Lokalne systemy plików

Łukasz Osipiuk, Paulina Kania, Paweł Ostrowski

5 grudnia 2002

Spis treści

1 Wstęp

1.1 Co to jest system plików

Większość systemów operacyjnych przechowuje informacje zarówno w szybkiej, ale nietrwałej pamięci RAM jak i stosunkowo wolnej, ale trwałej pamięci (dyskietki, dyski twarde, płyty CD, etc.). Sposób organizacji danych w takiej trwałej pamięci w logiczną strukturę nazywa się systemem plików. System plików zapewnia:

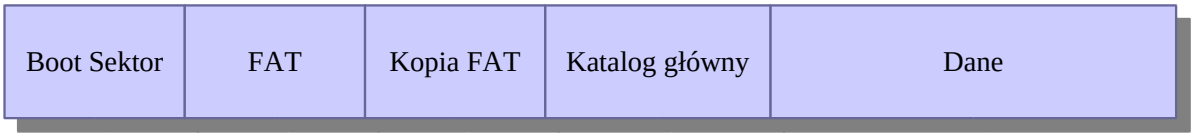
- dostęp do danych (często z uwzględnieniem uprawnień)
- spójność zapisanych danych (czyli, że odczytane dane będą identyczne z tymi, które zostały zapisane)
- dostęp do informacji o własnym stanie (np. ilość wolnego miejsca)
- dostęp do meta-danych, czyli wszystkich informacji dotyczących plików z wyjątkiem samych danych w plikach (np. czas dostępu, rozmiar, uprawnienia, etc.)

2 FAT

System plików FAT został zaimplementowany przez Billa Gatesa w 1977 roku. Początkowo związany był z jednodyskietkowym interpreterem BASIC'a (służył jako system plików dla dyskietki). Po pewnych zmianach przystosowano go do obsługi większych, wydajniejszych nośników (dysków twardych) i został użyty przez firmę Microsoft jako system plików w pierwszej wersji DOS'a.

Podstawową jednostką alokacji w systemie FAT jest klaster. Jego rozmiar ustalany jest podczas formatowania partycji.

W tabeli 1 przedstawionno parametry różnych wersji systemu plików FAT.



Rysunek 1: Schemat partycji FAT

Partycja podzielona jest na strefy jak pokazano na rysunku 1.

Boot sektor Skok do kodu ładującego system operacyjny, opis systemu plików

FAT Dwie kopie tablicy alokacji plików. Druga kopia utrzymywana jest jako backup.

Katalog Główny Zawiera metadane plików w katalogu głównym partycji.

Dane Informacje zapisane w plikach i puste miejsce.

	FAT 12	**FAT 16**	**FAT 32**
Zastosowanie	Dyskietki i dyski twarde o bardzo małej pojemności	Dyski twarde o małej i średniej pojemności	Dyski twarde o średniej i dużej pojemności
Wielkość wpisu w tablicy FAT	12 bitów	16 bitów	28 bitów
Maksymalna liczba klastrów	4,086	65,526	268,435,456
Wielkości używanych klastrów	0.5—4 KB	2—32 KB	4—32 KB
Maksymalny rozmiar partycji	16,736,256	2,147,123,200	ok. 2^{41}

Tablica 1: Różne wersje systemu FAT

2.1 Tablica FAT

Tablica FAT zawiera dane o tym, które sektory na dysku należą do poszczególnych plików. Każdy wpis w tablicy FAT odpowiada jednemu sektorowi w strefie danych. Zawiera on wskaźnik na wpis w tablicy FAT odpowiadający następnemu sektorowi aktualnego pliku.

Istnieje grupa wartości występujących we wpisach w tablicy FAT, które są zastrzeżone i oznaczają:

FREE - pusty klaster

- 0x0000 dla FAT 12 i FAT 16
- 0x00000000 dla FAT 32

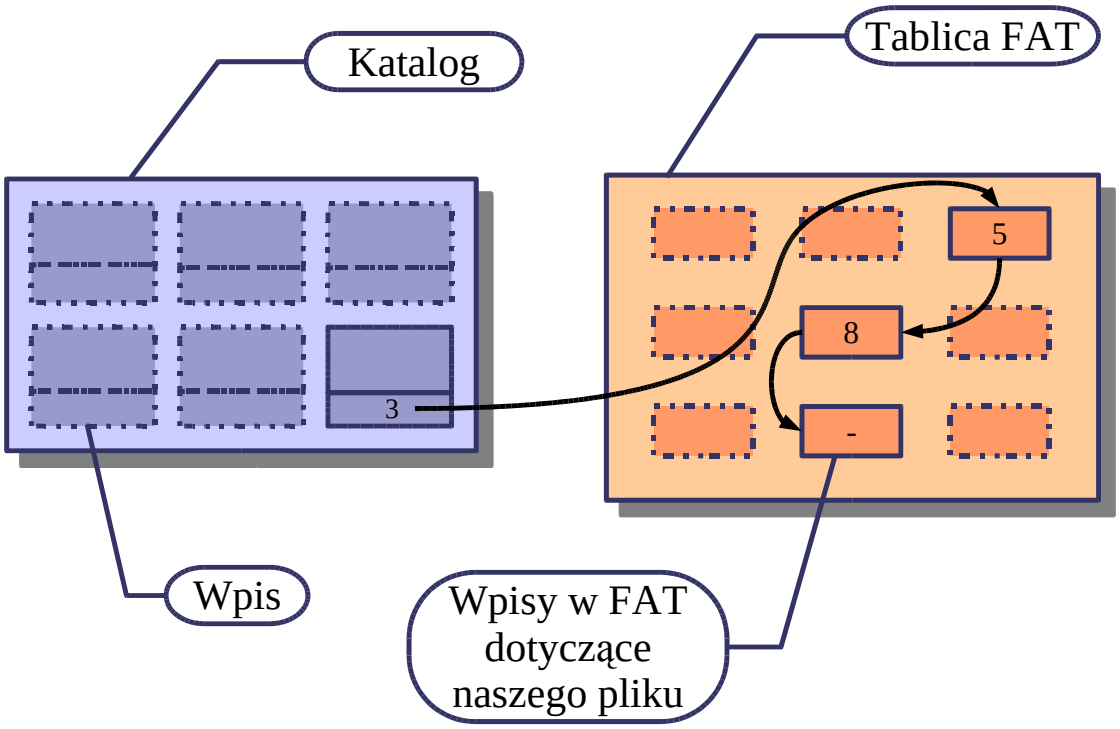
BAD CLUSTER - uszkodzony klaster

- 0x0FF7 dla FAT 12
- 0xFFF7 dla FAT 16
- 0x0FFFFFF7 dla FAT 32

EOC - koniec klastra (oznaczający koniec pliku)

- 0x0FF8 - 0x0FFF dla FAT 12
- 0xFFF8 - 0xFFFF dla FAT 16
- 0x0FFFFFF8 - 0x0FFFFFFF dla FAT 32

Lista pustych klastrów jest to po prostu lista klastrów, których wartość wynosi 0. Nie jest ona jednak przechowywana nigdzie na dysku - jednynie liczba pustych klastrów jest przeliczana podczas montowania nośnika. W przypadku FAT-u 32 wartość ta może (ale nie musi) być przechowywana w polu *BPB_FSInfo*.



Rysunek 2: FAT - zasada działania

Według specyfikacji może występować więcej niż 1 kopia zapasowa tablicy FAT , jednak w praktyce stosuje się zawsze tylko jedną taką kopię.
Z tego co nam wiadomo, większa ilość nie jest wspierana nawet przez systemy Microsoftu.

2.2 Katalog

Dane o nazwach plików przechowywane są w strukturach katalogowych. Każdy katalog może zawierać potencjalnie dowolną liczbę wpisów. Jedynie liczba wpisów w katalogu głównym jest ograniczona (dla dysku twardego zwykle do 512 wpisów). To ograniczenie nie dotyczy jednak FAT-u 32.

Wielkość jednego wpisu jest stała i wynosi 32 bajty.

Na rysunku 3 przedstawiono strukturę wpisu w katalogu. Znaczenie poszczególnych pól jest intuicyjne, jednak część z nich wymaga dokładniejszego omówienia.

Pole *name*

W polu zawierającym nazwę plików oprócz znaków alfanumerycznych (i innych mogących wystąpić w nazwie) są stosowane znaki specjalne:

```
struct directory {    // Nazwy krótkie w formacie 8.3
   unsigned char name[8];       // Nazwa pliku
   unsigned char ext[3];        // Rozszerzenie pliku
   unsigned char attr;          // Atrybuty
   unsigned char lcase;         // Wielkość liter nazwy
                                   i rozszerzania pliku
   unsigned char ctime_ms;      // Czas utworzenia w milisekndach
   unsigned char ctime[2];      // Czas utworzenia w formacie
                                   DOS'owym
   unsigned char cdate[2];      // Data utworzenia w formacie
                                   DOS'owym
   unsigned char adate[2];      // Czas ostatniego dostępu
   unsigned char reserved[2];   // Zastrzeżone
   unsigned char time[2];       // Czas ostatniej modyfikacji
   unsigned char date[2];       // Data ostatniej modyfikacji
   unsigned char start[2];      // Numer początkowego klastra
   unsigned char size[4];       // Wielkość pliku
}
```

Rysunek 3: Struktura wpisu w katalogu

00h - wpis jest pusty i nie był wcześniej używany

E5h - stary wpis został wymazany

2Eh - wpis specjalny występujący we wszystkich katalogach oprócz katalogu głównego oznaczający, że pierwszy znak jest kropką. Znak '.' - wskazuje na bieżący katalog, '..' - wskazuje na katalog macierzysty lub na NULL, jeśli katalog macierzysty jest katalogiem głównym.

Pole *attr*

Możliwe atrybuty pliku:

read_only Tylko do czytania

hidden Ukryty

system Systemowy

volume_label Etykieta dysku

directory Katalog

archive Archiwalny

reserved 2 bity zastrzeżone.

Pole *reserved*

W przypadku FAT 32 pole to zawiera 16 starszych bitów numeru początkowego klastra. W pozostałych wersjach wartość pola *reserved* jest ignorowana.

Pole *size*

W przypadku plików określa ich wielkość, a dla katalogów przyjmuje zawsze wartość 0.

DŁUGIE NAZWY PLIKÓW

Nazwy plików wykraczające poza format 8.3 są trzymane we wpisach dwojakiego typu. Pierwszą z nich jest (dla zachowania kompatybilności) standardowy wpis w katalogu (rysunek 3), przechowujący podstawowe informacje o pliku oraz początkową część nazwy. Drugą strukturę, przechowywującą dalsze fragmenty nazwy (każdy po 13 kolejnych liter), pokazano na rysunku 4.

Żeby odnaleźć kolejny fragment nazwy należy sięgnąć do poprzedniego wpisu w katalogu.

```
struct slot { // Up to 13 characters of a long name
   unsigned char id;               // Numer fragmentu
   unsigned char name0_4[10];      // Pierwsze 5 liter nazwy
   unsigned char attr;             // Atrybut
   unsigned char reserved;         // Zawsze = 0
   unsigned char alias_checksum;   // Suma kontrolna krótkiej
                                      wersji nazwy (8.3)
   unsigned char name5_10[12];     // 6 kolejnych liter nazwy
   unsigned char start[2];         // Numer początkowego klastra.
                                      Zawsze = 0
   unsigned char name11_12[4];     // 2 ostatnie litery nazwy
};
```

Rysunek 4: Fragment długiej nazwy w katalogu

Pole *id*

- Pierwsze 6 bitów odpowiada za numer fragmentu nazwy pliku. Stąd maksymalna długość nazwy wynosi 63 * 13 liter = 819. W praktyce długość nazwy nie przekracza 250 znaków.
- Bit siódmy wskazuje czy dany wpis jest ostatnim fragmentem nazwy.
- Bit ósmy jest ustawiony, jeżeli wpis został usunięty (w przypadku usunięcia pliku lub skrócenia jego nazwy).

Pole *attr*

We wpisie zawierającym fragment długiej nazwy pliku atrybut ten przyjmuje wartość 0x0f. W standardowym wpisie wartość 0x0f jest traktowana jako błąd. Pliki z atrybutem 0x0f są niedostępne dla starszego oprogramowania, ponieważ żaden z programów nie ruszy pliku, który ma ustawione flagi: tylko do czytania, ukryty, systemowy, etykieta dysku...

Pole *alias_checksum*

Suma kontrolna krótkiej wersji nazwy jest używana do sprawdzenia czy wpis zawierający fragment długiej nazwy pliku jest częścią właśnie odczytywanej nazwy.

3 ISO 9660

3.1 Wstęp

System plików ISO9660 powstał jako efekt pracy komitetu High Sierra, powołanego w celu opracowania standardowego systemu plików dla płyt CD-ROM. Przed opracowaniem tego standardu cały ciężar obsługi napędów i systemu plików dla CD-ROM‘ów spadał na twórców aplikacji. Standard High Sierra został zdefiniowany i zaproponowany **I**nternational **S**tandarts **O**rganization w maju 1986. Podczas dyskusji nad nowym standardem firmy zaangażowane w stworzenie High Sierra zaimplementowały go. W 1988 roku **ISO** opublikowało standard pod nazwą ISO9660, wprowadzając niewielkie zmiany, które spowodowały brak kompatybilności ISO9660 i propozycji High Sierra.

ISO9660 został zaprojektowany tak, aby umożliwić wymianę informacji niezależnie od systemu operacyjnego, zawiera więc minimalną liczbę informacji, która może być wykorzystana przez wiele, nieraz bardzo odmiennych systemów operacyjnych. Ponieważ ISO9660 jest przeznaczony dla CD-ROM‘ów, nie zapewnia żadnej możliwości zmieniania danych, które zawiera.

ISO9660 określa 3 poziomy wymiany, kompatybilności (ang. levels of interchange):

poziom 1

- pliki muszą być spójne, tzn. każdy plik musi być zapisany jako ciąg następujących po sobie bajtów.

- nazwa pliku nie może być dłuższa niż 8 znaków
- rozszerzenie pliku nie może być dłuższe niż 3 znaki
- nazwa katalogu nie może być dłusza niz 8 znaków

poziom 2 - jedynym wymogiem jest tu spójność plików

poziom 3 - brak ograniczeń dotyczących nazw plików, czy ich spójności

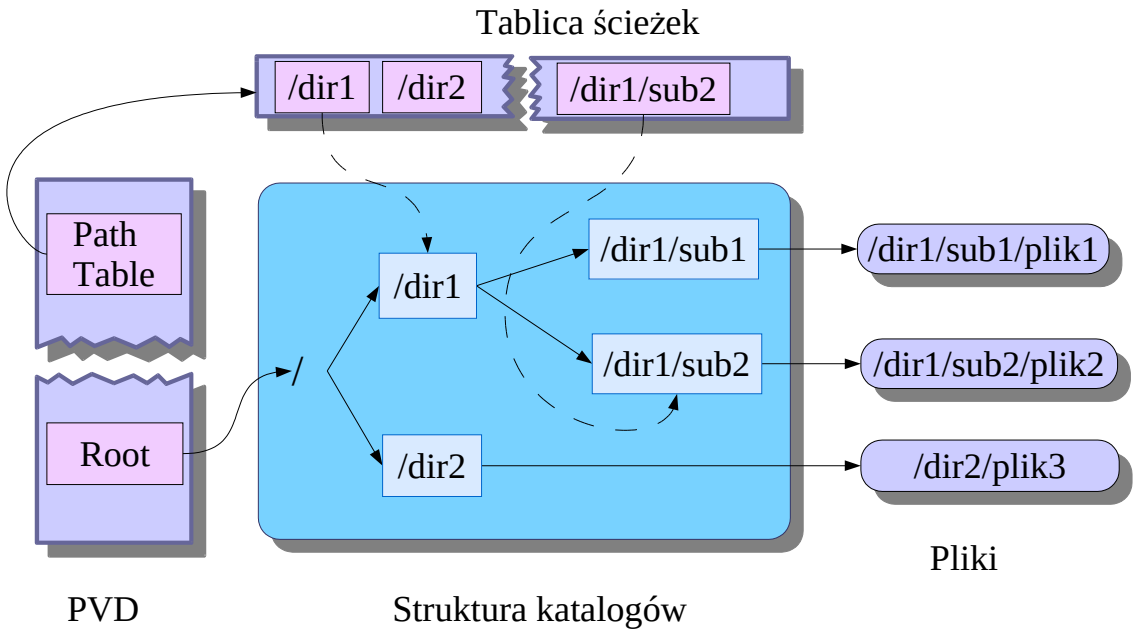
3.2 Struktura dysku ISO9660

Struktury danych dzielą się na 3 główne kategorie:

Volume Descriptors - deskryptory dysków

Directories - katalogi, pozwalają odwołać się do plików

Path Tables - zawiera skróty do każdego katalogu



Rysunek 5: Struktura Dysku ISO9660

3.2.1 Deskryptory dysku (Volume Descriptors)

ISO9660 określa 4 typy deskryptorów dysku:

Primary Volume Descriptor - pamięta pozycję głownego katalogu i tablicy ścieżek (Path Table), jest to zwykle jedyny stosowany rodzaj dekryptora

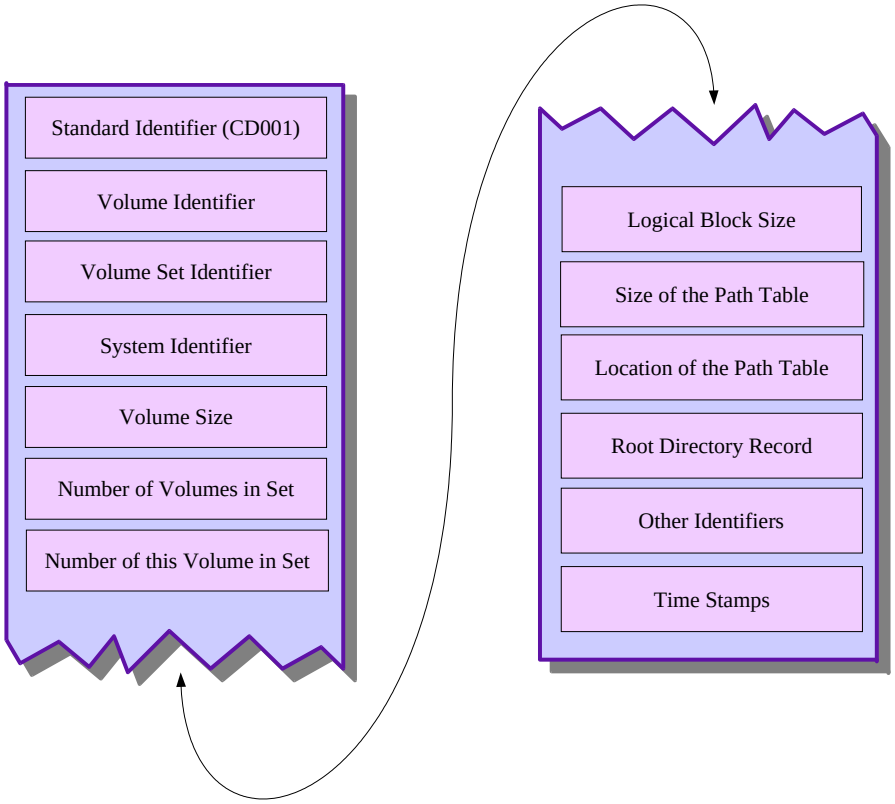
Boot Record -

Supplementary Volume Descriptor - może określać alternatywne kodowanie znaków dla systemów nie obsługujących ISO 646

Volume Partition Descriptor - może służyć do określenia logicznego podziału dysku na partycje, ale ISO9660 nie specyfikuje jak to zrobić

Deskryptory Dysku są nagrywane od 16 logicznego sektora płyty.

Główny Deskryptor Dysku



Rysunek 6: Zawartość Głównego Deskryptora Dysku (PVD)

Identyfikacja dysku zaczyna się od przeczytania głównego deskryptora dysku. Zawiera on między innymi pola:

Standard Identifier - identyfikator pozwalający odróżnić ISO9660 od innych standardów mających podobną strukturę (dla ISO9660 jest to ’CD001’, dla High Sierra ’CDROM’, a dla Compact Disc Interactive ’CD-I’)

Volume Identifier - nazwa dysku

Path Table - określa gdzie znajduje się tablica nazw katalogów, zwykle czyta się ją tylko raz i trzyma w pamięci operacyjnej, żeby przyśpieszyć dostęp do katalogów na dysku.

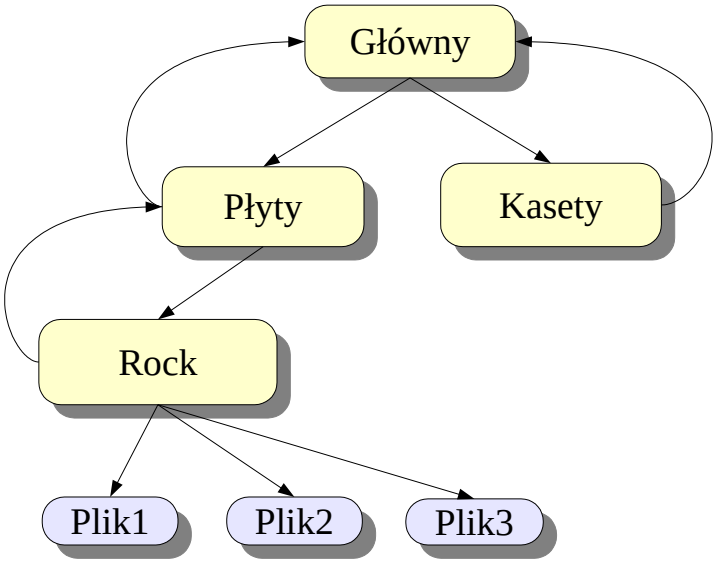
Path Table Size - wielkość tablicy nazw katalogów

Logical Block Size - najmniejsza ilość miejsca allokowana na dysku, może wynosić 512, 1024 lub 2048 bajtów, zwykle używa się bloków wielkości 2048

Root Directory Record - Rekord katalogu dla katalogu głównego: pozwala znaleźć na dysku i przeczytać główny katalog, jest to zwykły rekord katalogu, taki sam jak te opisujące każdy inny katalog na dysku.

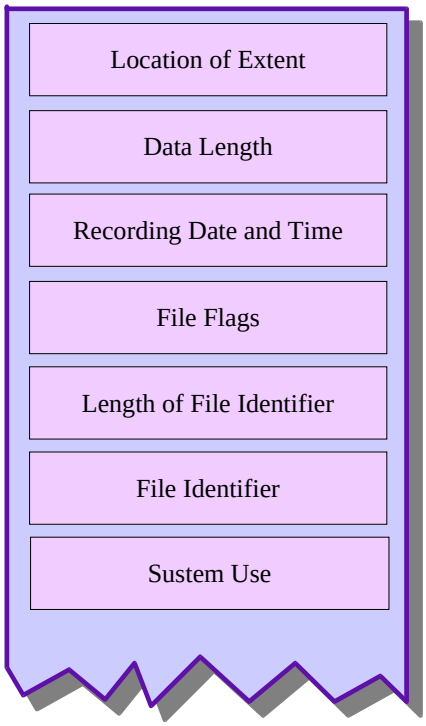
3.2.2 Katalogi

Struktura katalogów ISO9660 jest hierarchiczna. Na szczycie hierarchii znajduje się katalog główny (root). Na n-tym poziomie znajdują się katalogi umiejscowione w dowolnym z katalogów na poziomie n-1. Ze względu na kompatybilność z niektórymi systemami ISO9660 ogranicza dopuszczalną głębokość hierarchii do 8. Ograniczona jest również długość ścieżki (suma długości nazw katalogów na ścieżce, ich ilości i długości nazwy pliku) - nie może ona przekraczać 255.



Rysunek 7: Struktura katalogów ISO9660

Każdy katalog jest plikiem, w którym pamiętane są kolejne rekordy katalogu. Każdy z nich opisuje kolejny katalog lub plik należący do danego katalogu. Katalog zawiera też rekord katalogu dla swojego katalogu-rodzica.



Rysunek 8: Struktura rekordu katalogu

3.2.3 Tablica Ścieżek

Tablica ścieżek jest strukturą ułatwiającą znalezienie dowolnego katalogu na dysku ISO9660. Pozwala na odczytanie katalogu bez konieczności przechodzenia struktury katalogów od korzenia do tego katalogu.

3.3 Rozszerzenia ISO9660

Ponieważ ograniczenia narzucane przez ISO9660 sprawiają, że nie nadaje się on do niektórych zastosowań, opracowano szereg jego rozszerzeń (ang. extensions):

Apple ISO9660

System operacyjny MacOs potrzebuje dużo dodatkowych danych dla swojego graficznego interfejsu użytkownika. W systemie plików Appla, te dodatkowe informacje trzymane są w dwóch

miejscach: w strukturze opisującej katalog i w specjalnej części każdego pliku (resource fork). Te dodatkowe dane zawierają między innymi preferencje użytkownika, parametry okienka, układ graficzny menu, a w przypadku programów również kod wykonywalny. Pozostałe Dane pliku trzymane są w drugiej części (data fork), która odpowiada plikom w UNIX‘ach.
Nie ma problemu z zapisaniem w ISO9660 obu części plików. Data fork jest zapisywany jako normalny plik, a resource fork jako specjalny, skojarzony z nim plik.
Nie ma natomiast sposobu na zapisanie w standardowym ISO9660 części potrzebnych MacOs‘owi informacji przechowywanej normalnie w systemie plików HFS (np. wygląd ikonek, pozycje ikonek na ekranie, typ i atrybuty pliku). W tym celu Apple opracował rozszerzenie standardu ISO9660, które dla przechowywania informacji dla MacOs‘a wykorzystuje pole ’System Use Field’ w strukturze opisującej katalog.

Rockridge

Rock Ridge jest grupą firm, które od 1990 roku rozpoczeły prace nad przystosowaniem ISO9660 do użytku z systemami UNIX‘owymi. Propozycje grupy Rockridge dotyczą dwóch głównych dziedzin:

- Określenia sposobu na umieszczenie kilku niezależnych rozszerzeń ISO9660 w jednym systemie plików ISO9660
- Określenia sposobu na umieszczanie plików i katalogów zgodnych z POSIX w systemie plików ISO9660

Użycie rozszerzenia Rock Ridge pozwala na nagranie UNIX‘owej struktury katalogów (o głębokości większej niż podstawowe 8), wraz z uprawnieniami, bez żadnej straty informacji. W szczególności Rock Ridge umożliwia zastosowanie długich nazw plików.

Rock Ridge System Use Sharing Protocol (SUSP)

SUSP jest standardem umożliwiającym jednoczesne wykorzystanie wielu rozszerzeń korzystających z pola System Use Field (SUF) w rekordzie katalogu. SUSP określa ogólny format danych w każdym SUF, oraz określa zasady tworzenia więcej niż jednego SUF na rekord katalogu.

Joliet

Umożliwia stosowanie długich nazw (do 64 znaków), również w unikodzie. Jest to rozszerzenie używane przez rodzinę 32 bitowych systemów Windows.

ECMA 168

Propozycje Frankfurt Group dotyczą Standardu dla aktualizowalnych dysków ISO9660, są one dość elastyczne ale skomplikowane i trudne w implementacji.

Updatable ISO9660

Jest znacznie prostszą w implementacji alternatywą dla propozycji Frankfurt Group. Nagrywanie informacji odbywa się w sesjach. Podczas nagrania każdej nowej sesji, nagrywana jest całkowicie nowa struktura ISO9660. Napęd musi potrafić obsługiwać płyty z multisesjami, żeby 'widzieć' zmiany powstałe w kolejnych sesjach.

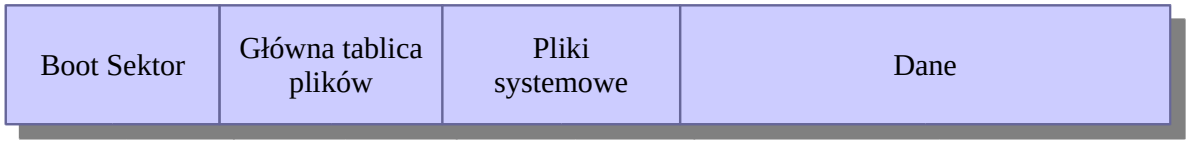
4 NTFS

System plików NTFS jest kolejnym system plików po FAT'cie, który powstał w firmie Microsoft. Jego powstanie wiąże się z projektem stworzenia przez Microsoft *poważnego* systemu operacyjnego który mogłby, jeśli nie wyprzeć, to przynajmniej konkurować z systemami UNIX-owymi i innymi systemami do zastosowań profesjonalnych.

Tak na początku lat 90-tych rozpoczęto prace nad nowym systemem operacyjnym Windows NT™. FAT nie nadawał się do zastosowania w systemie wieloużytkownikowym, chociażby z powodu nie obsługiwania żadnego systemu uprawnień do plików. W związku z tym musiano opracować od podstaw nowy system plików.

Jako że cały system miał być ładny, szybki i wogole fajny ;), wymyślono mu chwytliwą nazwę **New Technology**. Stąd nazwa systemu plików to **New Technology File System**

4.1 Struktura Partycji



Rysunek 9: Schemat NTFS

Na rysunku 9 pokazany jest ogólny schemat partycji NTFS. Nie należy sugerować się za bardzo kolejnością występowania stref na rysunku gdyż w rzeczywistości 3 ostatnie są przemieszane ze sobą.
Wyróżnione są strefy:

Boot sektor Skok do kodu ładującego system operacyjny, opis systemu plików

MFT Główna Tablica Plików.

Pliki Systemowe Patrz tabela 2

Pliki Dane plików użytkownika

4.2 MFT *Master File Table*

Informacja o wszystkich plikach zapisanych na partycji NTFS przechowywana jest w specjalnej strukturze danych określanej mianem **Głównej Tablicy Plików (ang. MFT)**. Informacja o umiejscowieniu jej początku przechowywana jest Boot Sektorze partycji NTFS.

Plik zależnie od wielkości i jego fragmentacji na dysku może używać jednego lub większej ilości wpisów w tablicy MFT. Jest on identyfikowany na partycji NTFS przez numer indeksu w tablicy MFT zawierający jego metadane.

Pierwsze 15 wpisów w MFT jest zarezerwowane dla specjalnych plików wykorzystywanych do trzymania dodatkowych informacji o systemie plików. W szczególności sama MFT jest reprezentowana jako plik w systemie (informacje o rozmieszczeniu MFT na dysku są trzymane w samej MFT), dzięki czemu struktura dopasowuje swoją wielkość do ilości danych przechowywanych na dysku. Lista zarezerwowanych plików przedstawiona jest w tabeli 2.

Plik	**Nazwa Pliku**	**Indeks w MFT**	**Opis**
MFT	$Mft	0	
Kopia MFT	$MftMirr	1	Duplikat pierwszych 4 sektorów MFT przechowywany dla bezpieczeństwa w środkowej części dysku.
Dziennik	$LogFile	2	Dziennik transakcyjny na wypadek awaryjnego przywracania spójności partycji (patrz punkt 4.5).
Opis woluminu	$Volume	3	Informacje dotyczące partycji takie jak nazwa partycji, wersja…
Definicje atrybutów	$AttrDef	4	Tablica nazw, numerów i opisów atrybutów, które mogą dotyczyć poszczególnych wpisów w MFT.
Katalog główny	. (kropka)	5	Plik zawierający katalog główny partycji.
Bitmapa zajętości	$Bitmap	6	Bitmapa zawierająca informacje, które klastry na partycji są zajęte, a które wolne.
Plik startowy	$Boot	7	Plik zawierający informacje niezbędne do wystartowania systemu z partycji.
Zepsute klastry	$BadClus	8	Informacje o zespsutych klastrach na partycji.
Zabezpieczenia	$Secure	9	?????????
Tablica konwersji wielkości znaków	$Upcase	10	Tablica odpowiadających sobie małych i wielkich liter w standardzie Unicode.
Rozszerzenia	$Extend	11	
		12-15	Zarezerwowane na przyszły użytek.

Tablica 2: Pliki systemowe NTFS

4.2.1 Wpisy w MFT

Jak była o tym mowa wyżej 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.
- Atrybutów, z których każdy ma *nagłówek* i ewentualnie *dane*. Lista jest uporządkowana zgodnie z numerami ID atrybutów.

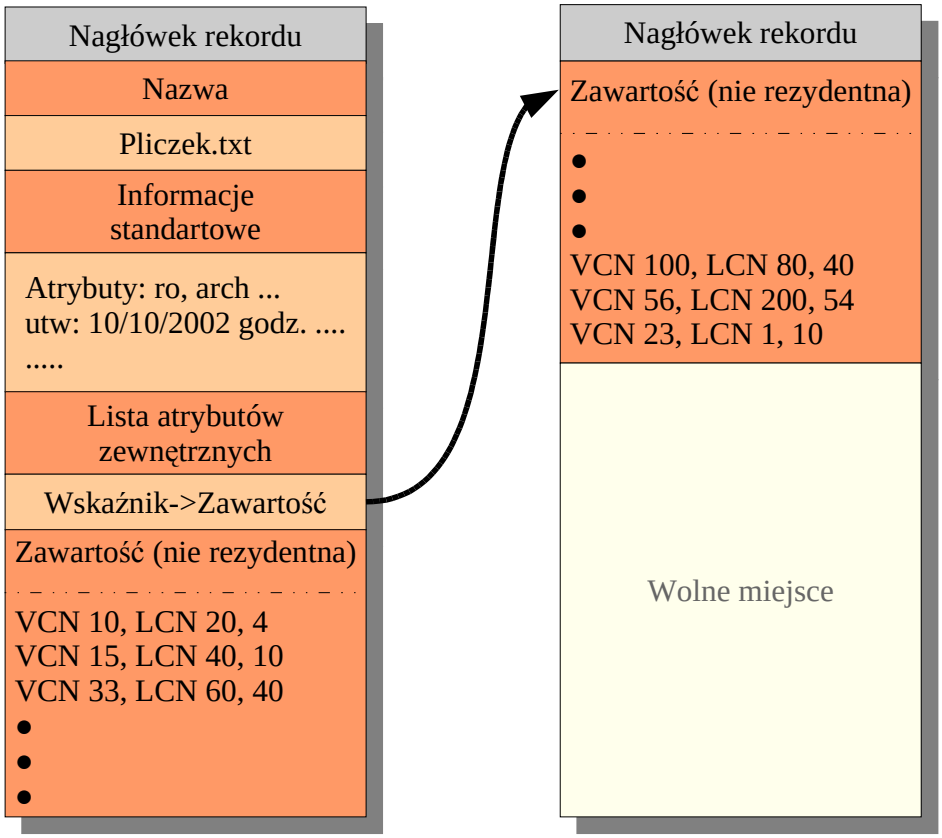
Dane atrybutu mogą być przechowywane wewnątrz rekordu MFT (wtedy mówimy o atrybutach *rezydentnych*) lub poza rekordem (wtedy mówimy o atrybutach *nie rezydentnych*).

Jeśli lista atrybutów nie mieści się w jednym rekordzie MFT wtedy na potrzeby pliku rezerwowany jest kolejny rekord.

Lista możliwych atrybutów przedstawiona jest w tabeli 3. Użytkownik może dodawać plikom nowe atrybyty. 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	Nazwa pliku
0x40	$Volume_version	Nie używane - usunięte w Windows 2000™.
0x40	$Object_id	Dodane w Windows 2000™. GUID (*Global Unique ID*) przypisany wpisowi.
0x50	$Security_descriptor	
0x60	$Volume_name	
0x70	$Volume_information	Nie używane - usunięte w Windows 2000™.
0x80	$Data	Dane 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 - usunięte w Windows 2000™.
0xC0	$Reparse_point	Dodane w Windows 2000™. ????
0xD0	$Ea_information	Dla zgodności z HPFS (używane przez podsystem OS/2 Windowsa i klientów Windows opartych na OS/2).
0xE0	$Ea	j.w.
0xF0	$Property_set	
0x100	$Logged_utility_stream	Dodane w Windows 2000™. Diagnostyczny strumień danych.

Tablica 3: Możliwe atrybuty wpisu w MFT



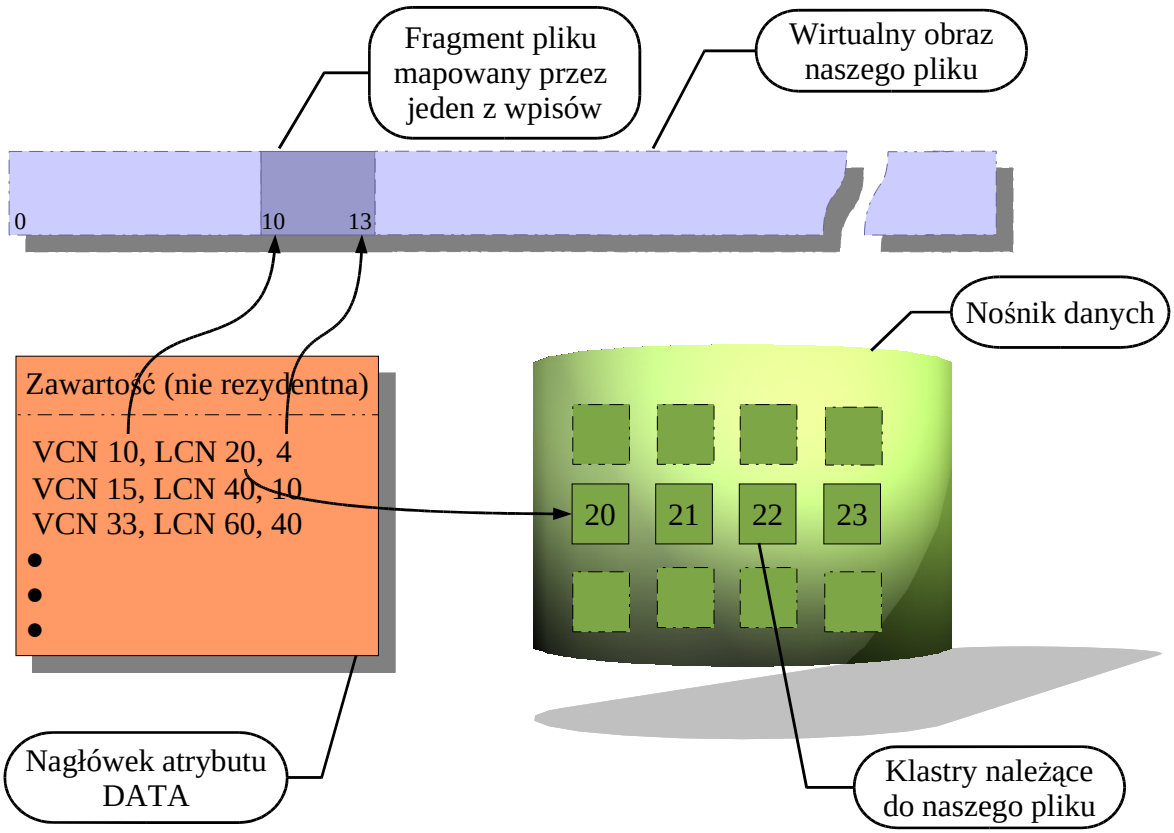
Rysunek 10: Wpis w MFT w NTFS

4.3 Odnajdywanie danych na dysku

W przypadku atrybutów rezydentnych odnajdywanie danych na dysku jest proste - po prostu sięgamy w opowiednie miejsce wpisu w MFT. W przypadku atrybutów nierezydentnych informacja o tym gdzie na dysku znajdują się dane przechowywana jest standartowo - w postaci przedziałów (ang. *data runs*).

Pamiętamy spójne przedziały bloków dyskowych i miejsce w którym powinny być one *wklejone* w logiczny obraz pliku. Ta metoda jest zobarazowana na rysunku 11.

Struktura wpisów na rysunku jest uproszczona w stosunku do rzeczywistości, dla ustalenia uwagi. Na prawdę w każdym wpisie przechowujemy informacje jakiej długości są poszczególne liczby określające miejsce na dysku i w pliku i nie przechowujemy wskaźników absolutnych lecz relatywne do poprzedniego wpisu. Z dokładnością do technicznych niuansów jest to to samo co pokazuje rysunek.



Rysunek 11: Kodowanie informacji i położeniu pliku w NTFS

4.4 Katalog

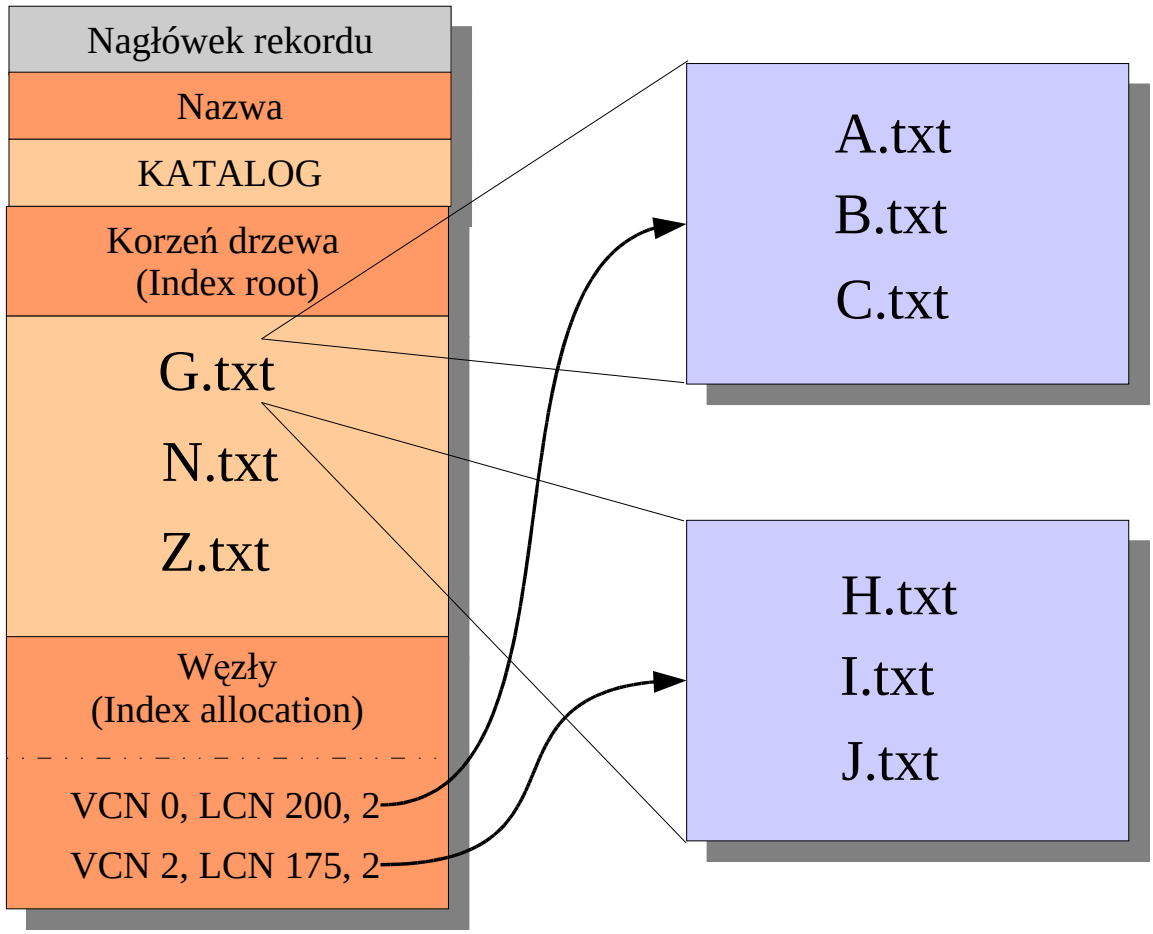
Do przechowywania danych katalogowych wykożystuje się 3 rodzaje atrybutów w MFT:

- $Index_root - korzeń drzewa reprezentującego katalog
- $Index_allocation - przechowuje węzły drzewa katalogu
- $Bitmap - kontrola wolnych miejsc w katalogu

W NTFS dane katalogowe przechowywane są 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 przechowywane są 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.

Przykład drzewa katalogu NTFS zamieszczono na rysunku 12.



Rysunek 12: Wpisy w katalogu NTFS

4.5 Dziennik

NTFS jak posiada dziennik tranzakcyjny. Jego struktura nie jest udostępniana przez Microsoftwięc pominę szczegóły implementacyjne. Wążna jest idea po co tworzy się coś takiego jak dziennik w systemie plików.

4.5.1 Idea dziennika

Nowoczesne systemy plików często używają dla zwiększenia bezpieczeństwa (tj. zmniejszenia ryzyka utraty danych i rozspójnienia systemu plików) dzienników transakcyjnych.

W przypadku awarii lub zawieszenia systemu dane na dysku mogą ulec rozspójnieniu. Wynika to z tego, że zwykle dopisanie lub poprawienie jakiejś informacji na dysku wymaga kilku fizycznych zapisów i może nie zostać w całości wykonane.

Systemy plików bez dziennika (takie jak FAT, ext2, HPFS) w takich sytuacjach, podczas startu systemu, są naprawiane specjalnym narzędziem (takim jak scandisc czy fsck), które odczytuje całość metadanych z dysku i stara się doprowadzić je do spójności. Jak łatwo się domyślić taki proces jest czasochłonny (trzeba przeczytać dużo danych z dysku) i podatny na błędy. Zależnie od systemu plików częściej bądź rzadziej może zdarzyć się sytuacja, że nie da się przywrócić spójnej struktury dysku. To prowadzi do utraty części danych i konieczności odbudowy stanu dysku na podstawie kopii zapasowej, co z kolei wiąże się z poniesieniem znacznych strat czasowych oraz ze stratą danych zapisanych na nośniku od momentu zrobienia backup-u do chwili obecnej.

Aby rozwiązać ten problem stosuje się dzienniki transakcyjne. Rozwiązanie to polega na grupowaniu operacji zapisu na dysku w niepodzielne transakcje (zależnie od systemu plików dotyczy to metadanych i/lub danych). Przed rzeczywistą zmianą części danych na dysku dokonywany jest wpis do dziennika. W przypadku awarii systemu, przy pierwszym uruchomieniu przeglądany jest dziennik transakcyjny i wszystkie niedokończone transakcje są wycofywane z nośnika do stanu sprzed ich rozpoczęcia.

Takie rozwiązanie jest znacznie szybsze (nie wymaga przeglądania całego dysku lecz jedynie przeczytania dziennika) i bezpieczniejsze niż skanowanie dysku w przypadku systemów plików bez dziennika. Niesie jednak pewne narzuty wydajnościowe, gdyż część operacji zapisu na dysk może wymagać dodatkowych zapisów do dziennika.

4.6 Kompresja

4.7 Szyfrowanie

Począwszy od wersji NTFS która pojawiła się z Windows 2000™ system plików udostępnia szyfrowanie na poziomie plików i katalogów.

W rzeczywistości szyfrowanie przesunięte jest z systemu plików o poziom wyżej.
W skrócie, Windows 2000™ wykorzystuje w tym celu dodatkowy moduł, blisko związany z sterownikiem od systemu plików (*EFS - Encrypting File System*). Gdy użytkownik chce aby jego plik został zaszyfrowany, sterownik NTFS przesyła dane do EFS, a ten pobiera z systemu klucz użytkownika i szyfruje dane przy pomocy algorytmu DES (*Data Encryption Standart*).

Deszyfrowanie danych przebiega analogicznie.

5 JFS

5.1 Wstęp

Po omówieniu systemów plików stosowanych w produktach firmy Microsoft przyszła kolej na omówienie systemów plików stosowanych w systemach UNIX-opodobnych.

JFS czyli Journaled File Sysytem powstał w 1990 roku w firmie IBM i był ściśle związany z unixowym systemem AIX 3.1. W 1995 roku zaczęto ueslastyczniać JFS-a oraz dodano mu wsparcie dla maszyn wieloprocesorowych. Zaczęto też poważnie myśleć o jego adaptacji dla innych systemów operacyjnych.

W grudniu 1999 roku trwały prace nad trzema systemami plików z dziennikiem przeznaczonymi dla systemu Linux. Ext2 był w trakcie dodawania dziennika do swojego systemu plików w wyniku czego powstał Ext3. Silicon Graphics Inc. zaczęło właśnie adaptować XFS-a, przeznaczonego dotychczas dla systemu operacyjnego IRIX. Dodatkowo Hans Reiser zaczął tworzyć nowy system nazwany później ReiserFS. Żaden z nich nie był jeszcze w pełni funkcjonalny. IBM wierzył, że przeniesienie tak silnego (ich zdaniem) narzędzia jakim był system JFS przyniesie Linuxowi wiele korzyści.

Trzy główne cele IBM-a związane z JFS-em:

1. JFS ma być systemem na licencji GPL.

2. Adaptacja JFS-a nie powinna wymagać żadnych zmian w jądrze (tzn. występuje jako osobny moduł).

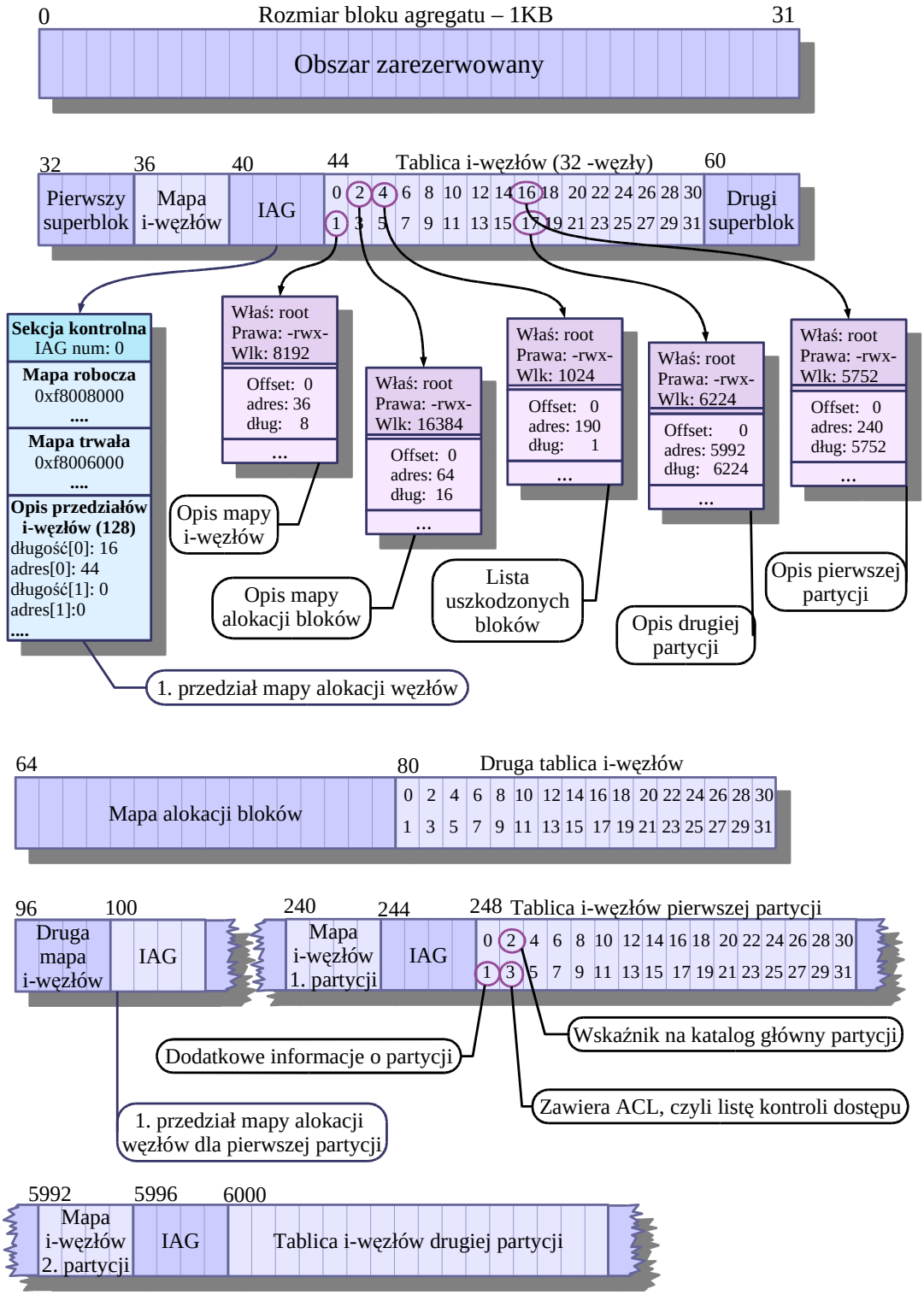
3. JFS powinien być dostępny dla wszystkich architektur wspieranych przez Linuxa.

W lutym 2000 roku wyszła pierwsza wersja kodu źródłowego JFS-a dla Linuxa. Zawierała tylko 3 funkcje: mount, unmount oraz ls.

5.2 Agregaty

Poprzez agregat (ang. *aggregate*) rozumiemy nośnik danych, taki jak dysk twardy czy macierz dyskowa. Natomiast partycja (ang. *fileset*) oznacza montowalny zbiór plików i katalogów. Nie należy mylić tejże partycji z fizyczną partycją na nośniku (tworzoną za pomocą FDISK-a). Jeden agragat może być związany z jedną lub większą ilością partycji. Jedna partycja jest przypisana do dokładnie jednego agregatu.

Dla każdego agregatu ustala się rozmiar bloku (ang. *aggregate block size*). Może on przyjmować jedną z wartości: 512, 1024, 2048 lub 4096 bajtów, jednak nie może być ona mniejsza od rozmiaru bloku partycji FDISK-owej (ang. *partition block size*, naogół 512B). Rozmiar bloku agregatu definiuje najmniejszą jednostkę jaką można zaalokować.



Rysunek 13: Struktura agregatu i partycji w JFS

Struktura agregatu

- 32K obszar zarezerwowany (na początku agregatu)
- Pierwszy i drugi superblok (ang. *Primary/Secondary Aggregate Superblock*)
  Trzymane są tu ogólne informacje dotyczące agregatu takie jak jego wielkość, rozmiar bloku, rozmiar grupy itp. Drugi superblok jest kopią pierwszego i przejmuje jego funkcje w przypadku awarii.
- Dwie tablice i-węzłów (ang. *Aggregate Inode Table*) - druga tablica jest kopią pierwszej.
  Jedna tablica zawiera 32 i-węzły opisujące struktury kontrolne agragatu:
  - i-węzeł 0 - zarezerwowany
  - i-węzeł 1 - opisuje mapę i-węzłów
  - i-węzeł 2 - opisuje mapę alokacji bloków
  - i-węzeł 3 - opisuje położenie log-a
  - i-węzeł 4 - opisuje położenie uszkodzonych bloków, odkrytych podczas formatowania agregatu. Bloki te są oznaczane jako zaalokowane.
  - i-węzły 5 - 15 - zarezerwowane na potrzeby dalszego rozwoju JFS-a
  - i-węzeł 16 i dalsze - opisują kolejne partycje. Wraz z dodawaniem nowych partycji tablica i-węzłów może się rozszerzać.
- Dwie mapy i-węzłów (ang. *Aggregate Inode Map*)
  Opisują tablice i-węzłów agregatu. Zawierają stan alokacji oraz położenie i-węzłów na nośniku.
- Mapa alokacji bloków (ang. *Block Allocation Map*)
  Zawiera stan alokacji wszystkich bloków agregatu.
- Obszar roboczy (ang. *fsck Working Space*)
  Dodatkowe miejsce na końcu agregatu dla fsck.
- Log (ang. *In-Line Log*)
  Miejsce przeznaczone do zapisu zmian metadanych.

5.3 Struktura partycji

- Tablica i-węzłów partycji (ang. *Fileset Inode Table*)
  - i-węzeł 0 - zarezerwowany
  - i-węzeł 1 - zawiera dodatkowe informacje o partycji nie zawarte w tablicy i-węzłów agregatu
  - i-węzeł 2 - zawiera wskaźnik na katalog główny partycji

  - i-węzeł 3 - ACL (ang. *Access Control List*)
  - i-węzeł 4 i dalsze - i-węzły używane przez pliki, katalogi, linki itp.

- Mapa alokacji i-węzłów (ang. *Fileset Inode Allocation Map*)
  Opisuje stan alokacji i-węzłów należących do partycji oraz ich fizyczne położenie.

5.4 Grupy alokacji

Grupy alokacji (ang. *AG = Allocation Group*) dzielą przestrzeń agregatu na na kawałki, w celu zwiększenia wydajności operacji we/wy. Polityka JFS-a, związana z efektywną alokacją i-węzłów:

1. JFS stara się grupować bloki dyskowe i i-węzły dla powiązanych ze sobą danych, ponieważ
   - często dostęp do plików jest sekwencyjny
   - często pliki z tego samego katalogu są używane jednocześnie
2. JFS stara się rozrzucić dane nie powiązane ze sobą po całym agregacie, aby w przypadku pojawienia się danych powiązanych można im było przydzielić miejsce gdzieś niedaleko.

Liczba grup alokacji jest ograniczona do 128. Wielkość jednej grupy musi być >= 8K bloków oraz musi być postaci 2^n* ilość bloków opisanych w jednej strukturze *dmap*. Przechowywana jest ona w superbloku.

5.5 Przedziały

Przedział (ang. *extent*) jest to spójny ciąg bloków traktowany (i alokowany) przez JFS jako jeden obiekt. Jego długość jest zmienna i może przyjmować wartości od 1 do 2^{24}-1 bloków.

Jeden przedział przypisany jest do dokładnie jednej partycji, jednak może się on rozciągać na wiele grup alokacji. Określają go dwie wartości: 24-bitowa długość oraz adres fizyczny. Przyjmując za wielkość bloku wartości 512B oraz 4096B otrzymujemy maksymalny rozmiar przedziału równy odpowiednio 8G i 64G.

JFS stara się zmaksymalizować alokowanie ciągłych obszarów, alokując jak najmniej jak największych przedziałów. Powoduje to znaczne przyspiesznie operacji odczytu i zapisu na dysk.

Wykorzystanie przedziałów w połączeniu z małym rozmiarem bloku (512B) pomaga obniżyć fragmentację wewnętrzną (ang. *Internal fragmentation*) nawet dla systemów z dużą ilością małych plików.

5.6 i-węzły

Każdy obiekt występujący w JFS-ie reprezentowany jest przez i-węzeł. i-węzeł zawiera specyficzne informacje o danym obiekcie oraz B^+ drzewo przechowujące przedziały z danymi obiektu.

Każdy i-węzeł liczy sobie 512 bajtów i składa się z czterech części:

- Pierwsza opisuje podstawowe atrybuty obiektu takie jak:
  - znacznik czy i-węzeł należy do partycji
  - numer partycji
  - deskryptor przedziału do którego należy
  - rozmiar i liczba zaalokowanych bloków
  - id właścicieli (użytkownika i grupy)
  - prawa dostępu
  - czas utworzenia, ostatniej modyfikacji, dostępu
- Druga zawiera informacje specyficzne dla konkretnego systemu operacyjnego oraz nagłówek B^+ drzewa
- Trzecia zawiera albo deskryptory do przedziałów albo deskryptory do korzeni B^+ drzew albo ciąg danych
- Czwarta zawiera rozszerzone atrybuty, kolejne deskryptory przedziałów lub dalszą część ciągu danych

W JFS-ie i-węzły są alokowane dynamicznie. Wiąże się z tym kilka zalet:

- i-węzły mogą być umieszczane w dowolnym miejscu na dysku. Ich identyfikator nie jest na sztywno związany z położeniem i dzięki temu zmniejszenie rozmiaru agragatu nie stanowi żadnego problemu.
- Nie ma potrzeby alokować "10 razy więcej i-węzłów niż będzie nam potrzebne" co jest szczególnie ważne przy tak dużym rozmiarze i-węzłów.
- Bardzo duże pliki zajmujące kilka grup alokacji mogą pozostać ciągłe, tymczasem w wersji statycznej na początku każdej grupy byłaby "dziura" zaalokowana dla i-węzła podczas inicjalizacji grupy.

Główną wadą dynamicznej alokacji i-węzłów jest przymus użycia odrębnej struktury mapującej i-węzły. W JFS-ie rolę tę odgrywa mapa alokacji i-węzłów.

i-węzły są alokowane w przedziałach i-węzłów. Każdy przedział i-węzłów liczy sobie 32 i-węzły i zajmuje 16KB.

5.7 B^+ drzewa

B^+ drzewa opisują rozmieszczenie danych przechowywanych w obiektach systemu JFS. B^+ drzewa zostały wybrane do tego celu z powodów wydajnościowych, tzn. by umożliwić jak najszybsze szukanie, czytanie i pisanie do przedziałów. Dodatkowo umożliwiają efektywne wstawianie i

usuwanie przedziałów z drzewa (czyli zmianę wielkości danych przechowywaniych w obiekcie).

Korzeniami B^+ drzew są deskryptory alokacji przedziałów (ang. ***E**xtent **a**llocation **d**escriptor*), nazywane strukturami *xad*. Każda taka struktura składa się z pól:

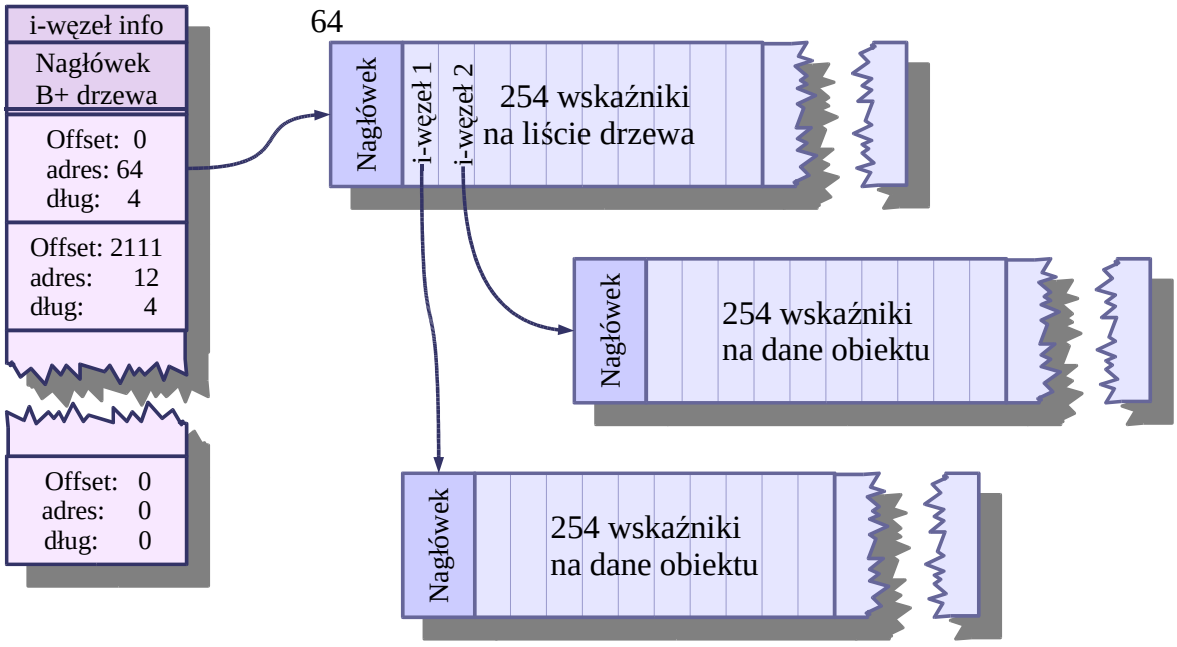
addr1/addr2 - adres fizyczny przedziału w agregacie

len - długość przedziału

off1/off2 - adres logiczny w obiekcie

flag - flagi dotyczące kompresji, zajętości itp

Węzły wewnętrzne B^+ drzewa są to tablice zawierające 254 deskryptory przedziałów (*xad*) oraz nagłówek wskazujący na pierwszy wolny przedział w tablicy (wszystkie po nim następujące także są wolne). Liście B^+ drzewa zawierają bezpośrednie wskaźniki na dane obiektu, a jego kluczami są adresy logiczne przedziałów. Maksymalna liczba liści przekracza 526 tysięcy.



Rysunek 14: B^+ drzewa w i-węzle w JFS

5.8 Mapa alokacji bloków

Mapa alokacji bloków opisuje zajętość bloków całego agragatu. Wykorzystuje się do tego pełne drzewo, w którym każdy z węzłów ma po 1024 synów. Zmiany zachodzące w mapie nie są zapisywane w dzienniku.

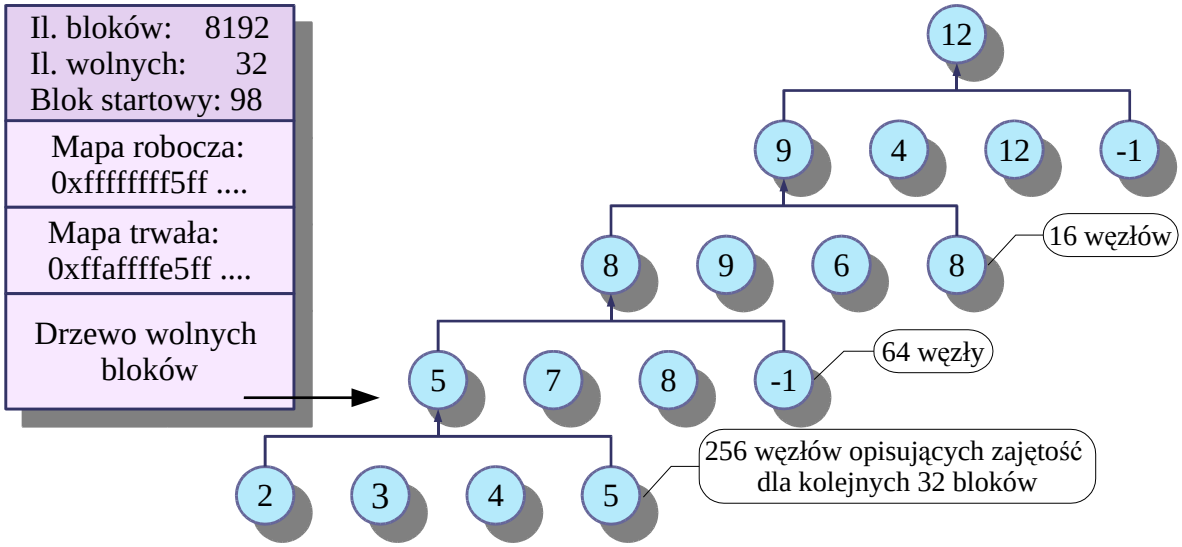
Jeśli agregat zaiwera kilka partycji to dzielą one jedną mapę alokacji bloków.

5.8.1 Węzły wewnętrzne

Węzłami wewnętrznymi drzewa są strony kontrolne (ang. *control pages*) zawierające informacje o numerze startowym bloku oraz najdłuższym ciągu pustych bloków wśród jego synów (oblicza się to algorytmem bliźniaków z 1024 liśćmi).

5.8.2 Liście

W liściach drzewa znajdują się struktury *dmap*, opisujące zajętość 8K bloków agregatu.



Rysunek 15: Struktura liścia (czyli struktura *dmap*) w JFS

Mapy Mapa robocza zawiera aktualny stan bloków opisywanych w danym liściu, natomiast mapa trwała jedynie zmiany zatwierdzone albo przez transakcje JFS albo poprzez sprawdzenie stanu na dysku lub w logu.

Drzewo wolnych bloków

Każdy liść tego drzewa dotyczy 32 kolejnych bloków przechowywanych przez strukturę *dmap*. Trzymana jest w nim wartość opisująca rozmiar najdłuższego ciągu pustych bloków zaczynającego się w danym liściu i będącego potęgą dwójki. Generowana jest ona algorytmem bliźniaków.

Każdy z węzłów wewnętrznych trzyma maksymalny rozmiar ciągu pustych bloków wyliczony na podstawie swoich czterech synów.

Algorytm bliźniaków dla 32 bitowych słów

W zerowym kroku odczytujemy wartości poszczególnych bitów z mapy roboczej i jeśli dany blok jest pusty to wpisujemy 0 wpp -1.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Mapa	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bliźniak	0	0	-1	-1	-1	0	0	0	0	0	-1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2^1	1		-1		0		1		1		-1		1		1		1		1		1		1		1		1		1		1	
2^2	1				1				1				2				2				2				2				2			
2^3	1								2								3								3							
2^4	2																4															
2^5	4																															

Rysunek 16: Algorytm bliźniaków dla słów 32 bitowych w JFS

W *n*-tym kroku bierzemy dwie kolejne wartości.

- Jeśli obie mają wartość -1 (nie ma żadnych pustych bloków) to wpisujemy -1
- Jeśli liczby są różne to wpisujemy wartość większej z nich
- Jeśli obie liczby są równe *n* (oba kawałki są zupełnie puste)) to wpisujemy *n*+1

Wartość uzyskana w piątym kroku wskazuje potęge dwójki, będącą liczbą elementów ciągu pustych bloków.

Algorytm bliźniaków dla wszystkich liści

Wartości uzyskane dla grup 32 bloków tworzą 256-cio elementowy ciąg. Wykonuje sie na nim kolejną część algorytmu - znajdowanie najdłuższego sekwensu wolnych bloków zaczynających się w konkretnej grupie 32 bloków.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dł. ciągu	2	3	4	5	5	5	5	5	5	5	5	5	5	5	5	5
2^6	2		4		6	-1	6	-1	6	-1	6	-1	6	-1	6	-1
2^7	2				7		-1		7		-1		7		-1	
2^8	2								8							
2^9	2															
Wynik	2	3	4	5	7	-1	-1	-1	8	-1	-1	-1	-1	-1	-1	-1

Rysunek 17: Algorytm bliźniaków - druga faza (wersja skrócona) w JFS

Uzyskane w ten sposób wartości zapisywane są w liściach drzewa wolnych bloków.

5.8.3 Rozszerzanie agregatu

JFS oferuje możliwość powiększania agregatu “w locie”, jednak najpierw musi on sprawdzić czy na naszym agregacie jest wystarczająco dużo miejsca do zaadresowania dodatkowej przestrzeni. Do tego właśnie celu stosuje sie mapę alokacji bloków.

Jeśli miejsca jest za mało to najpierw przyłączana jest taka część przestrzeni, na którą starcza miejsca. W ten sposób zyskujemy na agregacie dodatkową liczbę bloków mogących służyć do zaadresowania reszty przestrzeni.

5.8.4 Alternatywa dla algorytmu bliźniaków

W tym przypadku węzły składają się z czterek 32-bitowych wejść. Każde wejście posiada typ, wielkość i bitmapę. Typ reprezentuje stan alokacji 32 kolejnych bloków i może przyjmować następujące wartości: wolny (W), zajęty (Z), reprezentowany przez bitmapę stanu alokacji bloków (B) lub “bez znaczenia” (BZ). Jeśli typ jest “bez znaczenia” to bloki są opisane przez swoich lewych sąsiadów i ich rozmiar nie gra roli.

1 128 pustych bloków

Wlk	Typ	Bitmapa	Bloki
7	W		0-31
	BZ		32-63
	BZ		64-95
	BZ		96-127

2 64-95 zaalokowane przez A

Wlk	Typ	Bitmapa	Bloki
6	W		0-31
	BZ		32-63
5	Z		64-95
5	W		96-127

3 96-127 zaalokowane przez B

Wlk	Typ	Bitmapa	Bloki
6	W		0-31
	BZ		32-63
5	Z		64-95
5	Z		96-127

4 0-63 zaalokowane przez C

Wlk	Typ	Bitmapa	Bloki
6	Z		0-31
	BZ		32-63
5	Z		64-95
5	Z		96-127

5 61-63 oraz 64-95 zwolnione

Wlk	Typ	Bitmapa	Bloki
5	Z		0-31
	BZ	0xffffff8	32-63
5	W		64-95
5	Z		96-127

6 96-127 zwolnione

Wlk	Typ	Bitmapa	Bloki
5	Z		0-31
	B	0xffffff8	32-63
6	W		64-95
	BZ		96-127

Rysunek 18: Alternatywny algorytm bliźniaków w JFS

Dla każdego wolnego wejścia, jeśli jego lewy sąsiad jest wolny i ma ten sam rozmiar to typ prawego wejścia jest przekształcany na “bez znaczenia” a wielkość lewego wejścia jest

odpowiednio zwiększana.

Na rysunku przedstawiano działanie algorytmu dla liści złożonych z czterech grup 32-bitowych. W rzeczywistości wykorzystuje się go dla 256 grup (dla danych struktury *dmap*).

Po wykonaniu algorytmu, wartości z kolejnych czterech 32-bitowych grup są przekazywane do węzłów wewnętrznych (typ "bez znaczenia" i zajęty przekazują -1), które wybierają największą liczbę i przekazują ją wyżej.

Tak samo postępuje się w przypadku stron kontrolnych.

5.9 Alokacja i-węzłów

W związku z dynamiczną alokacją i-węzlów, numer i-węzła nie wskazuje na konkretny blok agregatu i trzeba zapewnić rozwiązanie dla 3 problemów:

1. Szukanie położenia i-węzła na podstawie jego numeru

2. Szukanie najbliżej położonych wolnych i-węzłów na podstawie numeru bloku partycji. JFS często stara się dane z jednego katalogu umieszczać niedaleko siebie.

3. Szukanie i-węzłów nie przyporządkowanych do żadnego przedziału i-węzłów.

5.9.1 Mapa alokacji i-węzłów

Rozwiązuje problem pierwszy. Jest to B^+ drzewo , w którego liściach znajdują sie struktury IAG (ang. *inode allocation group pages*, dosłowne tłumaczenie: grupy alokacji i-węzłów). Indeksowane jest ono położeniem wyżej wspomnianych struktur.

Jedna struktura IAG opisuje 128 przedziałów i-węzłów, czyli $128 * 32 = 4096$ i-węzłów. Zawiera:

- mapy fizycznego położenia przedziałów i-węzłów
- mapy ich zajętości (mapa robocza i trwała)
- liczbę wolnych i-węzłów i przedziałów itp.

Dzięki temu możliwe jest znalezienie adresu przedziału, do którego należy i-węzeł, i jego pozycji w tym przedziale, co już jednoznacznie wyznacza położenie i-węzła (jak działa ten algorytm można zobaczyć w materiałach IBM-a).

Mapa alokacji i-węzłów jest transakcjonowana.

Pozostałe problemy związane z alokacją daje się rozwiązać przy pomocy kilku dodatkowych struktur, których już tutaj nie będę omawiać.

5.10 Obiekty JFS

Pliki (ang. *files*) Reprezentowane są przez i-węzeł zawierający B^+ drzewo, wskazujące na dane pliku.

Linki symboliczne (ang. *symbolic links*) Reprezentowane są przez i-węzeł z ustawionym odpowiednim typem (*S_IFLNK*). Cała ścieżka przechowywana jest w i-węzle, jeśli starczy miejsca lub w B^+ drzewie w przeciwnym przypadku.

Katalogi (ang. *Directories*) Katalogi są metadanymi, których zmiany są przechowywane w dzienniku.

Każdy katalog składa sie z wielu wejść (do katalogów i pików znajdujących się w tym katalogu), które nazwie katalogu przyporządkowują numer i-węzła.

Wejścia te trzymane są albo w samym i-węzle, jeśli jest ich dostatecznie mało, albo w B^+ drzewie, indeksowanym po nazwie obiektu. Liście B^+ drzewa przechowują kompletne nazwy i wskaźniki na obiekty katalogu.

W węzłach wewnętrznych zastosowana jest kompresja przyrostków, która obcina z nazwy obiektu tyle, żeby się dało go odróżnić od nazwy poprzedniego obiektu. Na przykład dla słów:
alamakota, alarm, bakalie, banan
zostaną wygenerowane przedrostki:
a, alar, b, ban.

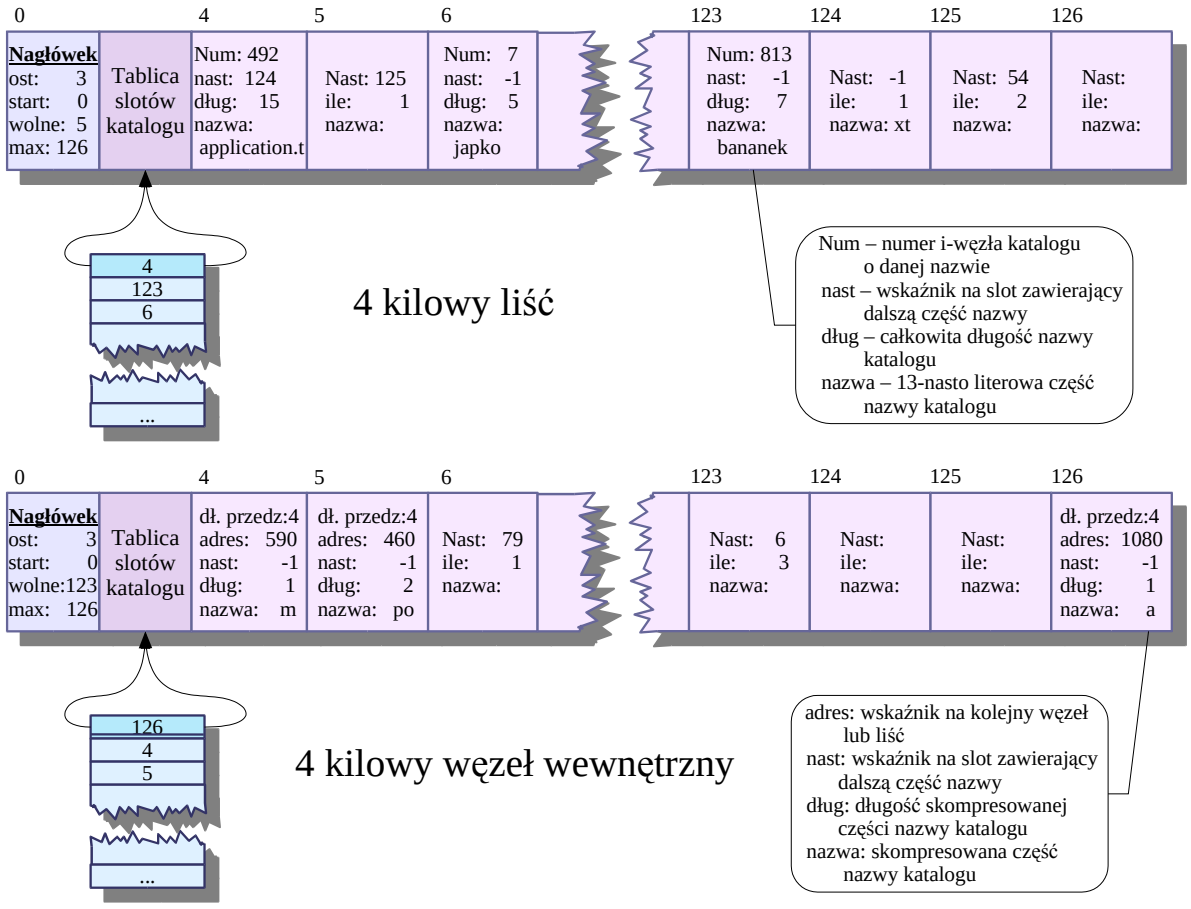
Każdy węzeł drzewa (rys. 19) w katalogu posiada nagłówek, zawierający

- flagi czy jest on liściem lub korzeniem
- pole start (ang. *stblindex*) - indeks pierwszego slotu używanego przez tablicę slotów katalogu
- pole ost (ang. *nextindex*) - indeks ostatniego slotu używanego przez tablicę slotów katalogu
- pole wolne (ang. *freelist*) - wskaźnik na pierwszy element listy pustych slotów
- pole max (ang. *maxslot*) - maksymalna liczba slotów w węźle

oraz tablicę slotów katalogu (ang. *Directory Slot Array*) zawierającą indeksy slotów, będących aktualnie w użyciu. Warto tutaj dodać, że tablica ta jest posortowana nie po samych indeksach a po nazwach slotów, na które te indeksy wskazują.

5.11 ACL czyli listy uprawnień

Z każdym i-węzłem partycji skojarzona jest lista uprawnień (ang. *Access Control List*). Zawiera ona informacje o uprawnieniach obiektu oraz jego właścicielach (użytkowniku i grupie). Katalogi jako jedyne mogą posiadać dwie listy uprawnień: jedną dla tworzonych w nim katalogów i drugą dla tworzonych w nim plików. Wszystkie operacje na listach uprawnień są zapisywane w dzienniku.



Rysunek 19: Węzły drzewa katalogu w JFS

6 ReiserFS

Innym systemem plików dedykowanym dla Linuxa jest ReiserFS. Jego twórca Hans Reiser jest zwolennikiem unifikowania nazw (i nazwanych obiektów) w jedną przestrzeń. W swoim systemie dążył do stworzenia jak najbardziej jednolitego interfejsu dostępu do różnych obiektów (takich jak katalogi, pliki, dane).

Ten artykuł omawia wersję 3 systemu plików ReiserFS. Na dniach powinna pojawić się wersja 4 która niesie sporo zmian takich jak:

- Przechowywanie danych w przedziałach (ang. *extents*)
- Zwiększona modularność dzięki wtyczkom (ang. *plugins*)
- Listy uprawnień (ang. *ACL*)

- ...[1]

6.1 Struktura Danych

Takie podejście zaprowadziło go do rozwiązania, w którym wszystkie dane na dysku (zarówno metadane katalogów i plików jak i same dane użytkownika) są przechowywane w jednej globalnej dużej strukturze danych. Jak nie trudno się domyślić tą strukturą danych jest słownik.
Autor chciał aby system był wydajny zarówno dla małych jak i dużych plików gdyż jak wynika z badań większość operacji (80%) odbywa się na plikach mniejszych niz 10KB. Hans Reiser twierdzi, że w rzeczywistości to wcale nie są małe pliki i gdyby systemy pozwalały na wydajne korzystanie z malutkich plików to większość odwołań dotyczyłaby plików wielkości 100B.

Autor postanowił do tego celu wykorzystać drzewa zrównoważone (B^+-drzewa). Nie zniechęcił go fakt, że zanim powstał jego projekt uważano że jest zbyt powolna i złożona struktura danych (w szczególności nie nadająca się do przechowywania dużych obiektów).

Jak się okazało, udało się jednak zaimplementować wydajny i bezpieczny system plików korzystający w zasadzie wyłącznie z tej struktury danych.

Pisząc ReiserFS przyświecała mu idea aby niezależnie od wielkości plików, na których się operuje i ich ilości w katalogu system działał szybko i efektywnie wykożystywał miejsce na dysku.

6.2 Ogólna struktura partycji

Rysunek .....

6.3 Drzewo - serce systemu

Jak pisałem na wstępie centralnym miejscem w którym przechowujemy dane jest B^+-drzewo postaci pokazanej na rysunku 20.

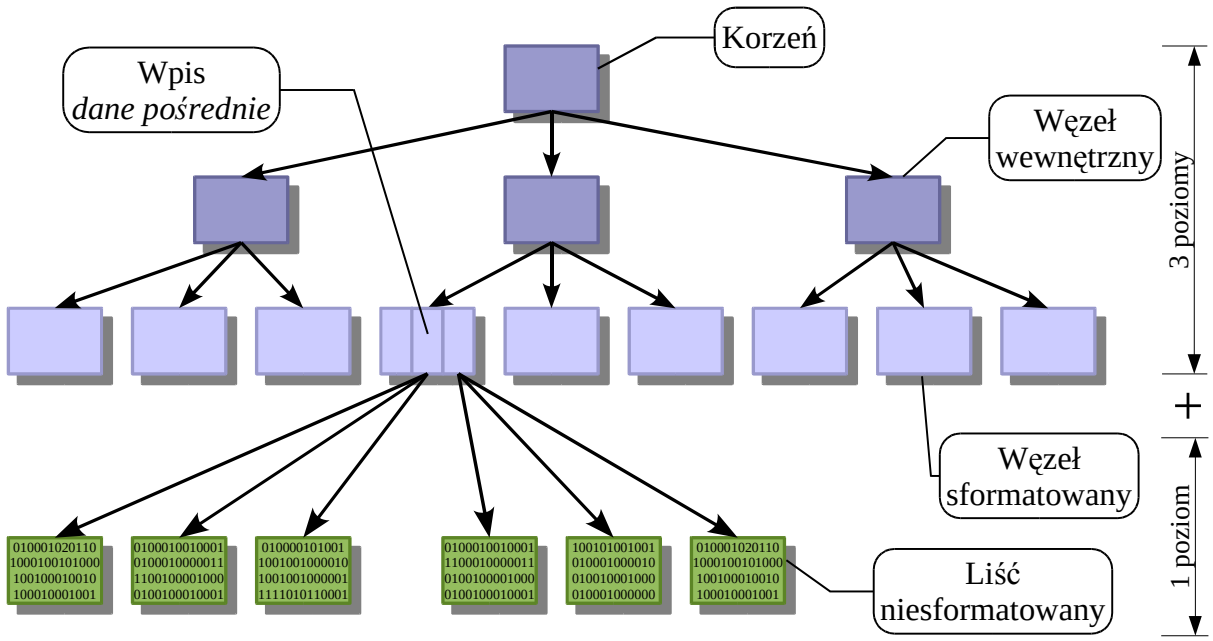
Wielkość węzła jest stała i zwykle jest ona równa wielkości bloku dyskowego czyli w Linuxie wynosci 4KB. Dane są przechowywane jedynie w liściach.

Porządek w drzewie ustalony zgodnie z kolejnością unikalnych kluczy przydzielanych przechowywanym węzłom. Długość klucza wynosi 16 bajtów. Stuktura klucza jest istotna i jest opisana poniżej.

6.3.1 Rodzaje węzłów w drzewie

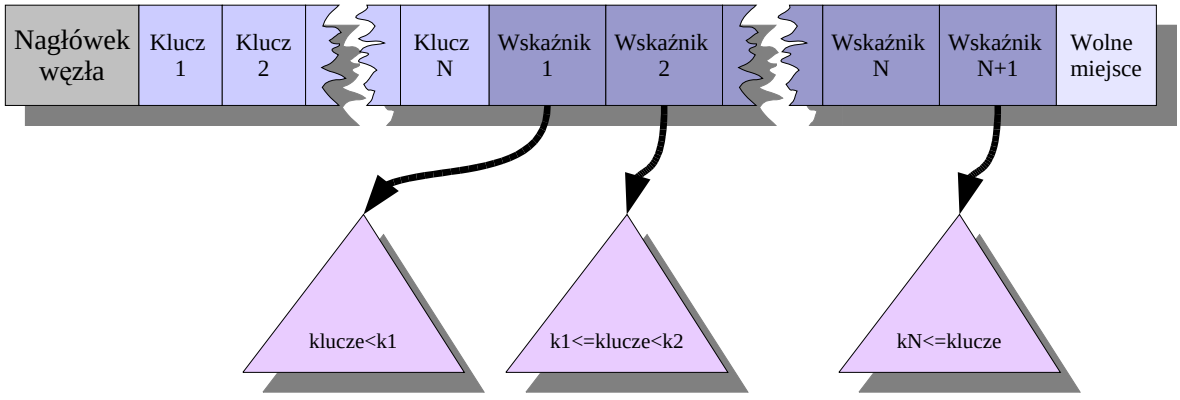
Wyróżniamy 3 rodzaje węzłów:

[1] Szczegóły w `http://www.namesys.com/v4/v4.html`



Rysunek 20: Drzewo główne w systemie ReiserFS

Węzły wewnętrzne (ang. *Internal nodes*) Nie zawierają danych lecz jedynie wskaźniki do węzłów o poziom niżej poprzedzielane kluczami.
Dla układu (k1, wsk, k2) wszystkie elementy w poddrzewie wskazywanym przez wsk mają klucze z przedziału [k2,k1). Struktura takiego węzła przedstawiona jest na rysunku 21

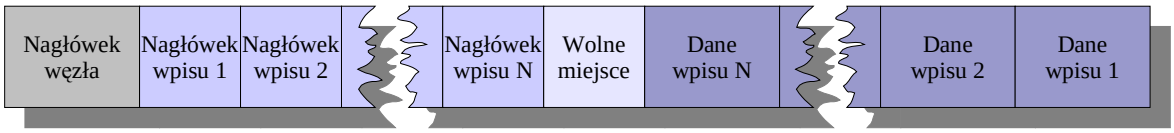


Rysunek 21: Struktura węzła wewnętrznego w RaiserFS

Węzły sformatowane (ang. *Formatted nodes*) Zawierają wpisy (ang. *items*). W 3 wersji ReiserFS wyróżniamy 4 rodzaje wpisów (dane bezpośrednie, dane pośrednie, wpisy katalogowe i wpisy informacyjne). Każdy wpis w węźle sformatowanym zawiera swój klucz.

Węzły sformatowane są bądź liśćmi bądź mają do siebie podczepione jeszcze liście niesformatowane. Nie występują na wyższych poziomach.

Każdy wpis w węźle sformatowanym ma swój nagłówek oraz dane. Struktura takiego węzła przedstawiona jest na rysunku 22



Rysunek 22: Struktura węzła sformatowanego w RaiserFS

Liście niesformatowane (ang. *Unformatted nodes*) Nie mają żadnej struktury - służą do przechowywania ciągu bajtów. W Liściach niesformatowanych nie ma kluczy.
Długość ścieżki do liścia niesformatowanego jest zawsze o 1 dłuższa niż długość ścieżki do węzła sformatowanego (jest to błąd projektowy powodujący spadek wydajności i w wersji 4 systemu ReiserFs zarówno wezły sformatowane jak i węzły niesformatowane będą zawsze liśćmi, przechowywanymi na tym samym poziomie drzewa)

6.3.2 Wpisy w węzłach sformatowanych

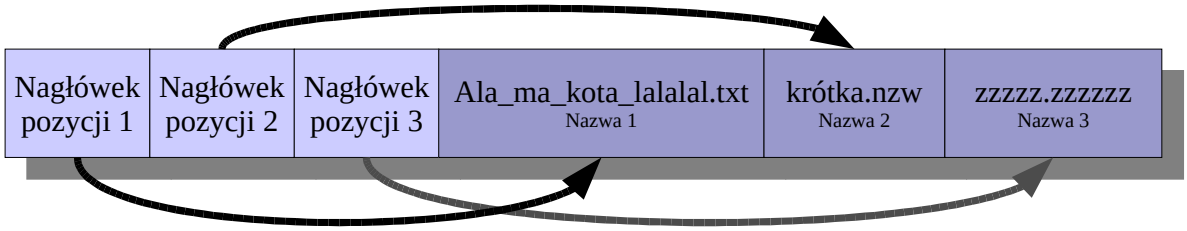
Poniżej opisane są możliwe typy wpisów w węzłach sformatowanych:

Dane bezpośrednie (ang. *Direct items*) Zawierają *ogonki* plików. Końcówki kilku plików (długość modulo wielkość bloku) nie zajmują całego bloku dyskowego aby oszczędzać miejsce. Ta właściwość jest szczególnie ważna w przypadku przechowywania na dysku wielu bardzo małych plików.

Dane pośrednie (ang. *indirect items*) Przechowują wskaźniki do liści niesformatowanych. W liściach niesformatowanych przechowywane są pozostałe (oprócz ogonków) dane plików. Dane takiego wpisu obrazuje rysunek 23

Rysunek 23: Wpis *Dane pośrednie* w węźle sformatowanym w RaiserFS

wpisy katalogowe (ang. *directory items*) Zawierają klucz pierwszej pozycji w katalogu, oraz kolejne pozycje. Pozycja zawiera identyfikator obiektu (patrz niżej) oraz jego nazwę. Pozycje posortowane są zgodnie z 4 pierwszymi literami nazwy. Dane wpisu katalogowego przedstawiono na rysunku 24



Rysunek 24: Wpis katalogowy w węźle sformatowanym w RaiserFS

wpisy informacyjne (ang. *stat data items*) Zawierają informacje o uprawnieniach, wilekości, czasie modyfikacji i utworzenia ....
Istnieje możliwość aby te dane były przechowywane nie jako osobne wpisy ale razem z wpisami katalogowymi. Prowadzone są testy które podejście jest wydjaniejsze.

6.3.3 Odnajdywanie danych w drzewie

Jak łatwo się domyśleć obiekty systemu plików takie jak katalogi i pliki mogą być bardzo duże i zajmować więcej niż jeden węzeł w drzewie. Aby można było je odlaźć w drzewie każdy taki obiekt ma przypożądkowany identyfikator (nie mylić z kluczem), który jest liczbą 32 bitową. Można myśleć o tym identyfikatorze jak o numerze inoda w przypadku systemu Ext2.

Nazwa	**Długość (w bajtach)**	**Opis**
k_dir_id	4	Identyfikator katalogu nadrzędnego
k_object_id	4	Identyfikator obiektu (pliku/katalogu) którego dotyczy klucz
k_offset	4	Pozycja w obiekcie której dotyczy ten wpis. W przypadku plików jest to pozycja w bajtach. W przypadku katalogów cztery pierwsze litery pierwszej pozycji która znajduje się w tym wpisie.
k_uniqness	4	Stała wartość zależna od typu wpisu.

Tablica 4: Struktura klucza obiektu w ReiserFS

Struktura klucza (patrz tabla 4), gwarantuje że węzły należące do tego samego pliku/katalogu będą obok siebie w porządku drzewiastym co umożliwi ich odlazienie.

Dodatkowo dzięki polu k_dir_id pliki i podkatalogi należące do jednego katalogu będą leżały blisko siebie w porządku drzewiastym.

7 xFS

7.1 Wstęp

Prace nad xFS‘em rozpoczęto w 1993 roku, w celu zastąpienia działającego na IRIX‘ie systemu plików EFS. EFS był 32-bitowym systemem plików i narzucał szereg ograniczeń (wielkość obsługiwanej partycji, maksymalna wielkość pliku), co przestało wystarczać wraz ze wzrostem pojemności dysków. Przenoszenie xFS‘a na Linuxa wymagało znacznego wysiłku. SGI postanowiło udostępnić i przystosować xFSdla Linuxa ponieważ:

1. Linux został uznany za system przyszłościowy z powodu swojego błyskawicznego rozwoju (open source/GPL)
2. SGI ma w planach przejście na Linuxa
    - Dla GNU/Linuxa jest dostępna ogromna ilość oprogramowania
    - SGI jest producentem sprzętu
3. XFS jest bardzo dobrym, nowoczesnym systemem plików
4. Możliwości xFSprzyczynią się do rozwoju linuxa

XFS jest nowoczesnym systemem plików o dużych możliwościach:

- Umożliwia tworzenie większych, w porównaniu z innymi linuxowymi systemami plików, systemów plików i plików (odpowiednio 18 i 9 milionów TB)
- Wszystkie operacje na metadanych są kronikowane (ang. journaling), co umożliwia szybki powrót do działania po awarii
- Jest szybki, przystosowany do działania w systemach wieloprocesorowych
- Pliki oparte są na przedziałach bloków dyskowych (ang. extent), zamiast na pojedyńczych blokach co zapewnia:
    - Maksymalną ciągłość plików na dysku - zmniejszenie fragmentacji zewnętrznej
    - Oszczędność miejsca - dziury w plikach nie zajmują miejsca (tzw. Sparse Files)
- Udostępnia Listy Uprawnień

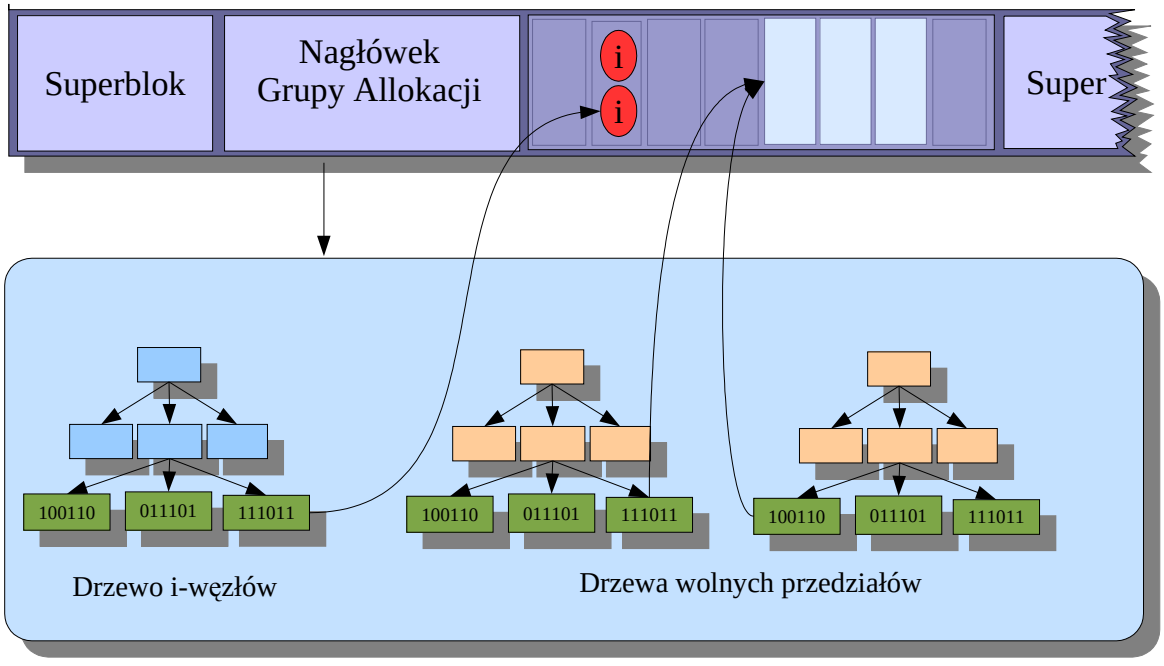
7.2 Struktura xFS

Tak jak większość innych systemów plików xFSoperuje na blokach o stałym rozmiarze (od 512B, do 64KB), określanym przy tworzeniu systemu plików.

7.2.1 Grupy Allokacji

xFS, podobnie jak JFS dzieli partycję na tzw. grupy allokacji czyli kawałki stałego rozmiaru o wielkości od 16MB do 4GB. Grupy allokacji ułatwają wielowątkowe zarządzanie systemem plików. Na początku każdej grupy allokacji umieszczony jest superblok, czyli blok opisujący cały system plików. Za superblokiem umieszczony jest nagłówek GA, zawiera on między innymi 3 ważne struktury:

1. Struktura mapująca wolne przedziały (ang. extents). Może to być:
   - para B^+ drzew. Oba jako wartości mają pary: (pierwszy blok przedziału, ilość bloków przedziału), pierwsze drzewo indeksowane jest pozycją pierwszego bloku, a drugie wielkością przedziału.
   - mapa bitowa - początkowo była rozważana
2. Drzewo i-węzłów - B^+ drzewo indexowane numerami i-węzłów. Pierwotna koncepcja nie wymagała użycia struktury mapującej, ponieważ numer i-węzła miał być bezpośrednio przeliczany na jego adres.
3. Wykaz bloków zajmowanych przez Drzewo wolnych przedziałów - pomaga szybko znaleźć wolne miejsce



Rysunek 25: Struktury danych w Grupie Allokacji

7.2.2 Dzienniki

Wszystkie zmiany metadanych xFSopakowane są transakcje, które rejestrowane są w dziennikach (ang. logs). Transakcja to po prostu ciąg zapisów bloków systemu plików. W razie awarii systemu, naprawa systemu plików polega na wykonaniu wszystkich niezatwierdzonych transakcji z dziennika. xFSumożliwia zapisywanie Dzienników na innym urządzeniu niż system plików którego dotyczą. Może to mieć znaczenie dla bezpieczeństwa i w przypadku synchronicznego trybu obsługi transakcji również dla wydajności.

Obsługa transakcji jest asynchroniczna, czyli dostęp do modyfikowanych bloków dyskowych blokowany jest dopiero w momencie zatwierdzania tranzakcji. Obsługa jest synchroniczna w przypadku udostępniania zasobów przez NFS.

7.3 B^+ drzewa

Są wykorzystane w xFS‘ie właściwie wszędzie gdzie się da :)

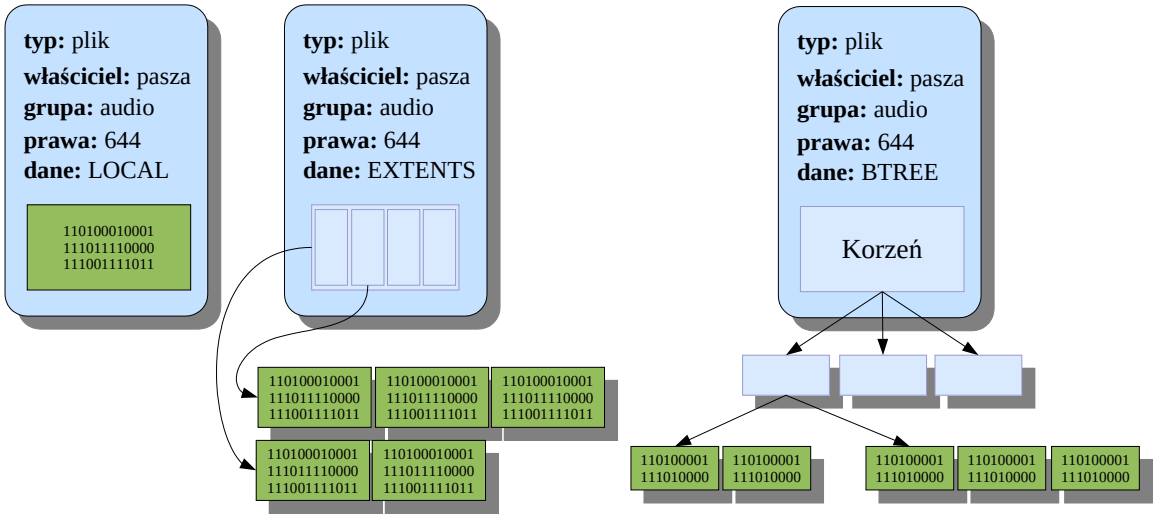
7.4 I-Węzły

Obiekty występujące w xFS‘ie reprezentowane są przez i-węzły. Wielkość i-węzła jest ustalana podczas Tworzenia systemu plików i jest istotnym parametrem mającym wpływ na wydajność i zużycie miejsca na dysku. Zwykle jest to 512 bajtów. Informacje w i-węźle można podzielić na 3 grupy:

1. Ogólne informacje o i-węźle:
   - typ obiektu (plik, katalog, symlink)
   - właściciele obiektu (użytkownik, grupa)
   - podstawowe uprawnienia dla obiektu
   - czas utworzenia, itd.
   - flagi mówiące o tym jaką postać przyjęła sekcja atrybutów i sekcja danych
2. Sekcja atrybutów, zależnie od ilości informacji w tej sekcji, może ona przybrać jedną z 3 postaci:
   (a) bezpośrednią (tzn. dane atrybutów mieszczą się bezpośrednio w i-węźle)
   (b) tablicy deskryptorów przedziałów zawierających dane atrybutów, jeśli te dane nie mieszczą się w i-węźle, ale dane deskryptorów tak
   (c) B^+ drzewa przedziałów zawierających dane atrybutów,
3. Sekcja Danych - podobnie jak sekcja atrybutów, w zależności od ilości danych, jest jednej z 3 postaci:
   (a) dane bezpośrednio w i-węźle

(b) tablica deskryptorów przedziałów z danymi

(c) B^+ drzewo przedziałów z danymi, indeksowane przesunięciem w pliku



Rysunek 26: Rozmieszczenie danych w i-węźle

7.5 Obiekty xFS

Linki symboliczne (ang. symbolic links)

Pliki reprezentowane przez i-węzeł, z wyżej wymienioną stukturą danych

Katalogi reprezentowane przez i-węzeł, zawierający między innymi numer i-węzła katalogu-rodzica oraz strukturę danych pozwalają znaleźć numer i-węzła odpowiadającego obiektowi w katalogu. Ta struktura może mieć jedną z 2 postaci:

1. jeśli katalog ma niewielką ilość krótkich wpisów to znajdują się one w samym i-węźle, w postaci tablicy par (nazwa, numer i-węzła)
2. jeśli wpisy nie mieszczą się w i-węźle, to w i-węźle przechowywany jest korzeń B^+ drzewa indeksowanego wartościami funkcji hashującej dla nazw wpisów,

7.6 Listy uprawnień

Listy uprawnień (ang. Access List - ACL) realizowane są dzięki danym z sekcji atrybutów i-węzła. Listy uprawnień pozwalają na wygodniejsze i bardziej elastyczne przyznawanie uprawnień niż w tradycyjnym modelu UNIX‘owym. Można np. dać prawa wybranemu użytkownikowi nie będącemu właścicielem obiektu, ani członkiem grupy posiadającej plik.

8 Podsumowanie

Nazwa	Maksymalny rozmiar systemu	Rozmiar bloku	Maksymalny rozmiar pliku
xFS	18 tys. PB	512B - 64KB	9 tys. PB
JFS	512B blok -> 4PB, 4KB blok -> 32PT	512B, 1KB, 2KB, 4KB	512TB do 4PB
RaiserFS	4GB bloków, 16TB	4 KB	4GB
ext3	4TB	od 1KB do 4 KB	2GB

Tablica 5: Podsumowanie systemów z dziennikiem - rozmiary

xFS i JFS wywodzą się z systemów komercyjnych, pisanych z myślą o bardzo dużych serwerach. Stąd właśnie wynikają tak duże dopuszczalne rozmiary systemu i plików. W systemie Linux wiekość systemu jest ograniczona do 2TB (plików do 2GB) i stąd takie “niskie” wyniki RaiserFS i ext3.

Nazwa	xFS	JFS	RaiserFS
Zarządzanie wolnymi blokami	B^+ drzewo indeksowane po offsecie i rozmiarze	drzewo + algorytm bliźniaków	bitmapa
Mechanizm ogonków	tak	nie	tak
B^+ drzewa w katalogach	tak	tak	tak
B^+ drzewa w plikach	tak	tak	tak
Obsługa przedziałów	tak	tak	od wersji 4
Dane linków symbolicznych w i-węzle	tak	tak	$\star$
Wejścia katalogu w i-węzle	tak	do 8	$\star$

$\star$ RaiserFS nie posiada jako takich i-węzłów. Cała struktura jest oparta na B^+ drzewie.

Tablica 6: Podsumowanie systemów z dziennikiem - możliwości

Ext3 nie wspiera żadnej z technik wymienionych w tabeli 6, ponieważ jest to system oparty na ext2. Głównym jego celem bylo wsparcie dla dziennika.

Ze względu na wspieranie mechanizmu ogonków najlepszymi systemami do zastosowań domowych są xFS i RaiserFS. Umożliwiają one wykorzystanie części wolnego miejsca w blokach dla nowych małych plików. Są one także bardzo dobrym wyborem w przypadku serwerów newsów, które każdą wiadomość trzymają w osobnym pliku.

JFS wydaje się bardzo dobrym wyborem dla osób operujących na bardzo dużych plikach, ponieważ mechanizmy grup alokacji i przedziałów znacznie przyspieszają sekwencyjny dostęp do plików.

Nazwa	Dynamiczna alokacja i-węzłów	Dynamiczne struktury odszukujące i-węzły	Rozrzedzone pliki
xFS	tak	B^+ drzewo	tak
JFS	tak	B^+ drzewo + przedziały i-węzłów	tak
RaiserFS	tak	B^+ drzewo	tak
ext3	nie	nie	tak

Tablica 7: Podsumowanie systemów z dziennikiem - inne możliwości

Główną zaletą systemu FAT jest prostota i wsparcie ze strony prawie wszystkich systemów operacyjnych. Jednak nie daje on niczego ponad to, a w szczególności nie nadaje się dla systemów wieloużytkowych.

Ext3 dla odmiany zachowuje pełną zgodność z popularnym systemem ext2, wspierającym wieloużytkowość. Do tego ma dosyć prostą budowę i wspiera transakcyjność.

9 Literatura

1. Dokładny opis systemu JFS i aktualny kod źródłowy:
   http://www-124.ibm.com/developerworks/oss/jfs/index.html
2. Opis działania ext2:
   http://rainbow.mimuw.edu.pl/SO/Wyklady-html/07_pliki/7_pliki.html
3. Kilka programów i dokumentacja ext2:
   http://e2fsprogs.sourceforge.net/ext2.html
4. Kilka linków do artykułów o ext3:
   http://www.zip.com.au/~akpm/linux/ext3/
5. Mocno nietechniczne porównanie FAT12, FAT16 i FAT32:
   http://www.pcguide.com/ref/hdd/file/fat.htm
6. Omówienie FAT16:
   http://www.alumni.caltech.edu/~pje/dosfiles.html
7. Specyfikacja FAT32 autorstwa Microsoft:
   http://www.microsoft.com/hwdev/hardware/fatgen.asp
8. Strona główna RaiserFS:
   http://www.namesys.com/

9. Artykuł autorstwa Marka Russinovich'a o "Inside NTFS":
   http://www.winntmag.com/Articles/Index.cfm?ArticleID=3455
10. Dokumentacja sterownika NTFS dla Linuxa
    http://linux-ntfs.sourceforge.net/ntfs/index.html
11. Dokumentacja sterownika NTFS dla Linuxa
    http://linux-ntfs.sourceforge.net/ntfs/index.html
12. Artykuł autorstwa Charles M. Kozierok'a z PCGuide:
    http://www.pcguide.com/ref/hdd/file/ntfs/
13. "Introduction to ISO9660" - Clayton Summers
    http://www.mp3ar.com/Literature/iso9660.pdf
14. Artykuł autorstwa Łukasza Spufińskiego - "Nowe systemy plików dla Linuxa" (xFS)
    http://www.pckurier.pl/archiwum/art0.asp?ID=4758
15. Dokumentacja projektu xFSdla Linuxa (miejscami nieaktualne)
    http://oss.sgi.com/projects/xfs/design_docs/xfsdocs93_pdf/

Spis rysunków

Spis tablic

10 Historia zmian

```
$Log: filesystems.tex,v $
Revision 1.33  2002/12/05 21:22:05  pasza
poprawiliśmy kilka błędów ortograficznych

Revision 1.32  2002/12/04 13:30:51  ofca
podsumoanie systemow

Revision 1.31  2002/12/04 11:24:11  pasza
dodałem literaturę do xfs‘a

Revision 1.30  2002/12/04 11:17:51  ofca
[raiserfs] zmiany pod rysunkami

Revision 1.29  2002/12/04 03:31:48  pasza
[xfs] - chyba koniec, dodałem obrazki
[iso] - poprawiłem obrazki (dodałem cienie)

Revision 1.28  2002/12/04 01:51:15  ofca
[jfs] koniec
```

```
Revision 1.27  2002/12/04 01:39:23  lucash
ladne kolory linkow

Revision 1.26  2002/12/04 01:07:33  lucash
poprawki ntfs

Revision 1.25  2002/12/04 01:02:17  ofca
[jfs] ag, zmiany w blizniakach, bledy ortograficzne

Revision 1.24  2002/12/03 20:27:09  pasza
[xfs] Chyba w miarę napisałem, zrobię jeszcze obrazki i pewnie coś
      dopiszę

Revision 1.23  2002/12/02 08:33:11  lucash
[jfs] skonczony wlasciwie, poprawki w raiserfs

Revision 1.22  2002/12/01 16:40:00  pasza
[iso9660] - no chyba w miarę koniec, wrzucam też obrazki

Revision 1.21  2002/12/01 15:05:15  ofca
[jfs] obrazek do katalogów

Revision 1.20  2002/12/01 03:20:59  ofca
[jfs] prawie skonczone, jeszcze tylko katalogi, moze acl...

Revision 1.19  2002/12/01 02:06:44  lucash
chyba z grubsza skonczony reiserfs. Trzeba to jeszcze przeczytac i
przeformatowac troche.

Revision 1.18  2002/11/30 23:45:51  ofca
[jfs] moje obrazki

Revision 1.17  2002/11/30 13:50:47  ofca
poprawione obrazki i powstawiane jak trzeba do filesystems.tex

Revision 1.16  2002/11/30 11:58:18  ofca
naprawa historii

revision 1.15  2002/11/30 00:54:29  lucash
zakomentowalem jeden rysunke bo go nie ma teraz w repozytorium.
Chcialem zeby sie dokument kompilowal.

revision 1.14  2002/11/30 00:48:31  lucash
```

```
dodany poczatek reisera

revision 1.13  2002/11/30 00:37:31  ofca
[jfs] extenty i inodyx

Revision 1.12  2002/11/29 15:08:32  lucash
rysuniki zamienione na pdf - dopisany kawalek JFS przez Paulinke

Revision 1.11  2002/11/28 23:54:22  lucash
[jfs] agregaty opisane

Revision 1.10  2002/11/28 16:43:53  lucash
dodalam wstep do jfs

Revision 1.9  2002/11/28 15:03:17  lucash
drobne poprawki

Revision 1.8  2002/11/28 14:22:31  pasza
Zacząłem o iso, napisałem o file systemach wogóle

Revision 1.7  2002/11/26 16:01:05  lucash
testowy commit

Revision 1.6  2002/11/25 17:14:21  lucash
bardzo male dodatki

Revision 1.5  2002/11/25 00:23:18  lucash
dwa sliczne rysunki i ntfs almost done

Revision 1.4  2002/11/24 00:05:29  lucash
fat prawie caly i ntfs rozgrzebany - obrazek dodany
(format obrazka to 2 pliki - eps i sxd - tylko tak
dodajemy obrazki)

Revision 1.3  2002/11/22 17:15:21  ofca
dopisalam troche o facie

Revision 1.2  2002/11/21 15:04:54  lucash
*** empty log message ***

Revision 1.1  2002/11/16 13:10:03  lucash
Dadany plik do cvs
```

```
Revision start  16/11/2002 14:00:00  lucash
Pusty plik
```