna długość nazwy pliku: 255 znaków.
- Obsługa "dziurawych" plików (nieużywane bloki nie zostają przydzielone).

Wady:

- Mało efektywna obsługa katalogów (choć użycie pamięci podręcznej znacznie ją poprawia)

- Niska wydajność dla bardzo małych plików (rzędu kilkuset bajtów) - duże straty na alokację i stosunkowo wolny dostęp
- Długotrwałe sprawdzanie systemu plików po niepoprawnym zamknięciu systemu (o idei walki z tym zjawiskiem, obecnym przecież w wielu systemach (np. FAT32), opowiemy jeszcze szerzej poniżej...)

Struktura systemu plików

Partycja systemu plików Ext2 podzielona jest na **bloki** o rozmiarze 1024, 2048 lub 4096 bajtów (na niektórych architekturach do 8192 b). Kolejne bloki połączone są w **grupy**, których rozmiar zależy od wybranego rozmiaru bloku. Podział na grupy zwiększa lokalność danych związanych z jednym plikiem, a co za tym idzie, przyspiesza dostęp do nich. Grupy zostają ustalone statycznie podczas tworzenia systemu plików.

Grupa:

Superblok	Deskryptory grup	Mapa bitowa bloków	Mapa bitowa i-węzłów	Tablica i-węzłów	Bloki danych

W oryginalnej wersji Ext2, w każdej grupie znajdował się **superblok**, opisujący cały system plików, oraz **deskryptory grup**, opisujące sumaryczne informacje o wszystkich grupach. Miało to na celu przyspieszenie dostępu do tych informacji, a także zabezpieczenie istotnych danych przed awarią dysku. W nowszej wersji wprowadzono możliwość pominięcia kopii w niektórych grupach, aby zmniejszyć ilość marnowanego miejsca. Kopie superbloku i deskryptorów grup znajdują się wtedy w blokach o numerach 0, 1, oraz będących potęgą 3, 5 lub 7.

Poza superblokiem i deskryptorami grup, identycznymi na całym dysku, w skład grupy wchodzą:

- Mapa bitowa bloków: zajmuje pojedynczy blok, w którym każdy bit odpowiada jednemu blokowi grupy. Bit ustawiony oznacza, że odpowiedni blok jest zajęty
- Mapa bitowa i-węzłów: opisuje zajętość i-węzłów w tej grupie
- Tablica i-węzłów: bloki, w ramach których przydzielane są metryczki plików (*i-węzły*). Liczba i-węzłów w ramach grupy ustalana jest statycznie podczas tworzenia systemu plików

Superblok

Zawiera informacje o całym systemie plików, takie jak:

- Rozmiar bloku
- Liczby wszystkich oraz wolnych bloków i i-węzłów
- Stan systemu plików (czy został poprawnie zamknięty)
- Licznik zamontowań i data ostatniego sprawdzenia - pozwalają wymusić sprawdzenie integralności danych co pewien czas
- Wersja systemu plików - zbiór 3 różnych kategorii rozszerzeń, w zależności od stopnia wprowadznych przez nie niezgodności. Są to:
    - *COMPAT* - zmiany należące do tej kategorii są zupełnie przejrzyste dla starszych wersji
    - *RO_COMPAT* - zmiany te nie powodują błędów przy odczycie, ale zapis w takim systemie przez wersję nie obsługującą rozszerzenia spowodowałby uszkodzenie danych. Stara wersja może używać takiej partycji w trybie tylko do odczytu
    - *INCOMPAT* - wprowadzone zmiany są na tyle poważne, że próba użycia systemu plików przez starszą wersję może skończyć się błędem.

Normalnie używana jest podstawowa wersja superbloku z 1024-go bajtu urządzenia. W przypadku jego uszkodzenia można użyć jednej z kopii.

Deskryptory grup

Jest to tablica rekordów, po jednym dla każdej grupy, opisujących sumaryczne dane (m.in. liczbę wolnych i-węzłów i bloków). Informacje te są używane podczas przydzielania bloków.

I-węzły

I-węzeł opisuje wszystkie atrybuty obiektu zapisanego w systemie plików, poza jego nazwą. Są to:

- Dowiązania do bloków danych (12 bezpośrednich i po jednym pojedynczo, podwójnie i potrójnie pośrednim)
- Prawa dostępu
- Właściciel (użytkownik i grupa)
- Typ obiektu i różne flagi
- Rozmiar obiektu i liczba używanych przez niego bloków
- Data dostępu, modyfikacji, zmiany metadanych i skasowania obiektu
- Liczba dowiązań (wpisów katalogu odwołujących się do tego i-węzła)
- Dodatkowe informacje (wersja, *Access Control List*, *Extended Attributes*)

Istnieje kilka pól o podwójnym znaczeniu, zależnym od typu pliku, a także pól zarezerwowanych do przyszłych zastosowań:

- Górna połowa 64-bitowego pola długości dla plików staje się dowiązaniem do *ACL* katalogów (na razie żadne z tych rozszerzeń nie jest zaimplementowane)
- Miejsca na rozszerzone, 32-bitowe pola właściciela i grupy

Specjalnie obsługiwane są **dowiązania symboliczne**. Jeśli nazwa wskazywanego pliku nie jest dłuższa niż 60 znaków, to zamiast w oddzielnym bloku zapisywana jest w samym i-węźle, na pozycjach normalnie używanych jako wskaźniki do bloków (w sumie jest ich 15, każdy po 4 bajty). Podobnie zaimplementowane są pliki urządzeń - wtedy przechowywany jest numer oznaczanego przez i-węzeł urządzenia. Używanie tej techniki daje bardzo duży zysk, zarówno pod względem czasu dostępu, jak i zużywanego miejsca. Planowane jest jej rozszerzenie na krótkie pliki.

Katalogi

Są zorganizowane jako standardowe pliki rekordów zmiennej długości. Zapisane są więc w tej samej przestrzeni, co bloki zwykłych plików. Każdy rekord zawiera pole określające jego długość, co pozwala "przeskakiwać" nad dziurami powstałymi po usunięciu krótkich wpisów. Poza tym rekordy zawierają nazwę pliku oraz numer jego i-węzła.

W nowszych wersjach systemu Ext2 także typ obiektu (plik, katalog, plik specjalny, dowiązanie symboliczne) zapisywany jest w katalogu. Pozwala to ograniczyć liczbę wczytywanych i-węzłów w operacjach przeszukiwania katalogu. Niestety, użycie tej techniki wymaga wsparcia programów trybu użytkownika, którego dotąd nie ma.

Obecnie używana implementacja katalogów nie jest zbyt efektywna, gdyż opiera się właściwie na listach jednokierunkowych. Ogranicza to używalną wielkość pojedynczego katalogu do 10-15 tysięcy wpisów. Już od pewnego czasu trwają prace nad zamianą obecnie używanych list jednokierunkowych na tablice haszujące. Pozwoliłoby to sprawnie używać katalogów o nawet 100 tys. do miliona wpisów.

Dodatkowe funkcje

- Konfigurowalny (podczas tworzenia systemu) rozmiar bloku. Mniejsze bloki powodują mniej strat na alokację, ale zwiększają narzut związany z zarządzaniem nimi.
- Pewna część bloków oraz i-węzłów zostaje zarezerwowana dla administratora. Poza względami bezpieczeństwa, pozwala to zmniejszyć fragmentację
- Możliwość wymuszenia synchronicznego zapisu metadanych (struktur opisujących pliki), jednak kosztem znacznego spadku wydajności

- Możliwość wybrania semantyki System V lub BSD podczas montowania systemu. Różnią się one grupą do jakiej należą nowo tworzone pliki (w BSD należą one do takiej grupy, jak nadkatalog, a w System V dodatkowo sprawdzana jest flaga *setgid*)
- Bezpieczne usuwanie pliku - pliki z ustawioną odpowiednią flagą zostają zamazane podczas kasowania tak, aby ich zawartości nie dało się odczytać narzędziami do edycji dysków
- Pliki niezmienialne (*immutable*) - nie mogą zostać przez nikogo zapisane ani usunięte
- Pliki tylko do dopisywania (*append-only*) - można zapisywać jedynie na ich koniec

Idea journallingu

Podczas używania standardowych systemów plików takich jak Ext2 mogą pojawić się problemy po nagłym wyłączeniu komputera. Może się zdarzyć, że system operacyjny przestanie działać w trakcie wykonywania pewnych operacji na dysku, pozostawiając system plików w niespójnym stanie. Może się to objawiać m.in:

- utratą części bloków, które nie należą już do żadnego pliku, ale nie zostały oznaczone jako wolne. Może się tak stać w przypadku, gdy system zdążył oznaczyć plik jako skasowany, ale nie zwolnił tworzących go bloków. Poza utratą części pojemności, jest to zjawisko stosunkowo niegroźne
- mogą nie zostać zaktualizowane pewne liczniki, np. wolnych bloków w grupie (w systemie Ext2). W wyniku nieprawidłowo może działać algorytm przydziału bloków
- adres nowo dodawanego bloku pośredniego może zostać wpisany do i-węzła zanim zostanie zapisany sam blok, w efekcie czego różne pliki mogą używać tych samych bloków. Gdy błędny plik zostanie skasowany, zwolnione (i zamazane) zostaną bloki współdzielone z innymi, dotąd poprawnymi plikami. Błędy tego typu są najpoważniejsze.

Używanie błędnego systemu plików na ogół powoduje rozszerzenie błędów i może prowadzić do utraty większości danych na partycji.

Aby temu przeciwdziałać, w systemie Ext2 używa się flagi mówiącej o tym, czy system został poprawnie zamknięty. Jeśli nie, wymuszane jest uruchomienie programu *fsck*, który znajduje i naprawia wszelkie niezgodności w badanym systemie plików.

Nie jest to jednak rozwiązanie idealne:

- Sprawdzanie dużego systemu plików może trwać bardzo długo. W tym czasie pliki na sprawdzanej partycji są niedostępne
- Niektóre pliki mogą zostać utracone, np. podczas dopisywania informacji do dużego pliku, zniszczona może zostać duża część jego poprzedniej zawartości

Journaling (kronikowanie) jest to funkcja systemów plików polegająca na prowadzeniu dziennika (journal), w którym zapisywane są operacje zlecone systemowi plików ale jeszcze nie zakończone **(transakcje)**. Czyli jest to mechanizm podobny do stosowanego w bazach danych.

Operacje zapisu, tworzenia lub usuwania plików lub katalogów nie są atomowe. System plików musi przechowywać oprócz danych zapisywanych w plikach, także dane o samej strukturze plików i strukturze systemu plików (tzw. metadane). Stąd biorą się problemy ze spójnością podczas awarii systemu, gdy nastąpi ona między operacjami na metadanych i operacjami na danych (w trakcie jednej operacji np. zapisu). Transakcje zapewniają zapis takiej jednej operacji dyskowej (zapisu, tworzenia lub usuwania pliku (katalogu)) do dziennika jako operacji atomowej. Jest ona zapisywana w całości i tylko jeśli wszystkie jej kroki przejdą pomyślnie może zostać uznana za wykonaną.
W momencie uruchomienia systemu po awarii wszystkie operacje w całości zapamiętane w dzienniku są wykonywane, a te które nie są zapisane w całości są wymazywane z dziennika i ignorowane (na pewno nie zostały nawet rozpoczęte na dysku).
System plików z kroniką po restarcie zwykle przywraca spójność dysku w czasie rzędu kilku, kilkunastu sekund.
Restart systemu po awarii w systemach z kronikowaniem **nie zależy od wielkości partycji**, co jest szczególnie istotne w przypadku wielkich dysków.

Tryby journallingu

Tylko metadanych

Najczęstszą praktyką w systemach plików jest journalling jedynie zmian w metadanych (katalogach, i-węzłach, superbloku itp.). Choć gwarantuje to spójność systemu plików, na końcu plików zapisywanych przed awarią mogą pojawić się niepoprawne dane.

Danych i metadanych

Najprostszym sposobem usunięcia tego problemu jest journalling tak metadanych, jak i samych danych pliku. Jest to jednak zbyt kosztowne -

każdy zapis dokonywany jest dwukrotnie, najpierw do dziennika a potem do docelowego bloku.

Tryb *ordered*

Journallowane są tylko metadane, ale przed ich zatwierdzeniem zostają zapisane wszystkie bloki danych. Działa to nadal nieco wolniej niż journalling samych metadanych, ale znacznie szybciej niż pełny.

Ext3

Jest to rozszerzenie do standardowego systemu plików Ext2, dodające do niego ***journalling***. Ext3 został stworzony przez zespół dr Stephena Tweedie, jednego z twórców Ext2.

System plików Ext3 jest nie tylko oparty na Ext2, ale poza dodanym *journalling*iem, nie różni się od oryginału prawie niczym. Zgodność ta posunięta jest do tego stopnia, że poprawnie odmontowaną partycję Ext3 można zamontować jako Ext2 (i odwrotnie).

Do przejścia z ext2 na ext3 wystarczy np. pakiet E2fsprogs w wersji powyżej 1.21

Journalling w systemie plików Ext3

Ext3 przeznacza pewien obszar dysku na plik dziennika. Wszelkie zmiany metadanych na partycji są najpierw zapisywane do dziennika, a dopiero potem na dysk. Jeśli nastąpi awaria, to podczas przeładowywania systemu wszystkie kompletne zmiany zapamiętane w dzienniku zostają zapisane na dysku, zaś zmiany, które nie zostały w całości zapisane do dziennika zostają zignorowane (gdyż z pewnością nie trafiły jeszcze na dysk).

Dziennikowanie w ext3 nie jest bezpośrednio zaimplementowana. Wykorzystane jest specjalne API nazywane warstwą *Journaling Block Device* (JBD). JBD zostało napisane by w prosty sposób dołączać dziennikowanie do dowolnego urządzenia blokowego.

W ext3 dziennik tworzony i zarządzany przez JBD trzymany jest w inode, czyli po prostu w pliku. Domyślnie jest to plik .journal w katalogu

głównym. Od wersji 0.9.5 ext3 umożliwia trzymanie dziennika na dowolnym innym urządzeniu blokowym. Można znacznie zwiększyć wydajność systemu plików przenosząc dziennik na partycje na innym dysku, bądź testować działanie ext3 trzymając dziennik w ramdysku. Dzięki takiemu sposóbowi trzymania dziennika zachowujemy dużą kompatybilność z ext2.

Większość systemów plików z dziennikowaniem przechowuje w dzienniku tylko różnice pomiędzy blokami danych do zapisania i tymi, które są zapisane na dysku (listę bajtów do zmiany), co nazywamy dziennikowaniem logicznym (*logical journaling*). JBD robi to inaczej. W dzienniku przechowywane są całe zmienione bloki. Takie podejście nazywa się dziennikowanie fizycznym (*physical journaling*). Mogłoby się wydawać, że zmniejsza to wydajność systemu ze względu na dużą ilość danych przesyłanych na dysk. Jednak pozwala to połączenie w pamięci kilku operacji na bloku zmniejszając ilość operacji zapisu na dysku. Również fakt, że w pamięci znajdują się gotowe do zrzucenia bloki zmniejsza nakład obliczeniowy procesora.

Wykorzystanie JBD przez ext3 dokonuje się na tej zasadzie, że sterownik ext3 przed zapisaniem czegoś na dysk informuje o tym JBD i czeka na zezwolenie na zapis.

Główne cechy Ext3

- Udostępnia wszystkie te funkcje, co Ext2
- Zrównoważony pod względem wydajności
- Udostępnia trzy sposoby journallingu:
    - samych metadanych
    - danych i metadanych
    - samych metadanych, ale z wcześniejszym zapisywaniem bloków danych (zwany *ordered*, jest to sposób domyślny)

Podstawową wadą tak Ext3, jak i innych systemów z *journalling*iem, jest teoretycznie mniejsza wydajność (te same informacje muszą zostać zapisane dwa razy). Okazuje się jednak, że w typowych zastosowaniach nie ma to negatywnego wpływu na wydajność.

Tryb *ordered*

Ext3 domyślnie używa trybu *ordered*.

Aby zapewnić poprawne działanie trybu *ordered*, Ext3 stosuje następujące techniki:

- *Revoke Records* - specjane rekordy zapisywane do dziennika. Umożliwiają pominięcie odtwarzania wskazanych bloków z wcześniejszych transakcji. Używane są po skasowaniu metadanych (np. katalogu), aby modyfikacje już nieistniejących struktur nie zamazały nowych bloków pliku.
- Ext3 nigdy nie przydziela na dane tych bloków, które zostały zwolnione, ale informacja o tym nie została jeszcze zatwierdzona w dzienniku. Zapobiega to zamazaniu bloków np. skasowanego katalogu.
- Lista "osieroconych" plików (*orphaned files*) - pliki, które zostały usunięte z wszystkich katalogów, ale są nadal otwarte przez pewne procesy, nie mogą zostać zwolnione. Gdyby w takim momencie przerwać działanie systemu, nie można by ich ani usunąć, ani odzyskać. Dlatego wszystkie takie pliki trzymane są na liście (wskazywanej z superbloku). Dodatkowo, lista ta umożliwia podzielenie operacji skrócenia pliku (która może angażować dowolnie wiele bloków) na mniejsze transakcje.

Kolejna cecha ext3:

Indeksowane katalogi

W ext3 jest możliwość włączenia indeksowanego formatu katalogu. Indeksowany format katalogu pozwala na szybsze wyszukiwanie pozycji w katalogu. Jest to przydatne zwłaszcza gdy ilość plików w katalogu jest stosunkowo duża.

Struktura pliku katalogu:

- 0: Root Block Index
- 1..511: Index Block/0
- 512: Directory Entry
- 513: Directory Entry
- ...

Bloki 0 do 511 to bloki indeksów, kolejne bloki są tradycyjnymi blokami przechowującymi wpisy katalogów. W blokach indeksów przechowywane są wpisy postaci:
klucz : wskaźnik do bloku
Klucz jest generowany na podstawie nazwy funkcją haszującą, najbardziej znaczący bit to flaga kolizji. Wskaźnik wskazuje zależnie od wysokości drzewa indeksowania na kolejny blok indeksu (1-511) bądź na blok z wpisami katalogowymi. Pierwszy blok indeksowy zawiera dodatkowo nagłówek, w którym określony jest typ indeksowania, wersja funkcji haszującej, ilość poziomów indeksowania.

Operacja szukania wpisu w katalogu polega na obliczeniu funkcji haszującej i schodząc w dół drzewa według wskaźników dojść do bloku zawierającego poszukiwany wpis. Dalsze wyszukiwanie odbywa się tak jak w bloku katalogu bez indeksowania. Operacja zapisu przebiega podobnie, z tym że potrzebny jest mechanizm rozwiązywania kolizji (przepełnienia bloku z wpisami katalogowymi):

- posortowanie według klucza i podzielenie na dwa bloki
- poprawienie drzewa indeksów

Gdy zdarzy się, że wszystkie wpisy w bloku przepełniającym się mają ten sam klucz to ustawiana jest flaga kolizji w kluczu i zapisuje się nowe wpisy do następnego bloku.

W obecnej implementacji uwzględniony jest tylko jeden poziom indeksowania. Pozwala to na trzymanie w katalogu około 90 000 wpisów. Dołączenie kolejnego poziomu umożliwiłoby zapis 50 milionów plików.

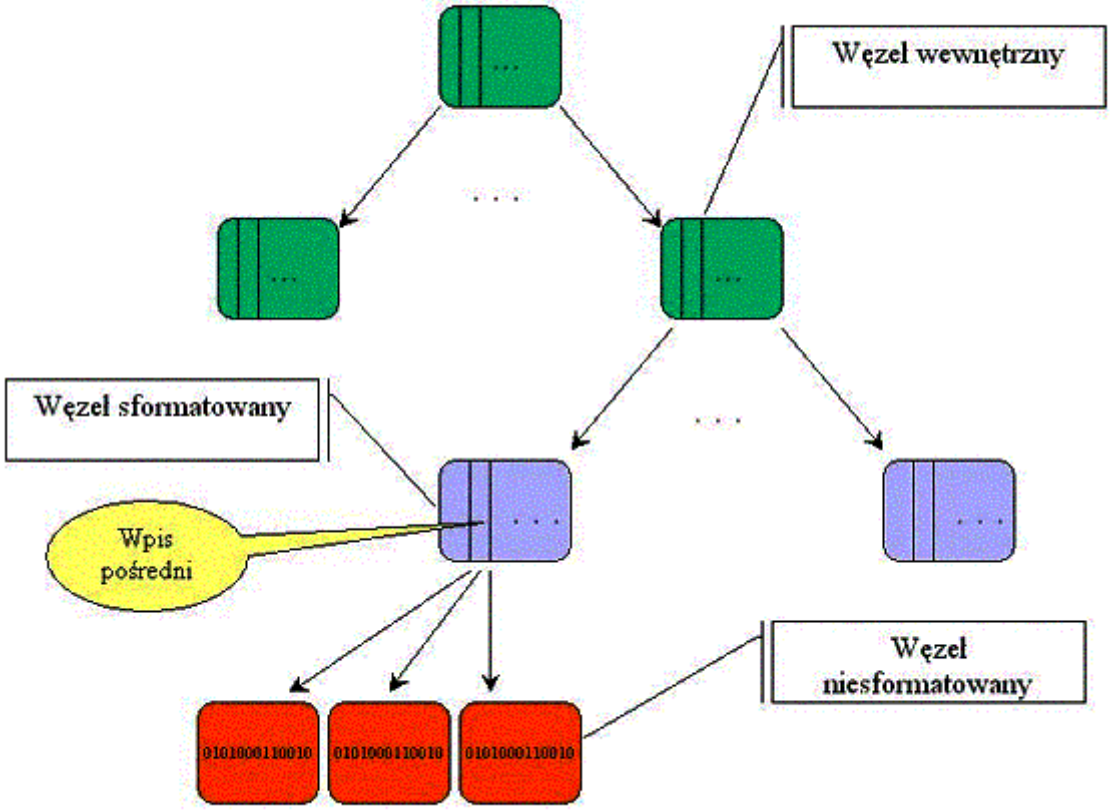
Czas utworzenia plików		
Ilość plików	Indeksowany	Normalny
10 0000	1s	23s
50 0000	7s	9m31s
90 0000	13s	33m18s

ReiserFS

Jest to system plików tworzony na zasadzie open source. Stworzony został przez Hansa Reisera.
Podstawowe cechy tego systemu plików to:

- - zastosowanie B+ drzew do przechowywania plików i ich i-węzłów. Jedno drzewo obejmuje wszystkie bloki partycji. Jest to więc w pełni oryginalne rozwiązanie.
- - zapamiętywanie wielu małych plików lub końcówek dużych plików w węźle drzewa (w metadanych, nie w liściu gdzie przechowywane są fragmenty dużych plików)
- - nie posiada i-węzłów (wszystkie informacje są w B+ drzewie)
- - kronikowanie metadanych

B+ drzewo:



Wyróżniamy trzy rodzaje węzłów (bloków):

- o **węzły wewnętrzne** (internal nodes) – zestawy kluczy i wskaźników do kolejnych węzłów
- o **węzły sformatowane** (formatted nodes) – są to liście drzewa, które składają się z wpisów (items). Każdy wpis ma swój unikatowy klucz wyszukiwania i może być jedną z postaci:
  - ☐ bezpośredni (direct item) – zawiera końcówki (ogony) plików (lub małe całe pliki)
  - ☐ pośredni (indirect item) – zawiera wskaźnik do węzła niesformatowanego, który zawiera kolejne bajty pliku, bez końcówki (ogona)
  - ☐ katalog (directory item) – zawiera klucz pierwszego wpisu do katalogu i nazwy wpisów (directory entries) i liczbę wpisów tego katalogu
  - ☐ stat data – przechowuje dodatkowe dane pliku lub katalogu
- **węzły niesformatowane** (unformatted nodes) – przechowują fragmenty dużych plików, są w pełni zajęte przez dane

Wyrównywanie plików do początków i końców bloków

Wyrównywanie plików do granic bloku ma następujące efekty:

- minimalizuje liczbę bloków dla pliku (jest to szczególnie zaletą dla dużych plików, gdy odwołujemy się do ich danych w sposób mało lokalny)
- marnuje przestrzeń dysku i bufora, przechowując całe, nie do końca zapisane bloki
- marnuje czas operacji wejścia-wyjścia na ściąganie całych, nie do końca zapisanych bloków
- zwiększa średnią ilość bloków potrzebnych dla dostępu do każdego pliku w katalogu
- ma prostszy kod

Przechowywanie plików w zrównoważonych drzewach

System ReiserFS przechowuje zarówno pliki, jak i nazwy plików w zrównoważonym drzewie. Małe pliki, katalogi, i-węzły oraz końce dużych plików (ogony) są efektywnie upakowane, dzięki odejściu od wyrównywania plików do bloków i i-węzłów o stałych rozmiarach. Duże pliki przechowywane są w niesformatowanych węzłach przyłączonych do drzewa, ale bez możliwości przenoszenia w algorytmach balansowania drzewa.

ReiserFS używa B+ drzew. B+ drzewa różnią się od B-drzew tym, że dane przechowywane są na samym dole drzew w liściach. W tej implementacji krótkie informacje (tzn. Katalogi, nazwy plików, małe pliki i ogony dużych plików) przechowywane są bezpośrednio w liściach, a duże pliki przechowywane są pod liśćmi.

Słabe strony pakowania wielu plików do bloku

- kiedy ogon pliku (pliki poniżej 4K są całe ogonem) urośnie na tyle, żeby zająć cały węzeł, zostaje usunięty ze sformatowanego węzła i przeniesiony do niesformatowanego węzła
- jeśli ogon jest mniejszy od całego węzła może zostać rozdzielony pomiędzy dwa węzły, co powoduje konieczność dwóch operacji dyskowych
- separowanie ogona od reszty plików może zmniejszyć efektywność czytania
- dodanie jednego bajta do pliku lub ogona, który nie jest na końcu węzła powoduje przeniesienie średnio połowy wielkości węzła w pamięci; sytuacja ta może się zdarzyć podczas niestandardowego niebuforowanego zapisu do pliku (funkcje biblioteczne I-O zapewniają buforowany dostęp do danych)

Zalety pakowania wielu plików do bloku

- większa efektywność wykorzystania operacji wejścia-wyjścia
- brak znaczących różnic w szybkości dla różnych rozmiarów bloków

Struktura drzewa

Drzewo zrównoważone w ReiserFS składa się z trzech rodzajów węzłów: internal nodes (węzły wewnętrzne), formatted nodes (liście) i unformatted nodes (podliście; węzły, które mogą wystąpić bezpośrednio pod liśćmi).

Internal nodes pełnią zwykłą funkcję węzłów wewnętrznych B-drzewa.

Formatted nodes są liśćmi B-drzewa. Składają się z **items**. Items zawierają unikatowy klucz do wyszukiwań i mogą być jednym z rodzajów:

- direct item – zawiera ogony plików
- indirect item – zawiera wskaźnik do unformatted node, zawierającego dane pliku (ale nie ogon)
- directory item – zawiera klucz pierwszego directory entry i liczbę directory entries, które zawiera
- stat data – zawiera dodatkowe dane dla pliku lub katalogu; znajduje się zawsze na początku pliku lub katalogu

Drzewo zapisane jest w blokach dysku. Każdy blok należący do drzewa zaczyna się od Block_head.

Internal node

Block_Head	Key 0	Key 1	---	Key N	Pointer 0	Pointer 1	---	Pointer N	Pointer N+1	....Free Space....

Formatted node

Block_Head	IHead 0	IHead 1	IHead 2	---	IHead N	....Free Space...	Item N	---	Item 2	Item 1	Item 0

Unformatted node

. . . . .

Struktury internal node

Disc block

Block_Head	Key 0	Key 1	---	Key N	Pointer 0	Pointer 1	---	Pointer N	Pointer N+1	....Free Space....

struct block_head

Field Name	**Type**	Size (bytes)	Description
blk_level	unsigned short	2	Level of block in the tree (1- leaf; 2,3,4,... - internal;
blk_nr_item	unsigned short	2	Number of Keys in an Internal block. Or Number of Items in a Leaf block.
blk_free_space	unsigned short	2	Block Free Space in bytes
blk_right_delim_key	struct key	16	Right delimiting key for this block (for Leaf nodes only)
	total	6 (8) 22 (24)	(6) 8 bytes for internal nodes ; (22) 24 bytes for leaf nodes

struct key

Field Name	**Type**	**Size (bytes)**	**Description**
k_dir_id	__u32	4	ID of the parent directory
k_object_id	__u32	4	ID of the object (also it is the number of inode)
k_offset	__u32	4	Offset from beginning of the object to the current byte of the object
k_uniqueness	__u32	4	Type of the item (StatData = 0, Direct = -1, InDirect = -2, Directory = 500)

	total	16	16 bytes

struct disc_child (wskaźnik do bloku dyskowego)

Field Name	**Type**	**Size (bytes)**	**Description**
dc_block_number	unsigned long	4	Disk child's block number.
dc_size	unsigned short	2	Disk child's used space.
	total	6	(6) 8 bytes

Struktury formatted (leaf) node

Struktura formatted node zawiera nagłówki i ciała items.

disc block

Block_Head	IHead 0	IHead 1	IHead 2	- - -	IHead N		Item N	- - -	Item 2	Item 1	Item 0

struct item_head

Field Name	**Type**	**Size (bytes)**	**Description**
ih_key	struct key	16	Key to search the item. All item headers is sorted by this key
u.ih_free_space u.ih_entry_count	__u16	2	Free space in the last unformatted node for an InDirect item; 0xFFFF for a Direct item ; 0xFFFF for a Stat Data item. The number of directory entries for a Directory item.
ih_item_len	__u16	2	total size of the item body

ih_item_location	__u16	2	an offset to the item body within the block
ih_reserved	__u16	2	used by reiserfsck
	total	24	24 bytes

struct stat_data

Field Name	Type	Size (bytes)	Description
sd_mode	__u16	2	file type, permissions
sd_nlink	__u16	2	number of hard links
sd_uid	__u16	2	owner id
sd_gid	__u16	2	group id
sd_size	__u32	4	file size
sd_atime	__u32	4	time of last access
sd_mtime	__u32	4	time file was last modified
sd_ctime	__u32	4	time inode (stat data) was last changed (except changes to sd_atime and sd_mtime)
sd_rdev	__u32	4	device
sd_first_direct_byte	__u32	4	Offset from the beginning of the file to the first byte of direct item of the file. (-1) for directory (1) for small files (file has direct items only) (>1) for big files (file has indirect and direct items) (-1) for big files (file has indirect, but has not direct item)
	total	32	32 bytes

Directory item

deHead 0	deHead 1	deHead 2	---	deHead N	**fileName N**	---	**fileName 2**	**fileName 1**	**fileName 0**

Struct reiserfs_de_Head (deHead)

Field Name	**Type**	**Size (bytes)**	**Description**
deh_offset	__u32	4	third component of the directory entry key (all reiserfs_de_head sorted by this value)
deh_dir_id	__u32	4	objectid of the parent directory of the object, that is referenced by directory entry
deh_objectid	__u32	4	objectid of the object, that is referenced by directory entry
deh_location	__u16	2	offset of name in the whole item
deh_state	__u16	2	1) entry contains stat data (for future) 2) entry is hidden (unlinked)
	total	16	16 bytes

direct item

.....Small File Body.....

undirect item

unfPointer 0	unfPointer 1	unfPointer 2	---	unfPointer N

unfPointer - pointer to unformatted block (unfPointer size = 4 bytes)

Wstawianie węzła do drzewa

Przeszukuję się bitmapę wolnych bloków zaczynając od lewego sąsiada ostatnio używanego węzła i poruszając się w tym samym kierunku, co ostatnio.

W testach okazało się, że metoda ta jest lepsza od następujących alternatyw:

- wyszukiwanie pierwszego wolnego bloku w bitmapie
- branie pierwszego za ostatnio przydzielonym i poruszanie się w tym samym kierunku, co ostatnio (3% szybsze przy zapisie i 10-20% wolniejsze przy odczycie)
- zaczynanie od lewego sąsiada i poruszanie się w kierunku, branym od prawego sąsiada

Okazało się również, że metoda jest o ~10% szybsza niż gdybyśmy zaczynali od aktualnego węzła, mimo, że ryzykujemy jeden odczyt sięgnięcia do ojca, jeśli lewego sąsiada by nie było.

Porządek w drzewie

Porządek użyty w drzewie ma duże znaczenie dla wydajności systemu. Wpływa na lokalność czytania danych i efektywność upakowywania ogonów.

Struktura klucza składa się z pól: locality_id, object_id, offset, uniqueness.

Locality_id standardowo wskazuje na object_id katalogu nadrzędnego, co zapewnia nam lokalność.

Każdy plik, czy katalog ma unikatowy object_id.

Optymalizacje w balansowaniu drzewa

Priorytety:

- minimalizacja ilości użytych węzłów
- minimalizacja ilości węzłów poddających się operacji balansowania
- minimalizacja ilości uncached węzłów poddających się operacji balansowania
- jeśli przenoszenie do innego formatted node jest potrzebne, maksymalizacja przenoszonych danych

Ostatni warunek oparty jest na lokalizacji. Istnieje duża szansa, że następna operacja dyskowa będzie w tym samym miejscu. W związku z tym, robimy sobie miejsce na następne operacje dyskowe.

Tak na koniec

ReiserFS używa innowacyjnych technik przechowywania danych, znanych wcześniej w systemach bazodanowych. Użycie zrównoważonych drzew wydaje się być kosztowne algorytmicznie, ale testy dowodzą, że

efektywność systemu jest porównywalna z ext2fs. ReiserFS doskonale się sprawdza przy przechowywaniu małych plików, oszczędzając dużo miejsca. Równie dobrze działa obsługa katalogów z bardzo dużą ilością plików (np. 100 000).

Reiser 4

Co istotne, Reiser 4 został napisany OD POCZĄTKU. Nie jest to rozszerzenie Reisera FS (tak, jak to miało miejsce w przypadku ext3 i ext2). Ten sam główny projektant, ale dzieło zupełnie nowe ☺

Wiele udoskonaleń nadeszło...:

- system transakcji, który jeszcze bardziej rozszerza pojęcie journalingu
- możliwość dołączania własnych plug-inów; użytkownik może na przykład stworzyć swoją własną abstrakcję katalogu
- lepsze zapewnianie bezpieczeństwa
- lepsza wydajność
- zmiana architektury systemu na bardziej obiektową
- używanie repackera – specjalnego programu, który upakowuje ogony, jeszcze bardziej oszczędzając miejsce
- zmiana struktury drzew na drzewa "tańczące"; w czasie działania systemu struktura drzewa zmienia się dopiero przy operacji flush lub commit, a nie dla każdej operacji dyskowej

Drzewa tańczące to dość nowa idea. Inne podejście. Taki QuickSort: dla pewnych przypadków może i mają gorszą złożoność i działają po prostu wolniej, ale tak naprawdę w systemach plików liczy się praktyka. A ta okazuje się i tym razem nieubłagana. W rzeczywistości są lepsze niż klasyczne B+ drzewa...

Główną ideą jest usunięcie z węzłów wewnętrznych jakiejkolwiek informacji zawartej w plikach. Są tam czyste metadane. Skąd przypuszczenie, że może to być wydajniejsze? Przecież taki mały plik ma szybki czas dostępu i pozornie wydaje się to być bardzo korzystne. Okazuje się jednak, że praktyka jak zwykle ma „swoje" do powiedzenia.

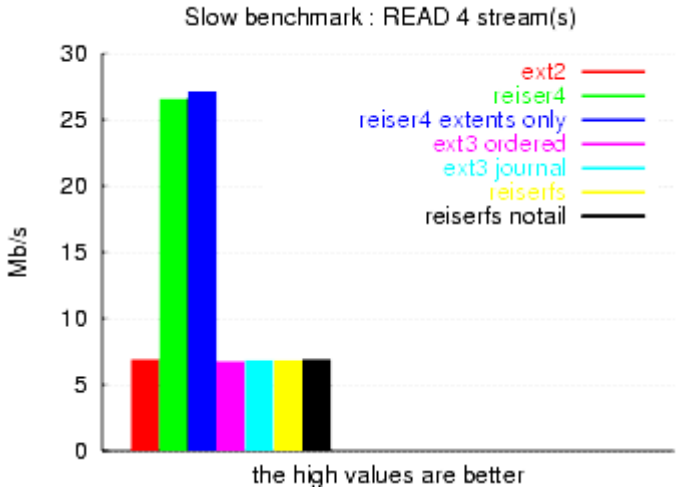
W celu wyjaśnienia tego fenomenu dobrze jest posłużyć się pewną abstrakcją, postarać się danym przydzielić pewną cechę. I tak informacji przydzielmy pewną gęstość lub może (jak piszą o tym na stronie Reisera) temperaturę. Niech będzie to stosunek częstości odwołań do wielkości. Przy czym cały plik traktujmy jak jedną informację. Podobnie czynimy z poszczególnymi metadanymi.

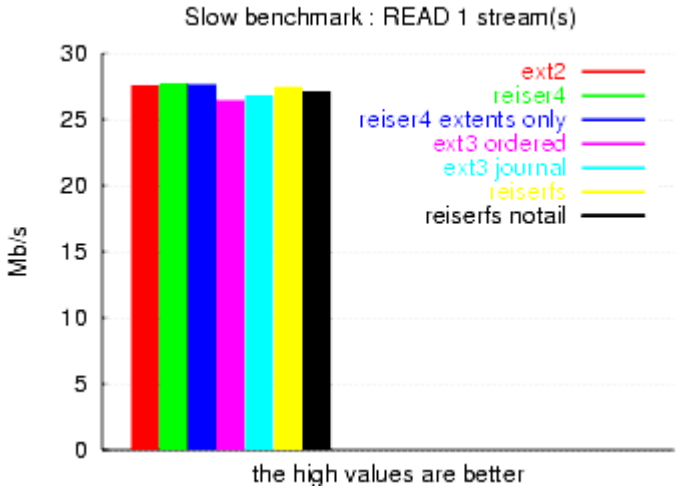
I tak oto okazuje się, że małe pliki mają niemal zawsze mniejszą temperaturę niż metadane! Zauważmy, że być może zajmują one niewiele, ale jednak na pewno więcej niż te kilka wskaźników. Jedni powiedzą: „No tak, ale przecież mamy do nich szybszy czas dostępu, więc może umieszczajmy tam często używane, małe pliki!?!”. I takie podejście okazuje się być błędem!! Plik zazwyczaj nie jest często używany przez dłuższy okres czasu!! Należy pamiętać o tym, że struktury naszego drzewa nie chcemy zbyt często zmieniać. A taki plik, gdy już „zostanie wykorzystany do granic”, będzie tylko zabierał cenne miejsce wysoko w strukturze drzewa, a jego temperatura, wcześniej być może wysoka, zacznie dość szybko spadać...

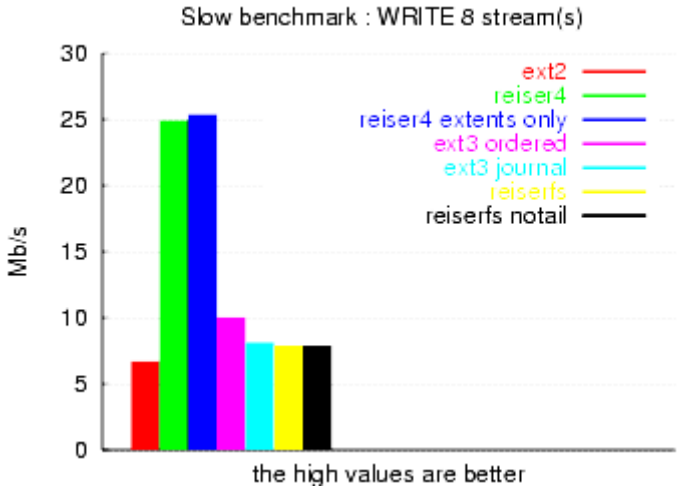
I tak oto, pomimo niewątpliwych zalet i wysokiej wydajności (przynajmniej o takiej mówią testy ze strony Reisera), system ten nie jest jeszcze domyślnym w jakiejś gałęzi jądra. Być może się to zmieni. Czas pokaże...

Pare słów o wydajnosci...

Tabelki porównawcze. Mówią wszystko...







Windows a Linux

Windows

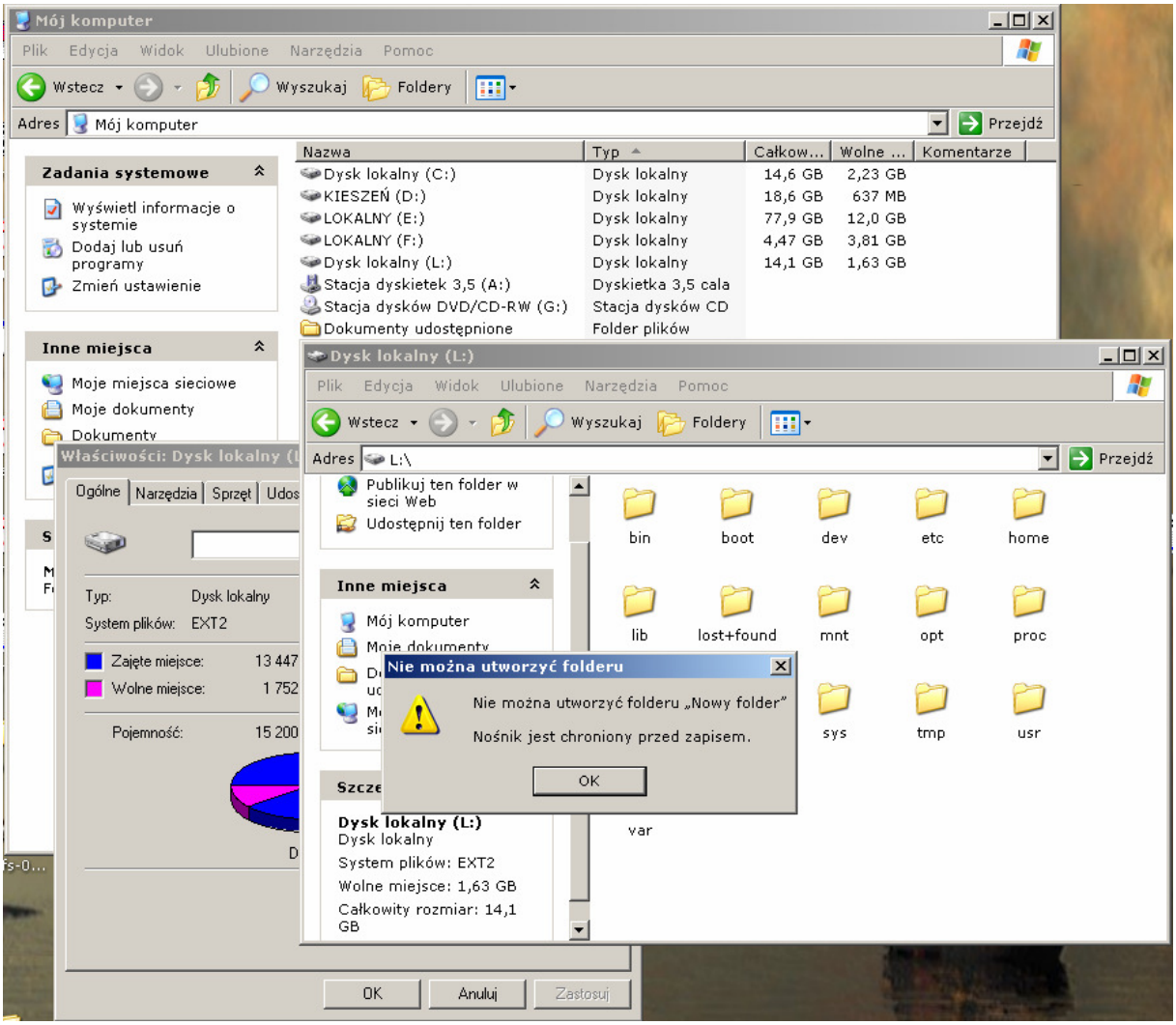
Omówiliśmy jak realizowane są systemy plików w systemach linuksowych i windosowych. Jak jest z współdzieleniem ich na jednym(kilku) dysku. Spotyka się opinię, że pod Windowsem nie ma łatwego wglądu do partycji linuksowych. A jednak odwiedźmy stronę internetową:

`http://uranus.it.swin.edu.au/~jn/linux/ext2ifs.htm`

Znajdujemy tam sterowniki(EXT2IFS) umożliwiające windowsowi na odczyt partycji linuxowych – ext2 oraz ext3. Po ściągnięciu pliku:

`http://uranus.it.swin.edu.au/~jn/linux/ext2ifs/ext2ifs-0.3.zip`

i rozpakowaniu. Uruchamiamy instalatora. Po zaznaczeniu paru opcji i restarcie komputera dostajemy w systemie nową partycję(partycje) linuxową widoczną w „Mój Komputer”.



Jak widać nie można modyfikować partycji(zarówno spod zwykłego konta jak i konta administratora).

Istnieje też inny program pod Windows odczytujący partycje linuksowe – Explore2fs. Dostępny pod adresem:

`http://uranus.it.swin.edu.au/~jn/linux/explore2fs.htm`

Również odczytuje on partycje ext2 i ext3 ale nie integruje się z systemem w tym samych stopniu co EXT2IFS. Partycje linuksowe są dostępne jedynie spod konta administratora.

Także partycje reiserFS można odczytać pod Windowsem. Potrzebny do tego jest program virtualrfstoll dostępny:

`http://sourceforge.net/projects/visualrfstool/`

Tak więc przy odrobinie wysiłku można korzystać z partycji linuksowych pod Windowsem.

Linux

Pod Linuxem w nowych wersjach jądra jest wsparta obsługa partycji windowsowych jak FAT czy NTFS(jako moduł). Korzystanie z owych partycji jest proste:

```
$ mount -t ntfs /dev/hda1 /mnt/Windows
```

lub:

```
$mount -t vfat /dev/hda6 /mnt/Lokalny
```

I od tej pory partycje są widoczne w hierarchi katalogów odpowiednio pod: /mnt/Windows I /mnt/Lokalny. Można też zamiast stałego wykonywania polecenia mount umieścić odpowiedni wpis w pliku /etc/fstab:

```
/dev/hda1   /mnt/Windows ntfs     defaults        1 0
/dev/hda2   /            ext3     defaults        1 1
/dev/hda6   /mnt/Lokalny vfat     defaults        1 0
/dev/cdrom  /mnt/cdrom   auto     noauto,owner,ro 0 0
/dev/fd0    /mnt/floppy  auto     noauto,owner    0 0
devpts      /dev/pts     devpts   gid=5,mode=620  0 0
proc        /proc        proc     defaults        0 0
none        /sys         sysfs    defaults        0 0
```

Wszelkie pliki z tychże partycji są zintegrowane z linuxem z dokładnością do praw dostępu czy atrybutu wykonywalności.

Porównanie Systemów plików

Maximum filename length	Maximum pathname length	Maximum file size	Maximum volume size
FAT16	Unicode except NUL	4GiB	16MiB to 8GiB
FAT32	Unicode except NUL	4GiB	GiB to 2TiB
NTFS	255 bytes	Any Unicode except NUL	16EB
ext2	255 bytes	2TB to 32TB	
ext3	255 bytes	2TB to 32TB	
ReiserFS V3	4032 bytes/255 characters	16TB	

Rich File type metadata	Stores file owner	POSIX file permiss -ions	Creation time-stamps	Last access/ read time-stamps	Last metadata change time-stamps	Last archive time-stamps	Access control lists	Security/ MAC labels	Alternate data stream / resource fork
FAT16	No	No	Yes	Yes	No	No	No	No	No
FAT32	No	No	Yes	Yes	No	No	No	No	No
NTFS	Yes	Yes	Yes	Yes	Yes	No	Yes	?	Yes
ext2	Yes	Yes	No	Yes	Yes	No	Yes	?	No
ext3	Yes	Yes	No	Yes	Yes	No	Yes	?	No
ReiserFS V3	Yes	Yes	No	Yes	Yes	No	Yes	?	No
Reiser4	No	Yes	Yes	No	Yes	Yes	No	No	?

Hard links	Soft links	Block journaling	Metadata-only journaling	Case-sensitive	Case-preserving	File Change Log
FAT16	No	No	No	No	No	No
FAT32	No	No	No	No	No	No
NTFS	Yes	Yes	Yes	Yes		
ext2	Yes	Yes	No	No	Yes	Yes
ext3	Yes	Yes	Yes	Yes	Yes	Yes
ReiserFS V3	Yes	Yes	Yes	Yes	Yes	No
Reiser4	Yes	Yes	?	Yes	Yes	Yes