Lokalne systemy plików

Łukasz Chmielewski
Andrzej Mizera
Mateusz Stachlewski

26 grudnia 2002

Spis treści

1 Sytem Plików FAT

1.1 Wstęp

System plików FAT został napisany przez Billa Gatesa w 1977 roku. Początkowo obsługiwał tylko dyskietki. Później zaadaptowano go do pracy z dyskami twardymi. Został podstawowym systemem plików DOSa.

Jednostką alokacji w systemie FAT jest klaster (ang. cluster), którego rozmiar ustalany jest podczas formatowania partycji. Klaster to odpowiednik bloku w innych systemach plików. Możliwe rozmiary klastra to 0.5kB, 1kB, 2kB, 4kB, 8kB, 16kB, 32kB oraz 64kB.

System FAT (ang. File Allocation Table czyli tablica przydziału plików) to odmiana przydziału listowego. Plik stanowi powiązaną listę bloków dyskowych. Na początku partycji zapisana jest tablica FAT o rozmiarze dokładnie równym liczbie klastrów danej partycji. Tablica ta indeksowana jest numerami klastrów (bloków) dyskowych. Pozycja w tablicy zawiera numer następnego bloku.

1.2 Wersje systemu plików FAT

Wersje systemu FAT to FAT12, FAT16 oraz FAT32. Liczba w nazwie (12, 16 lub 32) to rozmiar (w bitach) pozycji w tablicy FAT.

Wersja	**FAT12**	**FAT16**	**FAT32**
Zastosowanie	dyskietki i dyski twarde do 16MB	partycje do 2GB	partycje do 2TB
Wielkość wpisu w tablicy FAT	12 bitów	16 bitów	28 bitów
Maksymalna liczba klastrów	4,086	65,526	268,435,456
Rozmiar klastra	0,5kb do 4kB	2kB do 32kB	4kB do 64kB

Tablica 1: Parametry wersji systemu FAT

1.3 Fizyczna struktura partycji FAT

Partycja FAT ma następujący układ na dysku:

- **Boot sector** - kod bootujący, jeśli partycja ma ustawioną flagę Active,
- **FAT 1** - tablica FAT,
- **FAT 2** - zapasowa kopia tablicy FAT,
- **katalog główny** - klastry katalogu głównego,
- **obszar danych** - klastry z danymi, klastry wolne i uszkodzone,

1.4 Tablica FAT

Tablica FAT implementuje przydział listowy. Pozycja w tej tablicy definiuje:

- pusty(wolny) klaster (0 dla wszystkich wersji FAT),
- uszkodzony klaster (zarezerwowana sekwencja bitów),
- ostatni klaster pewnego pliku (zarezerwowana wartość),
- indeks następnego klastra (wartość różna od trzech powyższych).

Puste(wolne) klastry to po prostu zerowe wpisy w tablicy FAT. Dostęp do nich nie jest w żaden sposób optymalizowany (np. nie ma listy pustych klastrów).

1.5 Format katalogu

Katalog jest specjalnym rodzajem pliku. Składa się z listy 32 bajtowych struktur odpowiadających plikom i podkatalogom w danym katalogu.

W systemach FAT12 i FAT16 katalog główny przechowywany jest tuż za tablicami FAT oraz ma ograniczoną liczbę pozycji (max 512 wpisów). W systemie FAT32 katalog główny nie jest w żaden sposób wyróżniony tzn. może być przechowywany w dowolnych klastrach i nie ma sztywnego ograniczenia na jego wielkość.
Oto deklaracja struktury katalogu w języku C:

```
struct directory { // Nazwy krótkie w formacie 8.3
    unsigned char name[8]; // Nazwa pliku
    unsigned char ext[3]; // Rozszerzenie
    unsigned char attr; // Atrybuty pliku (1 bajt)
    signed char lcase; // Wielkość liter nazwy i rozszerzenia
    unsigned char ctime_ms; //Czas utworzenia pliku w milisekundach
    unsigned char ctime[2]; // Czas utworzenia
    unsigned char cdate[2]; // Data utworzenia
    unsigned char adate[2]; // Czas ostatniego dostępu
    unsigned char reserved[2]; // 2 zarezerwowane bajty
    unsigned char time[2]; // Czas ostatniej modyfikacji
    unsigned char date[2]; // Data ostatniej modyfikacji
    unsigned char start[2]; // Numer pierwszego klastra pliku
    unsigned char size[4]; // Logiczny rozmiar pliku
};
```

Możliwe atrybuty pliku to:

- System - plik systemu DOS,
- Archive - plik archiwalny używany do tworzenia kopii inkrementalnych,
- Hidden - plik ukryty,

- Read-Only - plik tylko do odczytu,
- Volume-Label - plik zawierający etykietę partycji,
- Directory - katalog.

W systemie FAT32 2-bajtowe pole **reserved** przechowuje 16 starszych bitów numeru pierwszego klastra pliku. W systemach FAT12 i FAT16 jest ono ignorowane.

1.6 Długie nazwy w systemie FAT32

System FAT32 wprowadził długie nazwy plików (max 819 znaków). W celu zachowania zgodności z poprzednimi wersjami FAT, długie nazwy przechowywane są w postaci kilku struktur **struct directory** opisanych powyżej. Każda struktura przechowuje maksymalnie 13 znaków długiej nazwy. Zatem długa nazwa pliku może być zakodowana za pomocą wielu takich wpisów katalogowych. Pola tych dodatkowych struktur są ustawiane na specjalne wartości, aby starsze oprogramowanie ignorowało te dodatkowe wpisy w katalogach.

1.7 Zalety systemu FAT

- **wyeliminowanie fragmentacji zewnętrznej**
- **prosta implementacja**
- **efektywny dla partycji o małych rozmiarach**

1.8 Wady systemu FAT

- **dostęp do pliku praktycznie tylko sekwencyjny** - koszt liniowy,
- **podatność na awarie** - cross-linked files, lost clusters itp,
- **nieefektywny dla partycji o dużych rozmiarach**
- **klastry przydzielone katalogowi zwalniane są dopiero przy jego kasowaniu** - skasowanie pliku A lub podkatalogu B z katalogu C zwolni bloki przydzielone plikowi A lub podkatalogowi B ale nie zwolni klastrów przydzielonych katalogowi C

2 Systemy plików Ext2 i Ext3

2.1 Wstęp

Ext2 czyli Second Extended File System to najczęściej używany linuksowy system plików. Opublikowano go w 1993 roku jako ulepszoną wersję pierwotnego systemu Extended File System. Jest to nowoczesny system plików, realizujący uniksową semantykę plików i oferujący zaawansowane funkcje. Istnieją implementacje Ext2 dla innych systemów uniksowych, dla GNU Hurd, dla Windows 95/98/NT i OS/2.

2.2 Fizyczna struktura partycji Ext2

Partycja Ext2 ma następujący układ na dysku:

- **sektor startowy (ang. boot sector)** - ignorowany przez Ext2
- **grupy bloków systemu Ext2**

Każda grupa ma poniższą postać:

- **superblok (1 blok)**
- **tablica deskryptorów grup**
- **bitmapa zajętości bloków (1 blok)**
- **bitmapa zajętości i-węzłów (1 blok)**
- **tablica i-węzłów**
- **bloki z danymi, bloki wolne i uszkodzone**

Bitmapy zajętości bloków i i-węzłów przechowywane są w pojedyńczych blokach dyskowych. W każdej grupie może więc być co najwyżej 8*t bloków, gdzie t jest rozmiarem bloku w bajtach. Dopuszczalne wielkości bloku to 1kB, 2kB oraz 4kB. Wszystkie grupy bloków w systemie plików mają ten sam rozmiar i są przechowywane sekwencyjnie, dzięki czemu jądro może pobrać położenie grupy na dysku za pomocą prostego indeksowania. Jądro stara się przechowywać bloki danych należące do danego pliku w tej samej grupie, minimalizując w ten sposób fragmentację.

Superblok (struktura ext2_super_block) przechowuje podstawowe informacje o systemie plików. Kopia superbloku trzymana jest w każdej grupie bloków. Jądro standardowo korzysta z kopii zapisanej w grupie nr 0 czyli w pierwszej grupie. Wybrane pola:

- całkowita liczba i-węzłów i bloków danych
- liczba wolnych i-węzłów i bloków danych

- liczba i-węzłów i bloków w każdej grupie
- punkt montowania systemu plików
- liczba operacji montowania od ostatniego sprawdzenia
- co ile montowań sprawdzana jest spójność systemu plików
- status systemu plików (zamontowany, poprawnie odmontowany, niepoprawny)

Tablica deskryptorów grup to tablica z deskryptorami wszystkich grup (struktura ext2_group_desc). W każdej grupie bloków trzymana jest jej kopia. Jądro domyślnie korzysta z kopii zapisanej w grupie numer 0. W deskryptorze grupy trzymane są między innymi następujące dane:

- liczba wolnych bloków oraz i-węzłów w grupie
- numer bloku mapy bitowej bloków
- numer bloku mapy bitowej i-węzłów
- numer pierwszego bloku tablicy i-węzłów

Mapy bitowe zajętości bloków i i-węzłów określają które z bloków (i-węzłów) w grupie są wolne. Ustawiony bit oznacza zajęty blok lub (odpowiednio) i-węzeł. Mapy bitowe są rozmiaru dokładnie jednego bloku dyskowego.
Tablica i-węzłów składa się z serii sąsiadujących bloków zawierających i-węzły danej grupy (struktury dyskowe ext2_inode). Najważniejsze pola i-węzła dyskowego:

- typ pliku i prawa dostępu
- licznik sztywnych dowiązań
- logiczna długość pliku w bajtach
- liczba bloków danych pliku
- wskaźniki do bloków danych (pole i_block)
- znaczniki czasowe: ostatni dostęp, ostatnia modyfikacja

2.3 Adresowanie bloków danych

Pole **i_block** w i-węźle pozwala adresować bloki danych na dysku. Bloki dyskowe mogą być adresowane:

- **bezpośrednio** tzn. adresy pierwszych 12 bloków z danymi są zapisane w samym i-węźle
- **pośrednio** czyli adresy bloków z danymi są zapisane w bloku, którego adres zapisany jest w i-węźle
- **podwójnie pośrednio** tzn. jak wyżej ale są 2 pośrednie bloki
- **potrójnie pośrednio** tzn. jak wyżej ale są aż 3 pośrednie bloki

2.4 Rodzaje plików

W systemie Ext2 **zwykły plik** ma przyporządkowane bloki dyskowe w których przechowywane są dane w dowolnym formacie (tzn. w formacie narzuconym przez procesy korzystające z pliku). Ext2 implementuje **katalogi** jako specjalny rodzaj pliku, przechowujący pary (nazwa pliku, numer i-węzła). **Dowiązanie symboliczne** wymaga pojedyńczego bloku danych tylko wtedy, gdy przechowuje ścieżkę o długości przekraczającej 60 znaków (wówczas nazwa nie mieści się w i-węźle). Pozostałe typy plików (**nazwane potoki, gniazda, pliki urządzeń znakowych i blokowych**) nie korzystają z bloków danych tzn. potrzebne informacje trzymane są w i-węźle.

2.5 Cechy systemu Ext2

- dynamiczna alokacja bloków danych: blok jest przyporządkowywany plikowi dopiero, gdy proces próbuje zapisać do niego dane
- system dzieli bloki dyskowe na grupy (jądro stara się przechowywać dane pliku w tej samej grupie co redukuje fragmentację pliku)
- system prealokuje dyskowe bloki zwykłych plików zanim zostaną użyte (mniejsza fragmentacja i czas dostępu, domyślnie 8 bloków)
- szybkie dowiązania symboliczne (ścieżki do 60 znaków trzymane są w i-węźle tzn typowe symlinki nie zajmują przestrzeni dyskowej)
- wybór rozmiaru bloku przy tworzeniu systemu plików (1, 2 lub 4kB)
- możliwość określenia liczby i-węzłów
- automatyczne sprawdzanie spójności systemu w czasie startu (po awarii systemu, po określonej liczbie montowań lub po pewnym czasie)
- przy nadawaniu ID grupy nowemu plikowi zgodność z semantyką BSD lub SVR4

2.6 Cechy systemu Ext3

- journalling tzn. mechanizm księgowania zwiększający bezpieczeństwo systemu i minimalizujący czas sprawdzania systemu po awarii
- fragmentacja bloków tzn. blok może przechowywać kilka małych plików we fragmentach o stałej dlugości
- listy kontroli dostępu (opcjonalne)
- obsługa skompresowanych i zaszyfrowanych plików (opcjonalnie)
- logiczne usuwanie (ang. undelete)

3 NTFS - New Technology File System

3.1 Historia powstania

Na początku lat 90-tych celem Microsoft'u było stworzenie nowego systemu operacyjnego (Windows NT), który byłby w stanie konkurować na rynku systemów operacyjnych przeznaczonych do zastosowań biznesowych. Ówczesne systemy Microsoft'u: MS-DOS i Windows 3.x nie miały szans z systemami takimi, jak np. UNIX. Słabym punktem tych systemów był przede wszystkim FAT, który nie spełniał wymagań stawianych systemom plików wykorzystywanych do przechowywania i zarządzania danymi w systemach sieciowych.

3.2 Główne cele

Głównym celem było stworzenie elastycznego (ang. flexible), dającego się łatwo dostosowywać (ang. adaptable), zapewniającego wysoki poziom bezpieczeństwa (ang. high-security) i niezawodności (ang. high-reliability) systemu plików. Aby zrealizować taki system, przy projektowaniu NTFS wzięto pod uwagę następujące elementy:

- **niezawodność (ang. reliability)** - jedną z najważniejszych cech systemu plików jest niedopuszczenie do rozspójnienia danych w przypadku wystąpienia awarii oraz możliwość szybkiego przywrócenia stanu sprzed awarii. W NTFS został zaimplementowany system transakcji, który umożliwia niepodzielne wykonywanie operacji, dzięki czemu przechowywane dane są spójne,
- **bezpieczeństwo i kontrola dostępu (ang. security and access control)** - główną z wad systemu FAT był brak kontroli dostępu do katalogów i plików przechowywanych na dysku,
- **zniesienie limitów związanych z rozmiarami partycji** - na początku lat 90-tych system FAT-16 umożliwiał tworzenie partycji, których rozmiary nie przekraczały 4GB, co w przypadku np. baz danych nie jest ograniczeniem do zaakceptowania,
- **długie nazwy plików** - NTFS pozwala na 255-cio znakowe nazwy plików, w porównaniu ze standardowymi 8 (nazwa) + 3 (rozszerzenie) w przypadku zwykłego FAT'a,
- **wymagania stawiane systemom sieciowym (ang. networking)**.

Powszechnie uważa się, że NTFS powstał od zera, ale tak naprawdę system ten korzysta z wielu kluczowych rozwiązań systemu HPFS, będącego sytemem plików systemu operacyjnego OS/2.

3.2.1 Wersje NTFS

- Najbardziej rozpowszechnioną wersją NTFS jest wersja, która występuje pod dwiema nazwami:

  - NTFS 1.1 - nazwa oficjalna ,
  - NTFS 4.0 - nazwa pochodząca od Windows NT 4.0 (najbardziej popularanej wersji Windows NT korzystającej z NTFS).

- NTFS 5.0 - wykorzystywana przez Windows 2000

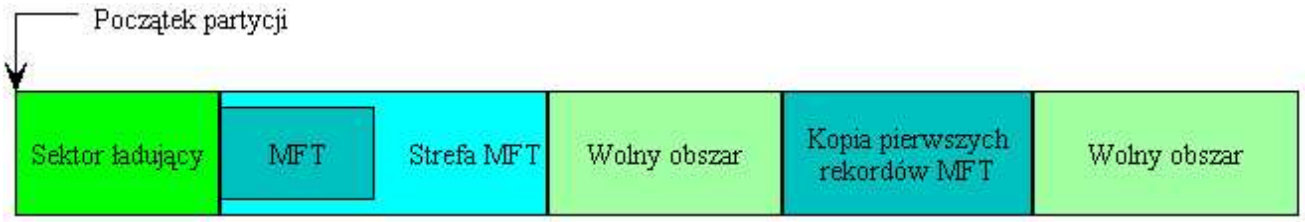
NTFS 5.0 różni się od poprzedniej wersji następującymi elementami:

- poprawiono część odpowiedzialną za bezpieczeństwo i zarządzanie prawami dostępu,
- wprowadzono punkty specjalne (ang. reparse points) - pliki i katalogi mogą mieć swoje metody, które są wywoływane przy dostępie do tych plików/katalogów,
- wprowadzono journalling (change journall) - partycje (dyski) przechowują zapis wszystkich operacji wykonywanych na przechowywanych plikach i katalogach (opisane w punkcie 3.5),
- szyfrowanie - NTFS 5.0 umożliwia szyfrowanie plików i automatyczne ich deszyfrowanie przy odczycie (EFS),
- wprowadzono zarządzanie przydziałem zasobów dyskowych (ang. disk quota)
- obsługa plików rozrzedzonych (ang. sparse files), czyli plików zawierających duże, puste obszary. Część aplikacji korzysta z bardzo dużych plików, które zawierają mało danych, a pozostała ich część jest pusta (wypełniona zerami). Pliki te często nie mogą być po prostu skompresowane, bo np. w przypadku baz danych spowodowałoby to spadek wydajności. W NTFS plik taki posiada atrybut, który nakazuje podsystemowi I/O alokowanie miejsca na dysku jedynie dla znaczących (niezerowych) danych. Aplikacja odwołująca się do takiego pliku otrzymuje dane tak, jakby plik ten był zwykłym plikiem w całości zapisanym na dysku.

Uwaga: Windows 2000 automatycznie zamienia NTFS 1.1 na NTFS 5.0 !!! Dotyczy to także dysków przenośnych.

3.3 Architektura i struktury NTFS

W NTFS wszytko jest plikiem. Wszystkie dane dotyczące partycji są przechowywane w zbiorze specjalnych plików, które są tworzone podczas tworzenia partycji NTFS. Są to tzw. **metapliki** (ang. metadata files). Jedyny wyjątek od reguły „wszystko jest plikiem" stanowi **sektor ładujący** (ang. volume boot sector), który znajduje się przed metaplikami na początku partycji.



Rysunek 1: Schemat partycji NTFS po sformatowaniu

3.3.1 Sektor ładujący - volume boot sector

Podczas tworzenia partycji NTFS w pierwszym sektorze partycji tworzony jest tzw. **sektor ładujący** (ang. volume boot sector). [1] Jego rola polega między innymi na ładowaniu systemu operacyjnego podczas uruchamiania komputera.

Składa się z dwóch podstawowych struktur:

- **BIOS Parameter Block** - na podstawie tego bloku rozpoznawana jest partycja NTFS; tu przechowywane są dane o etykiecie partycji (volume) i jej rozmiarze,
- **Volume Boot Code** - zawiera kod ładujący *NTLDR - NT loader program* - odpowiada za dalsze ładowanie systemu operacyjnego.

[1] Sektor ładujący zaczyna się w pierwszym sektorze partycji, ale może zajmować więcej niż ten jeden sektor - może zajmować nawet 16 kolejnych sektorów.

3.3.2 Główna tablica plików (MFT) i metapliki

W skrócie metapliki to pliki zawierające dane o danych. Pliki te są tworzone automatycznie podczas formatowania partycji i umieszczane na jej początku (za ewentualnym sektorem ładującym).

Centralnym elementem całego systemu jest **NTFS Master File Table** - główna tablica plików, w skrócie **MFT**. **MFT** to metaplik, który dla każdego pliku/katalogu znajdującego się na partycji (dysku) zawiera opisujący go rekord, dalej zwany rekordem bazowym. Ponieważ metapliki też są plikami, więc 16 początkowych rekordów **MFT** opisuje właśnie te pliki (w szczególności **MFT** zawiera rekord z informacją o sobie samym). Pliki te zostały wyszczególnione w tabeli 2. Pozostałe rekordy **MFT** zawieraja informacje dotyczące poszczególnych plików i katalogów na partycji (dysku).

Dla każdego utworzonego pliku lub katalogu tworzony jest odpowiedni rekord w **MFT**. Wielkość tego rekordu jest określona przez wielkość klastra, ale jest niemniejsza niż 1024 bajty i niewiększa niż 4096 bajtów.

Nazwa metapliku	**Nazwa pliku**	**Nr rekordu bazowego w MFT**	**Opis pliku**
Master File Table	$MFT	0	Zawiera jeden bazowy rekord pliku dla każdego pliku/katalogu partycji NTFS. Jeżeli informacji jest za dużo, by pomieściły się w jednym rekordzie, to alokowane są rekordy dodatkowe.
Master File Table 2	$MFTMirr	1	Kopia pierwszych czterech rekordów MFT. Jest przechowywana w środku, bądź na końcu partycji.
Log File	$LogFile	2	Dziennik transakcji partycji (jego zastosowanie zostało opisane w punkcie 3.4).
Volume Descriptor	$Volume	3	Zawiera podstawowe informacje o partycji takie, jak: nazwa, wersja NTFS, czas utworzenia, itd.
Attribute Definition Table	$AttrDef	4	Zawiera nazwy, numery i opisy atrybutów.
Root Directory	„ . ”	5	Katalog główny partycji.
Cluster Allocation Bitmap	$Bitmap	6	Bitmapa określająca, które klastry partycji są wolne, a które zajęte.
Volume Boot Code	$Boot	7	W przypadku partycji „bootowalnej” zawiera kopię kodu ładującego system operacyjny (lub wskaźnik do tego kodu).
Bad Cluster File	$BadClus	8	Zawiera listę uszkodzonych klastrów.
Quota Table	$Quota	9	Zawiera tablice z informacjami o przydziałach dyskowych użytkowników. Wykorzystywany przez NTFS w wersji 5.0.
Upper Case Table	$UpCase	10	Zawiera tablicę wykorzystywaną przy konwersji małych znaków na odpowiadające im wielkie znaki w standardzie UNICODE.
		11 - 15	Zarezerwowane dla przyszłych zastosowań.

Tablica 2: Matepliki w NTFS

Ponieważ **MFT** podczas tworzenia nowych plików/katalogów zwiększa swój rozmiar (potrzebne są nowe rekordy bazowe do opisu nowych obiektów), w celu uniknięcia fragmentacji **MFT**, na początku tworzenia partycji system przeznacza 12,5% (25%, 37,5%, 50% - konfigurowalne) pojemności partycji na tzw. strefę MFT - **MFT Zone**. Zwykłe pliki i katalogi nie będą umieszczane w tej przestrzeni, dopóki istnieje miejsce na dysku. Dopiero, gdy to miejsce się skończy pliki i katalogi bedą umieszczane w **MFT Zone** (ściślej **MFT Zone** jest zmniejszana o połowę).

W przypadku, gdy **MFT Zone** stanie się za mała dla samego **MFT**, system alokuje dodatkową pamięć na dysku. Prowadzi to jednak do fragmentacji, a w rezultacie spadku wydajności.

System przechowuje informacje o plikach i katalogach w rekordach **MFT** w postaci atrybutów (zostały opisane w następnym podrozdziale - 3.3.3).

3.3.3 Atrybuty

Do przechowywania informacji o plikach i katalogach NTFS wykorzystuje różnego rodzaju atrybuty. W przypadku NTFS 4.0 jest to 13 rodzajów atrybutów (szczegółowo opisane w tabeli 3).

Rodzaj atrybutu	**Opis**
$VOLUME_VERSION	Wersja NTFS danej partycji/dysku (wykorzystywany przez metaplik $Volume)
$VOLUME_NAME	nazwa partycji (wykorzystywany przez metaplik $Volume)
$VOLUME_INFORMATION	Dodatkowe informacje o partycji NTFS (wykorzystywany przez metaplik $Volume)
$FILE_NAME	nazwy pliku lub katalogu
$STANDARD_INFORMATION	stemple czasowe związane z utworzeniem, modyfikacją, dostępem, flagi: ukryty, systemowy, tylko do odczytu, itp.
$SECURITY_DESCRIPTOR	informacje związane z prawami dostępu (3.6), audytem, właścicielem
$DATA	właściwe dane pliku
$INDEX_ROOT	indeks plików katalogu
$INDEX_ALLOCATION	gdy indeks katalogu jest za duży, by zmieścić się w rekordzie bazowym **MFT**, atrybut ten zawiera wskaźniki do reszty indeksu
$BITMAP	opisuje zajętość klastrów

$ATTRIBUTE_LIST	"meta-atrybutźawierający wskaźniki do atrybutu, który sam jest nierezydentny [2]
$EXTENDED_ATTRIBUTE	wprowadzony dla zachowania kompatybilności z OS/2
$E_A_INFORMATION	wprowadzony dla zachowania kompatybilności z OS/2

Tablica 3: Atrybuty w NTFS

Atrybuty składają się z dwóch logicznych części: nagłówka i danych. Nagłówek przechowuje informacje o rodzaju atrybutu, ustawionych flagach oraz zawiera wskaźnik do danych danego atrybutu.

Ponieważ wielkość rekordu w **MFT** jest ograniczona, NTFS przechowuje atrybuty na dwa różne sposoby, w zależności od rozmiaru ich danych:

- rezydentny - dane tego atrybutu są przechowywane w rekordzie bazowym **MFT** pliku/katalogu (nagłówek atrybutu zawiera wskaźnik do tych danych),
- nierezydentny - dane nie mieszczą sie w rekordzie bazowym **MFT** pliku/katalogu; atrybut składa się wówczas jedynie z nagłówka i zawiera wskaźnik(i) do danych (sytuację tę obrazuje rysunek 3).

Atrybuty, takie jak nazwa pliku/katalogu, standardowe informacje, czy SD, są zawsze przechowywane w sposób rezydentny.

3.3.4 Pliki i katalogi w NTFS

NTFS nie operuje na pojedyńczych sektorach o wielkości 512 bajtów, ale łączy je w bloki zwane klastrami - pod tym względem podobny jest do FAT. NTFS określa rozmiar klastra na podstawie wielkości partycji. Dane plików (także metaplików) są przechowywane w klastrach.

NTFS stosuje podobną do FAT metodę organizacji plików - drzewo katalogów. Bazowym katalogiem jest główny katalog **root directory**, który w rzeczywistości jest metaplikiem. W ramach tego katalogu są przechowywane wskaźniki do plików i innych katalogów. Jedną z głównych różnic pomiędzy NTFS a FAT jest to, że katalogi w FAT przechowują znaczną część informacji o plikach, a pliki zawierają wyłącznie dane. W przypadku NTFS, gdzie plik jest zbiorem atrybutów, plik sam zawiera swój opis. W rezultacie katalog NTFS zawiera jedynie informacje o sobie samym, a nie plikach, które są w nim przechowywane.

[2] Ma to miejsce, gdy same wskaźniki z nagłówka danego atrybutu nie mieszczą się w rekordzie bazowym w **MFT**. Wówczas tworzone są dodatkowe rekordy w **MFT** dla danego pliku/katalogu, a atrybut $ATTRIBUTE_LIST zawiera wskaźniki do nagłówków nierezydentnego atrybutu w tych rekordach. Sytuacja ta została przedstawiona na rysunku 4.

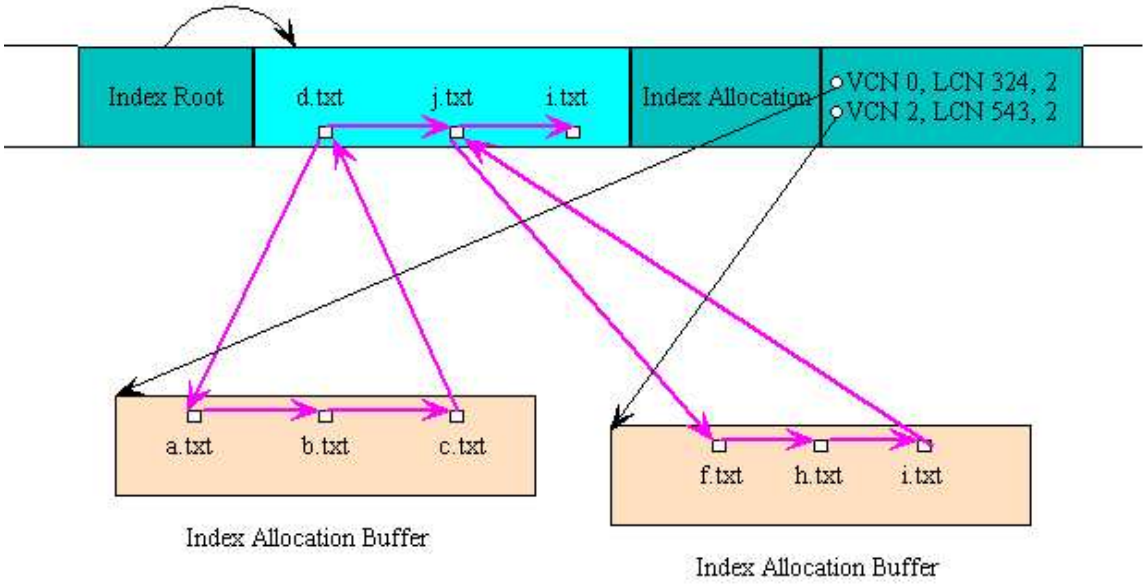
Katalogi

Zgodnie z reguła „wszystko jest plikem", katalogi w NTFS to nic innego tylko pliki. Każdy katalog ma swój rekord w **MFT** tak, jak każdy inny plik. Rekord **MFT** katalogu zawiera następujące informacje i atrybuty:

- **Nagłówek (H)** - zawiera numery porządkowe wykorzystywane przez NTFS, wskaźniki do atrybutów katalogu oraz wolnych miejsc w rekordzie. Nagłówek znajduje się w rekordzie **MFT**, ale **nie** jest atrybutem.
- **Standard information attribute (SI)** - zawiera standardowe informacje o plikach i katalogach takie, jak: czas utworzenia, modyfikacji, dostępu, czy jest tylko do odczytu, ukryty, itd.
- **File Name Attribute (FN)** - zawiera nazwę katalogu. Katalog w NTFS może mieć wiele atrybutów (FN): podstawowa nazwa (UNICODE), krótka nazwa MS-DOS, nazwa POSIX'owa.
- **Index Root Attribute** - zawiera aktualny indeks plików znajdujących się w danym katalogu. Indeks może w całości mieścić się w tym atrybucie (w **MFT**) lub częściowo (w przypadku dużych katalogów).
- **Index Allocation Attribute** - atrybut ten występuje w rekordzie bazowym katalogu w przypadku, gdy indeks katalogu jest za duży, by zmieścić się w całości w rekordzie bazowym. Atrybut ten zawiera wskaźniki do pozostałych części indeksu, umieszczonych w buforach indeksowych (ang. Index Allocation Buffer).
- **Security Descriptor (SD) Attribute** - zawiera informacje dotyczące bezpieczeństwa, które okreslają prawa dostępu do katalogu.

W celu zwiększenia wydajności w NTFS każdy katalog utrzymuje B-drzewo, które jest wykorzystywane przy wyszukiwaniu konkretnego pliku. Ma to istotny wpływ na wydajnosć w przypadku dużych katalogów. Dla porównania, w FAT informacje o plikach danego katalogu są przechowywane w klastrach, które są połaczone w nieposortowaną listę. Jest to mechanizm prosty do zaimplementowania, jednak przy każdym odwołaniu do katalogu trzeba przejrzeć cała listę, a następnie posortować przed wyświetleniem użytkownikowi. Zastosowanie B-drzew powoduje, że każdy katalog przechowuje pliki już posortowane po nazwie.

Na rysunku 2 został przedstawiony przykładowy rekord bazowy katalogu. Ponieważ zawartość katalogu jest zbyt duża, by cały indeks zmieścił się w **MFT**, rekord bazowy zawiera jedynie korzeń B-drzewa w **Index Root Attribute**. Dodatkowo NTFS umieszcza **Index Allocation Attribute**, który zawiera wskaźniki do pozostałych części indeksu. Części te znajdują się w buforach indeksowych poza **MFT**.



Rysunek 2: Przykładowy rekord bazowy katalogu w MFT

Pliki

W NTFS dane są przechowywane w plikach. Każdy plik jest zbiorem atrybutów. W szczególności dane przechowywane w pliku są jednym z jego atrybutów. W przypadku małych plików ma to ciekawą konsekwencję - cały plik jest przechowywany w rekordzie bazowym w **MFT**, co powoduje, że nie zabiera on więcej (poza tym rekordem) miejsca na dysku. Także czas dostępu ulega skróceniu - chcąc przeczytać plik, system odwołuje się do **MFT** i w odpowiednim rekordzie odnajduje cały plik, bez potrzeby ponownego odwoływania się do dysku.

Sposób przechowywania danych pliku w NTFS zależy od jego wielkosci. Z każdym plikiem są przechowywane następujące informacje (atrybuty):

- **Nagłówek (H)** - zawiera numery porządkowe wykorzystywane przez NTFS, wskaźniki

do atrybutów pliku oraz wolnych miejsc w rekordzie. Nagłówek znajduje się w rekordzie **MFT**, ale nie jest atrybutem.

- **Standard information attribute (SI)** - zawiera standardowe informacje o plikach takie, jak: czas utworzenia, modyfikacji, dostępu, czy jest tylko do odczytu, ukryty, itd.
- **File Name Attribute (FN)** - zawiera nazwę pliku. Plik w NTFS może mieć wiele atrybutów (FN): podstawowa nazwa (UNICODE), krótka nazwa MS-DOS, nazwa POSIX'owa.
- **Data Attribute (Data)** - przechowuje dane zawarte w pliku.
- **Security Descriptor (SD) Attribute** - zawiera informacje dotyczące bezpieczeństwa, które okreslają prawa dostępu do pliku.

Jeśli plik jest mały (wszystkie atrybuty mieszczą się w rekordzie **MFT**), to wówczas plik jest przechowywany w sposób rezydentny. W przeciwnym przypadku NTFS wykonuje następujący ciąg operacji:

1. próbuje umieścić cały plik w rekordzie bazowym w **MFT**,
2. jesli plik się nie mieści w rekordzie bazowym, atrybut danych (Data) jest umieszczany w sposób nierezydentny, tzn. nagłówek atrybutu danych zawiera wskaźniki do tzw. **data runs** (extents) - są to spójne obszary na dysku zajmowane przez dane pliku. Obszary te znajdują się poza **MFT**,
3. plik może być jednak tak duży, że same wskaźniki nie mieszczą się w rekordzie **MFT**. W takim przypadku lista wskaźników jest przechowywana w sposób nierezydentny. Atrybut **data** zawiera tylko część wskaźników do **data runs**. W rekordzie bazowym tego pliku umieszczany jest dodatkowy atrybut **Attribute_list**, który zawiera wskaźniki do nagłówków atrybutu **data** umieszczonych w dodatkowych rekordach **MFT** (plik może posiadać wiecej niż jeden rekord w **MFT**). Nagłówki te zawierają pozostałą część wskaźników do **data runs**. Sytuacja ta została zobrazowana na rysunku 4

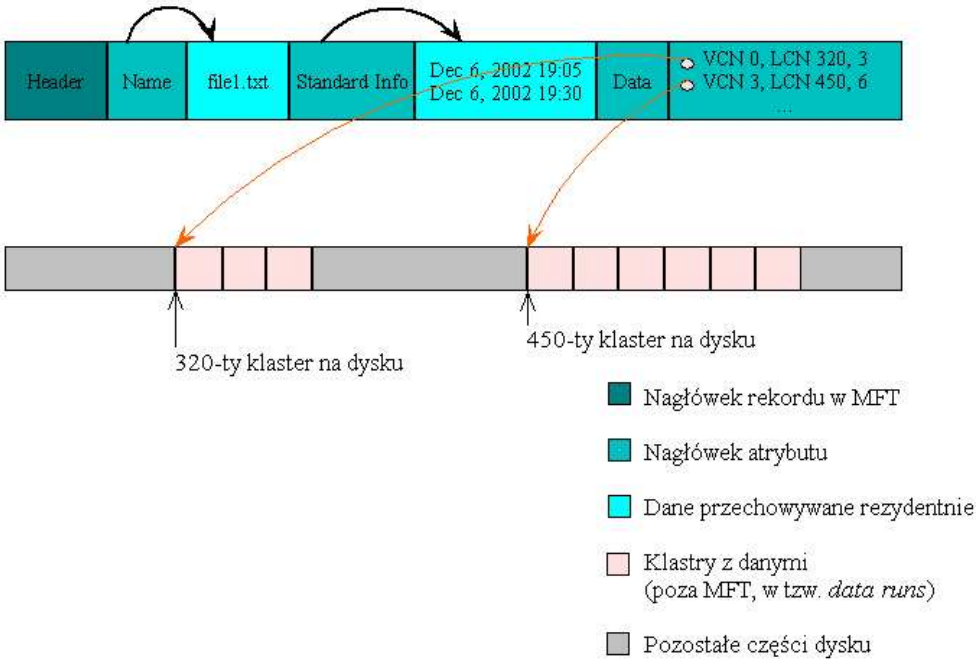
Wskaźnik do **data run** zawiera trzy wielkości:

- numer VCN (ang. virtual cluster number) - określa numer klastra w ramach pliku; ten klaster występuje jako pierwszy we wskazywanym **data run**,
- numer LCN (ang. logical cluster number) - określa położenie początku **data run** na dysku,
- długość - określa długość w klastrach wskazywanego **data run**.

Użycie schematu „wskaźnik+długość" powoduje, że przy odczytywaniu określonego klastra pliku, nie jest konieczne przeczytanie każdego poprzedniego.

Jedynym ograniczeniem na wielkość pliku w NTFS jest rozmiar dysku (partycji) pomniejszony o **MFT**.

Nazwa pliku może mieć długość do 255 znaków. Nazwa nie może zawierać znaków:

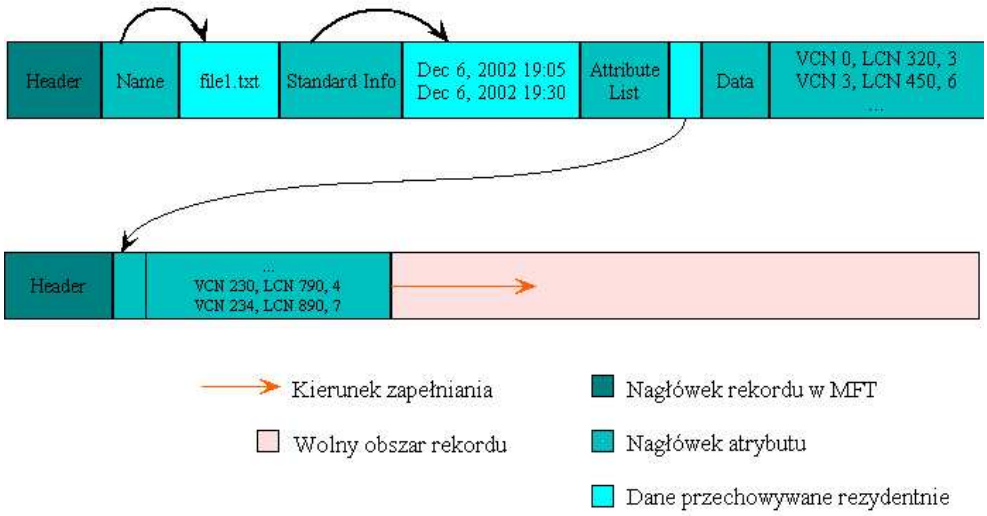


Rysunek 3: Rekord bazowy dużego pliku w MFT

? "/ { } < > * | :

Może zawierać zarówno duże, jak i małe litery, ale przy odwoływaniu się do plików wielkość liter nie ma znaczenia (nazwy: plik.txt, Plik.txt, pLik.txt odnoszą się do tego samego pliku).

Wszystkie nazwy plików w NTFS są przechowywane w formacie UNICODE - 16-bitowa reprezantacja znaków. W przypadku, gdy nazwa pliku jest dłuższa niż 8+3 NTFS tworzy alias w tym formacie (zachowanie kompatybilności ze starszymi aplikacjami). Plik w NTFS może posiadać wiele atrybutów nazw . Np. NTFS pozwala na tworzenie hard linków, które są przechowywane jako oddzielne atrybuty nazw w rekordzie bazowym w **MFT** (zachowanie standardu POSIX).



Rysunek 4: Rekord bazowy bardzo dużego pliku w MFT

3.3.5 Partycje i ich rozmiary

NTFS umożliwia tworzenie bardzo dużych partycji - maksymalny rozmiar partycji to

$$2^{64} = 18,446,744,073,709,551,616$$

bajtów.

3.4 Mechanizm transakcji w NTFS

Często operacja zapisu na dysku wymaga modyfikacji wielu struktur na nim zapisanych. Modyfikacje te muszą odbyć się w sposób atomowy. W NTFS każda operacja modyfikacji w systemie traktowana jest jako transakcja. System wykonuje podoperacje transakcji jako integralną całość.

Aby zagwarantować zakończenie lub wycofanie transakcji, NTFS zapisuje jej podoperacje w pliku rejestru transakcji. Po zapisaniu całej transakcji w rejestrze, NTFS wykonuje jej podoperacje działając na buforze partycji (dysku). Po aktualizacji bufora NTFS potwierdza transakcje zapisując w rejestrze (ang. commit), że została zakończona.

Po potwierdzeniu transakcji NTFS gwarantuje, że zostanie ona w całości zapisana na dysku. Podczas operacji odzyskiwania NTFS powtarza każdą potwierdzoną transakcję, którą znajdzie w pliku rejestru. Następnie NTFS wyszukuje w pliku rejestru transakcje, które nie zostały potwierdzone do momentu zajścia awarii systemu, a następnie wycofuje wszystkie ich podoperacje (ang. roll back).

Rozmiar pliku rejestru transakcji (dziennika transakcji) zależy od rozmiaru partycji i może sięgać do 4 MB. NTFS cyklicznie (co ok. 5 sekund) zapisuje dane z bufora na dysku i oczyszcza rejestr transakcji.

3.5 Dziennik zmian (ang. journall)

Dziennik zmian to nowa funkcja NTFS 5.0 (wykorzystywana w Windows 2000). Udostępnia ona rejestr zmian dokonywanych w plikach partycji. NTFS korzysta z tego dziennika w celu śledzenia informacji o dodawanych, modyfikowanych lub usuwanych plikach.

Z dziennika zmian mogą korzystać różnego rodzaju aplikacje w celu efektywniejszego uzyskiwania informacji o ostatnio zmodyfikowanych plikach.

Gdy dowolny plik jest tworzony, modyfikowany lub usuwany, NTFS dodaje odpowiedni rekord do dziennika zmian danej partycji. Każdy rekord w dzienniku 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.

3.6 Prawa dostępu i bezpieczeństwo

NTFS w porównaniu z poprzednimi systemami plików (np. FAT) sprawuje dużo większą kontrolę nad tym kto i jakiego typu operacje może wykonywać na różnych danych w ramach tego systemu plików. W NTFS prawa są nadawane użytkownikom lub grupom użytkowników. Wszystko zaczyna się od **MFT**. Każdy rekord bazowy pliku/katalogu zawiera atrybut SD (security descriptor), okreslający prawa dostępu do danego obiektu. Jedną z najważniejszych części tego atrybutu stanowi zbiór list **ACL** (Access Control Lists) okreslających, który użytkownik i w jakim przypadku ma prawo dostępu do obiektu. Każdy obiekt na partycji NTFS posiada dwa rodzaje list **ACL**:

- **System Access Control List (SACL)** - tą listą zarządza system. Służy do audytu prób dostępu do obiektu.

- **Discretionary Access Control List (DACL)** - lista ta zawiera prawa, które okreslają, którzy użytkownicy i które grupy użytkowników mają prawo wykonywać poszczególne operacje na obiektach.

Każda pozycja list ACL, tzw. **ACE** (Access Control Entry) zawiera identyfikator okreslający użytkownika bądź grupę, do której ta pozycja się odnosi. Dalej **ACE** zawiera informacje o prawach.

Użytkownik może należeć do wielu grup, których prawa dostępu do danego obiektu są sprzeczne ze sobą. W takim przypadku, przy próbie dostępu do danego obiektu uruchamiany jest proces „permission resolution" rozstrzygający konflikt na podstawie priorytetów praw. Np. w Windows NT:

- jeśli choć jedna z grup, do których należy użytkownik ma prawo, to użytkownik ma prawo,
- w przypadku, gdy choć jedna z grup, do których należy użytkownik ma ustowione „No Access" - użytkonik nie ma dostępu (należy uważać przy nadawaniu No Access).

W Windows 2000 jest to bardziej skomplikowane, ale oparte na tej samej zasadzie.

W Windows NT jest sześć rodzajów praw dla obiektów NTFS. Są to tzw. „specjalne prawa dostępu" (ang. special permissions), w odróżnieniu od „standardowych grup praw dostępu", używanych na wyższym poziomie.

Prawo	Skrót	W przypadku pliku	W przypadku katalogu
Read	R	zezwala na odczyt zawartości pliku	zezwala na odczyt zawartości katalogu
Write	W	zezwala na modyfikację zawartości pliku	zezwala na zmianę zawartości katalogu (tworzenie podkatalogów, plików)
Execute	X	zezwala na wykonywanie programu	zezwala na przechodzenie struktury podkatalogów
Delete	D	zezwala na usuwanie pliku	zezwala na usuwanie katalogu
Change Permissions	P	zezwala na modyfikowanie ustawień praw pliku	zezwala na modyfikowanie ustawień praw katalogu
Take Ownership	O	zezwala na stanie się współwłaścicielem pliku	zezwala na stanie się współwłaścicielem katalogu

Tablica 4: Specjalne prawa dostępu w NTFS

3.7 Inne ciekawe elementy systemu NTFS

3.7.1 Audyt

Jeśli audyt jest włączony, system może przechowywać zapis o określonych zdarzeniach, które zaszły w systemie. Jeśli takie zdarzenie zajdzie, informacja o nim (rodzaj zdarzenia, data, czas, użytkownik, itd...) jest umieszczana w specjalnym logu, który później może być odczytywany przez administratorów systemu.

3.7.2 Przemieszczanie klastrów (ang. Dynamic bad cluster remapping)

NTFS ma możliwosć (sterownik FTDISK) automatycznego przeniesienia danych z uszkodzonego klastra i zaznaczenia go jako uszkodzony. Taka możliwość istniała w FAT, ale trzeba było ręcznie uruchomić ScanDisc.

Dodatkowo sterownik FTDISK po zapisie danych do klastra próbuje je z niego odczytać, minimalizując w ten sposób ryzyko utraty danych.

3.8 Strony WWW o NTFS

- http://www.pcguide.com/ref/hdd/file/ntfs/index.htm
- http://linux-ntfs.sourceforge.net/ntfs/index.html
- http://www.winntmag.com/Articles/Index.cfm?IssueID=27&ArticleID=3455
- http://www.microsoft.com/poland/windows2000/win2000prof/rozdzial_17/roz_17_4.asp
- http://www.qvctc.commnet.edu/classes/csc277/ntfs.html

4 ISO-9660

4.1 Historia

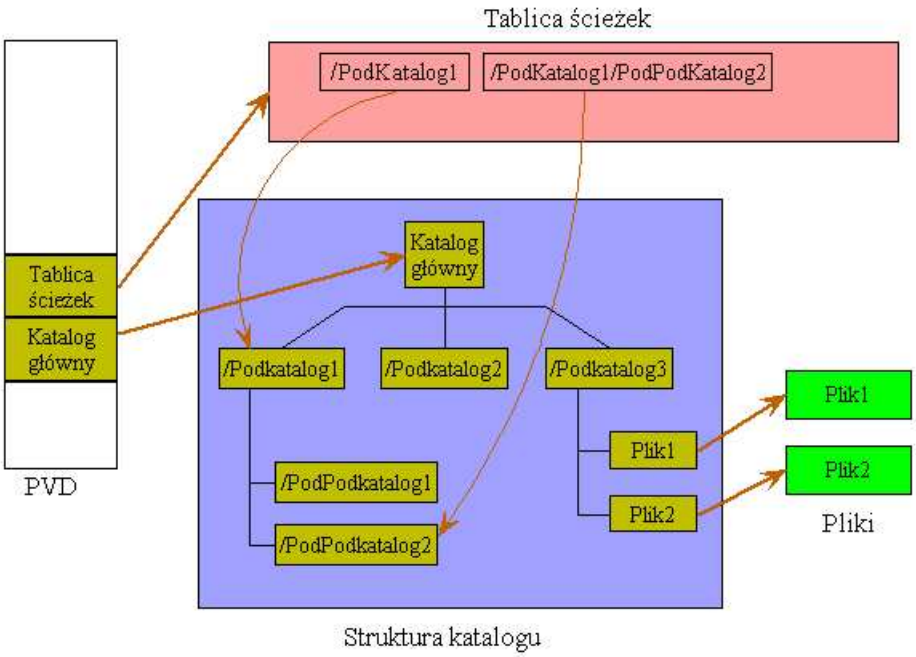
- Przed powstaniem ISO-9660 na producentach oprogramowania spoczywał obowiązek dostarczania odpowiednich sterowników do napędów CD-ROM dla swoich programów. To powodowało, że wiekszość zasobów była zużywana na prace oderwane od głównej działalności.
- Wprowadzenie standardu pozwoliłoby na zmniejszenie kosztów związanych z produkcją oprogramowania.
- Zostaje utworzony komitet High Sierra, którego zadaniem jest opracowanie standardowego systemu plików dla płyt CD-ROM.
- W maju 1986 roku standard High Sierra został zatwierdzony przez ISO (International Standards Organization).
- Firmy związane z opracowywaniem standardu High Sierra kontynuowały prace i zaimplementowały ten standard.
- W kwietniu 1988 roku został opublikowany ISO-9660. ISO wprowadziła pewne zmiany, co doprowadziło do powstania nowego standardu, nie w pełni kompatybilnego ze standardem High Sierra.

4.2 Przegląd struktur ISO-9660

- Płyta CD-ROM jest podzielona na 2048-bajtowe sektory fizyczne.
- Oprócz tego jest podzielona na bloki logiczne. Blok logiczny to najmniejsza porcja danych, jaka może być zapisana/odczytana na/z płyty. Na ogół rozmiar bloku logicznego jest równy wielkości sektora - 2048 bajtów.

Stuktury danych ISO-9660 można podzielić na trzy główne kategorie:

- Volume Descriptors - deskryptory dysku,
- Directory Structurs - struktury katalogów,
- Path Tables - tablice ścieżek.



Rysunek 5: Struktury ISO-9660

4.3 Deskrytory dysku

Standard ISO-9660 definiuje cztery typy deskryptorów dysku:

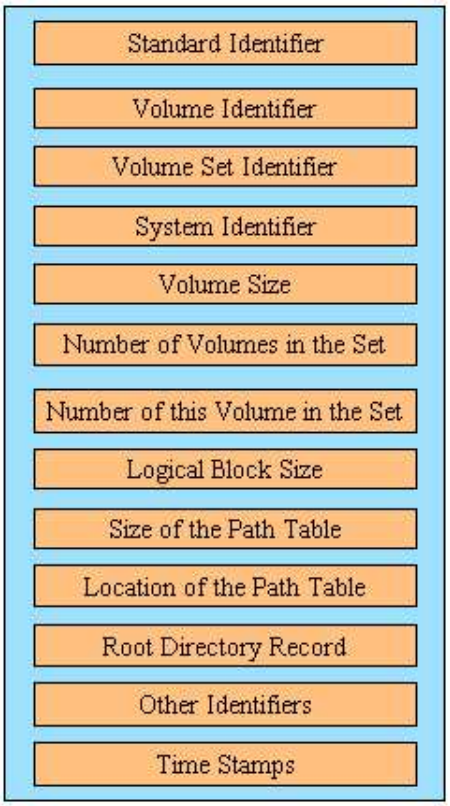
- **Primary Volume Descriptor** - podstawowy deskryptor dysku (najczęściej używany - pozostałe trzy deskryptory są opcjonalne),
- Boot Record - rekord startowy - wykorzystywany do wykonywania operacji inicjacyjnych, zanim użytkownik uzyska dostęp do dysku,
- Supplementary (secondary) Volume Descriptor - drugi deskryptor dysku - wykorzystywany przez systemy nie rozpoznające zestawu znaków ISO 646 (dodatkowy zestaw znaków),
- Volume Partition Descriptor - deskryptor partycji - wykorzystywany przy podziale dysku na logiczne partycje.

Deskryptory dysku zaczynają sie od 16-go sektora logicznego dysku.

Każdy deskryptor dysku ma rozmiar jednego sektora - 2048 bajtów.

Tablica deskryptorów kończy się tzw. Volume Descriptor Terminator.

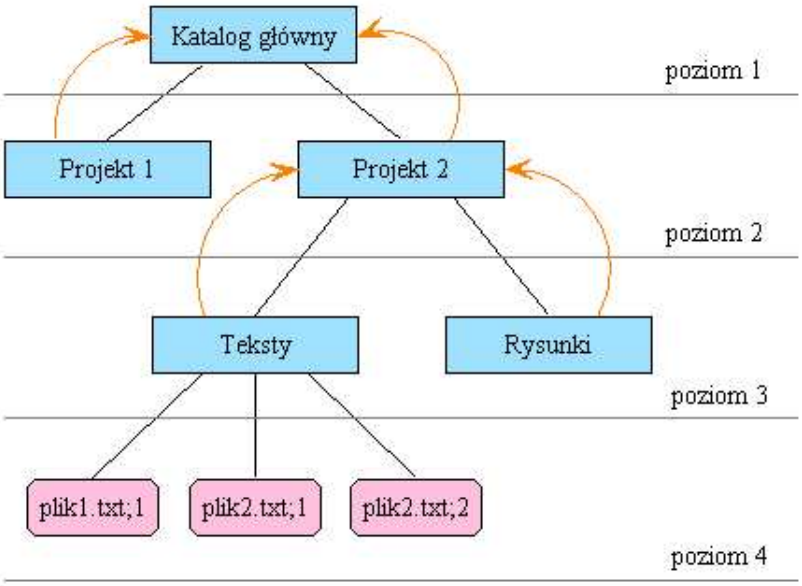
4.4 Primary Volume Descriptor



Rysunek 6: Schemat PVD

4.5 Struktura katalogów

- Struktura katalogów w ISO-9660 jest strukturą hierarchiczną, podobną do tej stosowanej we współczesnych systemach plików.
- ISO-9660 ogranicza wysokość drzewa katalogów do 8-miu poziomów.
- Długość bezwzględnej ścieżki określającej położenie pliku nie może przekroczyć 255-ciu znaków.
- Katalog w ISO-9660 jest plikiem zawierającym zbiór rekordów katalogów. Każdy rekord katalogu zawiera opis pliku, bądź podkatalogu. Każdy katalog posiada rekord opisujący swój nadkatalog. Nadkatalog zawiera rekord katalogu z informacjami o tym katalogu.
- Nadkatalogiem katalogu głównego jest on sam.



Rysunek 7: Hierarchia katalogów

4.6 Nazwy plików

Nazwa plików to inaczej identyfikator pliku.

Nazwa składa się z pięciu części:

1 Nazwa pliku	2 Separator 1	3 Rozszerzenie	4 Separator 4	5 Numer wersji pliku
d-znaki	.	*d-znaki*	;	*liczba z zakresu 1 do 32767*
plik	.	txt	;	1

Identyfiaktor pliku musi spełniać następujące warunki:

- jeśli nazwa pliku nie istnieje, rozszerzenie musi posiadać co najmniej jeden znak,
- jeśli rozszerzenie jest puste, to nazwa musi się składać przynajmniej z jednego znaku,
- suma długości nazwy i rozszerzenia nie może przekroczyć 30-stu znaków.

Nazwa katalogu składa się wyłącznie z części nazwa.

Rekordy w katalogu są posortowane według identyfikatora pliku.

4.7 Tablica ścieżek

- Tablica ścieżek umożliwia systemowi operacyjnemu szybszy dostęp do katalogów - bez konieczności przechodzenia drzewa.
- Dla każdego katalogu (poza katalogiem głównym) przechowywany jest rekord zawierający między innymi:
  - nazwę katalogu,
  - numer rekordu nadkatalogu w tablicy ścieżek,
  - położenia pierwszego logicznego bloku katalogu na dysku.

4.8 Poziomy wymiany (ang. levels of interchange)

Standard ISO-9660 określa 3 poziomy wymiany:

- Poziom 1
  - plik jest zapisany w ciągłym obszarze dysku;
  - nazwa pliku nie może zawierać więcej niż 8 d-znaków lub d1-znaków ;
  - rozszerzenie pliku nie może zawierać więcej niż 3 d-znaki lub d1-znaki ;
  - nazwa katalogu nie może być dłuższa niż 8 d-znaków lub d1 - znaków ;
- Poziom2
  - plik jest zapisany w ciągłym obszarze dysku;
- Poziom3
  - ograniczenia wynikające ze standardu ISO-9660;

4.9 Rozszerzenia ISO-9660

ISO-9660 współpracuje z wieloma systemami operacyjnymi. Jednak w niektórych przypadkach okazało się, że korzytanie z niego jest niewygodne, bądź wręcz niemożliwe.

Wprowadzono rozszerzenia do standardu ISO-9660:

- **Apple ISO 9660** - rozszerznie dla systemu operacyjnego komputerów Macintosh - MacOS, związane ze specyficznymi danymi niezbędnymi dla graficznego interfejsu użytkownika.

- **The Rock Ridge Proposals**

  Rock Rigdge to grupa firm, które w 1990 roku zajmowały się rozwiazaniem problemów związanych z używaniem ISO-9660 z UNIX'owymi systemami operacyjnymi. Problemy te były związane z długimi nazwami, małymi literami w nazwach, i wysokością drzewa katalogów.

  Powstały rozszerzenia:

  - **Rock Ridge System Use Sharing Protocol (SUSP)** ,
  - **Rock Ridge Interchange Protocol (RRIP)** .

- **Updatable ISO 9660** - umożliwia dogrywanie danych w tzw. sesjach.
- **ECMA 168** - bardziej elastyczne rozwiązania niż w Updatable ISO 9660, ale bardziej skomplikowane .
- **Joliet** - stworzona przez Microsoft wersja ISO-9660, umożliwiająca nagrywanie plików o nazwie o długości do 64 znaków.

4.10 Strony WWW o ISO-9660

- http://www.singlix.com/trdos/IntroductionToISO9660.pdf
- http://www.wikipedia.org/wiki/ISO_9660
- http://www.alumni.caltech.edu/p̃je/iso9660.html
- http://www.angelfire.com/pa2/mpx/iso9660.html

5 Wstęp do „nowoczesnych” lokalnych systemów plików w Linux’ie

5.1 Systemy plików z journalingiem

Najważniejsze pojęcia :

Blok dyskowy - logiczny element podziału dysku, stały w obrębie danego systemu plików

i-węzeł (ang. *i-node*, *index node*) - pozycja w tablicy i-węzłów, przechowuje informacje potrzebne do odnalezienia pliku lub katalogu (nazwę pliku, adresy bloków dyskowych zawierających dane itd.) oraz dodatkowe informacje wykorzystywane przez system operacyjny (tak zwane metadane)

metadane (ang. *metadata*) - dodatkowe informacje dotyczące pliku lub katalogu, np. typ pliku, liczba dowiązań do niego, atrybuty, znaczniki czasowe

superblok - obszar systemu plików (partycji), w którym przechowywane są metadane dotyczące całej partycji, takie jak liczba wolnych bloków oraz i-węzłów czy rozmiar systemu plików

kronikowanie (ang. *journaling*) - technika polegająca na zapisywaniu dokonywanych w systemie plików zmian zamiast uaktualniania konkretnych danych; kronikowanie wydatnie wpływa na zwiększenie bezpieczeństwa danych przechowywanych na dysku

5.2 Zaawansowane systemy plików - linuxowe projekty open source

System Plików	**Projektami kieruje**	**Maks. rozmiar FS (TB)**	**Rozmiar bloku (B)**	**Maks. rozmiar pliku (TB)**
XFS	SGI	18mld	512-65536	9mld
JFS	IBM	32mln	512/1024/2048/4096	4mln
ReiserFS	Hans Reiser	16	1024-65536	16

5.3 Nowe systemy plików, a drzewa B+

System Plików	**Wykaz wolnych plików**	**Wykaz bloków pliku**	**Wykaz i-węzłów**	**Struktura katalogów**
XFS	B+-drzewo ekstentów	B+-drzewo ekstentów	B+-drzewo	B+-drzewo
JFS	podział mapy bitowej bloków na mniejsze fragmenty i indeksowanie drzewem binarnym (wg. położenia)	B+-drzewo ekstentów	B+-drzewo z liśćmi o stałym rozmiarze 32 i-węzłów	B+-drzewo

System Plików	Wykaz wolnych plików	Wykaz bloków pliku	Wykaz i-węzłów	Struktura katalogów
Reiser FS	mapa bitowa	jedno B+-drzewo obejmujące cały system plików (adresowanie pojedyńczymi blokami zamiast ekstentów)	jedno B+-drzewo obejmujące cały system plików	jedno B+-drzewo obejmujące cały system plików

Problemy wydajnościowe spowodowały, że zostało uruchomionych kilka projektów nowoczesnych systemów plików w ramach prac nad Linuksem (oraz przeniesiono kilka z innych systemów).

Aby zrozumieć sposób działania nowoczesnych systemów plików, trzeba przyjrzeć się dotychczas stosowanym architekturom i wynikającymi z nich trudnościami oraz zastanowić się nad możliwymi rozwiązaniami.

W systemach UFS i ext2fs analiza dostępności bloków dyskowych odbywa się sekwencyjnie. Wolne bloki znajduje się, przeglądając po kolei zawartości map bitowych bloków w kolejnych grupach bloków. W skrajnym przypadku może to oznaczać konieczność przejrzenia wszystkich map bitowcy. Czyli:
131 071 grup bloków po 32 768 pozycji każda (16 TB = 131 072 * 32 768 * 4 kB = 131 072 * 128 MB).
Proponowanym rozwiązaniem tego problemu jest zastąpienie sztywnej struktury map bitowych indeksami (położenia oraz liczby wolnych bloków) w formie B+-drzew, które są często stosowane w systemach baz danych. W opisywanych w tej częsci pracy nowoczesnych systemach plików używane są B+-drzewa.

Ponieważ indeksowanie obejmuje również rozmiar całych ciągłych sekwencji wolnych bloków (ekstentów, od ang. *extents*), korzyści jest wiele - poza szybszym znalezieniem pierwszego wolnego bloku, można stosunkowo szybko znaleźć wolny blok, który byłby jednocześnie najlepszym ze względu na przewidywany przyrost wielkości pliku. W przypadku dużej ilości dużych plików (np. bazy danych) ekstenty pozwalają zaoszczędzić miejsce na dysku. Zamiast zapisywania informacji o każdym z małych bloków, można zapisywać obszerniejsze (w porównaniu do informacji o bloku) informacje o znacznie większych jednostkach alokacji.

Alokacja i-węzłów jest problemem podobnym do alokacji bloków. Oczywiście dotyczy on tylko systemów plików posługujących się klasycznym rodzajem i-węzła, czyli wydzieloną fizycznie strukturą w systemie plików.
Struktura grupy bloków narzuca ograniczenia dotyczące bloków:

- liniowa mapa bitowa bloków
- stała, identyczna liczba bloków we wszystkich grupach
- stały, taki sam rozmiar bloku we wszystkich grupach

Struktura grupy bloków narzuca również ograniczenia dotyczące i-węzłów:

- liniowa mapa bitowa i-węzłów
- stała, identyczna liczba i-węzłów we wszystkich grupach

Oba problemy są rozrozwiązywane - przez zastąpienie sztywnych struktur B+-drzewami.

Problemem podczas pracy z ext2fs'em bywa również obsługa dużej ilości plików w obrębie jednego katalogu. W ext2fs'ie katalog jest zaimplementowany w formie jednokierunkowej listy, zatem operacja wyszukania jakiegokolwiek pliku ma charakter liniowy. Przy dużych katalogach problem narasta. Tu także klasycznym rozwiązaniem są indeksy w postaci B+-drzew.

Poza technikami rozwiązującymi dotychczasowe problemy wydajnościowe i pojemnościowe nowe systemy plików implementują szereg innych, interesujących mechanizmów. Jednym z nich jest śledzenie wykorzystania przestrzeni dyskowej przez plik i alokacja tylko niezbędnych obszarów. Przykładowo ciągły zapis odpowiednio dużej (opłacalnej w odniesieniu do pojemności) ilości tych samych danych (zer, jedynek) do pliku może być zastąpiony zapisem specjalnego „skrótu". Co prawda technika ta przypomina kompresję pliku, ale ma od niej znacznie mniejszą złożoność obliczeniową. W praktyce można nawet pominąć taki skrót, gdy założymy, że plik skracamy tylko o ciągi zer, a kolejne pozycje niezerowe mają wynikać ze zdefiniowanych pozostałych fragmentów pliku. Nie jest to często stosowany mechanizm (nie wymagany od XFS'a).

Taki scenariusz dotyczy nowych systemów plików dla Linuksa. Kolejne fragmenty plików są tam definiowane przez ekstenty (lub bloki - w przypadku nieco spóźnionego pod tym względem systemu ReiserFS'a (z powodów które zostaną wyjaśnione przy omawianiu go)), które stanowią nie tylko jednostki alokacji o zmiennej wielkości, ale ich deskryptory (zapisane w B+-drzewach) dają także informację o tym, jakiego fragmentu pliku dotyczą. Po prostu zerowe zapisy (ekstenty składające się z samych zer) nie są przenoszone na fizyczny nośnik i nie marnują zasobów dyskowych i czasu procesora.

6 Dziennikowy system plików JFS

6.1 JFS - ogólnie

JFS Journaling File System

IBM luty 2000 rok udostępnienie publicznie, strona:
http://www-4.ibm.com/software/developer/library/jfs.html

JFS stawia na:

Niezawodność

Bezpieczeństwo

W tej częsci pracy zosatną omówione:

- Zalety JFS'a i jego „inność"
- Ekstenty
- Realizcja i struktury danych JFS'a

W tej części prezentacji opowiem o dziennikowym systemi plików JFS'ie w Linux'ie. Jest to system aktualizujący dane za pomocą transakcji. JFS zapewnia superblok oraz wywołania systemowe służące do wykonywania operacji na plikach i wymagane przez warstwę wirtualnego systemu plików Linux'a (VFS'a).

Choć zaprojektowano go pod kątem dużej przepustowości i niezawodności niezbędnej w ukierunkowanych na transakcje, wysoko wydajnych serwerach, nadaje się również do wykorzystania w konfiguracjach klienckich, w których pożądana jest duża wydajność i niezawodność.

Zacznę od omówienia zalet dziennikowych systemów plików. Następnie przedstawię krótko schemat alokacji opartej na ekstentach. Później opowiem ogólnie o realizacji oraz strukturach danych JFS'a. **/linux/include** powinno być dodane do wszystkich ścieżek, które używamy w tej części prezentacji.

6.2 Zalety dziennikowych systemów plików

- tradycyjny system plików *ext2fs* posiada wiele wad
- Szybki restart systemu po awarii
- Większa spójność strukturalna
- Bezpieczeństwo danych

Dzięki tym zaletom JFS stał się kluczową technologią w internetowych systemach plików.

Zalety dziennikowych systemów plików zaprezentowane zostaną na przykładzie JFS'a.
Wady *ext2fs'a*:

- *ext2fs* jest statyczny - nie śledzi zmian, aby upewnić się, że aktualizacje na dysku odbywają się w bezpieczny sposób.
- asynchroniczny zapis metadanych; metadane są zapisywane z opóźnieniem w stosunku do zawartości pliku; zatem co się stanie gdy sprzęt (np.: zasilanie) ulegnie awarii?!
- fsck (ang. *file system checker* - tester systemu plików) działa bardzo długo dla np.: *ext2fs* (rzędu kilku godzin, a dla dużych nośników danych nawet kilka dni), a bardzo szybko dla JFS'a (kilka sekund/minut)
- w najgorszym przypadku może wystąpić utrata lub zagubienie danych

Dzięki technikom wywodzącym się z baz danych JFS może w ciągu kilku sekund lub minut odzyskać spójność systemu plików (po awarii). W niedziennikowych może to zająć nawet kilka godzin. Dzięki zastosowaniu dzienników można uniknąć badania metadanych systemu.

6.3 Plik, Katalogi i Dzienniki w JFS'ie

- Plik
- Katalog
- Dziennik
- JFS obsługuje 2 organizcje katalogów
- Obsługa gęstych lub rorzedzonych plików

Plik jest reprezentowany przez i-węzeł zawierający korzen B+-drzewa, który opisuje ekstenty z danymi urzytkownika.

Katlog JFS to plik z metadanymi podlegający rejestrowaniu w dzienniku. Różnicą w stosunku do pliku jest indeksowanie B+-drzewa nazwami obiktów znajdującymi się w katalogu.

Dziennik jest prowadzony w każdym agregacie (opisany później) i służy do rejestrowania operacji na metadanych. Zatem jeśli w operacji zapisu jest błąd (ale operacja zwróci kod powodzenia) to JFS nie może o tym wiedzieć.

Organizacja katalogów w JFS zależy od ilości obiektów w katalogu:

- dla małej - treść umieszczna jest w i-węźle
- dla dużej - „normalnie" w B+-drzewie (katalogu)

JFS obsługuje gęste, lub rozrzedzone plik (ale tylko jedne w danym systemie).

- gęste -> długi ciąg zer lub jedynek jest zapisywany fizycznie na dysku (na przykład ktoś na początku przesówa się na dużą długość i tam coś zapisuje)
- rozrzedzone -> te ciągi są kompresowane (informacja o nich nie jest trzymana fizycznie na dysku - ich treść)

6.4 Cechy odróżniające JFS od innych systemów plików

- Dziennik operacji jest integralną częścią systemu
- Wewnętrzne limity JFS'a (dość duże, ale XFS ma większe)
- Nośniki wymienne (JFS ich „nie lubi")
- Rozmiar systemu plików
- Rozmiar pliku
- dynamiczna alokacja i-wezłów
- B+-drzewa - struktury adresowania

JFS'a zaprojektowano tak, aby dziennik operacji był integralną częścią systemu, a nie dodatkiem do już istniejącego systemu.

JFS to w pełni 64-bitowy system plików. Zatem musi obsługiwać duże pliki i partycje.

JFS nie pozwala używać dyskietek jako bazowego urządzenia systemu.

Rozmiar systemu plików:

- minimalny wynosi 16 mb
- 512 tb (tera) < maksymalny < 4 pb (peta)

Rozmiar pliku jest ograniczony przez VFS'a (jego budowę np.: 32-bitową).

Drzewo B+ posortowane wedug nazwy poprawia wydajnośc lokalizowania konkretnego wpisu katalogowego.

Pamięć na i-węzły jest przydzielana dynamicznie, przez co nie ma ograniczenia na ilość plików.

6.5 Alokacja oparta na ekstentach

JFS (IBM), XFS (SGI) i VxFs (Veritas) - systemy plików oparte na ekstentach

ekstent - ang. *extent*, jet opisany przez: logiczne przesunięcie, długość i adres fizyczny

drzewo B+ - ang. *B-tree*, struktura adresowa (używana w nowych systemach plików)

metadane jest ich mała liczba (ze względu na to, że opisują ekstenty)

sekwencyjny i losowy odczyt

Ekstent to ciągła sekwencja wielu bloków przydzielana jako jednostka i opisana przez trzy parametry: logiczne przesunięcie, długość i adres fizyczny. *Ekstent* jest opisany przez deskryptor alokacji systemu, który znajduje się w pliku **/linux/JFS/JFS_xtree.h** - structure xad (opisująca ekstent).
Struktura xad to wartości w drzewie B+. Wpisy są posortowane wedug przesunięcia w tej strukturze.

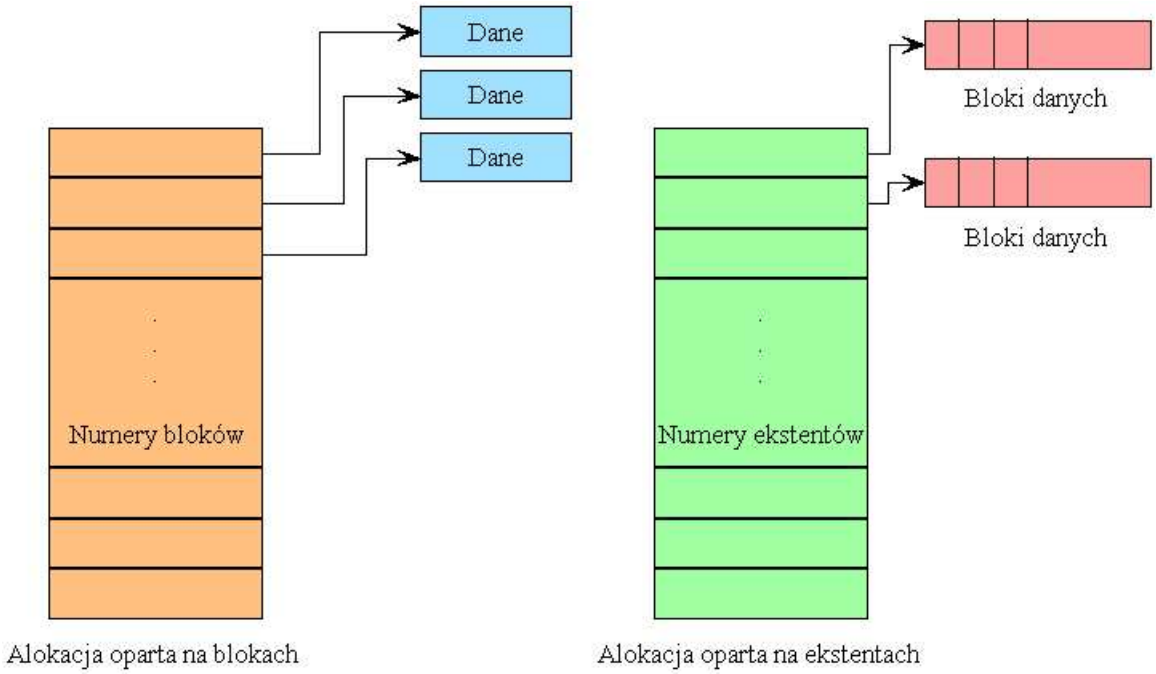
Struktura adresowa to B+ drzewo deskryptorów ekstentów (trójkami parametrów, zakorzenione w i-węźle) i indeksowane według logicznego przesunięcia pliku. O B+ drzewach było na wykładzie z ASD. Ale należy pamiętać, że:

- wskaźniki do danych znajdują się tylko w liściach
- zwykle rozmiar węzła zdefiniowany jest przez rozmiar strony
- są dobre bo mają płaską strukturę
- problem jest jednoczesne wykonywanie różnych operacji: wyszukiwania, wstawiania i usównania
- np.: wyszukiwanie i usównanie - istnieje 50% prawdopodobieństwo, że powstanie impas. Zatem pararelizm operacji na drzewach jest zły w systemach plików opartych na ekstentach

W alokacji blokowej każdy logiczny blok pliku wymaga numeru adresowego -> zatem pliki zawierają dużo metadanych. Plik składający się z kilku dużych ekstentów nie wymaga dużo metadanych.

Odczyt sekwencyjny jest dobry przy alokacji opartej na ekstentach, gdyż czytamy tylko kilka ekstentów. Odczyt losowy jest mało wydajny, tzn. wydajność działania jest zbliżona do wydajności odczytu przy alokacji opartej na blokach

6.6 Alokacja oparta na ekstentach - rysunek



6.7 Agregat, Superblok i I-węzeł

Agregat -> nośnik danych

Partycja -> montowany system (zbiór) plików (nie fizyczny)

Superblok (agregatu) zawiera informacje dotyczące agregatu

podstawowy i wtórny superblok

/linux/JFS/JFS_superblock.h -> structure JFS_superblock (struktura opisująca superblok)

I-węzeł (512 bajtów) zawiera 4 podstawowe zbiory informacji

/linux/JFS/JFS/dinode.h -> structure dinode

Agregat jest tablicą bloków dyskowych alokowanych zgodnie ze specyficznym formatem. Zawiera m.in.:

- Superblok, który zawiera daną partycję
- Mapę alokacji
- Dwie tablice I-węzłów opisujące dane agregatu

Dla każdego agregatu ustala się rozmiar bloku:

- 512 bajtów
- 1024 bajtów
- 2048 bajtów
- 4096 bajtów

Partycja zawiera:

- Tablice i-wezłół partycji -> (są tam również informacje o samej partycji) i-wezły plików
- Mapę alokacji I-węzłów

Superbloki zawierają informacje dotyczące całego agregatu:

- jego rozmiar
- rozmiar grup alokacji
- rozmiar bloku agregatu
- itd.

Wtórny superblok jest kopią podstawowego (dokładną). Wtórne superbloki znajdują się w ustalonych miejscach.

I-węzeł zawiera 4 podstawowe zbiory informacji:

- atrybuty POSIX obiektu JFS
- dodatkowe atrybuty m.in.: informacje dla VFS'a, informacje specyficzne dla systemu operacyjnego, nagłówek B+ drzewa
- albo deskryptor ekstentu zawierającego wierzchołek drzewa, albo dodatkowe dane (wplecione)
- rozszerzone atrybuty, więcej wplecionych danych, albo dodatkowe deskryptory alokacji ekstentów

I-węzeł opisuje:

- pliki, katlogi
- mapę alokacji
- położenie na dysku każdego i-węzła w zestawie plików

Wszystkie struktury metadanych (za wyjątkiem superbloku) są reprezentowane przez „pliki”.

Katalogi odwzorowują nazwy podane przez użytkownika na i-węzły przydzielane plikom i katalogom, tworząc tradycyjną hierarchię nazw.

Struktury adresowe oparte na ekstentach i zakorzenione w i-węźle służą do odwzorowywania danych pliku na bloki dysku.

Plik jest alokowany jako sekwencja ekstentów.

I-węzeł zawiera do 8 korzeni B+-drzewa. Każdy węzeł B+-drzewa to 4 kb. Dodawanie miejsca w i-węźle jest stosunkowo naturalne (gdy brakuje miejsca to jest tworzony nowy liść). Mogą powstawać węzły wewnętrzne (w wyniku tej operacji). Dane o B+ drzewie znajdują się w plikach **/linux/JFS/JFS_xtree.h** i **/linux/JFS/JFS_btree.h**.

6.8 Mapa alokacji bloków

mapa alokacji opisuje stan alokacji każdego bloku danych w agregacie

zestawy plików <-> systemy plików w **ext2fs**

plik opisany przez 2 i-węzeł agregatu jest mapą alokacji bloków

kontrolne bmap,kontrolne dmap,dmap - są zdefiniowane w:

linux/JFS/JFS_dmap.h -> structure dmap_t, structure dmapctl_t, structure dbmap_t

Mapa alokacji służy do śledzenia przydzielanych i zwalnianych bloków (jak w ext2fs’ie, tylko inaczej zorganizowana).

Zestawy plików współdzielą jedną pulę bloków.

W JFS’ie są możliwe dynamiczne zmiany wielkości (związane ze zmianą wielkości agregatu).

Mapa alokacji bloków zawiera 3 typy stron (4kb każda):

- kontrolne bmap -> jedna na początku mapy alokacji; zawiera sumryczne informacje przyspieszające wyszukiwnie („mocno” wolnych grup alokacji)
- kontrolne dmap -> zarządza dmap’ami (poprawiają wydajność); zawierają drzewa o 1024 liściach
- dmap -> tablica bitowa bloków (opisuje 8kb bloków)

Dwie mapy są zawarte w dmap:

- robocza, rejestruje bieżący stan alokacji
- trwała, to co zatwierdzone w dzienniku

Zmiany w mapie alokacji bloków nie są zapisywane w dzienniku (nie są rejestrowane).

6.9 Mapa alokacji I-wezłów i Lista wolnych i-węzłów grup alokacji

mapa alokacji i-wezłów ang. *Inode Allocation Group* = IAG - dynamiczna tablica grup alokacji i-węzłów

plik zawarty w agregacie (opisany przez własny węzeł agregatu/ zestwu plików) - reprezentacja IAG

linux/JFS/JFS_imap.h -> structure dinomap_t

linux/JFS/JFS_imap.h -> znajduje się w structure dinomap_t

Mapa alokacji i-węzłów rozwiązuje problem wyszukiwania zwykłego. Agregat i każdy zestaw plików utrzymuje mape alokacji i-węzłów. Mapa alokacji jest również nazywana tablicą i-węzłów agregatu/zestawu plików.

Grupa IAG:

- ma rozmiar 4 kb
- opisuje 128 ekstentów i-węzłów
- ponieważ każdy ekstent i-węzłów opisuje 32 i-węzły -> mapa alokacji opisuje 4096 i-węzłów

W pliku **/linux/JFS/JFS_imap.h** jest opisana struktura IAG (structure iag_t).

Wszystkie ekstenty **IAG** są umieszczone w jednej grupie alokacji.

Pierwsza 4kb strona **IAG** jest stroną kontrolną zawierającą sumaryczne informacje o mapie.

Mapa alokacji i-węzłów zestawu plików jest opisana przez i-węzeł zestawu plików. Jej strony są alokowane i zwalniane zgodnie z indeksowaniem drzewa B+ (klucz to przesunięcie bajtowe strony IAG).

Lista wolnych i-węzłów grup alokacji rozwiązuje problem wyszukiwania odwrotnego. Agregat i każdy zestaw plików utrzymuje mapę alokacji i-węzłów. Również nazywana tablicą i-węzłów agregatu/zestawu plików.

Liczba nagłówków list wolnych i-węzłów grup alokacji jest stała (znajdują się one na stronie kontrolnej mapy alokacji I-wezłów).

N-ty wpis jest nagłówkiem podwójnie powiązanej listy wszystkich wpisów mapy alokacji I-wezłów dysponujących wolnymi i-węzłami i zawartych w n-tej grupie alokacji. Numer grupy IAG jest indeksem na tej liście.

6.10 Lista wolnych grup IAG i I-węzły mapy alokacji zestawu plików

Lista wolnych grup IAG jest opisana w pliku:
[/linux/JFS/JFS_dinode.h]-> structure inomap_t.
A I-węzły mapy alokacji zestawu plików są opisane w pliku:
[linux/JFS/JFS_dinode.h]-> structure dinode_t.

Dzięki liście wolnych grup IAG, JFS może łatwo znaleźć grupę IAG bez zaalokowanych ekstentów i-węzłów.

I-węzły mapy alokacji zestawu plików tablicy i-wezłów agregatu są „super i-węzłami" zestawu plików. Przechowują:

- informacje specyficzne dla zestwu plików
- położenie mapy alokacji i-węzłów zestawu plików za pomocą B+-drzew.

6.11 Lista Uprawnień

Lista Uprawnień - ang. *Access Control List*

i-węzeł posiada własną listę uprawnień (każdy)

Lista uprawnień zawiera:

- uprawnienia obiektu
- dane użytkownika (właściciela, grupy)

6.12 Posumownie informacji o strukturach

Superblok agregatu, mapa alokacji dysku, deskryptor pliku, mapa i-węzłów, i-węzły, lista uprawnień, katalogi i stuktury adresowe są strukturami kontrolnymi JFS'a, czyli metadanymi tego systemu plików.

6.13 Standartowe narzędzia administracyjne

mkfs tworzenie systemu plików

fcsk sprawdzanie oraz odzyskiwanie systemu plików

logredo czasami używane zamiast *fcsk*

lost+found tu trafiają „sieroty"

Wadą programu *fcsk* jest to, że w razie niespójności usuwa dane.

Działanie *fcsk* dla JFS'a:
ilość potrzebnej pamięci wirtualnej zależy od ilości plików (nie bloków) (32 bajty na plik (około)).

7 Dziennikowy system plików ReiserFS

7.1 Wstęp do ReiserFS'a

RaiserFS Reiser File System (omawiany dla Linux'a)

Hans Reiser (twórca) prezentuje naukowe i intelektualne podejście do projektowania i programowania

wada ogólnie ResierFS jest „super", ale ma słabą interkację z innymi składnikami jądra Linux'a

zunifikowana przestrzeń nazw -> koncepcja „wszystko jest plikiem" postawiona na głowie

szybkie przywracanie sprawności po awarii

W tej części prezentacji omówimy:

- zalety ReiserFS'a i jego oryginalność
- realizcję i struktury danych ReiserFS'a

W tej części prezentacji opowiemy o dziennikowym systemie plików ReiserFS'ie w Linux'ie. Jest to system oparty na działaniu drzew zrównoważoncyh. ReiserFS zapewnia superblok oraz wywołania systemowe służące do wykonywania operacji na plikach i wymagane przez warstwę wirtualnego systemu plików Linux'a. Hans Reiser wierząc w zasadę doskonałego projektowania rozpoczął przedsięwzięcie, które miało dowieść słuszności jego koncepcji. Rezultat - ReiserFS jest ciekawy ze względu na naukowe i intelektualne podejście do projektowania i programowania. Głównym celem ReiserFS'a jest szybkie przywracanie sprawności systemu po awarii oraz transakcyjne aktualizacje metadanych systemu plików.

Niestety ReiserFS ma słabą interkację z innymi składnikami jądra Linuxa (np.: NFS'em).

Koncepcja „wszystko jest plikiem" oznacza, że wszystko powinno być traktowane (stosunkowo) tak samo (identyczne przechowywanie plików i katalogów (one zawierają metadane)).

Przykład Reiser'a:

Nie możesz znaleźć Świętego Mikołaja bez reniferów, chyba że wiesz, jak rozłożyć relację dostawca-napędu-dla na kanoniczne składniki.

Unifikacja i zamknięcie przestrzeni nazw jest wciąż niezbadanym terenem.

Obecnie jest dopiero początek histori ReiserFS'a. Wiele pomysłów jeszcze nie zostało zrealizowanych.

ReiserFS jako produkcyjny system plików może się okazać niezawodny i wydajny, ale pozostanie dowodem pewnej tezy oraz narzędziem rozwojowo-badawczym, nie zaś gotowym produktem.

Zacznę od omówienia zalet systemu plików RFS.

Później opowiem ogólnie o realizacji oraz strukturach danych RFS'a.

Ta część prezentacji dotyczy ReiserFS'a w wersji trzeciej (niedługo ukaże się kolejna wersja).

7.2 Zalety ReiserFS

Powody, dlaczego ReiserFS jest wspaniały dla Ciebie:

- Szybki jounalling (księgowanie / dziennikowanie)
- Szybki restart systemu po awarii
- Oparty na drzewach zrównoważonych
- Większa wydajność zajmowania miejsca na dysku

Powyższy akapit to przetłumaczony tekst ze strony **http://www.namesys.com**.

Drzewa zrównoważone to mocne narzędzie. Efektywnie zaimplementowano klasyczne algorytmy dla drzew zrównoważonych (jako drzewa zrównoważone używane są B+-drzewa). Dzięki zastosowaniu drzew zrównoważonych 100000 plików w katalogu to nic złego.

Dużo małych plików jest wrzucane do jednego bloku (zbieranie wielu małych plików w jednym bloku -> nowość). Takie zastosowanie pozwala zaoszczędzić około 6% przestrzeni dyskowej. Większośc plików na dysku to małe pliki.

7.3 Jak działa ReiserFS?

- Wyrównywanie granic plików z blokami na dysku
- Drzewo zrównoważone
- Małe pliki -> trzymane razem (w jednym bloku)
- Duże pliki -> przechowywane w niesformatowanych węzłach (poza końcówkami)
- Wady -> m.in.: brak ekstentów, wielkośc kodu
- Duże uwarstwienie budowy
- Listy zachowywania (ang.*preserve lists*)

Mimo, że ReiserFS jest bardzo wydajny dla małych plików nie oznacza to, że działa niefektywnie dla dużych plików.

Wyrównywanie granic plików z blokami dyskowymi daje:

- minimalizuje liczbe bloków zajmowanych przez plik
- brak marnotrawienia miejsca na dysku i w przestrzeni buforowej przez trzymanie niedopełnionych bloków
- zmniejsza średnią liczbę bloków potrzebną do uzyskania dostępu do każdego pliku w katalogu

Hans Reiser od początku chciał agregować małe pliki, tak aby uniknąć marnotrawienia miejsca na dysku. Początkowo były próby gromadzenia plików w obrębie katalogu (ale to nie jest idealny pomysł, niesie dużo problemów, np.: co robić gdy zwiększy się liczba plików w katalogu, jak zrównoważyć wtedy drzewo).

Idealną strukturą do przechowywania różnych rzeczy (np.: plików lub katalogów) jest słownik np.: drzewo zrównoważone (B+-drzewo).

W ReiserFS zarówno pliki jak i nazwy plików są przechowywane w drzewie zrównoważonym. Dzięki temu - a także dzięki małym plikom, wpisom katalogowym i końcówkom dużych plików - można wydajniej upakować dane (w wyniku rozluźnienia wymogu wyrównywania do bloku) i nie trzeba przydzielać stałej pamięci na i-węzły.

Zawartość dużych plików jest przechowywana w niesformatowanych węzłach (które są przechowywane w drzewie, ale nie podlegają przenoszeniu przez algorytmy równoażące).

Ani NTFS ani XFS nie zapewnia tak wydajnego upakowywania plików.

Wadą ReiserFS'a jest wielkość kodu : 30 kb dla RFS i 6 kb dla ext2fs'a. Jest to opłata za używanie drzew zrownoważonych.

Semanyka (pliki), pakowanie (bloki i węzły), buforowanie (np.: ilość danych wczytywanych z wyprzedzeniem) oraz sprzętowe interfejsy dysku (sektory) i stronnicowanie (strony) wiążą się z kwestiami ziarnistości. Ich optymalna ziarnistość bywa rózna, ale jeśli podzieli się je na grupy-warstwy, tak aby nie miały na siebie wplywu, to poprawia to wykorzystanie przestrzeni dyskowej i prędkość działania systemu plików. Innowacją ReiserFS'a jest to, że warstwa semantyczna (pliki) przekazuje do innych warstw dane nie granulwane przez granice plików.

Wady:

- gdy końcówka pliku (pliki < 4 kb to końcówki) urośnie to : *węzeł sformatowany* przekształca się w *węzeł niesformatowany*
- końcówka pliku mniejsza niż jeden węzeł może być w 2 węzłach
- oddzielenie ciała pliku od końcówki
- zapisywanie danych w końcówkach mioze powodować nieciekawy efekt (średnio połowa węzła jest przesówana)
  funkje z bibliotek C rozsądnie buforują działania na końcówkach (co rozwiązuje w pewnym stopniu problem)
- brak optymalizacji dla dużych plików
- wspomniane już problemy z współpracą z innymi składnikami jądra Linux'a

ReiserFS stosuje nową metodę zapewniania odtwarzalności - listy zachowywania - i unika nadpisywania starych metadanych, zapisując w pobliżu nowe metadane. Listy spójności (zachownia) nie są bardzo wydajne (istnieją potencjalnie wydajniejsze rozwiązania).

7.4 Drzewa w RaiserFS'ie

- Cel:
  - optymalizowanie lokalności odwołań
  - optymalizowanie wyszukiwania za pomocą kluczy w dzrzewie
  - efektywne pakowanie obiektów
- drzewo zrónoważone w ReiserFS'ie = B+-drzewo
- Klucz („funkcjonalność i-węzła")
- Trzy rodzaje węzłów:

  wewnętrzne

  sformatowane zawierają: elementy bezpośrenie, pośrednie, katalogowe i statystyczne

  niesformatowane zawierają zawartość pliku oprócz końcówki

W ReiserFS celem drzewa jest optymalizowanie lokalności odwołań oraz efektywne pakowanie obiektów. Klucze są zdefiniowane zgodnie z tym algorytmem (są używane w systemie zamiast numerów węzłów). Klucze są dłuższe niż numery i-węzłów, ale nie trzeba ich tyle trzymać (w jednym węźle może znjdować się wiele plików).

Zawart węzłów wewnętrznych i sformatowanych jest posortowana według kolejności kluczy (węzły niesformatowane nie zawierają kluczy).

Węzły wewnętrzne:

- składają się ze wskaźników do poddrzew oddzielonych ograniczającymi je kluczami.
- służą do znajdowania wlaściwych kluczy w poddrzewie.

Klucz, który poprzedza wskaźnik do poddrzewa, jest duplikatem pierwszego klucza w pierwszym sformatowanym węźle tego drzewa.

ReiserFS zaczyna od węzła głównego i schodzi na sam dół w poszukiwaniu rządanego klucza (węzła).

Pierwszy dolny poziom zawiera węzły niesformatowane, drugi - węzły sformatowane, a reszta B+-drzewa składa się z wewnętrznych węzłów. Liczbę poziomów zwiększa się w razie potrzeby dodając nowy węzeł główny.

Wszystkie ścieżki od korzenia drzewa do każdego sformatowanego węzła mają tą samą długość. Ścieżki do liści niesformatowanych również mają tę samą długość, ale są o jeden węzeł dłuższe. Równa długość ścieżek jest kluczowa dla wydajności.

Węzły sformatowane składają się z czterech typów elementów : bezpośrenich, pośrednich, katalogowych i statystycznych.

Te elementy są posortowane po kluczu.

- element bezpośredni = końcówki plikow

- element pośredni = wskaźnik do niesformatowanego węzła
- element katalogowe = zawierają klucz pierwszego wpisu katalogu przechowywanego w elemencie, po którym następuje pewna liczba wpisów katalogowych
- element statystyczne = pierwszy element pliku lub katalogu zawiera elementy statystyczne

W pewnych przypadkach końcówki moga byc w węzłach niesformatowanych.

Katalogi składają się z elementów katalogowych, które z kolei składają się z wpisów katalogowych. Wpisy katalogowe zawierają nazwę i klucz pliku.

Podczas równoważenia drzewa następuje próba łączenia trzech węzłów w dwa.

7.5 Struktura klucza

Każdy element plikowy ma klucz o następującej strukturze:

locality_id identyfikator lokalności

object_id identyfiaktor obiektu

offset przesunięcie

uniqueness niepowtarzalność

Identyfikatorem lokalności jest domyślnie identyfiaktor obiektu katalogu nadrzędnego.

Identyfikator obiektu to niepowtarzalny identyfiaktor pliku, ustawiany na pierwszy nieużywany identyfiaktor obiektu podczas tworzenia pliku. Może powodować to przechowywanie obiektów z tego samego katalogu blisko siebie (co może być korzystne).

Przesunięcie (w przypaku pliku) wskazuje pierwszy bajt elementu jako części logicznego obiektu.

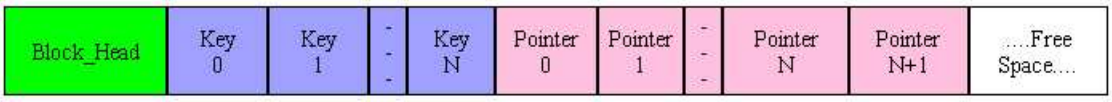
Przesunięcie (w przypaku katalogu) wskazuje pierwsze 4 bajty nazwy pliku.

Niepowtarzalność pozwala odróznić nazwy katalogów, gdy pierwsze 4 bajty są takie same. Jest to dość kontrowersyjne rozwiązanie.

Klucze mogą sie zmieniac (pomaga to w obsłudze demonów NFS'a), ale zawsze są unikalne w skali B+-drzewa.

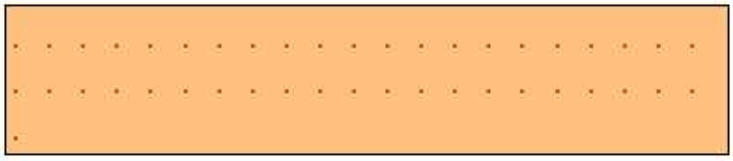
7.6 Sruktury - bloki

- Max_Height B+-drzewa = N (domyślnie 5)
- Wewnętrzny węzeł -> blok:



- Liść drzewa (n elementów) -> blok:

- Niesformatowane węzły drzewa -> blok:



Drzewo ReiserFS jest przechowywane w blokach na dysku. Każdy blok należący do drzewa ReiserFS ma nagłówek bloku.

Maksymalna liczba obiektów przestrzeni ReiserFS: (2 do 32) - 4 = 4294967292.

7.7 Sruktura key

Oto opis struktury key:

Nazwa pola	**Typ**	**Rozmiar (w bajtach)**	**Opis**
k_dir_id	_u32	4	Identyfikator katalogu nadrzędnego
k_object_id	_u32	4	Identyfikator obiektu (również numer i-węzła)
k_offset	_u32	4	Przesunięcie od początku obiektu do bieżącego bajtu obiektu
k_uniqueness	_u32	4	Typ obiektu (dane statystyczne=0, bezpośredni=-1, pośredni=-2,katalogowy=500)
razem		16	16 bajtów

7.8 Sruktura disk_child

Ta struktura jest rzeczywistym wskaźnikiem do bloku dysku.

Nazwa pola	Typ	Rozmiar (w bajtach)	Opis
dc_block_number	unsigned long	4	Numer bloku potomka
dc_size	unsigned short	2	Przestrzeń zajmowana przez potomka
razem		6	(6) 8 bajtów

7.9 Struktura block_head

To jest struktura nagłówka bloku dysku.

Nazwa pola	Typ	Rozmiar (w bajtach)	Opis
blk_level	unsigned short	2	Poziom bloku w drzewie (1=liść, 2, 3, 4, ... =wewnętrzny)
blk_nr_item	unsigned short	2	Liczba kluczy w bloku wewnętrznym albo liczba elementów w bloku-liściu
blk_free_space	unsigned short	2	Wolna przestrzeń bloku w bajtach
blk_right_delim_key	struct key	16	Klucz ograniczający ten blok z prawej strony (tylko w węzłach liściach)
razem		22	(22) 24 bajtów

7.10 Struktura - katalog i inne struktury

obiekt katalogowy

element bezpośredni

element pośredni

Inne struktury:

- item_head
- stat_data
- reiserfs_de_head

Obiekt katalogowy zawiera tylko nazwy plików.

Mały plik jest nazywany elementem bezpośrednim gdyż można go zaadresować za pomocą jednego wskaźnika.

Większe pliki (zajmujące więcej niż jeden blok dysku) wymagają wskaźnikowej akrobatyki w celu znalezienia kolejnych bloków, dlatego są nazywane elementami pośrednimi.

Każdy nagłówek elementu zawiera różne zmienne, dzięki którym element zna swoje położenie w B+-drzewie i może określić zajmowaną i wolną przestrzeń. To jest zawarte w item_head.

W strukturze stat_data przechowywane są informacje o strukturze.

Znajdowanie niesformatowanego bloku odbywa się dzięki reiserfs_de_head.

7.11 Używanie drzewa do optymalizowania układu plików

Istniją cztery poziomy optymalizowania układu plików:

1. Odwzorowywanie logicznych numerów bloków na fizyczne położenia na dysku.
2. Przypisywanie i-węzłów do logicznych numerów bloków.
3. Porządkownie obiektów w drzewie.
4. Równoważenie obiektów między węzłami, w których są upakowane.

Odwzorowywanie logicznych numerów bloków na fizyczne położenia na dysku zajmuje się program dostarczony przez producenta dysku, lub sterownik urządzenia. Oczywiście w grę może wchodzić oprogramowanie wyższego poziomu (Logical Volume Manager - LVM).

Gdy ReiserFS umieszcza węzeł drzewa na dysku, wyszukuje pierwszy pusty blok w bitmapie używanych numerów węzłów, zaczynając od lewego sąsiada węzła w B+-drzewie, a następnie poruszając się w tym samym kierunku co poprzednio. Eksperymenty wykazały skuteczność tej metody.

7.12 Równoważenie drzewa

Priorytety równoważenia drzewa:

1. Minimalizować liczbę używanych węzłów.
2. Minimalizować liczbę węzłów podlegających operacji równoważenia.
3. Minimalizować liczbę niebuforowanych węzłów podlegających operacji równoważenia.
4. Jeśli potrzebne jest przesunięcie do innego sformatowanego węzła, maksymalizować liczbę przesówanych bajtów według priorytetu.

Przyjmuje się, że miejsce wstawienia bajtów do drzewa jest wyznacznikiem przyszłego miejsca wstawienia, i że założenie 4 zmniejsza liczbę sformatowanych węzłów podlegających przyszłym operacjom równoważenia.

7.13 Mechanizmy dziennikowe ReiserFS

Podstawowe operacje, związane z dziennikowaniem:

Transakcje - Każda transakcja w dzienniku składa się z bloku opisu, po którym następuje pewna liczba bloków danych oraz blok zatwierdzenia. Blok opisu i zatwierdzenia zawierają sekwencyjną listę rzeczywistych dyskowych lokacji wszystkich bloków w dzienniku. Dziennik musi zachować kolejność aktualizacji, więc jeśli zapisujący rejestruje bloki A, B, C i D, a potem znów blok A, zostaną one ułożone w dzienniku w kolejności B, C, D, A. Gdy blok jest w niezatwierdzonej transakcji, musi pozostać czysty, a liczba odwołań do niego musi być równa przynajmniej jedności. Gdy transakcja ma wszystkie bloki na dysku, prawdziwe bufory są oznaczane jako brudne i zwalniane.

Transakcje wsadowe - łączenie wielu transakcji w jedną atomową jednostkę

Asynchroniczne zatwierdzenia - rozszerzenie transakcji wsadowych; transakcja może się zakończyć jeszcze przed zapisaniem wszystkich bloków dziennika na dysku; wprowadza to komplikacje ale jest znacznie szybsze (niż synchroniczny zapis)

Nowe bloki mogą ginąc w buforze - jeśli blok zostanie przydzielony/zarejestrowany, a potem zwolniny jeszcze przed zapisaniem na dysku/przed zakończeniem transakcji to nigdy nie zapisuje się go na dysku/w dzienniku

Wybiórcze opróżnianie buforów - gdy blok jest w niezatwierdzonej transakcji, nie można oznaczyć go jeko brudnego i zapisać na dysku; zanim jednak będzie można ponownie wykorzystać obszar dziennika, wszystkie zawarte w nim transakcje muszą zapisać swoje bloki w rzyczywistych lokacjach dysku

Rejestrowanie bloków danych - bloki danych są rejestrowane podczas konwersji elementu bezpośredniego na pośredni

7.14 Podsumowanie wiadomości o ReiserFS

- RaiserFS -> moc w małych plikach .
- Sumowanie małych obiektów w systemie plików .
- 80% to małe pliki (poniżej 10kb)! Więc warto je gromadzić!
- Niezwodność -> realizacja dziennikowania (rejestrowania).
- Niedługo pojawi się wersja ReiserFS 4!
- Materiały:
  http://www.namesys.com
  Dokumentacja: .../linux/Dokumentacja/filesystems...
  Książka: „Linux Systemy Plików” Moshe Bar

ReiserFS jest przyszłościowym systemem plików, który ciągle ulega rozwojowi. Nie należy zapominać, że jest przdewszystkim realizacją pewnej idei i narzędziem rozwojowo-badawczym.

8 Dziennikowy System Plików XFS

8.1 Wstęp do XFS'a

XFS -> Firma **Silicon Graphics**

1993 - grudzień 1994 prace nad XFS'em na system IRIX (EFS sobie „nie radził")

2000 - 2001 przeniesienie XFS'a na Linux'a

Doug Doucette z SGI -> kierownik

W tej części prezentacji zostaną omówione

- zalety XFS'a
- podstawowa architektura XFS'a
- realizcja i struktury danych XFS'a

Pomysł na XFS'a pojawił się już na początku lat dziewięćdziesiątych.

Projektem najbardziej skalowalnego systemu plików dla Linuksa jest próba adaptacji systemu XFS. Twórcy XFS jest znany producent superkomputerów oraz wydajnych, graficznych stacji roboczych - firma Silicon Graphics.

Prace nad XFS rozpoczęto w 1993 roku. Główną przyczyną był wzrost wydajności i skalowalności pamięci masowych, z czym nie radził sobie system plików EFS dotychczas stosowany w systemie operacyjnym IRIX (odmiana Uniksa dla maszyn SGI). System EFS wywodził się z BSD FFS, obsługiwał partycje o rozmiarze do 8 GB oraz miał klasyczne ograniczenie 32-bitowego systemu plików - maksymalny rozmiar pliku wynosił 2 GB.

Pierwsza wersja nowego systemu plików dla IRIX-a ukazała się w grudniu 1994 roku. W roku 1996 XFS stał się domyślnym systemem, wypierając poprzednika. Pierwotnie XFS był projektowany pod kątem systemu operacyjnego IRIX. Przeniesienie go na Linux'a wymagało poniesienia znacznych nakładów pracy, a zwłaszcza stworzenia nowych styków systemu plików z resztą systemu operacyjnego. Architektura XFS stawia specyficzne wymagania dla warstwy wirtualnego systemu plików (VFS'a) oraz warstwy buforów dyskowych.

W projekcie zdecydowano się nie tworzyć nowych styków, lecz dodatkowy kod pośredniczący, który mapowałby wywołania funkcji zgodnych z Linuksem na wywołania zgodne z IRIX-em. Pierwsza z warstw - **linvfs** - odpowiada za współpracę z linuksowym VFS, druga - **pagebuf** - za styk z buforem blokowym. Tego typu podejście ma swoje wady i zalety. Wadą są dodatkowe opóźnienia w porównaniu z implementacją dla bezpośrednio dla Linux'a. Zaletą tego podejścia jest wspólna linia rozwojowa samego XFS dla obydwu platform (IRIX'a i Linux'a).

Podczas projektowania i tworzenia XFS'a przyjęto następujące podstawowe cele:

1. XFS ma być użyteczny w:
    - naukowych serwerach plików

- komercyjnych serwerach przetwarzających dane
- serwerach mediów elektronicznych

2. musi zapewniać dużą dostępność danych i szybko odzyskiwać sprawność po awarii zachowując dane dyskowe w spójnym stanie
3. musi wydajnie obsługiwać bardzo duże (64-bitowe) pliki
4. niedopuszczalne jest liniowe wyszukiwanie w strukturach danych systemu plików w celu dotarcia do określonej części (bloku) pliku
5. musi wydajnie obsługiwać rozrzedzone pliki
6. musi wydajnie obsługiwać bardzo małe pliki (mniejsze niż 1 kb)
7. musi wydajnie obsługiwać operacje na dużych katalogach
8. powinien obsługiwać listy kontroli dostępu i inne funkcje POSIX
9. powinien obsługiwać wiele logicznych rozmiarów bloków

8.2 Podstawowa architektura XFS'a

XFS ma strukturę modułowa:

Menedżer Przestrzeni Dyskowej ang. *Space Manager*

Menedżer wejścia/wyjśćia - ang. *I/O Manager*

Menedżer Katalogów - menadżer przestrzeni nazw ang. *Directory Manager*

Menedżer Transakcji - dziennika ang. *Log manager*

Menedżer Ryglowania ang. *Lock Manager*

Menedżer Pamięci Podręcznej Buforów ang. *Buffer Cache Manager*

Menedżer Atrybutów *Attribute Manager*

XFS ma architekturę modułową (patrz rysunek). Najważniejszym modułem jest menedżer przestrzeni dyskowej (Space Manager), którego najistotniejszym zadaniem jest zarządzanie wszystkimi obiektami w systemie plików, a zwłaszcza obsługa i-węzłów oraz ekstentów. Modułami pośredniczącymi pomiędzy VFS IRIX-a a menedżerem przestrzeni są menedżer wejścia/wyjścia (I/O Manager) oraz menedżer katalogów (Directory Manager).

Pierwszy z nich zajmuje się całościową obsługą plików, a drugi - jak sama nazwa wskazuje - katalogów. Menedżer transakcji zarządza aktualizacją danych. Odpowiada również za operacje księgowania (journaling), które pozwalają na szybką naprawę systemu po awarii. Księgowanie polega na transakcyjności zmian w systemie plików oraz ich logowaniu. W XFS obejmuje ono

tylko metadane, takie jak drzewa i-węzłów, i-węzły, zawartość katalogów, drzewa ekstentów zawartości plików, drzewa wolnych ekstentów, bloki nagłówkowe grup alokacji oraz superbloki. Czyli podobnie jak w JFS'ie.

XFS używa również menedżera woluminów logicznych (ang. *Logical Volume Manager*), który zajmuje się obsługą woluminów. Jest to dodatkowy poziom logiczny pomiędzy partycjami a funkcjami operującymi na dysku.

Menedżer przestrzeni dyskowej zarządza alokacją przestrzeni dyskowej w systemie plików. Jest odpowiedzialny za odwzorowanie pliku na (sekwencji bajtów) na sekwencje bloków dysku. Steruje wewnętrzną strukturą systemu plików:

- grupami alokacji (cylindrów)
- i-węzłami
- wolną przestrzenią
- mechanizmami odwzorowującymi

Menedżer dziennika szeregowo rejestruje wszystkie zmiany metadanych w odrębnym obszarze dysku (dzienniku).

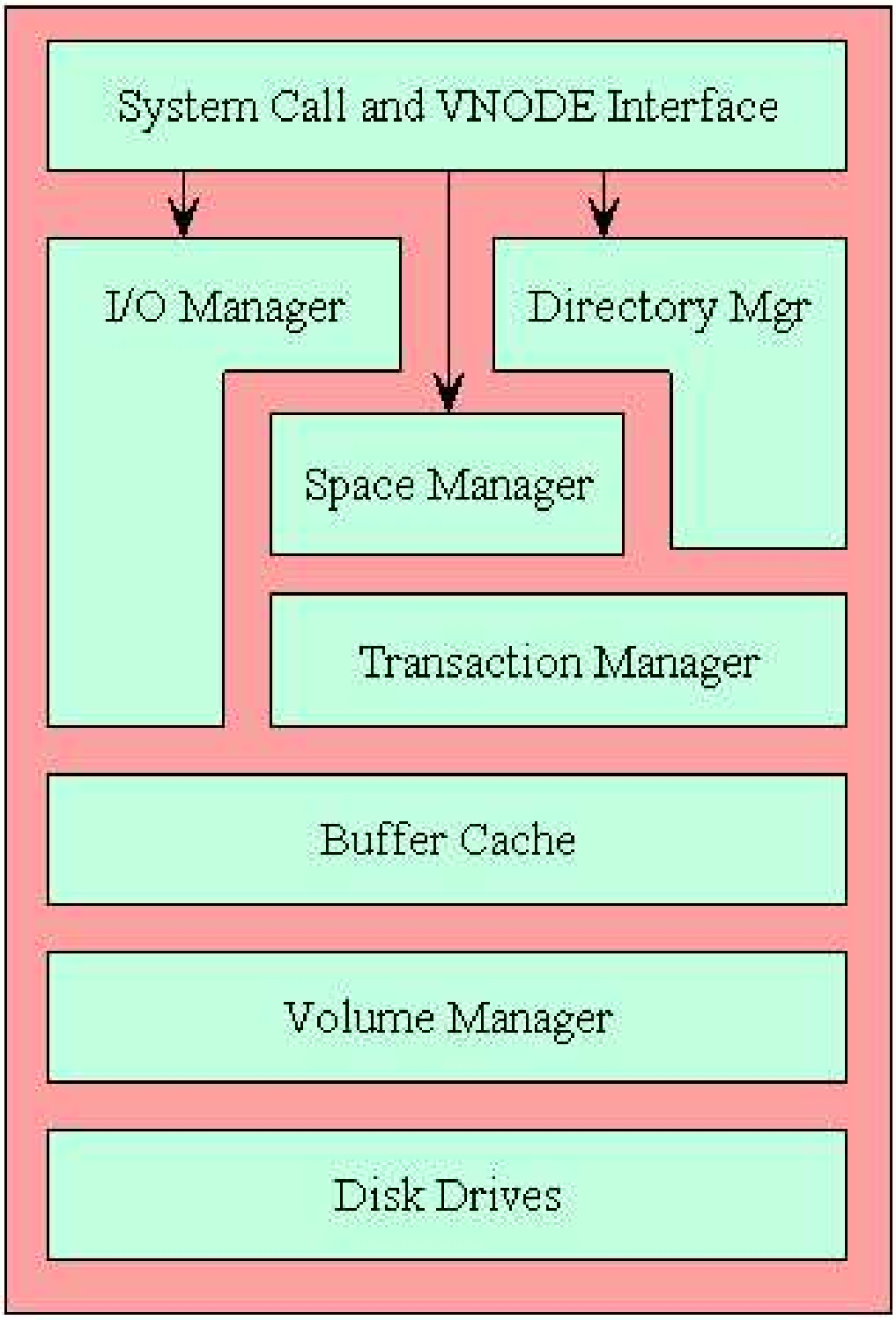
Menedżer pamieć podręcznej buforów przechowuje bloki dysku z różnych systemów plików działających lokalnie na komputerze. Żądania odczytu, jak i zapisu mogą trafiać do tego menedżera (standartowo trafiają). W pamięci podręcznej buforów przechowywane są zarówno metadane systemu plików jak i dane plików. Żądania użytkowników mogą pomijać ten menedżer dzięki znacznikom (**O_DIRECT**).

Menedżer ryglowania (rygli) korzysta z zaawansowanych algorytmów, aby ryglowanie trwało jak najkrócej. Menedżer rygli jest też przygotowany do działania w rozproszonym systemie plików, pozwalając na ryglowanie obiektów przez wiele węzłów klastra.

Menedżer atrybutów realizuje operacje na atrybutach systemu plików. Atrybuty to dołączone do i-węzła struktury (postaci: nazwy - wartości).

Menedżer przestrzeni nazw realizuje operacje nazewnicze i tłumaczy ścieżki na odwołania do plików. Plik jest identyfikowny przez swój i-węzeł i system plików. Numer i-węzła jest etykietą i-węzła w konkretnym systemie plików. Pliki są także identyfikowane przez unikatową liczbową wartość (niepowtarzalny identyfikator pliku). System plików jest identyfikowany przez „magiczne ciasteczko" (zwykle adres głównego i-węzła w pamięci), albo za pomocą niepowtarzalnego identyfikatora systemu pliów.

8.3 Podstawowa architektura XFS - rysunek



8.4 Struktury

- 0.5 kb < |Blok| < 64 kb
- **Grupy alokacji dyskowej** ang. *Alloocation Group*; wielkość od 16 mb do 4 gb
- superblok (a AG): *fs/xfs/xfs_sb.h*
- drzewo wolnych ekstentów
- drzewo i-wezłów
- struktura xfs_agfl

Podobnie jak i inne systemy plików, XFS operuje na blokach o stałej wielkości. Rozmiar ten możemy wybrać przy tworzeniu systemu plików, a zakres dopuszczalnych wartości mieści się w przedziale od 0,5 kB do 64 kB. Poza tym XFS dzieli partycję na kilka jeszcze większych części. Pierwsze z nich to tzw. grupy alokacji (allocation groups), czyli porcje przestrzeni o rozmiarze od 16 MB do 4 GB przeznaczone na właściwy system plików. Grupy alokacji umożliwiają sprawne wielowątkowe zarządzanie systemem plików. Oczywiście przy tworzeniu XFS-a należy wziąć pod uwagę ograniczenia narzucane przez maksymalny rozmiar grupy alokacji oraz liczbę procesorów w naszym systemie. Większa liczba procesorów zwiększa wydajność XFS, ale jednocześnie większa liczba grup alokacji przy mniejszej liczbie procesorów zmniejsza wydajność systemu plików.

Na początku każdej grupy alokacji znajduje się superblok - (tak jak w ext2fs'ie) blok opisujący cały system plików. Za superblokiem umiejscowiony jest nagłówek grupy alokacji. W jego skład wchodzą trzy struktury: drzewo wolnych ekstentów, drzewo i-węzłów, struktura pomocnicza przyspieszająca znajdowanie wolnej przestrzeni.

W ramach **AG** (grupy alokacji) dostępna wolna przestrzeń całego systemu plików jest indeksowana w formie pary B+-drzew. Pierwsze drzewo jest indeksem położenia wolnych ekstentów, drugie - ich rozmiarów. Złożoność obliczeniowa operacji wyszukiwania w B+-drzewie jest rzędu O(log n), a więc znacznie mniejsza od złożoności przeszukiwania map bitowych, która w skrajnym wariancie (jak np. w ext2fs'ie) może wynosic O(n), gdzie n oznacza liczbę jednostek alokacji (i-węzłów, bloków czy ekstentów).

W grupie alokacji, tak jak wolne ciągłości, i-węzły są zapamiętywane w specjalnych B+-drzewach. I-węzły XFS nie różnią się wiele od klasycznych uniksowych i-węzłów. Również mają swój niepowtarzalny numer, według którego są indeksowane w B+-drzewach.

Xfs_agfl jest strukturą pomocnicza przyspieszająca znajdowanie wolnej przestrzeni (nie opisywana w tej prezentacji).

8.5 Rejestracja transakcji i Alokowanie ekstentów

Transakcje (ważne pojęcia):

log - wpis dziennikowy

transakcje

nagłówek log'a

Alokowanie ekstentów (ważne pojęcia):

- bufory ekstentów
- rezerwowanie bloków
- pomijanie buforów

Tuż za grupami alokacji znajduje się obszar informacji o ostatnio wykonywanych transakcjach w ramach systemu plików (tak zwane logi). XFS rejestruje zmiany dokonywane na metadanych (w superblokach, nagłówkach grup alokacji, drzewach wolnych ekstentów, drzewach i-węzłów, w i-węzłach, w katalogach). Operacje są połączone w transakcje będące ciągiem zmian w blokach systemu plików. Działa to podobnie jak w JFS'ie.

Obsługa transakcji w XFS jest domyślnie asynchroniczna (dostęp do modyfikowanych bloków systemu plików jest blokowany dla innych żądań dopiero na etapie zatwierdzania transakcji). Jest to rozsądne podejście jeśli chodzi o wydajność. Inaczej sytuacja się ma, gdy chodzi o zasoby udostępniane przez sieciowy system plików NFS, gdzie używany jest synchroniczny tryb zapisu (blokowanie dostępu przy każdej nowej modyfikacji metadanych w ramach transakcji). XFS umożliwia tworzenie wpisów do dziennika na innym urządzeniu niż logowany system plików.

Nazwy plików i katalogów są ograniczone przez 255 znaków i są indeksowane poprzez funkcję skrótu, która generuje odpowiadające tym nazwom 32-bitowe unikatowe klucze.

W ramach XFS alokacja ekstentów oraz zapis pliku nie odbywa się od razu:

1. Najpierw na dysku rezerwowane są bloki na możliwie największy ekstent.
2. Następnie w pamięci tworzony jest specjalny bufor ekstentu (zbiór buforów blokowych).
3. Dopiero gdy zaistnieje potrzeba zwolnienia pamięci lub zostanie przekroczony rozmiar zarezerwowanych bloków, ekstent jest kopiowany na fizyczny system plików, a bufor usuwany.

Niewątpliwą zaletą tego typu buforów ekstentów jest zmniejszenie fragmentacji systemu plików, liczby operacji na metadanych oraz szybka obsługa plików tymczasowych. W skrajnym przypadku wcale nie muszą być zapisywane na dysku.

Inną właściwością XFS'a jest elastyczny mechanizm zarządzania równoczesnym dostępem do konkretnego pliku. W tradycyjnych Linux'owych systemach plików (np. ext2fs'ie) obowiązuje zasada - jedno zadanie zapisuje, wiele zadań odczytuje. W XFS zaimplementowano dodatkowe funkcje dla programów użytkownika, które dzięki mechanizmowi kolejkowania żądań zapisu pozbawione są tego ograniczenia.

8.6 Struktury w XFS'ie - podsumowanie

- B+-drzewa i-węzłów są zdefiniwane w pliku *fs/XFS/XFS_btree.h*
- B+-drzewa ekstentów są zdefiniwane w pliku *fs/XFS/XFS_bmap_btree.h*
- Cechy ekstentu:
  - 54-bitowy numer bloku w ramach pliku
  - 52-bitowy numer bloku w ramach systemów plików
  - 21-bitowy rozmiar (liczba bloków)
- Zawartość katalogów jest przechowywana w B+-drzewa

Podstawowe obiekty systemu plików (wolne ekstenty, i-węzły) są zorganizowane w B+-drzewa. Struktura takich drzew jest zdefiniowana w pliku źródłowym fs/XFS/XFS_btree.h. Do pojedynczego i-węzła, odnoszą się liście w drzewach i-węzłów.

8.7 Podsumowanie wiadomości o XFS'ie

- Wspólna linia rozwojowa samego XFS dla obydwu platform: IRIX'a i Linux'a
- Silicon Graphics Inc.
- Generalnie szybki zapis (w porównaniu do ext2fs)
- B+ drzewa i ekstenty
- Materiały:
  http://www.oss.sgi.com/projects/xfs...
  Artykuł : http://www.pckurier.com/...
  Dokumentacja: .../linux/Dokumentacja/filesystems...
  Książka: „Linux Systemy Plików" Moshe Bar

XFS jest jednym z najbardziej obiecujących dziennikowych systemów plików dla Linux'a. Jest rozwijany równocześnie dla dwóch systemów operacynych (IRIX'a i Linux'a). Należy pamiętać, że to Linux jest tym systemem dla którego był adaptowany (z czego wynika odrobinę zmniejszona wydajność).

9 Podsumowanie informacji o zaawansowanych systemach plików dla Linuxa

9.1 Podsumowanie

W tej części prezentacji zostały zaprezentowane różne linux'owe systemy plików:

JFS

ReiserFS

XFS

Ich główne zalety to:

- stosowanie B+-drzew
- częsta dynamiczna alokacja (różnych obiektów)
- bloki -> ekstenty

Mamy nadzieję, że udało się nam przekonać was, że są to bardzo ciekawe i wnoszące wiele przyszłościowych rozwiązań systemy plików.

Na konieć pokażemy kilka testów.

9.2 Tabelki testowe

Typ	**JFS**	**EXT3**	**ReiserFS**
zapis	11.04	10.51	9.52
zapis los.	0.54	0.54	0.54
odczyt	202.47	199.60	201.01
odczyt los.	184.22	184.24	186.08

Typ	**JFS**	**EXT3**	**ReiserFS**
zapis	12.17	1.24	5.05
zapis los.	0.44	0.62	0.62
odczyt	310.32	310.50	308.25
odczyt los.	298.50	302.31	301.56

Przykładowe tabelki z testami dla : **ReiserFS'a**, **ext3fs'a**, **JFS'a**.

Specyfikacja testów:

1. 4-way SMP 500 MHz, 2.4 GB RAM, 2-9.2 GB hard drives.
2. Linux kernel 2.5.40

3. tiobench command: tiotest -f120 -t[numberofthreads] -b4096

4. All journaling filesystems were selected and configured with menuconfig, and were created and mounted with default value.

Wartości podawane są w tabelkach w MB/sec.
Pierwsza tableka thread = 1.
Druga tableka thread = 6.