Lokalne Systemy Plików

Eryk Kopczyński
Maciej Osiński
Katarzyna Sokołowska

20 grudnia 2002

Wstęp

Raport „Lokalne Systemy Plików” ma na celu przedstawienie Czytelnikowi podstaw działania najczęściej używanych obecnie systemów plików (lokalnych). Autorzy skoncentrowali się na specyficznych cechach, odróżniających poszczególne systemy plików oraz stanowiące o ich przydatności do specyficznych zastosowań.

Szczególnie dokładnie został opisany mechanizm księgowania (zobacz m.in. 4), jako że jest on integralną częścią każdego nowoczesnego systemu plików.

Podział pracy pomiędzy autorów przedstawia się następująco:

Eryk Kopczyński systemy Fat (rozdział 2), Fat32 (sekcja 2.3), NTFS (rozdział 3) oraz Iso9660 wraz z rozszerzeniami (rozdział 1)

Maciej Osiński systemy Ext2 (rozdział 5), Ext3 (rozdział 6), ReiserFS 3.6 (rozdział 9) oraz ReiserFS 4 (rozdział 10)

Katarzyna Sokołowska systemy JFS (rozdział 8) i XFS (rozdział 7) oraz uwagi o nowoczesnych systemach plików (rozdział 4)

Raport powstał w ramach zajęć z *Systemów Operacyjnych* na Wydziale Matematyki, Informatyki i Mechaniki Stosowanej Uniwersytetu Warszawskiego i stanowi materiał towarzyszący prezentacji pt. *Lokalne Systemy Plików*.

Wszelkie techniczne niedostatki niniejszego raportu są udziałem Macieja Osińskiego, jako osoby odpowiedzialnej za skład raportu oraz wygenerowanie slajdów.

Zapraszamy do lektury.

Spis treści

Spis rysunków

Rozdział 1

ISO 9660

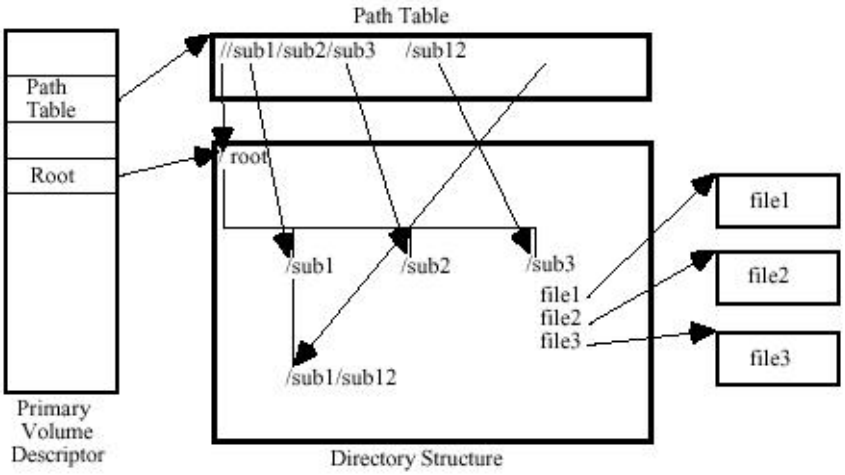
W przeciwieństwie do pozostałych systemów plików, służących do zapisu danych na nośniku wielokrotnego zapisu, ISO 9660 został stworzony do zapisu danych na CD-ROMach. System plików zapisanych w formacie ISO 9660 jest tworzony od razu w swojej „ostatecznej" postaci, na przykład jako kopia fragmentu dysku twardego. W związku z tym pewne rozwiązania, które trzeba było stosować w pozostałych systemach plików (np. zmniejszanie fragmentacji, system transakcji itp.) nie muszą być stosowane w ISO 9660 — jest stosowana alokacja ciągła. ISO 9660 jest dosyć starym systemem plików i ma pewne ograniczenia.

ISO 9660 jest zestandaryzowaną (przez ISO 9660) wersją formatu High Sierra. Format High Sierra został stworzony (w tym samym celu) wspólnie przez kilka firm w roku 1985.

1.1 Struktura

Dysk CD-ROM w formacie ISO 9660 podzielony jest na sektory rozmiaru 2048 bajtów. Zawierają one po kolei następujące dane:

- 16 pustych sektorów
- jeden lub więcej deskryptorów dysku (*Volume Descriptors*). Każdy deskryptor ma długość 1 sektora i zawiera informacje takie, jak nazwa systemu operacyjnego, nazwa dysku, długość *path table* i numer sektora, pod którym się ona znajduje, długość i numer sektora katalogu głównego itp. Po ostatnim z nich jest specjalny sektor do oznaczenia, że już nie ma więcej deskryptorów. Różne deskryptory mogą być przeznaczone np. dla różnych systemów operacyjnych.
- Tablica ścieżek (*path table*), czyli lista wszystkich katalogów w systemie plików. Ta informacja jest w pewnym stopniu redundantna, bo są też opisy katalogów. (Mogą być dwie takie tablice — różne deskryptory mogą używać różnych tablic.) Na początku jest katalog główny, potem alfabetycznie jego podkatalogi (stopnia 2) itd. Długości nazw mają zmienną długość (zatem także wpis w tablicy ścieżek ma zmienną długość), ale są ograniczone przez 31.



Rysunek 1.1: Struktura ISO-9660

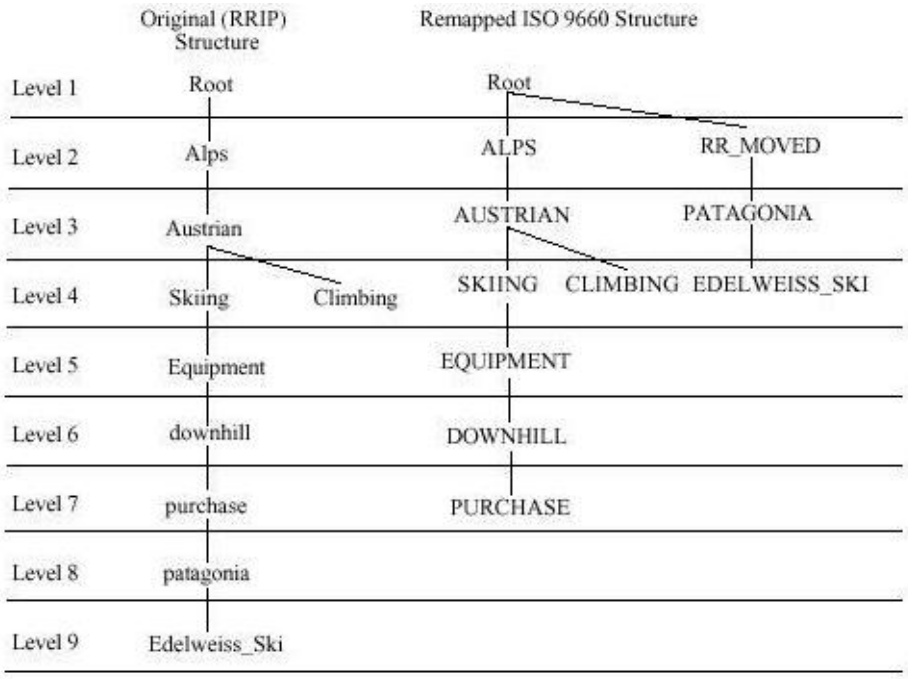
- dane o katalogach i plikach. Podobnie jak w FAT, katalog jest przechowywany tak, jak plik, ale ma ustawiony specjalny atrybut. Każdy wpis w katalogu odpowiada jednemu plikowi lub podkatalogowi (są też wpisy dla . i ..) i ma zmienną długość (zależną np. od długości nazwy pliku). Zawiera informacje o pierwszym sektorze i długości pliku, czasie, atrybutach (np. czy jest katalogiem) i nazwie pliku. Na końcu wpisu może powstać miejsce, które może zostać wykorzystane do trzymania informacji właściwych dla konkretnego systemu operacyjnego (np. rozszerzenia *Rocky Ridge*). Wpisy są uporządkowane alfabetycznie.

1.2 Uwagi

- Większość liczb w deskryptorach dysku, tablicach ścieżek i opisach katalogów jest trzymana w obu formatach *little endian* i *big endian*, żeby ułatwić czytanie na różnych procesorach.

- Standard ISO 9660 pozwala tylko na 8 poziomów zagnieżdżenia katalogów (łącznie z katalogiem głównym).

- Standard ISO 9660 nakłada ograniczenia na nazwy plików (długość ≤ 31; tylko znaki A-Z, 0-9, kropka i _; co najwyżej 1 kropka (oddzielająca nazwę i rozszerzenie); brak kropki w nazwach katalogów). Nazwy zazwyczaj są w formacie 8.3, by mogły być czytane w systemie DOS. Na końcu nazwy jest średnik i liczba służąca do oznaczania numeru wersji, np. `NAZWA;1`. Te reguły nie zawsze są respektowane.



Rysunek 1.2: Obchodzenie ograniczenia na zagnieżdżenie katalogów w Rocky Ridge

1.3 Rozszerzenia Rocky Ridge

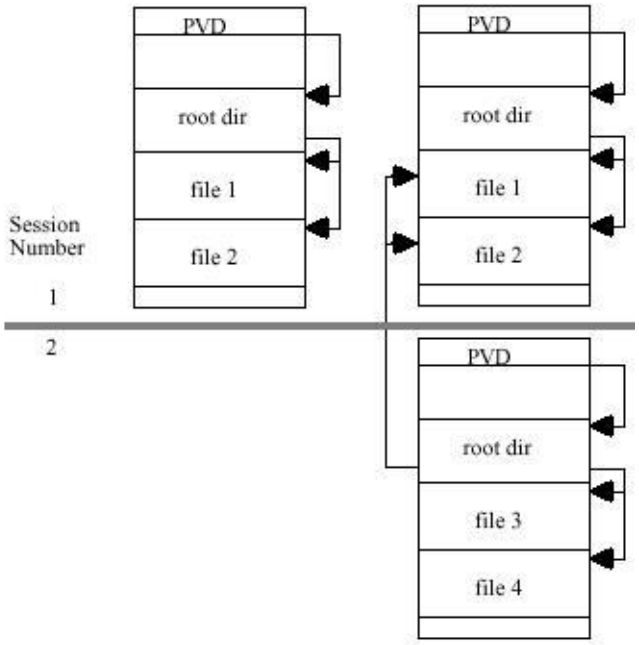
Rozszerzenia *Rocky Ridge* pozwalają na używanie uniksopodobnych właściwości plików (takich, jak właściciel, prawa dostępu i dowiązania symboliczne). Pozwalają także na więcej niż 8 poziomów zagnieżdżenia — dla zgodności z ISO 9660 katalogi na dziewiątym poziomie są przenoszone do katalogu głównego, ale system operacyjny czyta je tak, jakby były na dziewiątym poziomie.

1.4 Rozszerzenia Joliet

Rozszerzenia Joliet zostały stworzone na potrzeby Windows 95. Pozwalają one na używanie długich nazw (do 64 znaków) w standardzie Unicode. Zniesione jest ograniczenie zagnieżdżenia katalogów do 8. Informacje dla systemu Joliet (tablica ścieżek, katalogi) są zapisane na dysku niezależnie od informacji podstawowego ISO 9660.

1.5 Dyski CD-R

Nagrywarki CD-R pozwalają na dogrywanie nowych danych do częściowo już nagranego dysku. Robi się to w ten sposób, że w nowej sesji tworzy się nowe meta-dane (deskryptory



Rysunek 1.3: Dogrywanie sesje do plyty CD-R

dysku, tablicę ścieżek, dane o katalogach). Wskaźniki do plików mogą wskazywać i na starą (ew. stare), i na nową sesję. Stare meta-dane przestają być używane.

1.6 Universal Disk Format

Nowszym systemem plików służącym do zapisu danych na CD-R, CD-RW i DVD jest UDF (Universal Disk Format). Format ten ma mniejsze ograniczenia niż ISO 9660, co pozwala na używanie go do DVD. Format UDF został stworzony na potrzeby *nagrywania pakietowego*. Pozwala ono na używanie dysków CD-RW i CD-R tak, jak zwykłych dysków wielokrotnego zapisu — tzn. można dodawać i zmieniać zawarte na nich dane, co jest bardziej wygodne i efektywne niż ISO, wymagające za każdym razem tworzenia nowej sesji. (Oczywiście, w przypadku CD-R miejsce używane wcześniej przez stare dane się marnuje.)

Rozdział 2

FAT i FAT32

FAT, zwany też systemem plików DOS, jest jedynym systemem plików używanym w systemie MS DOS. System FAT jest dobry dla dysków poniżej 200 MB, np. na dyskietkach — na takich dyskach nie ma sensu stosowanie systemów takich, jak FAT 32 lub NTFS, bo wiązałoby się to z dużymi stratami na struktury systemu plików. Nie obsługuje dysków powyżej 2 GB (4 GB w Windows NT).

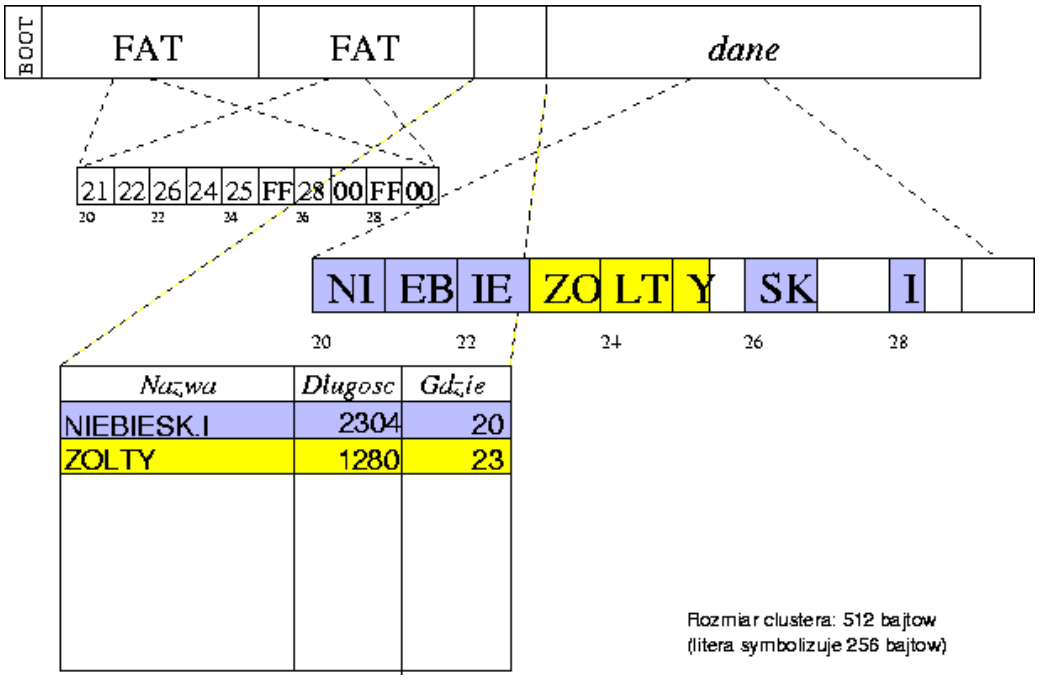
2.1 Struktura

Partycja w systemie 16-bitowym systemie FAT składa się z następujących części:

- *Boot sector*
- Tablica FAT (File Allocation Table)
- Kopia tablicy FAT
- Katalog główny
- Obszar danych

Obszar danych podzielony jest na clustery (jednostki alokacji), zwykle składające się z kilku sektorów dysku. Tablica FAT dla każdego clustera zawiera 16-bitową informację. Informacja o clusterze mówi o tym, czy jest on wolny, należy do jakiegoś pliku, czy uszkodzony. Jeśli należy do pliku, to może on być ostatnim clusterem lub nie — w tym drugim przypadku zapisany jest numer kolejnego clustera. Liczba clusterów musi zatem być trochę mniejsza niż 2^{16}.

Katalog główny ma ustalony w trakcie formatowania rozmiar — może on zawierać ustaloną liczbę plików i katalogów. Dla każdego pliku jest wpis w katalogu, o rozmiarze 32 bajtów. W oryginalnym FAT dla każdego pliku podana jest jego nazwa (8 znaków plus 3 znaki rozszerzenia), rozmiar w bajtach, czas modyfikacji, atrybuty i numer jego pierwszego clustera (trochę miejsca zostało zarezerwowane na następne wersje). Atrybutami mogą być (każdy zajmuje 1 bit):



Rysunek 2.1: Struktura FAT

- *archive* — plik, który trzeba skopiować przy następnej kopii zapasowej;
- *read only* — takiego pliku nie można skasować ani zmienić bez zmiany tego atrybutu;
- *hidden*, *system* — oznaczenia plików systemowych, które dla bezpieczeństwa nie powinny być widziane przez normalne programy;
- *directory* — katalog; katalog jest "plikiem" (nie może być czytany w normalny sposób) tego samego formatu, co katalog główny, ale może mieć zmienny rozmiar;
- *volume ID* — tak jest oznaczony jeden plik w katalogu głównym; jego 11-znakową nazwą jest etykieta dysku.

2.2 Uwagi i rozszerzenia

- W celu bezpieczeństwa są dwie kopie tablicy FAT — operacje zapisu są wykonywane na obu naraz, chociaż system operacyjny „używa" tylko oryginału. Jedyne zastosowanie kopii jest takie, że program naprawiający błędy na dysku może porównać obie kopie i wybrać „lepszą".
- Na dyskietkach i bardzo małych partycjach dysków twardych FAT jest 12-bitowy (tzn. liczba clusterów jest mniejsza niż 2^{12}).

- W systemie MS Windows 95 dodano długie nazwy plików tak, żeby nowe programy widziały długie nazwy, a stare programy wciąż działały. Zrobiono to tak, że plik z długą nazwą ma wpis w katalogu z nazwą w formacie 8.3 (wybieraną w sposób unikatowy i kojarzący się z długą nazwą — zazwyczaj jest to `abcdef~1.ext`, gdzie `ext` jest rozszerzeniem a `abcdef` pierwszymi sześcioma znakami), po czym następują wpisy zawierające długą nazwę. Wpisy te są oznaczone specjalnymi atrybutami, żeby nie były widziane przez stare programy. Bardzo długie nazwy (do 253 znaków) muszą używać kilku takich wpisów.
- Krótkie nazwy plików są pisane w katalogach dużymi literami, zatem nie ma informacji o tym, czy litery są duże, czy małe (niektóre programi wyświetlają nazwy plików małymi literami). Długie nazwy "pamiętają" wielkość liter, ale użytkownik nie musi zwracać uwagę na wielkość liter przy odwoływaniu się do plików.
- W systemie Windows 95 dodano czas stworzenia i ostatniego użycia pliku.
- Plik kasuje się przez zwolnienie clusterów w tablicy FAT i oznaczenie w specjalny sposób pierwszego znaku jego nazwy (oznaczenie jest inne niż znak końca listy plików). Dzięki temu mogą istnieć programy typu *undelete*, które pozwalają na przywrócenie usuniętego pliku (trzeba podać pierwszy znak nazwy).
- W systemie DR DOS firmy Digital Research istniały innego rodzaju rozszerzenia, np. pozwalające na ochronę niektórych plików hasłami.

2.3 FAT 32

System FAT 32 jest lepszą wersją systemu FAT, wprowadzoną w nowszych wersjach Windows 95 (OEM 2). W przeciwieństwie do FAT, obsługuje on partycje rozmiaru powyżej 2 GB.

FAT 32 jest najlepszym systemem plików dostępnym w Windows Millenium Edition i poprzednikach. Jest on 32-bitowy, więc dysk może zostać podzielony na więcej clusterów niż w FAT 16. Na dysku wielkości 2 GB rozmiar clustera w FAT jest 32 kB, a w FAT32 4 kB — małe pliki marnują więc mniej miejsca. Rozmiar clustera dostosowany jest do rozmiaru dysku. Dla dysków poniżej 8 GB rozmiar clustera jest 4 KB, od 8 GB do 60 GB jest 8 KB i tak dalej. Jednak wciąż nie ma wielu użytecznych właściwości NTFS, jak na przykład niezawodność i zabezpieczenia.

- Katalog główny jest przechowywany tak, jak każdy inny katalog (tzn. jako łańcuch clusterów), czyli ma zmienną długość.
- W FAT 32 możliwe jest wybranie, która z dwóch kopii tablicy FAT jest ważniejsza, oraz wyłączenie drugiej kopii. W FAT 16 błąd na dysku w obszarze „oryginału" powoduje nieużywalność dysku, a FAT 32 w takiej sytuacji może używać drugiej kopii jako ważnej.

Rozdział 3

New Technology File System

System plików NTFS (New Technology File System) został stworzony na potrzeby Microsoft Windows NT. Nie ma on ograniczeń FAT i FAT32 (od razu ma długie nazwy plików, prawa dostępu, *journaling* i inne ulepszenia ważne w nowoczesnych systemach plików) i ma strukturę, do której łatwo dodawać nowe możliwości, co oznacza, że nie powinno być potrzeby tworzenia nowego systemu plików w najbliższej przyszłości. W Windows NT 4.0 używana jest wersja NTFS 1.2, czasami nazywana też NTFS 4.0.

3.0.1 Struktura

Tak samo jak w FAT, dysk podzielony jest na clustery, rozmiaru zazwyczaj 4kB (poza małymi dyskami, gdzie są mniejsze i starszymi wersjami NTFS, gdzie dla dużych dysków są większe). Z punktu widzenia NTFS, wszystkie dane zapisane na dysku, poza tymi, które muszą się znajdować na początku (*boot sector*), są „plikami", w tym zwykłe pliki z danymi, katalogi (foldery), i pliki z metadanymi (*metadata files*), które zawierają informacje o systemie plików. (W FAT obie tablice FAT i katalog główny nie są plikami.) Oznacza to, że mogą być one umieszczone w dowolnym miejscu na dysku. Metadane mają nazwy zaczynające się od $. Najważniejszym z plików z metadanymi jest $MFT (Master File Table).

3.0.2 MFT

MFT zawiera informacje o wszystkich plikach w systemie. Każdy plik ma wpis w MFT, ustalonej wielkości (np. 2 kB). Plik w NTFS jest kolekcją atrybutów — dla plików z danymi jednym z nich jest *Data*, który zawiera „prawdziwą" zawartość pliku. Pozostałe atrybuty to na przykład *Name* zawierające nazwę pliku, *Standard Information* zawierający standardową informację taką, jak na przykład czasy stworzenia, modyfikacji i użycia. Program używający systemu NTFS może dodawać własne atrybuty.

Atrybuty mogą być umieszczane wewnątrz wpisu MFT lub w innym miejscu dysku (niektóre muszą być umieszczone wewnątrz). Dla małych plików może się zdarzyć, że wszystkie atrybuty, łącznie z *Data*, zmieszczą się we wpisie w MFT — wtedy plik nie

zajmuje dodatkowego miejsca na dysku. W przeciwnym przypadku wpis zawiera tylko wskaźniki do atrybutów, które się w nim nie zmieściły. Atrybut może być sfragmentowany, wówczas tych wskaźników może być więcej. W skrajnej sytuacji może być tak, że lista wskaźników znajduje się poza MFT.

3.0.3 Inne metadane

Pierwsze 16 wpisów w MFT zawiera informacje o wszystkich plikach z metadanymi. Najciekawsze z nich to:

- $MFT — informacja o samym MFT,
- $MFTMirr — kopia zapasowa pierwszych 16 wpisów w MFT,
- $LogFile — przechowuje informację o ostatnich zmianach w systemie plików,
- $Volume — informacje o dysku (nazwa, wersja NTFS, itd.)
- $AttrDef — lista nazw atrybutów i ich znaczenia,
- $Quota — informacja o quotach (w Windows 2000),
- . — katalog główny,
- $Bitmap — mapa bitowa wskazująca, które sektory są wolne.

3.0.4 Inne możliwości NTFS 1.2

- Nazwy plików w systemie NTFS mogą mieć długość 256 znaków i są trzymane w międzynarodowym 16-bitowym formacie Unicode. Informacja o nazwach plików trzymana jest w atrybutach (w MFT). Jeden plik może mieć wiele nazw. Może mieć dodatkową nazwę w formacie 8.3 (w celu zachowania zgodności ze starymi programami). Poza tym, jest możliwość tworzenia dowiązań — jeden plik może występować w kilku katalogach, pod różnymi nazwami.
- Dla każdego katalogu NTFS używa B-drzew do przechowywania informacji o liście plików do niego należących. Oznacza to, że wyszukanie zadanego pliku nawet w bardzo dużym katalogu jest bardzo efektywne.
- Użytkownik ma możliwość kompresji wybranych przez siebie plików i katalogów. Zazwyczaj stosuje się to do rzadko używanych plików, np. archiwalnych. Skompresowanego pliku używa się tak samo, jak zwykłego — informacja jest automatycznie kompresowana przy zapisie i dekompresowana przy odczycie, bez udziału użytkownika.
- W atrybutach plików jest przechowywana informacja o prawach dostępu do niego dla różnych użytkowników. Jest to część systemu zabezpieczeń w Windows NT.

- NTFS jest zgodny ze standardem POSIX.1, w celu uproszczenia przenoszenia programów między Unixem a Windows NT.

- NTFS rezerwuje trochę miejsca w pobliżu MFT — nie pozwala na wstawianie tam innych plików, chyba, że już całe pozostałe miejsce na dysku jest zajęte. Zmniejsza to fragmentację MFT.

- Skoro strumień z danymi to jeden z atrybutów, jest możliwe stworzenie pliku, który składa się z kilku różnych strumieni. Strumienie te mają wspólną nazwę pliku (odwołuje się do nich przez *nazwa pliku:nazwa strumienia*) i wspólne atrybuty i prawa dostępu.

3.0.5 NTFS 3.0

System Windows 2000 używa nowszej wersji systemu plików NTFS (3.0, zwanej też 5.0).

- Mechanizm *reparse points* polega na zaznaczaniu w specjalny sposób pliku lub katalogu. Kiedy użytkownik chce otworzyć taki plik lub katalog, zostanie wywołany odpowiedni program, który będzie zastępował normalne operacje na pliku. Mechanizm ten może być używany m. in. do symulacji dowiązań symbolicznych do plików i katalogów oraz do montowania.

- Użytkownik może zaszyfrować wybrane przez siebie pliki i katalogi. Zajmuje się tym usługa EFS (*Encrypting File System*). Wprawdzie system zabezpieczeń w Windows NT powoduje, że z wnętrza systemu nie można dostać się do prywatnych plików, ale gdyby nie były one zaszyfrowane, byłaby możliwość dostania się do nich z zewnątrz (z innego systemu operacyjnego). Szyfrowanie jest oparte na algorytmie szyfrowania z kluczem publicznym. Podobnie jak w przypadku kompresji, szyfrowanie i odszyfrowywanie jest automatyczne.

- NTFS 3.0 udostępnia usługę dla plików rzadkich (*sparse files*), czyli takich, które są duże, ale w większości wypełnionych zerami. Taki plik można specjalnie oznaczyć — wtedy fragmenty wypełnione zerami nie będą zajmowały miejsca na dysku. (Ta metoda jest dużo szybsza od kompresji.)

- Jest możliwość ograniczenia zużycia miejsca na dysku dla poszczególnych użytkowników (*quota*).

3.0.6 Niezawodność

NTFS używa systemu transakcji przy zapisie danych na dysk (tzn. operacje zmieniające kilka miejsc na dysku są wykonywane atomowo). Każda operacja (transakcja) wykonywana na dysku zapamiętywana jest w pliku `$LogFile`. W razie awarii (np. zasilania), przy następnym montowaniu system sprawdza, czy są jakieś niezakończone transakcje i je

wykonuje lub usuwa ich efekty. Dzięki temu jest mało prawdopodobne, że awaria spowoduje, że część metadanych będzie zawierała nieaktualne informacje i system plików będzie niespójny (np. część metadanych mówi, że dana część dysku jest wolna, podczas gdy w rzeczywistości jest ona zajęta przez coś ważnego). To trochę zmniejsza efektywność zapisu, ale zwiększa niezawodność systemu. W NTFS 3.0 istnieje również *Change journal*, który zawiera wszystkie informacje na temat zmian w systemie plików (jakie plik został zmieniony i w jaki sposób — treść zmiany nie jest zapisywana).

Rozdział 4

Nowoczesne Systemy Plików

Systemy plików z księgowaniem (*Journal File Systems*) to specjalne systemy plików o podwyższonej wytrzymałości na uszkodzenia i awarie.

Ich działanie polega na tym, że najpierw zapisują/odczytują w specjalnym miejscuna dysku pakiet danych, zanim faktycznie dokonają zapisu/odczytu i przekażą informacje o tym do aplikacji. Dzięki temu, jeśli system zawiesi się, zabraknie zasilania lub ulegnie awarii np. sektor dysku, operacja zostanie pomyślnie zakończona po przywróceniu funkcjonowania systemu.

Dodatkowo systemy te mogą radzić sobie z dużymi partycjami twardego dysku, łatwo skalować się na tysiące plików, szybko odzyskiwać dane po załamaniu systemu, zwiększać wydajność operacji We/Wy, radzić sobie dobrze zarówno z małymi jak i dużymi plikami, zmniejszyć wewnętrzną i zewnętrzną fragmentację.

4.1 Definicje

4.1.1 B+ drzewa

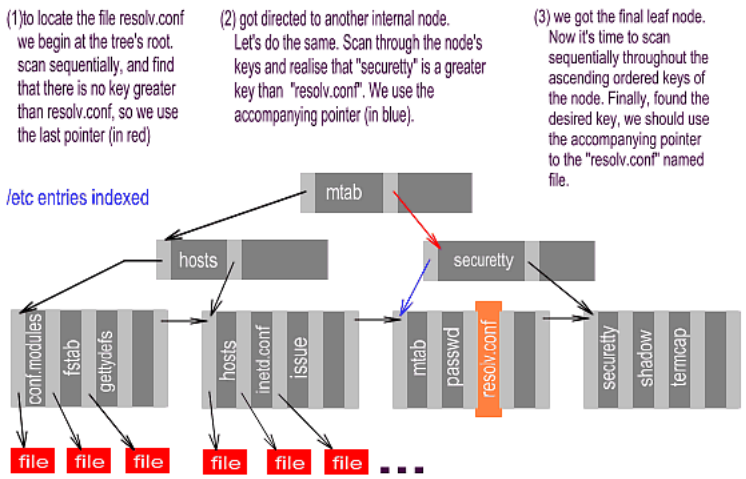
Struktura B+ drzew wykorzystywana była dotychczas w strukturach indeksujących baz danych. Znak + oznacza, że B-drzewo zostało zmodyfikowane tak, że zawiera wskaźnik z każdego liścia na następny, zapewniając dostęp sekwencyjny.

B+ drzewo składa się z dwóch rodzajów węzłów - wewnętrznych i liści. Wskaźniki wewnętrznych węzłów wskazują inne węzły, natomiast wskaźniki liści wskazują bezpośrednio końcową informację.

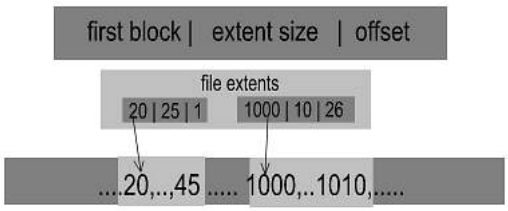
Poniższy rysunek przedstawia przykładowe B+ drzewo:

4.1.2 Extenty

Ekstenty są zbiorami sąsiednich bloków logicznych wykorzystywanymi przez niektóre systemy plików. Deskryptor ekstentu jest czymś jak *początek, rozmiar ekstentu, offset*, gdzie początek jest adresem bloku, w którym ekstent się zaczyna, rozmiar ekstentu jest rozmiarem w blokach, a offset jest przesunięciem, jakie zajmuje w pliku pierwszy bajt ekstentu.



Rysunek 4.1: B+ drzewo



Rysunek 4.2: Ekstent

Ponieważ bloki wewnątrz ekstentu są sąsiednie, polepsza się czas przeszukiwania, gdyz potrzebnych jest mniej ruchół głowicy. Używanie ekstentów redukuje zewnętrzną fragmentację, gdyż więcej bloków przechowywanych jest razem.

Poniższy rysunek przedstawia przykładowy ekstent:

4.2 Księgowanie

4.2.1 Czym jest księgowanie?

Księgowanie wywodzi się z systemów baz danych. Większość z nich stosuje transakcje, zapewniające właściwości ACID - niepodzielność (*Atomicity*), spójność (*Consistency*), izolacja (*Isolation*) i trwałość (*Durability*). Z punktu widzenia systemów plików najważniejsza jest niepodzielność. Stosują one podobną technikę jak w bazach danych, tj. zapisywanie wszystkich zmian w dzienniku (*Log, Journal*). Najpierw następuje zapis/odczyt do/z dziennika, a dopiero później faktyczny zapis/odczyt danych. Dzięki temu, jeśli załamanie

FS	Max. rozmiar systemu plików	Rozmiary bloków	Max. rozmiar pliku
XFS	18 tys. petabajtów	512B do 64KB	9 tys. petabajtów
JFS (blok 512B)	4 petabajty	512, 1024, 2048, 4096 bajtów	512 Tb
JFS (blok 4 kB)	32 petabajty	512, 1024, 2048, 4096 bajtów	4 petabajty petabajty
Reiser	16 Tb	64KB obecnie 4KB	4GB, 2^{10} petabajtów (3.6)

systemu nastąpi zanim zatwierdzony zostanie wpis do dziennika, oryginalne dane nadal są na dysku i tracimy jedynie zmiany. Jeśli natomiast nastąpi załamanie w trakcie aktualizacji (np. po zatwirdzeniu wpisu do dziennika), wpis w dzienniku będzie zawierał informację o tym, co miało zostać wykonane. W ten sposób po restarcie systemu można odczytać zadania do wykonania i dokończyć przerwaną aktualizację.

Dodatkową zaletą jest fakt, że czas związany z odzyskiwaniem właściwego stanu systemu plików poprzez użycie dziennika jest znacznie krótszy (rzędu kilku sekund) i niezależny od wielkości i ilości danych (w przeciwieństwie do programu fsck, który sprawdza wszystkie meta-dane).

4.3 Zwiększenie skalowalności

UNIX File System (UFS) oaz ext2 zostały zaprojektowane, gdy twarde dyski i inne nośniki danych miały znacznie mniejszą pojemność. Wzrost pojemności nośników danych przyczynił się do powstania większych plików, katalogów i rozmiarów partycji, powodując pojawienie się problemów związanych z systemami plików. Nowe systemy plików zostały zaprojektowane z myślą o rozwiązaniu tych problemów.

4.3.1 Nowe limity

Większość nowych systemów plików zwiększyła ilość bitów dla niektórych pól, w celu ominięcia istniejących limitów. Nowe limity dla wybranych systemów plików przedstawia tabela niżej:

4.3.2 Unikanie niwłaściwego użycia

Struktury wolnych bloków

W systemach UFS i ext2 analiza dostępności poszczególnych bloków odbywa się sekwencyjnie. Wolne bloki można znaleźć przeglądając po kolei zawartości map bitowych bitów z kolejnychgrup bloków.

Nowe systemy proponują szybsze rozwiązanie, oparte na zastąpieniu sztywnej struktury mapy bitowej indeksami (położenia oraz liczby wolnych bloków) w formie B+ drzew, często stosowanych w bazach danych.

Indeksowanie to obejmuje także rozmiar całych ciągłych sekwencji wolnych bloków (*ekstentów*), co powoduje szybsze znalezienie jakiegokolwiek wolnego bloku, jak również bloku, który byłby najlepszy ze względu na przewidywany przyrost wielkości pliku.

Podejście takie pozwala również na zaoszczędzenie miejsca na dysku w przypadku dużej ilości dużych plików - zamiast zapisywania informacji o każdym z małych bloków można zapisywać niewiele obszerniejsze (w porównaniu di informacji o bloku) informacje o znacznie większych jednostkach alokacji.

Alokacja i-węzłów

W ext2 obsługa dużej ilości plików w ramach jednego katalogu może stać się problemem, ponieważ katalog zaimplementowany jest w postaci jednokierunkowej listy. Operacja wyszukiwania jakiegokolwiek pliku ma charakter liniowy, co przy dużych katalogach prowadzi do długiego czasu wyszukiwania. Omawiane systemy plików używają B+ drzew indeksowanych nazwami, zmniejszając czas wyszukiwania. Po otrzymaniu żądania konkretnego pliku z tego katalogu, w B+ drzewie szybko wyszukiwany jest i-węzeł.

W omawianych systemach wykorzystanie B+ drzew jest zależne od konkretnego systemu. W niektórych z nich dla każdego katalogu przechowywane jest oddzielne B+ drzewo, podczas gdy w innych istnieje jedno B+ drzewo dla całego drzewa katalogów danego systemu plików.

Duże pliki

W omawianych systemach plików używa sie B+ drzew do zorganizowania bloków plików. Bloki są indeksowane poprzez przesunięcie (*offset*) w pliku. Następnie, po nadejściu żądania konkretnego offsetu w danym pliku, program systemu plików przeszuka B+ drzewo, lokalizując właściwy blok. Szczegóły rozwiązania zależą od konkretnego systemu plików.

Dodatkowo w niektórych z omawianych systemów używa się ekstentów w celu zgromadzenia kilku bloków logicznych razem (zamiast wskaźników pośrednich), co rozwiązuje część problemów związanych z dużymi plikami. Dodatkowo ekstenty mogą być zorganizowane w B+ drzewie, zmniejszając tym samym czas dostępu. Nowe i-węzły zwykle zawierają pewne bezpośrednie wskaźniki na ekstenty, natomiast w przypadku, gdy plik potrzebuje ich więcej, organizowane są one w B+ drzewo. W celu poprawy wydajności przy dostępie do małych plików, omawiane systemy przechowują dane pliku w samym i-węźle. Dzięki temu, jeśli tylko uzyskamy dostęp do i-węzła pliku, otrzymujemy także dostęp do jego danych. Technika ta jest szczególnie użyteczna w przypadku dowiązań symbolicznych, gdzie dane pliku są wyjątkowo małe.

Tabela 4.3 omawia stosowane techniki w różnych systemach plików.

FS	Zarządzanie wolnymi blokami	Ekstenty dla wolnej przestrzeni
XFS	B+ drzewa indeksowane przez rozmiar i offset	TAK
JFS	Drzewo + Binary Buddy	NIE
Reiser	oparte o mapy bitowe	na razie nie obsługiwane

FS	B-drzewa dla i-węzłów	Ekstenty do adresowania bloków pliku
XFS	TAK	TAK
JFS	TAK	TAK
Reiser	jako poddrzewo głównego drzewa systemu plików	wewnątrz głównego drzewa systemu plików

FS	Dane wewnątrz i-węzłów (małe pliki)	Dane dowiązań symbolicznych wewnątrz i-węzłów	Ekstenty katalogów wewnątrz i-węzłów (małe katalogi)
XFS	TAK	TAK	TAK
JFS	NIE	TAK	do 8
Reiser	od wydania 4	B* drzewo	B* drzewo

Rysunek 4.3: Zaawansowane techniki w systemach plików

Inne techniki	Dynamiczna alokacja i-węzła	Dynamiczne struktury śledzenia i-węzłów	Obsługa rzadkich plików
XFS	TAK	B+ drzewo	TAK
JFS	TAK	B+ drzewo z ekstentami i-węzła	TAK
ReiserFS	TAK	główne B+ drzewo systemu plików	TAK

Rysunek 4.4: Inne ulepszenia w systemach

4.4 Inne ulepszenia

4.4.1 Rzadkie pliki

Obsługa rzadkich plików wykorzystuje pole óffset wewnątrz pliku"deskryptora ekstentu. System śledzi wykorzystanie przestrzeni dyskowej przez plik i alokuje tylko niezbędne obszary. Przykładowo, jeśli zapisujemy kilka bajtów na początku pliku, a następnie piszemy na offsecie 10000 tego pliku, powstaje przerwa pomiędzy tymi offsetami. Omawiane systemy plików nie poszukują wolnych bloków do pokrycia tej przerwy. Zamiast tego system tworzy nowy extent z właściwym polem óffset wewnątrz pliku". Jeśli jakaś aplikacja chce przeczytać jakiś bajt z powstałej przerwy, zwracana jest wartość *null*, ponieważ nie ma tam żadnej informacji. W końcu przerwa zostanie wypełniona przez inne aplikacje, które pisały na offsecie wewnątrz przerwy.

4.4.2 Dynamiczna alokacja i-węzłów

Jednym z głównych problemów systemów plików podobnych do UFS jest ustalona liczba i-węzłów. Ogranicza to maksymalną liczbę tworzonych obiektów. Z drugiej strony, ustalenie większej liczby i-węzłów, które potem nie będą wykorzystywane, powoduje marnowanie miejsca.

Omawiane systemy plików stosują dynamiczną alokację i-węzłów, zorganizowanych w B+ drzewa. Dodatkowo istnieją struktury, które pomagają zaalokować i-węzły blisko innych obiektów systemu plików.

Użycie dynamicznych i-węzłów jest złożone i czasochłonne, pozwala jednak na przekroczenie limitów starych systemów plików.

Tabela 4.4 przedstawia techniki wykorzystywane w konkretnych systemach.

Rozdział 5

Second Extended File System

Drugi Rozszerzony System Plików (*Second Extended File System*, w skrócie *Ext2*) pojawił się w pierwszej wersji alfa w styczniu 1993r. Autorem systemu był Rémy Card z Universite Pierre et Marie Curie (Paryż). System powstał jako rozszerzenie systemu Extfs, który z kolei miał na celu zastąpienie Minixa - pierwszego systemu plików w Linuxie.

W chwili obecnej Ext2 jest najpowszechniej używanym systemem plików w Linuxie. Do niedawna Ext2 był podstawowym systemem plików dla wielu dystrybucji Linuxa.

5.1 Podstawowe cechy systemu Ext2

System Ext2 został zaprojektowany z uwzględnieniem możliwości rozszerzenia jego funkcjonalności z zachowaniem kompatybilności wstecz, z czego osoby odpowiedzialne za rozwój tego systemu wielokrotnie korzystały. W efekcie, powstało wiele rozszerzeń i drobnych usprawnień. W tym rozdziale skoncentruję się na „standardowej" wersji systemu Ext2.

Do cech systemu Ext2 należą m.in.:

Typy plików Ext2 wspiera Unixowe typy plików (zwykłe pliki, katalogi, dowiązania symboliczne, specjalne pliki urządzeń)

Nazwy plików mogą mieć do 255 znaków; limit może zostać powiększony do 1012

Szybkie dowiązania - nazwy plików do 60 znaków są trzymane w i-węźle

Rozmiar bloku - 1024, 2048 lub 4096 bajtów

BSD/System V - wybór sposobu zachowania systemu przy tworzeniu plików

Zapis synchroniczny - zmniejsza ryzyko utraty spójności kosztem spowolnienia pracy

Automatyczne sprawdzanie systemu po awarii oraz co określony czas

Superblok	Bitmapa bloków	Bitmapa i-węzłów	Tablica i-węzłów	Bloki danych

Rysunek 5.1: Schemat grupy bloków w systemie Ext2

Struktury meta-danych systemu Ext2 zawierają pola umożliwiające (teoretycznie) korzystanie z udogodnień takich jak *fragmentacja*, czyli trzymanie kilku małych plików w obrębie bloku czy *Access Control List*, umożliwiającą stosowanie bardziej skomplikowanej hierarchii uprawnień dostępu do plików niż tradycyjna (właściciel, grupa, wszyscy). W praktyce, możliwości te nie są wykorzystywane.

5.2 Struktura fizyczna

Fizyczna struktura systemu Ext2 jest oparta na pojęciu *bloku*, który jest podstawową jednostką danych na dysku. Takie podejście pochodzi z systemu FFS z BSD, choć tam bloki były bliżej związane z fizyczną reprezentacją danych na dysku (bloki cylindrów). Rozmiar bloku jest stały w ramach całego systemu plików i może wynosić 1024, 2048 lub 4096 bajtów (wybierane przy tworzeniu systemu plików). Poza *superblokiem* (zobacz 5.2.2) wszystkie struktury *meta-danych* są dopasowane do rozmiaru bloku.

Cały system składa się z grup bloków (zobacz 5.2.1) i ewentualnie sektora startowego umieszczonego na początku.

5.2.1 Grupa bloków

Schemat grupy bloków został przedstawiony na rysunku 5.1. Znaczenie poszczególnych pól:

superblok zawiera podstawowe informacje o systemie plików (zobacz 5.2.2); w każdej grupie bloków jest kopia superbloku systemu plików

deskryptory grup (Group Descriptor Table) tablica, zawierająca informacje o wszystkich grupach bloków; w każdej grupie jest kopia całej tablicy deskryptorów grup (zobacz 5.2.3)

bitmapa bloków mapa zajętości bloków w obrębie grupy

bitmapa i-węzłów mapa zajętości i-węzłów (zobacz 5.3.1) w obrębie grupy

tablica i-węzłów przechowuje i-węzły związane z tą grupą

bloki danych bloki przechowujące dane

5.2.2 Superblok

Superblok to struktura przechowująca bardzo podstawowe informacje o systemie plików, wykorzystywane m.in. przy montowaniu systemu. Pola struktury `ext2_sb_info` określają m.in.:

- liczbę wszystkich oraz wolnych i-węzłów i bloków
- liczbę bloków i i-węzłów w grupie
- czas ostatniego dostępu i montowania oraz punkt montowania
- stan systemu plików (przydatne przy odtwarzaniu po załamaniu systemu):

  EXT2_VALID_FS system odmontowany
  EXT2_ERROR_FS system zamontowany lub niepoprawnie odmontowany

5.2.3 Deskryptor grupy

Deskryptor grupy zawiera:

- mapy bitowe zajętych bloków oraz i-węzłów w obrębie grupy
- tablicę i-węzłów w grupie
- liczbę wolnych bloków oraz i-węzłów w grupie
- liczbę i-węzłów zaalokowanych dla katalogów w grupie

5.3 Pliki i katalogi

System plików Ext2 tworzy hierarchię plików i katalogów korzystając z *i-węzłów* (zobacz 5.3.1). Każdy plik w systemie ma przypisany swój i-węzeł. Każdy i-węzeł ma unikalny numer w obrębie systemu plików. Katalogi są traktowane jak specjalne pliki (zobacz 5.3.3).

5.3.1 I-węzeł

Tak jak bloki dla danych, tak i-węzły są podstawą budowy meta-danych w systemie Ext2. Każdy plik (katalogi traktujemy jak specjalny rodzaj plików - zobacz 5.3.3) w systemie jest jednoznacznie identyfikowany przez i-węzeł. I-węzły są trzymane w *tablicy i-węzłów* w grupie bloków (zobacz 5.2.1). Dzięki mapie zajętości i-węzłów możemy śledzić, które i-węzły są wolne, a które zajęte w obrębie grupy bloków.

Do ciekawszych pól struktury i-węzła należą:

i_mode prawa dostępu do pliku oraz typ pliku

i_uid, i_gid właściel pliku

czas stworzenia, modyfikacji, dostępu oraz usunięcia pliku

i_size rozmiar pliku w bajtach

i_blocks liczba zajętych węzłów; nie musi wynikać z *i_size*, ponieważ system potrafi sobie radzić z „dziurami" w plikach; **Uwaga:** *i_blocks* odnosi się do bloków 512 bajtowych

i_block tablica używana do zlokalizowania danych (zobacz 5.3.2)

i_flags flagi decydujące o zachowaniu się systemu przy dostępie do pliku określonego przez ten węzeł

5.3.2 Lokalizowanie bloków z danymi

Węzeł w systemie Ext2 jest szczególnie dobrze dostosowany do obsługi małych plików. Bloki z danymi mogą być adresowane:

bezpośrednio - adresy pierwszych `EXT2_NDIR_BLOCKS` (z reguły 12) bloków z danymi są zapisane bezpośrednio w i-węźle

pośrednio - adresy bloków z danymi są zapisane w bloku, którego adres znajduje się w i-węźle

podwójnie-pośrednio - jw., ale są dwa pośrednie bloki

potrójnie-pośrednio - jw., ale mamy aż trzy pośrednie bloki

Sposób adresowania bloków z danymi został przedstawiony na rysunku 5.2.

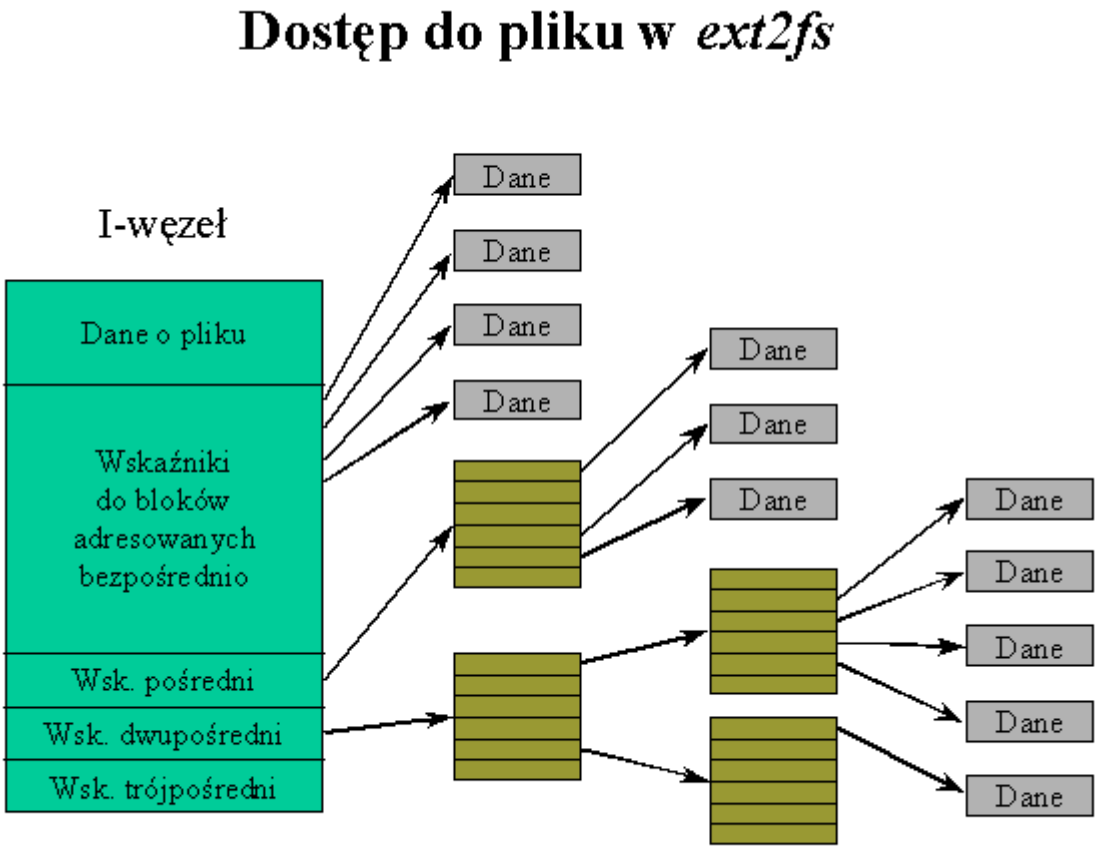
5.3.3 Struktura *ext2_dir_entry*

Katalogi są przechowywane tak samo jak pliki. Można je odróżnić na podstawie typu pliku w polu *i_mode* w węźle (zobacz 5.3.1). Dla opisu zawartości katalogu system przechowuje po jednej strukturze *ext2_dir_entry* dla każdego pliku wewnątrz katalogu. Struktura ta zawiera numer i-węzła, nazwę pliku i jej długość oraz długość całego wpisu (do znalezienia kolejnej).

5.4 Awarie systemu

Zamierzeniem autorów systemu Ext2 było stworzenie systemu szybkiego, ale przede wszystkim bezpiecznego i solidnego (robust). Celowo zrezygnowano ze stosowania b-drzew jako struktur danych. Według autorów, skutki awarii podczas operacji równoważenia drzewa mogłyby być bardzo trudne do naprawienia.

System Ext2 posiada kilka cech, ułatwiających okresowe sprawdzanie poprawności systemu oraz naprawianie go po awarii.



Rysunek 5.2: Dostęp do bloków z danymi z i-węzła

5.4.1 Stan systemu plików

System Ext2 przechowuje w superbloku znacznik określający, czy system został poprawnie odmontowany (zobacz 5.2.2). Dzięki temu przy ładowaniu systemu można sprawdzić, czy system plików wymaga sprawdzenia, czy też został poprawnie odmontowany i dane są w spójnym stanie.

5.4.2 Liczniki montowań

System Ext2 przechowuje w superbloku datę ostatniego sprawdzenia systemu plików oraz liczbę montowań od sprawdzenia (zobacz 5.2.2). Możemy wymusić automatyczne sprawdzanie poprawności systemu plików co określoną liczbę montowań (pole *s_max_mnt_count* w superbloku, domyślnie stała *EXT2_DFL_MAX_MNT_COUNT* równa 20) lub co pewien czas (pole *s_checkinterval* w superbloku, domyślnie wyłączone).

5.4.3 fsck

Program *fsck* jest bardzo silnym narzędziem umożliwiającym sprawdzanie i naprawianie systemów plików (z bardzo dobrym wsparciem dla Ext2). Przy sprawdzaniu systemu Ext2 wykorzystuje redundację związaną z zapisywaniem superbloku (5.2.2) oraz tablicy deskryptorów grup (*sec:gb*) w każdej grupie bloków.

5.5 Zastosowania systemu Ext2

Ext2 był do niedawna podstawowym systemem plików dla stacji roboczych. Ze względu na jego stabilność, długą obecność na rynku oraz bardzo dobre narzędzia (zobacz 5.4.3), był również wybierany dla serwerów (w tym dla wielkich maszyn, obsługujących bardzo duże ilości danych).

Brak mechanizmu *journalingu* oraz niska wydajność dla dużych ilości plików sprawiły, że Ext2 jest obecnie wypierany przez nowsze systemy plików.

Rozdział 6

Third Extended File System

Trzeci Rozszerzony System Plików wywodzi się bezpośrednio z Ext2 (zobacz 5). Został stworzony przez Stephen'a Tweedie z firmy RedHat. Wersja alfa pojawiła się w 1999 roku.

Podstawową różnicą w stosunku do poprzednika jest mechanizm księgowania (journaling), który wprowadzono w celu skrócenia czasu „wstawania" systemu po awarii. Dodatkowo, wprowadzono na stałe kilka usprawnień, które były wcześniej dostępne jako łaty dla systemu Ext2.

6.1 Podstawowe cechy systemu Ext3

Ze względu na duże podobieństwo, zarówno w sposobie działania jak i w strukturach danych, system Ext3 będzie omówiony w kontekście dodatkowych cech w stosunku do poprzedniej wersji (zobacz 5.1). Warto wymienić następujące możliwości systemu Ext3:

Journaling - mechanizm księgowania (zobacz 4) zwiększa bezpieczeństwo systemu (niepodzielność operacji), ale przede wszystkim skraca do minimum czas sprawdzania systemu plików po awarii.

Indeksowane katalogi znacznie zwiększają wydajność systemu przy dużej ilości plików (zobacz 6.3.2)

Zapis synchroniczny - w najnowszych wersjach systemu Ext3 (jądro 2.4.19) działa ponad 10 razy szybciej od wersji z Ext2 (dotyczy zapisu do zwykłych plików)

6.1.1 Kompatybilność z Ext2

Celem autora systemu Ext3 było usprawnienie Ext2, a nie tworzenie zupełnie nowego systemu. Dało to następujące korzyści:

- Możliwość montowania systemu Ext3 jako Ext2 (o ile system został poprawnie odmontowany); w chwili obecnej nie jest możliwe montowanie systemu Ext2 jako systemu Ext3, ale ma to być możliwe w przyszłości

- Możliwość korzystania z wielu sprawdzonych programów narzędziowych (m.in. fsck)
- Możliwość łatwej i bezpiecznej migracji z Ext2 (nie trzeba nawet odmontowywać systemu Ext2 podczas konwersji do Ext3) przy pomocy programu *tune*
- Przewidywalność, stabilność i uniknięcie wielu błędów
- Zaufanie użytkowników

6.2 Struktura fizyczna

Nie wprowadzono wielu zmian w stosunku do Ext2. Do najistotniejszych należy wprowadzenie pól związanych z obsługą księgowania.

Należy zauważyć, że mechanizm księgowania jest niezależny od systemu Ext3 (zobacz 6.4.1).

6.2.1 Superblok

Wprowadzono dodatkowe pola związane z obsługą księgowania:

s_journal_inum numer i-węzła pliku księgującego

s_journal_dev urządzenie, na którym znajduje się ten plik

s_last_orphan początek listy z i-węzłami do usunięcia

6.3 Pliki i katalogi

6.3.1 Struktura *ext3_dir_entry2*

Struktura *ext3_dir_entry_2* różni się od struktury *ext3_dir_entry* tym, że w systemie Ext3 pole *name_len* jest liczbą 8 bitową (więc długość nazwy pliku nie może przekroczyć 255) a pozostałe 8 bitów jest wykorzystywane do przechowywania pola *file_type*.

Warto zauważyć, że w systemie Ext3 istnieje również starsza wersja struktury *dir_entry*, a w nowszych wersjach systemu Ext2 występuje również nowa wersja tej struktury.

Dodatkowe pole umożliwia nam szybszy dostęp do informacji o typie pliku (nie musimy pobierać i-węzła). Z drugiej strony, zmniejszenie *name_len* do 8 bitów uniemożliwia korzystanie z bardzo długich nazw plików (co było teoretycznie możliwe w Ext2), choć nie jest to specjalna strata.

6.3.2 Indeksowane katalogi

W systemie Ext3 wprowadzono możliwość wydajniejszej obsługi katalogów zawierających dużo plików. W tym celu zastąpiono strukturę listową stosowaną do tej pory przez mechanizm oparty na haszowaniu i strukturach drzewiastych. Do włączenia mechanizmu indeksowania katalogów służy flaga *EXT3_INDEX_FL* i-węzła.

Uwaga: opis dla bloków o wielkości 4096 bajtów.

Struktura indeksująca

System rezerwuje pierwsze 512 bloków (do zerowego do 511.) katalogu dla przetrzymywania struktury indeksującej. Pozostałe bloki-liście (tzn. od 512.) to zwykłe bloki z wpisami, więc mogą być obsługiwane przez standardową funkcję *ext3_readdir*.

Blok zerowy jest korzeniem struktury drzewiastej i przechowuje na pierwszych 8 bajtach nagłówek (długość nagłówka, typ indeksowania, wersja, głębokość w drzewie). Pozostała część korzenia jest traktowana jak każdy blok indeksujący.

Blok indeksujący jest złożony z 512 wpisów postaci: *(klucz, adres)*, gdzie klucz jest wartością funkcji haszującej (na ostatnim bicie jest **znacznik kolizji** mówiący, że kolidujące klucze zostały rozbite pomiędzy dwa bloki) a adres to logiczny adres odpowiedniego bloku lub kolejnej struktury indeksującej w przypadku struktur wielopoziomowych.

W ten sposób można obsłużyć około 90000 plików w katalogu. Jeśli to nie wystarcza, można dołożyć kolejne poziomy – dla następnego otrzymujemy już około 50 milionów wpisów.

Szukanie

Przeszukiwanie katalogu przebiega w następujący sposób:

1. obliczenie klucza na podstawie nazwy pliku, służy do tego funkcja
   `int ext3fs_dirhash(const char *name, int len, struct dx_hash_info *hinfo)`
2. przeczytanie korzenia struktury indeksującej
3. przeszukiwanie (binarne lub liniowe w starszych wersjach) indeksów, aż do najniższego poziomu
4. przeczytanie liścia i postępowanie jak w wersji ze strukturą listową
5. w przypadku kolizji – przeszukiwanie kolejnych bloków indeksujących

Wstawianie

Wstawianie jest bardziej złożone:

1. przeszukanie, jak w poprzednim punkcie

2. jeśli blok pełny, to rozbicie (następny punkt)

3. wstawienie do liścia w standardowy sposób

Rozbijanie Rozbijanie pełnego węzła (węzła w drzewie indeksującym) polega na przydzieleniu jego kluczy do dwóch bloków. Oczywiście zależy nam, aby w obu nowych blokach było około połowy kluczy z pierwotnego bloku. W tym celu sortujemy klucze, wybieramy środkowy i dokonujemy podziału.

Kolizje W przypadku kolizji kluczy, staramy się umieścić wpisy w tym samym bloku. Jeśli nie jest to możliwe, to zaznaczamy najmniej znaczący bit w kluczu, co oznacza, że należy szukać w kolejnym bloku.

Funkcje haszujące

Krytyczne znaczenie dla wydajności systemu ma jakość wybranej funkcji haszującej. Wyniki zaprezentowane w następnym punkcie pokazują, że dla losowych danych system działa bardzo wydajnie. W celu umożliwienia użycia lepszej funkcji w przyszłości, dodano identyfikator funkcji haszującej (w nagłówku struktury).

Polepszenie wydajności

Wyniki testów dla tworzenia wielu plików w katalogu zostały zebrane w Tabeli 6.1. Wyniki te uwidaczniają różnicę pomiędzy haszowaniem (czas bliski liniowemu) w Ext3 i strukturą listową (czas kwadratowy) w Ext2.

Plików	10000	20000	30000	40000	50000	60000	70000	80000	90000
Ext3	1.35s	2.72s	4.33s	5.89s	7.04s	8.61s	9.98s	12.06s	13.4s
Ext2	23.6s	1m20s	3m9s	5m48s	9m31s	13m52s	19m24s	25m36s	33m18s

Tablica 6.1: Tworzenie dużej liczby plików w Ext3 (z indeksowanymi katalogami) i Ext2 (katalogi oparte na listach)

Użycie indeksowanych katalogów potrafi drastycznie zwiększyć wydajność systemu, płacimy jednak 2MB za stworzenie struktury. Dlatego też, indeksowanie należy stosować tylko dla katalogów, w których przewidujemy bardzo dużą liczbę plików.

6.3.3 Nowy sposób indeksowania

W najnowszych, rozwojowych wersjach jądra (oraz w niektórych gałęziach wersji stabilnej) zmieniono opisaną wyżej prostą, drzewiastą strukturę indeksującą na opartą o drzewa czerwono-czarne.

Struktura *fname*

Struktura *fname* zawiera węzły drzewa r-b, używane do przechowywania wpisu o katalogu. Ciekawe pola tej struktury to:

hash czyli klucz (podstawowy)

minor_hash niższa wartość; umożliwia przechowywanie 64 bitowych kluczy (łącznie z *hash*, ale obecnie **nieużywane** ze względu na problemy związane ze współpracą z VFS

rb_hash węzeł drzewa r-b

next do łączenia w listę w przypadku kolizji

inode i-węzeł pliku odpowiadającego wpisowi

file_type dzięki temu od razu znamy typ pliku (bez zaglądania do i-węzła)

6.4 Księgowanie

Mechanizm księgowania był głównym powodem powstania systemu Ext3. Journaling w Ext3 różni się w kilku znaczących szczegółach w porównaniu z podejściami zaprezentowanymi w innych systemach plików.

6.4.1 Journaling Block Device

Mechanizm księgowania jest realizowany niezależnie od systemu plików Ext3. Do właściwego journalingu służy interfejs JBD – rozwijany niezależnie od Ext3 interfejs, umożliwiający dodanie księgowania do dowolnego systemu plików na urządzeniu blokowym. Z poziomu Ext3 obsługa księgowania ogranicza się do wywoływania odpowiednich funkcji JBD.

Journal jest przechowywany w i-węźle.

.journal

Jeśli system Ext3 powstał z systemu Ext2 bez odmontowywania, to journal będzie widoczny jako plik *.journal* w korzeniu struktury katalogów.

6.4.2 Journal

JBD zachowuje w journalu całe fizyczne bloki zamiast pojedynczych bajtów. Zalety i wady takiego postępowania zostaną omówione w następnych punktach.

Zalety przechowywania bloków

Podstawową przewagą przechowywania całych bloków nad przechowywaniem modyfikowanych bajtów jest olbrzymie uproszczenie całego systemu. Nie ma potrzeby budowania specjalnych struktur do opisywania modyfikacji, wystarczy przechowywać dane tak samo, jak czynimy to przy zapisywaniu ich na dysk.

Przechowywanie danych w blokach zmniejsza zużycie procesora, ponieważ przesyłanie i zapisywanie bloku nie wymaga przesyłania ani przetwarzania danych w pamięci w celu przygotowania ich do zapisania na dysku.

Wady przechowywania bloków

Do wad takiego postępowania należy zaliczyć duże zużycie pamięci (zapisujemy cały blok nawet przy modyfikacji pojednyczego bitu). Teoretycznie, przechowywanie dużych bloków zamiast pojedynczych bajtów powinno znacznie obniżyć wydajność systemu, ale tak nie jest (zobacz poprzedni punkt).

System stara się optymalizować zużycie pamięci poprzez „ściskanie" (*squish*) bloków – sprawdza, czy nie przechowuje już tego samego bloku w związku z poprzednią modyfikacją i zapisuje obydwa tylko raz.

6.4.3 Księgowanie danych

Cechą odróżniającą system Ext3 od innych systemów plików z księgowaniem jest możliwość księgowania nie tylko *meta-danych*, ale również *danych*. Oznacza to, że np. o ile reiserfs zachowa prawidłowe struktury systemowe, ale być może utraci dane z pliku, który był modyfikowany w trakcie awarii, o tyle Ext3 (w trybie *journal*– zobacz następny punkt) zarówno naprawi własne struktury, jak i odzyska nasz plik.

6.4.4 Tryby działania

Mechanizm księgowania ma następujące tryby działania (ustalane przy montowaniu systemu plików):

data=writeback księgowanie tylko *meta-danych*, podobnie jak w pozostałych systemach plików; teoretyczne, powinno dawać najwyższą wydajność

data=journal pełne księgowanie (*meta-dane* i *dane*); zapisuje bloki dwa razy – najpierw do journalu, potem na właściwą pozycję; daje pełne bezpieczeństwo, choć (teoretycznie) kosztem niskiej wydajności; duży obszar przeznaczony na journal znacząco zwiększa wydajność

data=ordered opcja dodana niedawno; zapisywane są tylko *meta-dane*, jednak, dzięki specjalnemu postępowaniu, zachowuje się podobnie do pełnego księgowania:

- łączy modyfikacje danych i odpowiadających im meta-danyh w logiczne całości, zwane *transakcjami*
- zapisuje na dysk (na docelowe pozycje) dane
- dopiero po zapisaniu danych na dysk zapisuje meta-dane do journala

W ten sposób zachowana jest spójność całości przy mniejszej redukcji wydajności, niż w przypadku pełnego księgowania.

6.4.5 Mechanizm księgowania od wewnątrz

Księgowanie (zobacz 4) w Ext3 przebiega w następujący sposób:

- Modyfikowane bloki są umieszczane w *buforach*
- Bufory są łączone w jednostki logiczne zwane *transakcjami*; transakcja działa atomowo – modyfikacje dotyczą wszystkich buforów w obrębie transakcji lub żadnego
- Transakcja przechodzi przez *punkty kontrolne*
- Zakończona(logicznie) transakcja może zostać *zatwierdzona* (commit) i zapisana do logu, oznacza to, że modyfikacje należy zapisać na dysku
- Zatwierdzona transakcja jest zapisywana (flush) na dysk; w razie awarii wszystkie dane można odzyskać, bo są już zapisane w logu
- Mechanizm księgowania odnotowuje, że transakcja została zakończona (fizycznie)

Najważniejsze struktury danych związane z księgowaniem zostały omówione w kolejnych punktach.

Transakcja – transaction_t

Fundamentalnym pojęciem w procesie księgowania w Ext3 jest *transakcja*, reprezentowana przez strukturę *transaction_t*. Do jej ciekawszych pól należą:

t_journal journal dla tej transakcji (zobacz punkt *journal_t*)

t_state stan transakcji; używane są flagi:

T_RUNNING akceptuje kolejne zmiany

T_LOCKED zmiany w trakcie, ale nowych już nie przyjmujemy

T_FLUSH zmiany zakończone, ale wciąż zapisujemy na dysk

T_COMMIT dane zapisane, zapisujemy „commit record”

T_FINISHED zakończone, transakcję przechowujemy dla checkpoint'ów

t_log_start gdzie w logu zaczyna się commit dla tej transakcji

t_buffers lista cykliczna (dwukierunkowa) buforów dla meta-danych transakcji

t_nr_buffers liczba buforów na liście *t_buffers*

t_reserved_list zarezerwowane dla transakcji bufory, które nie były jeszcze modyfikowane

t_sync_datalist lista buforów, które należy „flushować" przed operacją commit (ochraniane przez *journal_datalist_lock*)

t_checkpoint_list jw, ale przed osiągnięciem checkpoint

t_cpnext, t_cpprev następna i poprzednia transakcja, spośród oczekujących na checkpoint

t_forget lista buforów do usunięcia, po operacji commit

Transakcja śledzi wszystkie bufory zmodyfikowane oraz bufory zatwierdzone (commit) ale jeszcze nie zapisane (flush).

Bufor – journal_block_tag_t

Pojedynczy bufor jest opisany przez prostą strukturę *journal_block_tag_s*:

t_blocknr numer bloku na dysku

t_flags flagi

Superblok journala – journal_superblock_t

Superblokiem dla journala jest struktura *journal_superblock_t*, do jej ciekawszych pól należą:

s_blocksize rozmiar bloku

s_maxlen ilość bloków w pliku journala

s_first pierwszy blok dla logu

s_max_transaction limit bloków dla transakcji

s_max_trans_data limit bloków danych dla transakcji

Journal – journal_t

Struktura *journal_t* przechowuje wszystkie informacje o stanie księgowania dla jednego systemu plików. Jest używana również do przechowywania informacji o stanie zapisywania logów.

j_flags dostępne flagi to:

- **JFS_UNMOUNT** wątek księgowania jest niszczony
- **JFS_ABORT** księgowanie anulowane z powodu błędów
- **JFS_FLUSHED** superblok journala został zapisany (flush)
- **JFS_LOADED** superblok journala został załadowany

j_superblock superblok dla journala

j_running_transaction bieżąca transakcja

j_commiting_transaction transakcja wysyłana do zapisu na dysk

j_wait_transaction_locked kolejka dla transakcji oczekujących na rozpoczęcie potwierdzania (commit)

j_wait_done_commit jw, na zakończenie potwierdzania

j_wait_logspace jw, dla oczekujących na zakończenie punktów kontrolnych

j_wait_updates jw, oczekujące na zakończenie modyfikowania

j_checkpoint_sem semafor do ochrony punktów kontrolnych

j_sem główny semafor

j_head pierwszy nieużywany blok w journalu

j_tail najstarszy używany blok w journalu

j_free ile wolnych bloków

j_first, j_last numer pierwszego bloku, który można użyć oraz pierwszego za ostatnim

j_dev, j_blocksize, j_blk_offset urządzenie oraz rozmiar bloku dla lokalizacji journala

j_fs_dev urządzenie, na którym trzymany jest księgowany system plików; jeśli lokalnie, to równe *j_dev*

j_maxlen maksymalny rozmiar dla journala

System plików	ReiserFS	Ext2	Ext3 (ordered)	Ext3 (writeback)	Ext3 (journal)
Czas odczytu	67	78	93	74	7

Tablica 6.2: Czas (sekundy) odczytu 16MB danych z systemu plików, na który jednocześnie prowadzony jest zapis

j_tail_sequence najstarsza transakcja w logu (numer)

j_commit_sequence transakcja ostatnio zatwierdzona

j_max_transaction_buffers ograniczenie na liczbę bloków z meta-danymi dla transakcji

j_commit_interval ograniczenie na czas trwania transakcji przed zatwierdzeniem

j_all_journals lista wszystkich journali

6.5 Wydajność

Przy wyłączonej usłudze indeksowania katalogów (zobacz 6.3.2 oraz wyniki testu w Tabeli 6.1), system Ext3 ma zbliżoną wydajność oraz zużycie procesora do Ext2. Przy operacjach zapisu, jest o kilka procent wolniejszy (dane są jeszcze księgowane).

6.5.1 Zaskakujący test

Teoretycznie, tryb księgowania *data=journal* (zobacz 6.4.4) powinien mieć najniższą wydajność, jako że dane są zapisywane na dysk dwukrotnie. Jednak w specyficznych warunkach może się okazać inaczej, co pokazał Andrew Morton w interesującym eksperymencie.

Test polegał na odczycie 16 MB porcji danych z systemu plików, na który jednocześnie zapisywane są duże ilości danych. Rezultaty zebrane w Tabeli 6.2 pokazują, że Ext3 wyjątkowo dobrze nadaje się do systemów, w których dane są jednocześnie zapisywane i odczytywane, a wydajność odczytu ma dla nas krytyczne znaczenie. Zbliżone wyniki uzyskamy również, gdy zapisujemy na system plików Ext3 i czytamy jednocześnie z innego systemu plików.

6.6 Zastosowanie

System Ext3 można uznać za ewolucyjnego następcę systemu Ext2, oferującego stabilność i renomę poprzednika plus garść cech, które pozwalają umieścić go pośród „nowoczesnych" systemów plików. Jako taki, wydaje się być bardzo dobrym rozwiązaniem dla stacji roboczych i komputerów domowych. Opinię tę potwierdza m.in. firma RedHat, która sugeruje

Ext3 jako system plików dla swojej dystrybucji linuxa. Gwarancją dalszego wsparcia firmy RedHat dla Ext3 jest również fakt, że tam właśnie pracuje twórca Ext3.

Z drugiej strony, dzięki indeksowanym katalogom (zobacz 6.3.2), a przede wszystkim dzięki wydajnemu i stabilnemu mechanizmowi księgowania (zobacz 6.4), system Ext3 można polecić jako system dla serwerów.

W chwili obecnej, system Ext3 nie nadaje się do następujących zastosowań:

- serwery pocztowe, jako że wymagają one szybkiego zapisu synchronicznego na specyficznych strukturach katalogów; Ext3 nie wspiera jeszcze w pełni tego mechanizmu
- serwery z dużą ilością bardzo małych plików, z względu na przewagę wydajnościową oraz lepsze wykorzystanie przestrzeni dyskowej przez inne systemy – reiserfs (zobacz 9) oraz xfs (zobacz 7)
- bardzo szybkie serwery, ze względu na słabą skalowalność

Rozdział 7

XFS

We wczesnych latach 90-tych SGI zdało sobie sprawę, że dotychczasowy system plików EFS, stosowanym systemie operacyjnym IRIX nie radzi sobie ze wzrostem wydajności i skalowalności pamięci masowych. System ten obsługiwał partycje o rozmiarze do 8 GB, miał klasyczne ograniczenie 32-bitowego systemu plików - maksymalny rozmiar pliku wynosił 2 GB oraz wolno przywracał spójność po krachu systemu (używał fsck).

Prace nad *XFS* rozpoczęto w 1993 roku. Pierwsza wersja XFS ukazała się w grudniu 1994 roku. W 1996roku XFS stał się domyślnym systemem plików dla IRIX. W 1999 roku SGI podało, że XFS zostanie wydany na licencji open source i zintegrowany z jądrem Linuxa.

7.1 Podstawowe cechy systemu XFS

System XFS został zaprojektowany w celu zaspokojenia następujących wymagań:

- szybkie odzyskiwanie spójności po krachu systemu
- obsługa dużych systemów plików
- obsługa dużych, rzadkich plików
- obsługa dużych, śąsiednich"plików
- obsługa dużych katalogów
- obsługa dużej liczby plików

Do cech systemu XFS należą m.in.:

Journaling - księgowanie (*Journaling*) pozwala na szybkie odzyskanie spójności systemu po niespodziewanym krachu, niezależnie od ilości zarządzanych plików.

Szybkie transakcje - XFS wykorzystuje zalety księgowania, jednocześnie poprzez swoje struktury i algorytmy księgowania zapewniając niezwykle szybkie zapisywanie transakcji.

Wysoka skalowalność - XFS jest 64-bitowym systemem plików zapewniającym spójność i teoretycznie może mieć wielkość 18 milionów terabajtów. System ten jest podzielony na regiony nazywane grupami alokacji (*AG - Allocation Groups*).

Quota - XFS wspiera quoty zarówno grupy, jak i użytkownika

Posix Access Contrl Lists (ACLs) - Linuxowy XFS wspiera semantykę i interfejsy ACL opisane w standardzie Posix 1003.1e.

Maksymalny rozmiar pliku - 9 mln terabajtów dla IRIX i 16TB dla 4K rozmiaru strony oraz 64TB dla rozmiaru strony 16K w przypadku Linuxa 2.4.

Maksymalny rozmiar systemu plików - 18 mln TB dla IRIX i 2 TB dla Linuxa 2.4.

Rozmiar bloku - 512B do 65536B (IRIX)

Kompatybilność - kompatybilny z NFS wersja 3.

7.2 Struktura fizyczna

XFS ma architekturę modułową. Struktura XFS jest zbliżona do tej w tradycyjnych systemach plików, z dodatkiem menadżera transakcji i wolumenów. XFS obsługuje wszystkie standardowe interefjsy plików UNIX i jest całkowicie zgodny z POSIX i XPG4.

XFS jest podzielony na kilka części, z których każda jest odpowiedzialna za inną część funkcjonalności systemu plików.

Najważniejszym modułem jest menadżer przestrzeni (*Space Manager* - zobacz 7.2.1). Pozostałe to:

- Menadżer we/wy (*I/O Manager* - zobacz 7.2.2)
- Menadżer katalogów (*Directory Manager* - zobacz 7.2.3)
- Menadżer transakcji (*Transaction Manager* - zobacz 7.2.4)
- Menadżer wolumenów (*Volume Manager* - zobacz 7.2.5)
- Bufor (*Buffer Cache*)
- Sterowniki dyskowe (*Disk Drivers*)

7.2.1 Menadżer przestrzeni

Głównym zadaniem menadżera przestrzeni jest obsługa wszystkich obiektów w systemie plików, a zwłaszcza zarządzanie i-węzłami oraz ekstentami.

Menadżer przestrzeni zarządza alokacją przestrzeni dyskowej w systemie plików. Jest odpowiedzialny za odwzorowanie pliku (ciągu bajtów) na ciąg bloków dyskowych. Dodatkowo zarządza wewnętrzną strukturą systemu plików, tzn. grupami alokacji, i-węzłami i wolną przestrzenią.

Struktury dyskowe

Niżej opisane zostały logiczna struktura i fizyczny układ całej przestrzeni dyskowej, za którą odpowiedzialny jest menadżer przestrzeni.

superblok - jest korzeniem wszystkich informacji systemu plików. Umieszczony jest na początku systemu plików (offset 0). Zawiera informacje umożliwiające znalezienie wszystkich innych części systemu plików. Jedynymi polami superbloku, które ulegają zmianie podczas normalnych operacji są pola statystyczne wykorzystywane przez **df**. Zmiany tych informacji muszą być księgowane w celu zapewnienia spójności systemu. Dodatkowo przy dynamicznej zmianie rozmiaru systemu plików zmieniają się pola rozmiar systemu plików oraz ćałkowity rozmiar"i ćałkowita liczba- zmiany te są księgowane. Przechowywane są kopie superbloku - w każdej grupie alokacji i na końcu systemu plików.

nagłówek grupy alokacji - XFS dzieli partycję na kilka większych części, tzw. grupy alokacji, czyli porcje przestrzeni o rozmiarze od 16 MB do 4 GB każda (być może poza ostatnią). Grupy alokacji ułatwiają wielowątkowe zarządzanie systemem plików - poprawia to działanie systemu, szczególnie w przypadku systemów wieloprocesorowych. Należy jednak mieć na uwadze, że wieksza liczna grup alokacji przy mniejszej liczbie procesorów zmniejsza wydajność systemu plików. W skład nagłówka grupy alokacji wchodzą:

- **drzewo wolnych ekstentów** - w ramach AG dostępna wolna przestrzeń jest indeksowana w formie pary B+ drzew. Pierwsze z nich to indeks położenia wolnych ekstentów, drugie - ich rozmiarów.
- **drzewo i-węzłów** - w grupie alokacji i-węzły zapamiętywane są w specjalnych B+ drzewach. I-węzły XFS nie różnią się bardzo od klasycznych uniksowych i-węzłów. Również mają swój niepowtarzalny numer, według którego są indeksowane w B+ drzewach
- **struktura pomocnicza** - przyspiesza znajdowanie wolnej przestrzeni

tablica i-węzłów - w XFS i-węzły są alokowane na żądanie, w małych grupach, dynamicznie. Każda grupa alokacji zarządza swoimi i-węzłami. Każda grupa wykorzystuje B+ Drzewo do indeksowania swoich i-węzłów. I-węzły są alokowane w grupach po 64

Numer i-węzła rodzica	licznik wejść

numer i-węzła	długość	tekst nazwy pliku
numer i-węzła	długość	tekst nazwy pliku

i-węzły. B+ drzewo alokacji i-węzłów każdej AG przechowuje ścieżkę do lokalizacji danej grupy i-węzłów oraz informację, czy każdy z i-węzłów w grupie jest wykorzystywany. Same i-węzły nie są przechowywane w B+ drzewie - umieszczone są tam jedynie informacje o tym, gdzie w danej AG znajduje ię dany i-węzeł.

numery i-węzłow - numery i-węzłów w XFS podzielone są na dwie części. Bardziej znaczące bity przechowują numer grupy alokacji, natomiast mniej znaczące - numer i-węzła wewnątrz grupy.

Wewnętrzne komponenty i interfejsy

Menadżer grup alokacji - zawiera wybór grupy alokacji dla pliku i katalogu

Alokator bloków i ekstentów - dla wolnej przestrzeni wewnątrz grupy alokacji; przestrzeń może być alokowana gdziekolwiek w obiekcie, lub w pobliżu określonego bloku, lub w określonej lokacji

Alokator i-węzłów

Menadżer przestrzeni plików

Zarządzanie statystykami - zawiera VFS_STATVFS

7.2.2 Menadżer we/wy

Modułem pośredniczącym pomiędzy VFS IRIX-a a menadżerem przestrzeni jest menadżer we/wy.

Jest on odpowiedzialny za obsługę żądań we/wy i zależy od menadżera przestrzeni w alokowaniu i przechowywaniu informacji o przestrzeni plików. Zajmuje się on całościową obsługą plików.

7.2.3 Menadżer katalogów

Menadżer katalogów zajmuje się całościową obsługą katalogów.

Format wewnętrznych i-węzłow

- katalogi przechowywane wewnątrz i-węzła mają strukturę:
- rozmiar i-węzła rośnie/zmniejsza się dokładnie do wymaganego rozmiaru
- jeśli i-węzeł przestaje wystarczać, przenosi się informacje do pojedynczego liścia
- nie ma sortowania nazw - przeszukiwanie liniowe

Bloki liści

Liście mają strukturę:

Węzły wewnętrzne

Węzły wewnętrzne mają postać:

Struktury danych

Argumenty operacji - przechowywane w strukturze *xfs_dir_name*

bloki B-drzewa - wszystkie informacje o bloku w B-drzewie przechowywane są w strukturze *xfs_dir_state_blk*

ścieżka do korzenia - ścieżka do korzenia, która prowadzi do danego miejsca przechowywana jest w strukturze *xfs_dir_state_path*

kontekst operacji katalogowej - kompletny kontekst danej operacji katalogowej przechowywany jest w strukturze *xfs_dir_state*

7.2.4 Menadżer transakcji

Menadżer transakcji zarządza aktualizacją danych.

Odpowiada on także za operacje księgowania, pozwalające na szybką naprawę systemu po awarii. Jego zadaniem jest zapewnienie atomowości wszystkich aktualizacji metadanych, tj. drzew i-węzłow, i-węzłów, zawartości katalogów, drzew ekstentów zawartości plików, drzew wolnych ekstentów, bloków nagłówkowych grup alokacji oraz superbloków.

7.2.5 Menadżer wolumenów

Menadżer wolumenów używany przez XFS, znany jako XLV, zapewnia warstwę abstrakcji pomiędzy XFS i urządzeniami dyskowymi.

XFS nie wie nic o warstwie urządzeń, na których działa. Ułatwia to implementację systemu plików.

7.2.6 Zarządzanie Superblokiem

Superblok to struktura opisująca cały system plików.

zarządzanie

bufor superbloku - superblok XFS przechowywany jest w prywatnym buforze systemu plików. Dostęp do bufora odbywa się poprzez wywołania *xfs_ getsb()* i *xfs_ trans_ getsb()*. Bufor jest czytany jedynie w momencie montowania, a następnie jest przechowywany z kopią dyskową superbloku.

modyfikacje superbloku - w celu zmniejszenia czasu przez jaki bufor superbloku jest zablokowany podczas transakcji, superblok w rzeczywistości nie jest blokowany ani zmieniany aż do czasu na krótko przed zatwierdzeniem transakcji.

superblok in-core - może być używany do wyszukiwania informacji o systemie plików, który się nie zmienia, jak również systemowych informacji podsumowujących. Dostęp do bufora superbloku powinien mieć miejsce wyłącznie wtedy, gdy coś jest zmieniane. liczbę wszystkich oraz wolnych i-węzłów i bloków

interfejsy zarządzania superblokiem

xfs_trans_mod_sb() - używany do zapisania zmian w licznikach superbloku bez natychmiastowego blokowania bufora superbloku

xfs_trans_getsb() - blokuje buforsuperbloku podczas transakcji. Powinien być wykorzystywany wyłącznie wtedy, gdy transakcja musi zmienić pola superbloku inne niż te, które można zmodyfikować używając xfs_trans_mod_sb()

xfs_getsb() - działa tak jak xfs_trans_getsb() poza tym, że może być używany poza transakcją. Zwraca wskaźnik na zablokowany bufor superbloku. Bufor ten nigdy nie powinien być modyfikowany, ponieważ superblok może być modyfikowany jedynie podczas transakcji.

xfs_mod_incore_sb() - używany do modyfikowania kopii in-core superbloku.

xfs_mod_incore_sb_batch() - pobiera tablicę struktur xfs_mod_sb_t, z których każda określa pole i delte dla tego pola. Pozwala na wykonanie wielokrotnych aktualizacji i ogranicza przez to blokowanie konieczne dla wielokrotnych wywołań xfs_mod_incore_sb()

7.3 Linux XFS a IRIX XFS

XFS pod Linuxa różni się od XFS dla IRIX. Główne różnice są następujące:

- w XFS dla Linuxa 1.0 rozmiar bloku systemu plików jest ograniczony do rozmiaru strony pamięci
- interfes dla ACL i EA (*Extended Attributes*) jest różny w XFS/Linux i IRIX. Jednakże biblioteki poziomu użytkownika zarówno dla ACL, jak i dla EA są dokładnie takie same w Linuxiei IRIX.
- niezapisane ekstenty nie są obsługiwane w XFS Linux 1.0
- system plików XFS Linux jest ograniczony do 2 TB ze względu na ograniczenia w warstwie urządzeń we/wy Linuxa
- Maksymalny osiągalny offset pliku Linux XFS to 16 TB w przypadku 4K i 64 TB w przypadku 16K rozmiaru strony
- Pod IRIX zewnętrzne logi są określane przez menadżera wolumenów, XLV lub XVM. Pod Linuxem zewnętrzne logi mogą być określone jako rozszerzenie opcji -l polecenia mkfs
- interfejs wywołania systemowego quotacl() różni się pomiędzy IRIX i Linuxem, a narzędzi quoty w Linuxie pod wieloma względami różnią się od swoich odpowiedników pod IRIX
- polecenie mkfs w Linux XFS używa standardowo wersji 2 formatu katalogu
- Linux XFS obsługuje montowanie przez etykietę i uuid
- menadżer wolumenów xlv nie jest dostępny w Linux XFS. Obsługa wolumenów w Linux XFS odbywa się za pomocą standardowych Linuxowych menadżerów wolumenów lvm i md

7.4 Awarie systemu

Zamierzeniem autorów systemu XFS było stworzenie systemu szybkiego, jak również bezpiecznego i solidnego (robust).

System XFS wykorzystuje metodę księgowania, co umożliwia bardzo szybkie odzyskanie spójności systemu po jego awarii. Jeśli podczas startu systemu wykrywane jest jego nieprawidłowe zamknięcie, następuje odczyt dziennika (*logu*) i aktualizacja danych na podstawie zawartych w nim wpisów.

7.5 Zastosowania systemu XFS

XFS został zaprojektowany z myślą o dużych systemach plików, dużych plikach i katalogach.

Ze względu na swoje zalety zostały rozpoczęte prace nad przystosowaniem go do innych systemów operacyjnych, takich jak np. Linux.

Rozdział 8

Journaled File System

Journaled File System (w skrócie *JFS*) został stworzony dla systemów zorientowanych na transakcje.

8.1 Podstawowe cechy systemu JFS

System JFS został zaprojektowany w celu zaspokojenia następujących wymagań:

- skalowalność
- niezawodność
- wydajność

8.2 Struktura fizyczna

8.2.1 Partycje

JFS jest zbudowany na partycji. Partycja ta posiada:

- stały rozmiar bloku partycji, z dozwolonymi wartościami 512, 1024, 2048 lub 4096 bajtów. Rozmiar bloku partycji definiuje najmniejszą jednostkę we/wy obsługiwaną przez partycję
- rozmiar *PART_NBlocks*, który jest liczba bloków partycji dyskowej
- abstrakcyjną przestrzeń adresową, *[0..PART_NBlocks-1]*, bloków partycji

8.2.2 Agregaty

JFS oddziela pojęcie obszaru alokacji przestrzeni dyskowej, zwanego agregatem, od pojęcia poddrzewa montowalnego systemu plików, nazywanego zestawem plików. Istnieje dokładnie jeden agregat na partycji, może jednak istnieć wiele zestawów plików w agregacie.

Agregat jest tablicą bloków dyskowych posiadających specjalny format, zawierających superblok i mapę alokacji. Superblok identyfikuje partycję jako agregat JFS, natomiast mapa alokacji opisuje stan alokacji każdego bloku danych wewnątrz agregatu.

8.2.3 Grupy alokacji

Grupy alokacji (*Allocation Groups - AG*) dzielą przestrzeń agregatu na kawałki i pozwalają zwiększyć wydajność operacji we/wy JFS.

JFS stara się grupować bloki i i-węzły dyskowe zawierające związane dane. Pliki są często czytane i zapisywane sekwencyjnie, a do plików wewnątrz jednego katalogu często odwołuje się razem.

Trzeba wybrać rozmiar grup alokacji. Aby zmniejszyć liczbę aktualizacji koniecznych do przeprowadzenia, kiedy agregat zostanie wyczerpany, grupy alokacji mają ograniczoną maksymalną ilość grup, 128. Dodatkowo minimalnyrozmiar AG jest ustalony na 8192 bloki agregatów. Rozmiar grupy alokacji przechowywany jest w superbloku agregatu.

8.2.4 Zestawy plików

Zestaw plików jest zestawem plików i katalogów tworzącym niezależnie montowalne poddrzewo. Zestaw plików jest przechowywany całkowicie wewnątrz pojedynczego agregatu.

Pliki i katalogi reprezentowane są trwale przez i-węzły. Każdy i-węzeł opisuje atrybuty pliku lub katalogu i służy jako punkt startowy do wyszukiwania danych pliku lub katalogu na dysku.

8.2.5 Ekstenty

Ekstent jest zestawem ciągłych bloków agregatów. Ekstent całkowicie zawiera się wewnątrz pojedynczego agregatu, jednak duże ekstenty mogą rozciągać się na wiele grup alokacji.

Ekstent może mieć rozmiar od 1 do 2(24)-1 bloków agregatu.

W przypadku 512-bajtowego rozmiaru bloku (najmniejszy możliwy), maksymalny ekstent ma niewiele poniżej 8G. Możliwe limity dotyczą wyłącznie pojedynczego ekstentu; nie mają wpływu na całkowity rozmiar pliku. Adres jest adresem pierwszego bloku ekstentu. Adres wyraża się także w jednostkach bloków agregatu: jest to offset bloku od początku agregatu.

JFS używa bazujących na ekstentach struktur adresowych, łącznie z taktyką alokacji bloków, w celu uzyskania efektywnych, skalowalnych struktur do odwzorowania logicznych offsetów wewnątrz pliku na fizyczne adresy na dysku. Ekstent jest ciągiem sąsiednich bloków zaalokowanych dla pliku jako jednostka i opisanym przez trójkę o następującym

składzie: <logiczny offset, długość, fizyczny>. Struktura adresowa jest B+ drzewem zawierającym deskryptory ekstentów (powyższe trójki), zakorzenionym w i-węźle, którego kluczem jest logiczny offset wewnątrz pliku.

8.2.6 I-węzły

I-węzły dyskowe JFS mają po 512 bajtów. I-węzeł zawiera cztery podstawowe grupy informacji. Pierwsza opisuje atrybuty POSIX obiektu JFS. Druga opisuje dodatkowe atrybuty obiektów JFS. Obejmuje ona informacje konieczne do obsługi VFS, informacje specyficzne dla środowiska systemu operacyjnego oraz nagłówek dla B+ drzewa. Trzecia grupa zawiera albo deskryptory alokacji ekstentów w korzeniu B+ drzewa, albo wewnętrzne (?) dane. Czwarta grupa zawiera dodatkowe atrybuty, dane lub dodatkowe deskryptory alokacji ekstentów. Definicja struktury i-węzła opisana jest w pliku jfs_dinode.h, struct dinode.

JFS alokuje i-węzły dynamicznie. Dzieje się to poprzez alokację ekstentów i-węzłów, które są po prostu sąsiednimi kawałkami i-węzłów na dysku. Z definicji, ekstent i-węzłów w JFS zawiera 32 i-węzły.

Przy alokacji nowego i-węzła nie jest inicjalizowany nowy ekstent. Jednakże po zaznaczeniu i-węzła w ekstencie jako używanego, inicjalizowane są jego numer zestawu plików, numer i-węzła, znacznik i-węzła, oraz adres bloku grupy alokacji i-węzła.

Dynamiczna alokacja i-węzła sprawia, że nie ma bezpośredniego związku pomiędzy numerem i-węzła a jego adresem na dysku. Mapa Alokacji I-węzłów pozwala na znalezienie i-węzła na dysku.

W JFS istnieje jeden licznik generacyjny (?) i-węzłów (?) zwiększany przy każdej alokacji i-węzła, zamiast tradycyjnego jednego licznika na i-węzeł, który zwiększany jest, gdy i-węzeł jest ponownie wykorzystywany. Rozwiązanie takie wynika z faktu, że przy dynamicznej alokacji i-węzłów, kiedy zwalnia się i-węzeł, jego przestrzeń dyskowa może być ponownie wykorzystana dla celów innych niż i-węzeł (tzn. przestrzeń ta może być wykorzystana doprzechowywaniazwykłych danych pliku).

8.2.7 B+ drzewa

B+ drzewa zostały użyte w JFS w celu poprawy wydajności czytania i zapisywania ekstentów, które są najbardziej popularnymi operacjami JFS. B+ drzewa zapewniają szybkie wyszukiwanie do czytania określonego ekstentu pliku. Zapewniają także efektywny sposób dołączania lub wstawiania ekstentu do pliku.

Deskryptor alokacji ekstentów (struktura *xad*) opisuje ekstent oraz ma dwa dodatkowe pola konieczne do reprezentowania plików: offset, opisujący logiczny adres bajtowy, który reprezentuje ekstent, oraz pole flag. Postać struktury deskryptora alokacji ekstentu przedtawia się następująco:

```
struct xad {
unsigned flag:8;
unsigned rsvrd:16;
```

```
unsigned off1:8;
uint32 off2;
unsigned len:24;
unsigned addr1:8;
uint32 addr2;
} xad_t;
```

gdzie:

flag jest 8-bitowym polem zawierającym różne flagi. Flagi te mogą wskazywać copy-on-write, jeśli ekstent jest zaalokowany, ale nie zapisany, informacje do kompresji, itp (?)

rsvrd jest 16-bitowym polem zarezerwowanym na przyszłość. Zawsze ma wartość zero.

off1, off2 jest 40-bitowym polem, zawierającym logiczny offset pierwszego bloku ekstentu. Logiczny offset reprezentowany jest w jednostkach rozmiaru bloków agregatu (?); innymi słowy, aby uzyskać go w bajtach, należy przemnożyć offset przez rozmiar bloku agregatu

len jest 24-bitowym polem, zawierającym długość ekstentu, reprezentowaną rozmiarem bloku agregatu

addr1, addr2 jest 40-bitowym polem zawierającym adres ekstentu, reprezentowanym jako rozmiar bloków agregatu (?)

Struktura xad opisuje dwie abstrakcyjne przestrzenie:

- fizyczną przestrzeń bloków dyskowych na dysku. Zaczyna się ona na numerze bloku agregatu *xad_address* i rozciąga na *xad_length* bloków agregatu.
- logiczną przestrzeń bajtów wewnątrz pliku. Zaczyna się ona na numerze bajtu *xad_offset * AGBS* (rozmiar bloku agregatu - *aggregate block size*) i rozciąga na *xad_length * AGBS* bajtów.

W JFS istnieje tylko jedna ogólna struktura indeksowa w postaci B+ drzewa dla wszystkich obiektów (poza katalogami). Indeksowane dane zależą od obiektu. B+ drzewo indeksowane jest przez offset zawarty w *xad* opisywanych danych. Pozycje są posortowane według offsetów struktur *xad*. w węzłach B+ drzewa przechowywane są struktury *xad*.

Rysunek 8.1 przedstawia pojedynczą strukturę *xad* oraz sposób, w jaki opisuje ona zarówno przestrzeń bajtów wewnątrz pliku jak i fizyczną lokalizację bajtów na dysku.

Dolna część drugiej sekcji i-węzła dyskowego zawiera deskryptor danych mówiący, jakiego rodzaju dane przechowywane są w drugiej części i-węzła. Druga część może zawierać wewnętrzne dane dla wystarczająco małych plików. Jeśli dane nie zmieszczą się w przestrzeni wewnętrznych danych i-węzła, przechowywane będą w ekstentach, a i-węzeł zawierać będzie korzeń węzła B+ drzewa. Nagłówek wskazuje ile *xad-ów* jest używanych

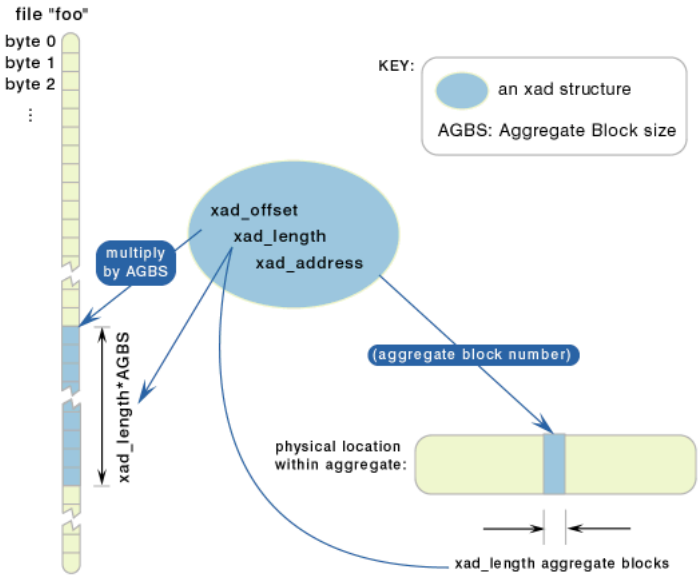


FIGURE 3. xad describes two "ranges"

Rysunek 8.1: B+ drzewo

i ile dostępnych. Ogólnie i-węzeł zawiera 8 takich struktur dla korzenia B+ drzewa. Jeśli istnieje 8 lub mniej ekstentów w pliku, wtedy te 8 struktur *xad* jest także liśćmi B+ drzewa. Opisują one ekstenty (zobacz rysunek 8.2, przykład 1). W przeciwnym przypadku 8 struktur *xad* i-węzła wskazuje albo na liście, albo na węzły wewnętrzne B+ drzewa.

Po wypełnieniu wszystkich 8 struktur *xad* w i-węźle następuje próba wykorzystania ostatniej ćwiartki i-węzła na więcej struktur *xad*. Jeśli ustawiony jest bit *INLINEEA* na polu *di_ mode* i-węzła, wtedy można wykorzystać jego ostatnią ćwiartkę.

Gdy wszystkie dostępne struktury *xad* w i-węzłach są w użyciu, B+ drzewo musi się rozdzielić. JFS alokuje 4K przestrzeni dyskowej na liść B+ drzewa. Liść jest logicznie tablicą pozycji *xad* z nagłówkiem. Nagłówek wskazuje na pierwszy wolny *xad* w węźle, wszystkie następujące po nim *xad-y* także nie są zajęte. 8 struktur jest kopiowanych *xad* z i-węzła do liścia, nagłówek wskazuje 9 pozycję jako pierwszą wolną. JFS aktualizuje korzeń b+ drzewa na pierwszą strukturę *xad* i-węzła - struktura ta wskazuje na nowo zaalokowany liść. Offsetem nowej struktury *xad* staje się offset pierwszej pozycji w liściu. Nagłówek i-węzła jest aktualizowany tak, by wskazywał, że tylko jedna struktura *xad* jest używana w B+ drzewie. Nagłówek aktualizuje się dodatkowo tak, by wskazywał, że i-węzeł zawiera teraz wyłącznie korzeń B+ drzewa (zobacz rysunek 8.2, przykład2).

Wraz z dodawaniem nowych ekstentów do pliku, dodawane są one do tego samego liścia we właściwym porządku, aż do wypełnieni liścia. Następnie alokuje się nowe 4K przestrzeni dyskowej na nowy liść B+ drzewa. Druga struktura *xad* z i-węzła wskazywać będzie na nowo-zaalokowany węzeł (zobacz rysunek 8.2, przykład 3).

Postępowanie takie jest kontynuowane do czasu, gdy wypełni się wszystkie 8 struktur *xad* i-węzła, kiedy to następuje podział B+ drzewa. Podział tworzy wewnętrzne węzły B+ drzewa, które wykorzystywane są do wyznaczania ścieżek poszukiwań w B+ drzewie. JFS alokuje 4K przestrzeni dyskowej na węzeł B+ drzewa. Węzeł wewnętrzny wygląda dokładnie tak samo jak liść. * struktur *xad* kopiowanych jest z i-węzła do węzła wewn., nagłówek wskazuje 9 pozycję jako pierwszą wolną. JFS aktualizuje korzeń B+ drzewapowodując, że pierwsza struktura *xad* i-węzła wskazuje na nowo zaalokowany węzeł wewn. Nagłówek i-węzła wskazuje, że wykorzystywana jest tylko jedna struktura *xad* dla B+ drzewa (zobacz rysunek 8.2, przykład 4).

W pliku *jfs_xtree.h* opisany jest nagłówek dla korzenia B+ drzewa w strukturze *struct xtpage_t*. W pliku *jfs_btree.h* opisany jest nagłówek dla węzła wewnętrznego lub liścia w strukturze *struct btpage_t*.

8.2.8 Mapa alokacji bloków

Mapa alokacji bloków wykorzystywana jest do śledzenia zaalokowanych lub zwolnionych bloków dyskowych dla całego agregatu. Ponieważ wszystkie zestawy plików wewnątrz agregatu dzielą tę samą pulę bloków dyskowych, podczas alokowania lub zwalniania bloków mapa alokacji jest wykorzystywana przez wszystkie zestawy plików wewnątrz agregatu.

Mapa alokacji bloków jest plikiem opisanym przez i-węzeł 2 agregatu. Kiedy tworzony jest agregat, alokowane są bloki danych dla mapy pokrywającej przestrzeń agregatu. Mapa może rosnąć lub zmniejszać się dynamicznie w miarę wzrostu lub zmniejszania się agregatu.

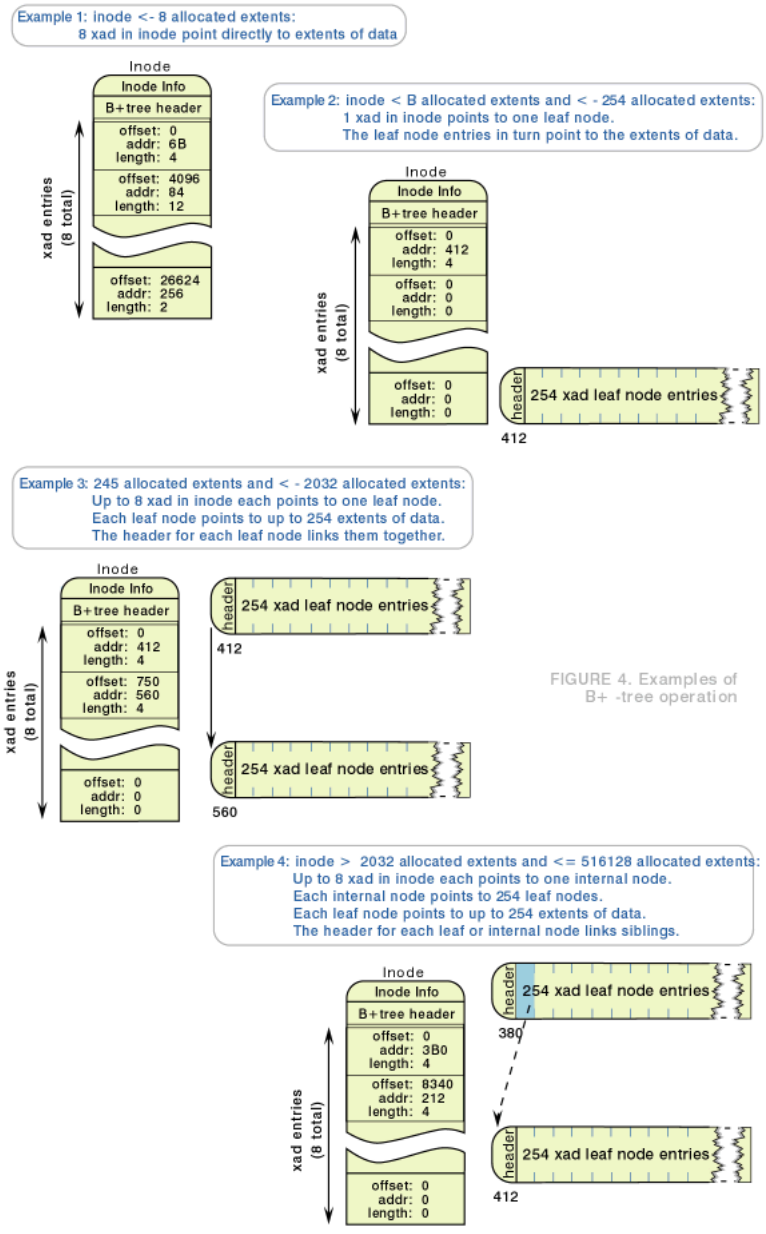
Mapa alokacji bloków zawiera informacje o ym, czy każdy pojedynczy blok agregatu jest zaalokowany lub zwolniony.

Rysunek 8.3 pokazuje logiczną i fizyczną strukturę mapy. Każda strona mapy ma 4K długości. Mapa zawiera trzy rodzaje stron: strony kontrolne *bmap*, strony kontrolne *dmap* i strony *dmap*.

Każdy *dmap* zawiera pojedynczy bit reprezentujący każdy blok agregatu. I-ty bit reprezentuje status alokacji i-tego logicznego bloku agregatu. Każdy *dmap* pokrywa 8K bloków agregatu.

Ponieważ mapa alokacji bloków może zawierać wiele stron dmap, zorganizowane są one przez strony kontrolne dmap. Strony te poprawiają wydajność znajdowania dużych ekstentów lub wolnych bloków. Rozmiar agregatu determinuje, ile takich stron i poziomów jets potrzebnych. Maksymalnie są to 3 poziomy, co pozwala na maksymalny rozmiar 2(42) bloków agregatu. Jeśli nie wszystkie poziomy są potrzebne, mapa alokacji bloków jest rzadkim plikiem z przerwami na pierwszą stronę każdego nieużywanego poziomu.

JFS wykorzystuje strategię zapewniającą właściwe aktualizowanie danych kontrolnych. Oznacza to zachowanie spójnego stanu struktur i zasobów alokacji w przypadku załamania systemu. W celu zapewnienia spójności stanu mapy alokacji bloków JFS przechowuje dwie mapy w strukturze *dmap* - mapę roboczą i mapę trwałą. Mapa robocza zapisuje aktualny stan alokacji. Mapa trwała zapisuje zatwierdzony stan alokacji, składający się ze stanów alokacji na dysku lub opisanych w logu JFS, lub zatwierdzonych transakcjami JFS. Kiedy zwalniany jest blok agregatu, najpierw aktualizowana jest mapa trwała. Kiedy alokowany



Rysunek 8.2: B+ drzewo

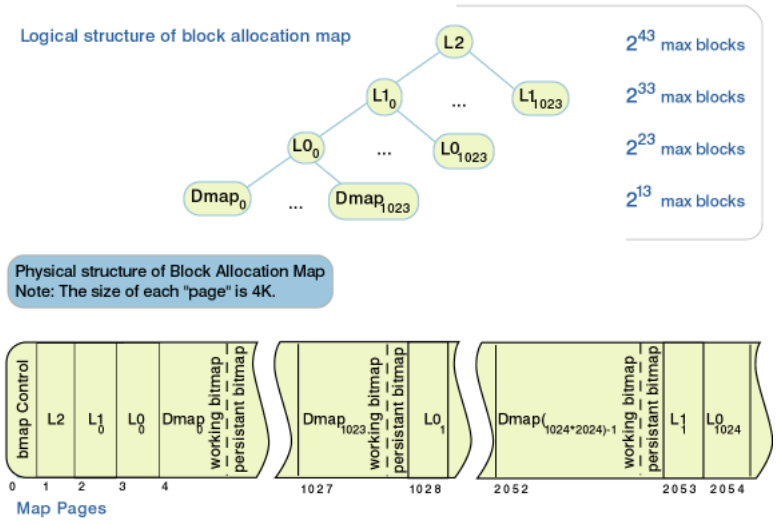


FIGURE 5. Block Allocation Map

Rysunek 8.3: Struktura mapy

jest blok agregatu, najpierw aktualizuje się mapę roboczą. Bit 0 reprezentuje zwolniony zasób, a 1 zaalokowany.

Strony kontrolne dmap mapy alokacji bloków zawierają drzewo podobne do drzewa struktury dmap, poza poziomem liści zawierającym 1024 elementy.

Rysunek 8.4 pokazuje dokładnie pola drzewa jednej struktury dmap. Zauważmy, że pole w strukturze dmap jest płaską tablicą, ale reprezentuje drzewo jak pokazano. Drzewo śledzi maksymalną liczbę sąsiednich bloków na każdym poziomie poza najniższym. Najniższy poziom, tree[85] do tree[341] zawiera reprezentację *binary buddy* mapy roboczej. Pozostałe poziomy drzewa zawierają maksymalną liczbę sąsiednich wolnych bloków z czterech przedziałów następnego niższego poziomu.

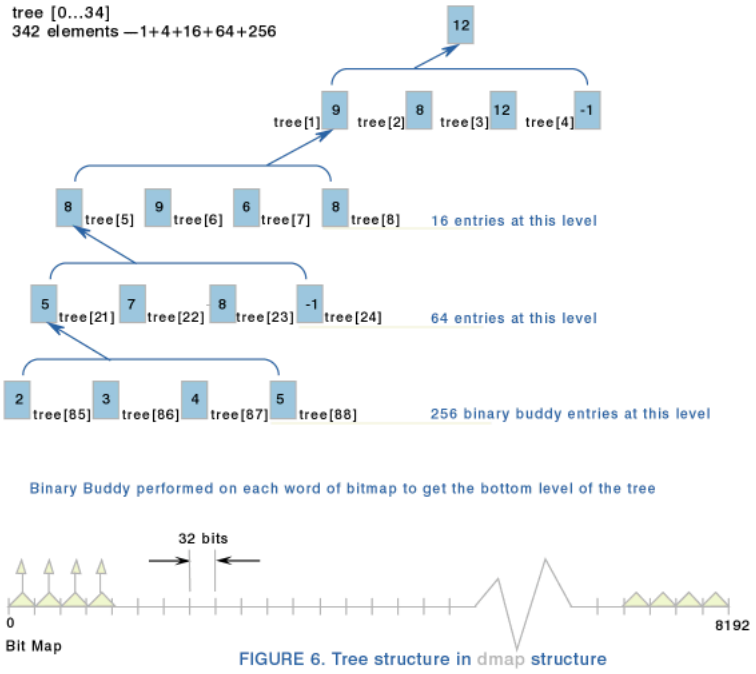
System *binary buddy* wykorzystywany jest do uzupełnienia poziomu liści każdego z podsumowujących drzew. Drzewo w dmap powstaje poprzez uzyskanie najdłuższego ciągu binary buddy wolnych bitów dla każdego słowa w bitmapie. Ciągi są kodowane jako potęgi 2, gdzie -1 reprezentuje wszystko zaalokowane.

Rysunek 8.5 pokazuje przykład znajdowania najdłuższego ciągu binary buddy wolnych bitów dla słowa.

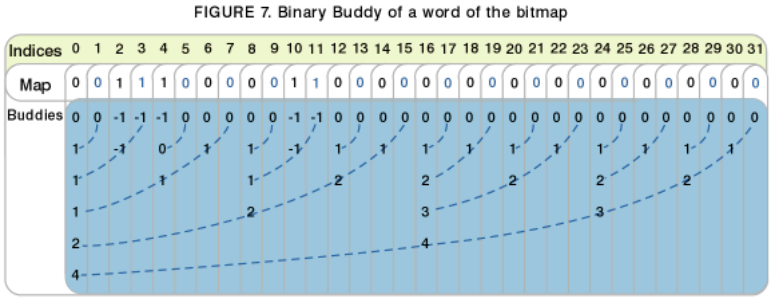
Następnie system binary buddy jest wykorzystywany do wypełnienia liści drzewa. Drzewo jest formowane poprzez wzięcie najdłuższej liczby wolnych bloków zaczynając na określonym indeksie, włączając jedynie buddy reprezentowane jako potęga 2.

Rysunek 8.6 przedstawia skrócony przykład liścia drzewa dmap.

Na szczycie mapy alokacji bloków istnieje struktura kontrolna mapy, *struct dbmap_t*. Zawiera ona informacje podsumowujące, przyspieszające znajdowanie grup alokacji mających ponadprzeciętną ilość wolnego miejsca. Opis tej struktury można znaleźć w pliku



Rysunek 8.4: Struktura drzewiasta w strukturze dmap



Rysunek 8.5: Binary Buddy

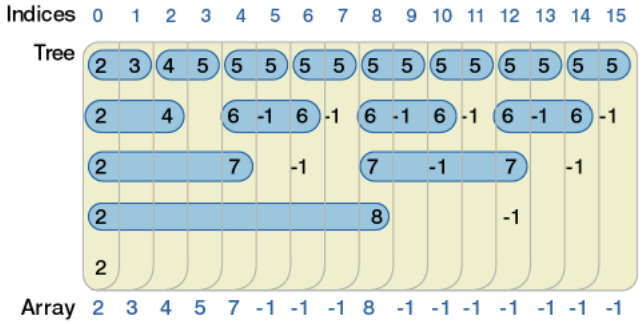


FIGURE 8. Binary Buddy

Rysunek 8.6: Liść drzewa dmap

jfs_dmap.h.

Mapa alokacji bloków nie jest księgowana: może być naprawiona w czasie odzyskiwania systemu przez *logredo* lub zrekonstruowana przez *fsck*. Zarówno mapa robocza jak i trwała muzą być w tym samym stanie po użyciu *fsck* lub *logredo*.

8.3 Alokacja i-węzłów

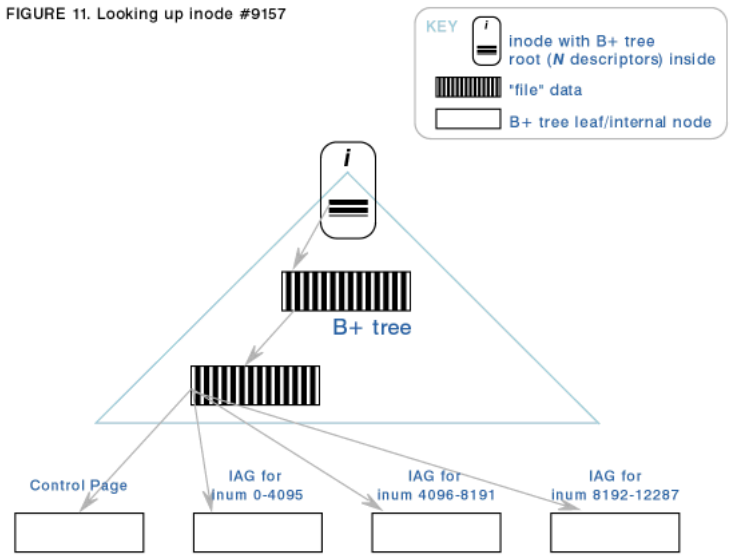
Przy dynamicznej alokacji i-węzłów, numer i-węzła nie odwzorowuje się bezpośrednio na określony logiczny blok dyskowy agregatu, w związku z czym potrzeba nowych struktur danych do obsługi następujących typów operacji:

przeszukiwanie w przód (ang. *forward lookup*) - mając numer i-węzła znajdź i-węzeł na dysku. Typowym przykładem jest przeszukiwanie pliku.

przeszukiwanie w tył (ang. *reverse lookup*) - mając numer grupy alokacji znajdź najbliższy wolny i-węzeł. Sytuacja ta pojawia się przy alokacji nowego i-węzła, gdy JFS stara sie zaalokować i-węzeł fizycznie najbliższy wybranej grupie alokacji (tak by na przykład pliki z jednego podkatalogu miały i-węzły położone blisko siebie).

wyszykiwanie numeru wolnego i-węzła - aby zaalokować ekstent nowego i-węzła, znajdź następne 32 i-węzły, które nie mają odpowiadającego im zaalokowanego ekstentu i-węzła. Sytuacja pojawia się, gdy wszystkie aktualnie zaalokowane i-węzły są wykorzystywane, gdy JFS potrzebuje i-węzłów dla grupy alokacji i nie alokował i-węzłów wcześniej, lub gdy nie ma wolnych i-węzłów dla grup alokacji.

Następujące problemy wyszukiwaniarozwiązane są za pomocą mapy alokacji i-węzłów. Agregat i każdy zestaw plików posiadają mapę alokacji i-węzłów, która jest dynamiczną tablicą grup alokacji i-węzłów (ang. *Inode Allocation Groups - IAG*). IAG może istnieć gdziekolwiek w agregacie. IAG stanowi dane dla mapy alokacji i-węzłów.



Rysunek 8.7: Znajdowanie i-węzła

Każdy IAG ma rozmiar 4K i opisuje 128 fizycznych ekstentów i-węzłów na dysku. Wszystkie ekstenty i-węzłów dla IAG istnieją w jednej grupie alokacji - IAG jest związana z grupą alokacji do czasu, gdy zwolnione zostaną wszystkie ekstenty jego i-węzłów.

Pierwsze 4K strony mapy alokacji i-węzłów jest stroną kontrolną. Zawiera ona podsumowujące informacje dla mapy alokacji i-węzłów.

Abstrakcyjnie, mapa alokacji i-węzłów jest dynamicznie rozszerzalną tablicą struktur IAG: *struct iag inode_ allocation_ map [1..N];* Fizycznie jest plikiem wewnątrz agregatu.

W celu zapewnienia spójności przechowywane są dwie mapy - robocza i trwała dla każdego IAG. Ich wykorzystanie jest analogiczne do map alokacji bloków. Wewnątrz każdej sekcji kontrolnej IAG istnieje mapa podsumowująca służącapoprawie wydajności znajdowania wolnych i-węzłów. Mapa podsumowująca odwzorowuje się na mapę roboczą IAG.

IAG zawiera także deskryptory ekstentów i-węzłów opisujące właściwe ekstenty i-węzłów. Jest ich 128 na każdy IAG.

Na rysunku 11 8.7 pokazano przykład znajdowania i-węzła nr 9157.

8.3.1 Lista wolnych i-węzłów AG

Lista wolnych i-węzłów AG rozwiązuje problem wyszukiwania wstecz. JFS tworzy maksymalną liczbę dozwolonych AG na agregat. Dzięki temu istnieje określona liczba nagłówków list wolnych i-węzłów AG. Nagłówek dla listy znajduje się na stronie kontrolnej mapy alokacji i-węzłów. Numer IAG używany jest jako indeks listy - (-1) wskazuje na jej koniec. Każda sekcja kontrolna IAG zawiera wskaźniki w przód i w tył na listę.

Rysunek 8.8 pokazuje listę wolnych i-węzłów AG.

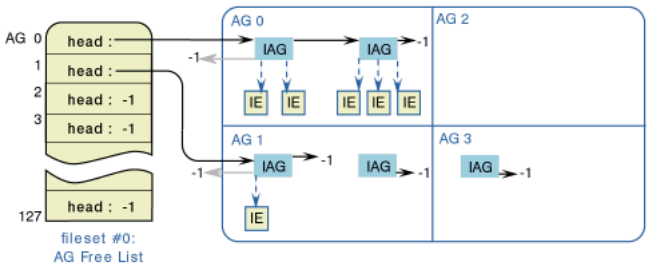


FIGURE 12. AG Free list

Rysunek 8.8: Lista wolnych i-węzłów AG

Lista nie jest księgowana - może być naprawiona przez *logredo* lub *fsck*.

8.3.2 Lista ekstentów wolnych i-węzłów

Lista ta pomaga rozwiązać problem wyszukiwania wstecz i wyszukiwania numeru wolnego i-węzła. Pozwala JFS na znalezienie następnego ekstentu w IAG dla określonej AG, która jeszcze nie była zapisana na dysk. Każdy zestaw plików ma własną listę ekstentów wolnych i-węzłów AG dla każdej AG. Działanie tej listy jest analogiczne do poprzedniej.

Rysunek 8.8 przedstawia tę listę.

8.3.3 Lista wolnych IAG

Lista ta jest rozwiązaniem problemu wyszukiwania numeru wolnego i-węzła. Pozwala JFS na znalezienie IAG bez odpowiadających ekstentów zaalokowanych i-węzłów. Agregat ma swoją własną dołączoną listę, tak jak każdy zestaw plików.

8.3.4 IAG Free Next

Licznik IAG Free Next pozwala JFS na znalezienie numeru wolnego i-węzła dla następnego IAG, który powinien być zaalokowany. Każdy agregat i każdy zestaw plików mają swoje własne liczniki. Znajdują się one w stronie kontrolnej mapy alokacji i-węzłów. Raz zaalokoawny, IAG nigdy nie jest usuwany.

8.4 I-węzły alokacji zestawów plików

I-węzły mapy alokacji zestawu plików w tablicy i-węzłów agregatu są specjalnym typem i-węzłów. Ponieważreprezentują zestawy plików, są śuper-i-węzłem"dla zestawu plików.

Zawierają specyficzne dla zestawu plików informacje w górnej części i-węzła zamiast normalnych danych i-węzła. Śledzą także lokalizację mapy alokacji i-węzłów zestawu plików w swoim B+ drzewie. Struktura jest zdefiniowana w pliku *jfs_ dinode.h*, przez strukturę *struct dinode*.

8.5 Plik

Plik reprezentowany jest przez i-węzeł zawierający korzeń B+ drzewa opisującego ekstenty zawierające dane użytkownika. B+ drzewo jest indeksowane przez offset ekstentów.

8.5.1 Pliki rzadkie i gęste

JFS obsługuje zarówno pliki rzadkie, jak i gęste.

Rzadkie pliki pozwalają na zapisywanie danych w losowych miejscach w pliku. Zgłaszany rozmiar pliku jest największym zapisanym bajtem, ale nie następuje faktyczna alokacja bloku, na którym nie przeprowadzono operacji zapisu.

W przypadku plików gęstych, zasoby dyskowe alokowane są na pokrycie rozmiaru pliku.

8.6 Dowiązania symboliczne

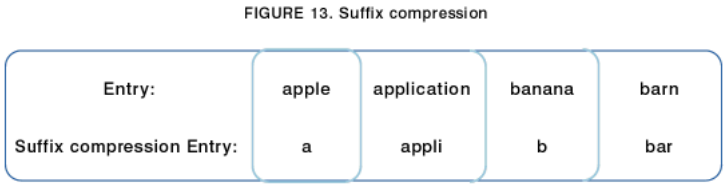
Dowiązanie symboliczne reprezentowane jest przez i-węzeł z polem *di_ node* wskazującym dowiązanie symboliczne (*S_ IFLNK*). Pełna nazwa ścieżki dowiązanego lpiku przechowywana jest wewnątrz i-węzła, jeśli jest na to miejsce. W przeciwnym przypadku przechowywana jest jako dane dla tego i-węzła w ekstencie indeksowanym przez B+ drzewo dla i-węzła.

8.7 Katalogi

Katalog stanowi księgowany plik meta-danych w JFS. Katalog składa się z pozycji katalogu wskazujących obiekty zawarte w tym katalogu. Pozycja katalogu łączy nazwę z i-węzłem. Określony i-węzeł opisuje obiekt o określonej nazwie. W celu poprawy wydajności lokalizowania określonej pozycji katalogu wykorzystywane jest B+ drzewo sortowane po nazwie.

W JFS występują dwie organizacje katalogów. Pierwsza, dla małych katalogów, przechowująca zawartość katalogu wewnątrz i-węzła katalogu eliminuje konieczność oddzielnych bloków we/wy katalogu jak również potrzebę alokowania oddzielnego magazynu. Wewnątrz i-węzła może być przechowywanych do 8 pozycji, wyłączjąc pozycje (.) i (..), przechowywane w innych częściach i-węzła. Druga organizacja wykorzystywana jest dla większych katalogów i reprezentuje każdy katalog jako B+ drzewo indeksowane nazwą.

Pole *di_ size* i-węzła katalogu reprezentuje strony liści B+ drzewa katalogu. Gdy liść katalogu przechowywany jest w i-węźle, pole *di_ node* ma wartość 256.



Rysunek 8.9: Kompresja sufixowa

Katalog nie zawiera specjalnych pól dla siebie - self(".") ani rodzica - parent(".."). Reprezentowane są one w samym i-węźle. *Self* jest własnym numerem i-węzła katalogu. *Parent* jest specjalnym polem i-węzła, *idotdot*, opisanym przez strukturę *struct dtroot_t*, w pliku *jfs_dtree.h*.

I-węzeł katalogu zawiera korzeń swojego B+ drzewa w sposób podobny do normalnego pliku. Jednak to B+ drzewo jako klucze ma nazwę. Liście B+ drzewa katalogu zawierają pozycje katalogu i są indeksowane pełną nazwą pozycji. B+ drzewo katalogów używa kompresji sufiksowej dla ostatnich węzłów wewnętrznych B+ drzewa. Pozostałe węzły wewnętrzne uzywają tego samego skompresowanego sufiksu. Kompresja sufiksowa obcina nazwę do tylu znaków, ile wystarcza do rozróżnienia obecnej pozycji od poprzedniej. Rysunek 8.9 pokazuje przykład kompresji sufiksowej.

Rysunek 8.10 przedstawia elementy węzła B+ drzewa katalogu:

8.8 Listy kontroli dostępu

Różne listy kontroli dostępu (ang. *Access Control Lists - ACL*) związane są z każdym i-węzłem JFS. ACL reprezentują różne aspekty, takie jak prawa, identyfikatory użytkowników lub grup. Pola ACL są ignorowane dla i-węzłów agregatów.

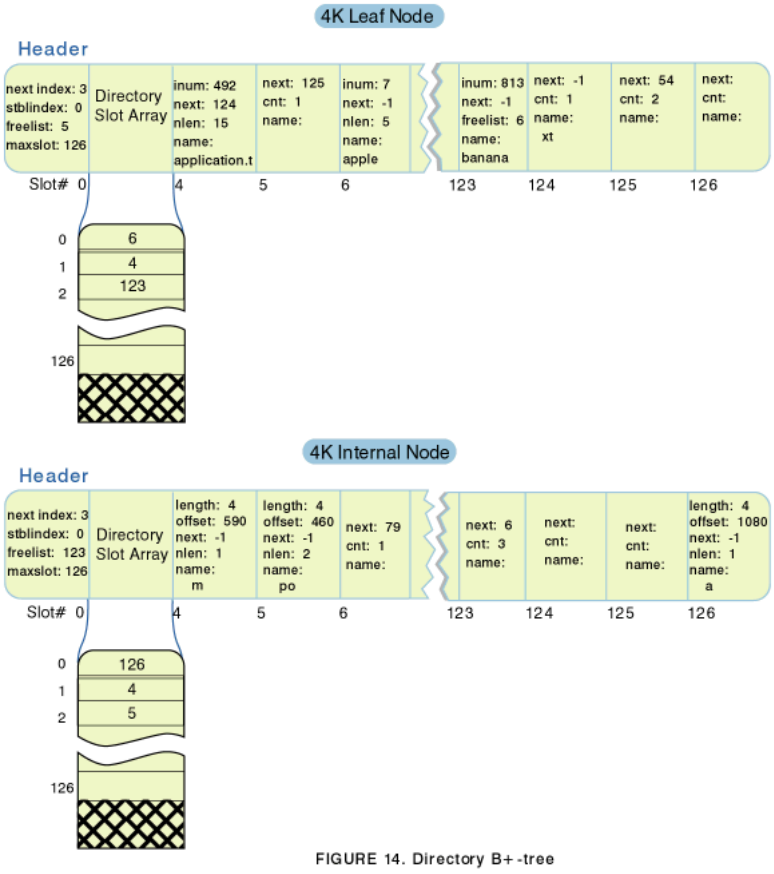
Chociaż nie ma wymagań co do reprezentacji ACL na dysku i w pamięci, przyjęta jest reprezentacja źewnętrzna". Jedynym limitem na rozmiar ACL jest konieczność zmieszczenia się zewnętrznej reprezentacji w 8192-bajtowej strukturze *dfs_acl*.

Każdy obiekt JFS może zostać związany z ACL, która zarządza dowolnym dostępem do tego obiektu; mówi się o niej jako o regularnej ACL. Dodatkowo obiekty JFS mogą mieć związane jeszcze dwie opcjonalne listy ACL wykorzystywane w momencie tworzenia obiektu - początkowa lista ACL katalogu i pliku. Jeśli istnieje, początkowa ACL dołączana jest do każdego pliku w danym katalogu.

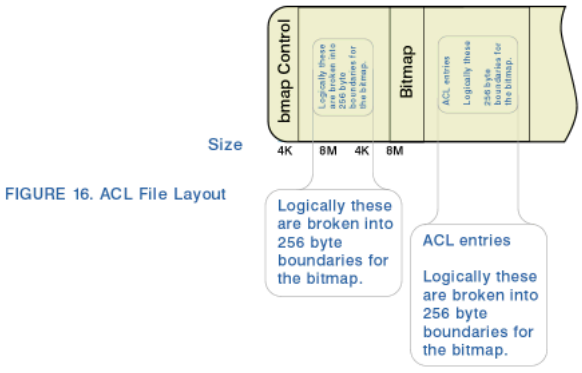
Plik ACL musi mieć bitmapę do lokalizowania wolnych regionów do przechowywania ACL-i. Plik ACL ma 4K bitmapę, po której następuje 8M pozycji ACL, powtórzonych, jeśli trzeba. Jeden bit w bitmapie reprezentuje 256 bajtów sąsiedniej przestrzeni dyskowej; bitmapa nie opisuje siebie.

Rysunek 8.11 pokazuje plik ACL.

Dane ACL nie są księgowane.



Rysunek 8.10: Elementy węzłą B+ drzewa katalogu



Rysunek 8.11: Plik ACL

8.9 Rozszerzony atrybut

Rozszerzony atrybut (ang. *Extended Attribute - EA*) jest ogólnym magazynem i mechanizmem dostępu do danych załączonych do obiektu JFS. EA są przechowywane sąsiadująco w przestrzeni EA (*EAS*), zdefiniowanej przez deskryptor EA w i-węźle dla obiektu JFS. Deskryptor EA jest deskryptorem ekstentu zdefiniowanym w pliku *jfs_types.h*, strukturą *struct dxd_t*.

EA może być przechowywany w i-węźle lub oddzielnym ekstencie. Pole flagowe deskryptora EA wskazuje sposób przechowywania. Ponieważ przestrzeń ta może być wykorzystana także dla dodatkowych pozycji xad dla xtree pliku, pole di_mode i-węzła wskazuje, czy przestrzeń ta jest dostępna. Jeśli ustawiony jest bit INLINEEA, przestrzeń jest dostępna.

Jeśli EA jest przechowywany w i-węźle, pola offsetu i długości deskryptora EA są ignorowane.. Rozmiar deskryptora EA wskazuje liczbę bajtów danych.

Jeśli EA przechowywany jest w ekstencie, deskryptor EA opisuje ekstent. JFS nie zakłada, żeby dane EA były bardzo duże, więc nie obsługuje więcej niż jeden ekstent EA na i-węzeł.

Pozycja EA zawiera zarówno nazwę EA ja i jego wartość. Aby uzyskać dostęp do pojedynczego EA, JFS przeszukuje dane EA liniowo.

Dane EA nie są księgowane, są jednak zapisywane synchronicznie (oznacza to, że zawsze będą to stare lub nowe dane, nigdy częściowo zapisane). JFS księguje. gdzie dane EA są ulokowane. Wewnętrzne(?) dane EA są księgowane.

8.10 Strumienie

Strumień wykorzystywany jest do dołączania danych do pliku lub katalogu. Te dodatkowe dane podobne są do danych katalogu, ponieważ można się do nich odwoływać przez nazwę.

Rysunek 8.12 przedstawia strumienie.

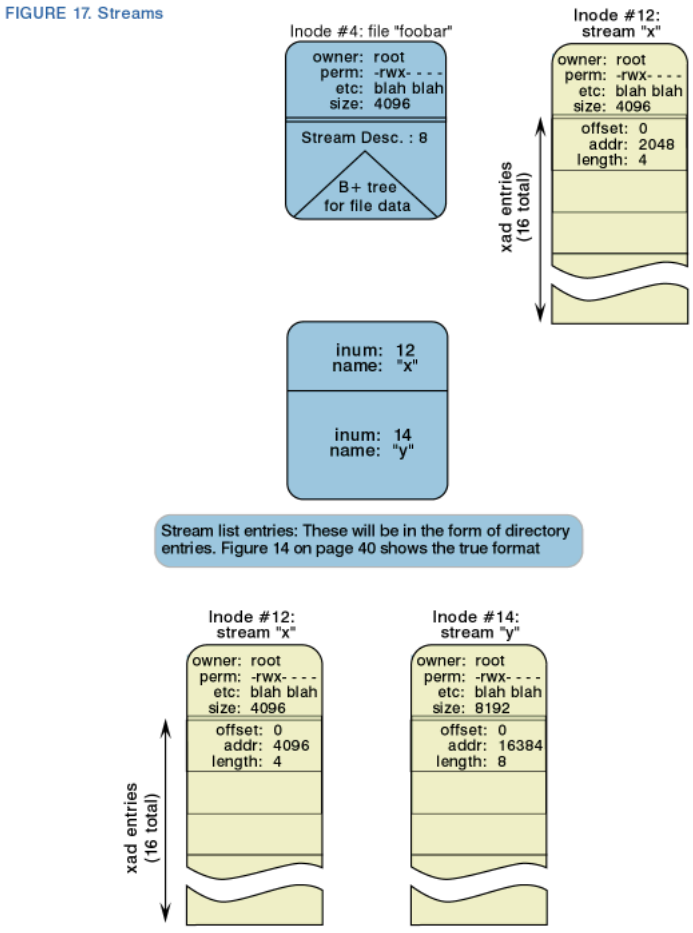
Strumienie nie są księgowane.

8.11 Agregaty z zestawami plików

Rysunek 8.13 przedstawia agregat zawierający zestaw plików.

8.12 Potencjalne wewnętrzne limity JFS

JFS jest pełnym 64-bitowym systemem plików. Wszystkie właściwe pola struktur systemu plików mają rozmiar 64 bitów. Pozwala to JFS na obsługę zarówno dużych plików jak i partycji.



Rysunek 8.12: Strumienie

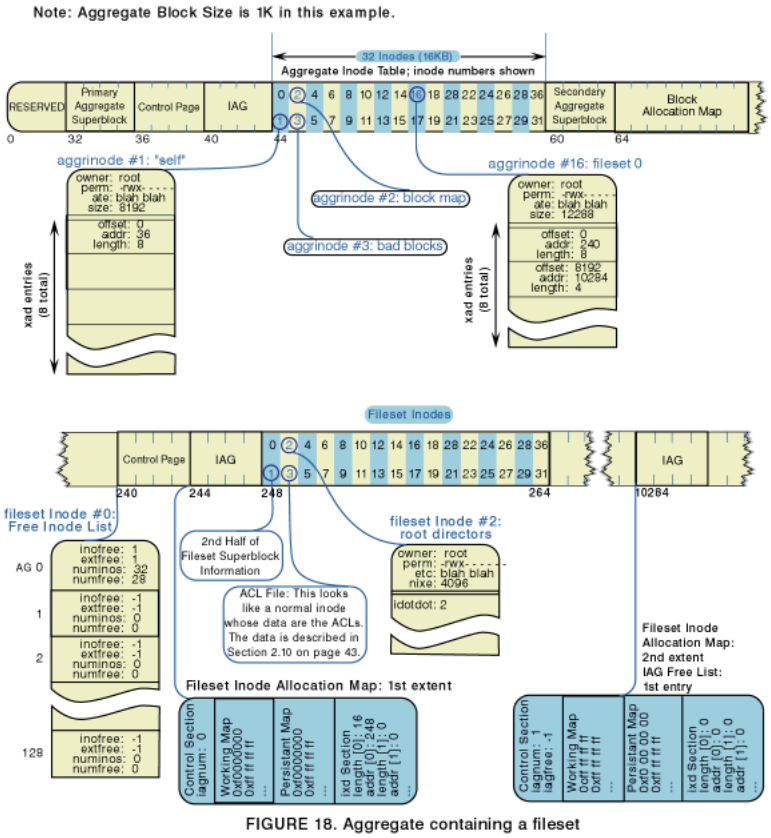


FIGURE 18. Aggregate containing a fileset

Rysunek 8.13: Agregat z zestawe plików

8.12.1 Rozmiar systemu plików

Minimalny rozmiar systemu plików obsługiwany przez JFS wynosi 16MB. Maksymalny rozmiar jest funkcją rozmiaru bloku systemu plików i maksymalnej liczby bloków obsługiwanej przez struktury meta-danych systemu plików. JFS może obsłużyć maksymalnie system plików wielkości 512 TB (z rozmiarem bloku 512B) do 4 PB (z rozmiarem bloku 4KB).

8.12.2 Rozmiar pliku

Maksymalny rozmiar pliku jest największym rozmiarem pliku obsługiwanym przez strukturę wirtualnego systemu plików.

8.12.3 Urządzenia przenośne

JFS nie obsługuje dyskietek (?) jako stanowiących podstawę systemu plików urządzeń.

8.13 Zastosowania systemu JFS

Celem twórców JFS było stworzenie stabilnego, wydajnego systemu plików.

JFS jest kluczową technologią dla Inernetowych serwerów plików, ponieważ dostarcza szybkich czasów restartu systemu plików w przypadku załamania systemu. Wykorzystując techniki księgowania, JFA przywraca system do spójnego stanu w ciągu kilku sekund lub minut.

Rozdział 9

Reiserfs, wersja 3.6

W przeciwieństwie do większości systemów plików w Linuksie, Reiserfs został stworzony zupełnie od zera, a nie przeniesiony z innej platformy jak np. JFS (zobacz 8) i XFS (zobacz 7) czy stworzony na bazie starszego systemu plików (jak np. Ext3 – zobacz 6). Autorem systemu jest Hans Reiser z firmy Namesys.

Brak oparcia w żadnym istniejącym systemie plików niesie za sobą pewne koszty – Reiserfs przez długi czas (nawet w wersji z jądra 2.4.16) miał duże problemy ze stabilnością działania. Od wiosny' 2002 nie znaleziono w systemie Reiserfs (seria 3.6.x) większych błędów.

9.1 Podstawowe cechy

Podstawowym założeniem przy tworzeniu pierwszej wersji było stworzenie działającego i możliwie prostego systemu, a następnie rozszerzanie go o kolejne cechy. W chwili obecnej Reiserfs posiada następujące możliwości:

- Bardzo dobra wydajność dla małych plików (8 do 15 razy lepsza niż *Ext2* – zobacz 5)
- Dobra wydajność, również przy operacjach na katalogach z dużą liczbą plików dzięki zastosowaniu drzew zrównoważonych
- Alternatywne podejście do obsługi małych plików – w innych systemach było to z reguły realizowane przez nakładkę na system działającą na zasadzie bazy danych, tutaj mechanizm obsługi małych plików jest wbudowany w system; dzięki temu oszczędzamy około 6% miejsca
- Małe pliki (do 3984 bajtów, przy blokach wielkości 4kB) są trzymane bezpośrednio w drzewie
- Dane nie są przetrzymywane w blokach stałej wielkości, aby nie marnować miejsca na dysku i zwiększyć wydajność

- Szybki mechanizm księgowania (księgowanie *meta-danych*)

9.2 Struktura „logiczna”

Ta sekcja omawia sposób organizacji systemu plików, bez zagłębiania się w techniczne szczegóły.

9.2.1 Drzewa zrównoważone

System Reiserfs działa w oparciu o drzewa zrównoważone (jedno drzewo dla systemu plików, „adres” w superbloku). Wszystkie liście są na jednym poziomie. W chwili obecnej wysokość drzewa może wynieść maksymalnie 5.

B+drzewa

B+drzewa użyte w systemie ReiserFS różnią się od tradycyjnych b-drzew brakiem danych w węzłach wewnętrznych. Dane są umieszczane wyłącznie w liściach lub specjalnych węzłach umieszczonych poniżej poziomu liści (zobacz niżej), służących do obsługi dużych plików (zobacz 9.2.6).

Takie podejście umożliwia użycie miejsca w węzłach na dodatkowe klucze i wskaźniki, a w rezultacie na zwiększenie stopnia rozgałęzienia drzewa. Z oczywistych powodów zmniejsza to wysokość drzewa.

9.2.2 Balansowanie drzewa

Operacji balansowania drzewa jest optymalizowana poprzez przestrzeganie następujących zasad:

1. minimalizacji liczby używanych węzłów
2. minimalizacji liczby węzłów zmienianych przez operację balansowania
3. minimalizacji liczby węzłów zmienianych przez operację balansowania, które nie są umieszczone w pamięci podręcznej
4. jeśli przenosimy dane pomiędzy węzłami, to należy przenieść jak najwięcej (tylko dla liści)

Typy węzłów

B+drzewo w ReiserFS 3.6 ma trzy typy węzłów:

węzeł wewnętrzny zawiera wskaźniki do poddrzew rozdzielone kluczami (zobacz 9.3.2); klucz poprzedzający wskaźnik jest jednocześnie równy pierwszemu kluczowi z poddrzewa, do którego odnosi się ten wskaźnik

liść znajduje się na pierwszym poziomie drzewa; zawiera nagłówki pozycji (IHead, zobacz 9.3.3) oraz pozycje (*item*) następujących typów:

pozycje pośrednie zawiera wskaźniki do *węzłów niesformatowanych* (zobacz niżej)

pozycje bezpośrednie zawierają *ogony* plików (zobacz 9.2.5), umieszczane *bezpośrednio* w pozycji

pozycje katalogowe zawierają *wpisy katalogów*, czyli nagłówki *deHead* (zobacz 9.3.5) oraz nazwę pliku

pozycje Stat pełnią role struktur *inode* z innych systemów plików; nie zawsze są używane jako oddzielne pozycje, można umieszczać je we wpisach katalogów; zobacz 9.3.4

Każda pozycja zawiera jednoznaczny *klucz*, używany do szukania i sortowania.

węzeł niesformatowany zawierają bloki z samymi danymi (części dużych plików), poza *ogonami* plików (zobacz 9.2.5)

Jak widać, *liść* jest pojęciem logicznym, ponieważ poniżej jego też występują węzły. Jednak *liść* jest najniżej położonym w drzewie węzłem o ustalonej strukturze.

Zawartość węzłów wewnętrznych i liści jest posortowana po wartości klucza (zobacz 9.3.2) obiektu (zobacz niżej).

Drzewo zrównoważone służy do szybkiego odnajdywania obiektów (zobacz niżej), gdy znamy odpowiednią wartość klucza. Schemat drzewa został przedstawiony na rysunku 9.1.

9.2.3 Obiekt

Obiektem może być plik lub katalog. Każdy obiekt składa się z pewnej liczby pozycji (item). Obiekt posiada *identyfikator obiektu* oraz *klucz* (zobacz 9.3.2).

Górne ograniczenie na liczbę obiektów w systemie plików to $2^{32} - 4 = 4294967292$.

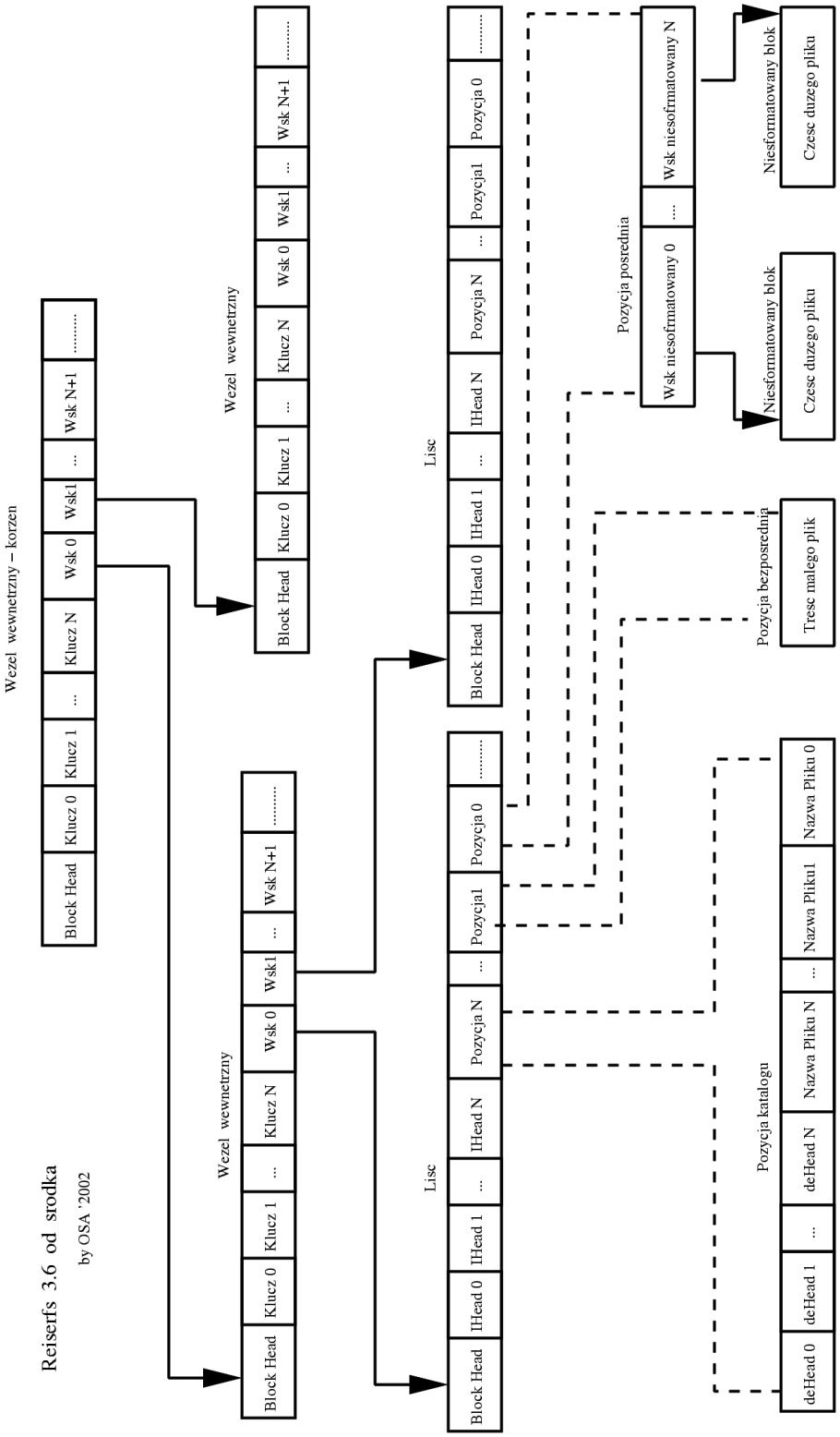
9.2.4 Pliki

Plik składa się ze zbioru *pozycji pośrednich*, wskazujących na bloki z danymi, oraz do dwóch *pozycji pośrednich* (jeśli *ogon pliku* jest rozbity na dwa bloki). Pierwsza pozycja jest typu *Stat* (zobacz 9.3.4).

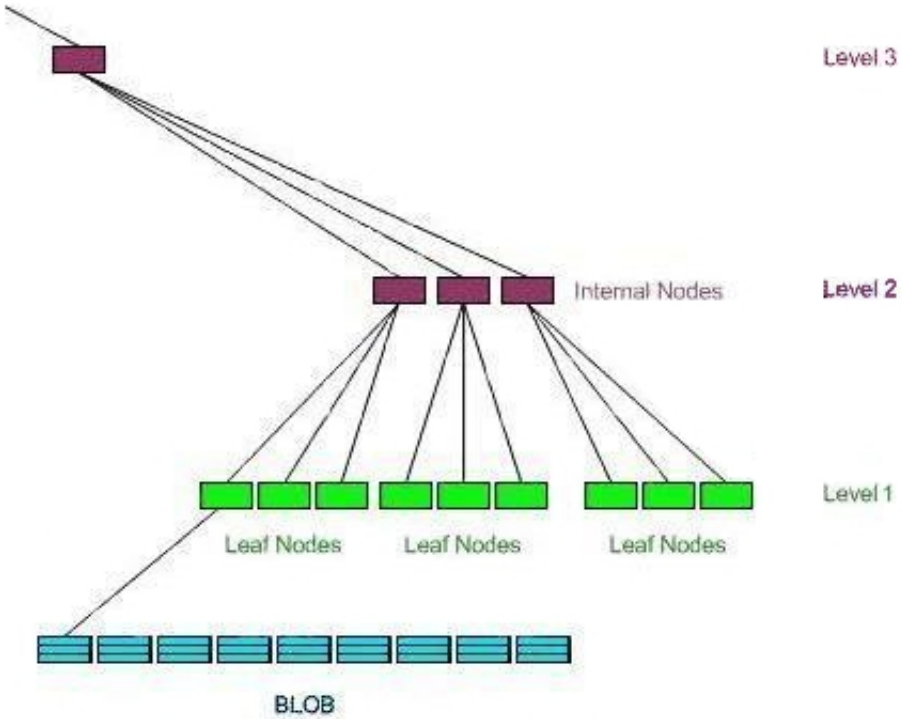
Nazwa pliku jest ograniczona przez długość_bloku - 64, czyli 4032 znaki dla bloków o wielkości 4kB.

9.2.5 Ogon pliku

Ogon pliku to ostatni fragment pliku wielkości długość pliku modulo wielkość bloku. Dla małych plików (mniejszych od rozmiaru bloku), ogon oznacza cały plik. Małe pliki możemy



Rysunek 9.1: Drzewo zrównoważone w systemie Reiserfs.



Rysunek 9.2: Dostęp do dużych plików w ReiserFS 3.6.

przetrzymywać bezpośrednio w *węźle bezpośrednim*, co zaoszczędza jedno przejście przez wskaźnik.

Ogony plików są obsługiwane przy pomocy mechanizmu *upychania ogonów*. Powoduje to, że system stara się upychać do jednego bloku kilka ogonów, dzięki czemu zyskuje się około 6% miejsca więcej niż w Ext2 (o Ext2 czytaj w 5). Jest to działanie domyślne.

Z drugiej strony, obsługa ogonów niesie ze sobą pewne problemy – jeśli dopisujemy dane na końcu pliku, to system musi przenosić ogon do innego bloku lub rozbijać ogon na dwa bloki. Możemy wyłączyć mechanizm upychania ogonów, co zwiększy wydajność systemu kosztem większego zużywania miejsca.

9.2.6 Duże pliki

Dostęp do dużych plików wymaga przejścia w dół całego b+drzewa, a następnie podążenia za linkiem z liścia. Sytuację tę ilustruje rysunek 9.2.

Takie podejście powoduje, że obsługa dużych plików jest mało efektywna. Zostanie to polepszone w wersji 4 systemu ReiserFS (zobacz 10 oraz rysunek 10.1).

9.2.7 Katalogi

Katalog składa się ze zbioru *pozycji katalogowych*, które z kolei są złożone z *wpisów katalogu* (zobacz niżej). Pierwsza pozycja jest typu *Stat* (zobacz 9.3.4).

9.3 Struktura fizyczna

Ta sekcja przedstawia techniczne szczegóły dotyczące pojęć omówionych w poprzedniej sekcji.

Będziemy rozważali bloki dyskowe. Każdy blok (poza blokiem reprezentującym *węzeł niesformatowany* – zobacz 9.2.1) posiada *nagłówek bloku*.

9.3.1 Nagłówek bloku

Nagłówek bloku jest reprezentowany przez strukturę *block_head*, o następujących polach:

blk_level poziom w drzewie (1 dla liści)

blk_nr_item liczba kluczy w węźle wewnętrznym lub liczba pozycji w liściu

blk_free_space wolne miejsce w bloku (w bajtach)

blk_right_delim_key klucz ograniczający blok z prawej strony (dla liści)

9.3.2 Klucz

Klucz służy do wyszukiwania obiektów w drzewie. Klucz posiada następujące pola:

k_dir_id identyfikator katalogu nadrzędnego

k_object_id identyfikator obiektu (równy numerowi węzła); unikalny

k_offset przesunięcie względem początku obiektu

k_uniqeuness rodzaj pozycji: 0 dla Pozycji Stat, -1 dla pozycji bezpośredniej, -2 dla pozycji pośredniej, 500 dla katalogu

W chwili obecnej, możliwe jest używanie identyfikatorów obiektu jako kluczy (są unikalne), jednak to może się zmienić w następnych wersjach. Dlatego do identyfikowania obiektów należy używać całych kluczy, a nie tylko identyfikatorów obiektu.

9.3.3 IHead – nagłówek pozycji

Nagłówek dla pozycji. Zawiera następujące pola:

ih_key klucz (zobacz 9.3.2)

unia ih_free_space wolne miejsce w ostatnim niesformatowanym węźle pozycji bezpośredniej, **0xFFFF** dla pozycji pośredniej lub pozycji *Stat*

ih_entry_count liczba wpisów dla pozycji katalogu

ih_item_len długość opisywanej pozycji

ih_item_location przesunięcie do opisywanej pozycji

ih_reserved zarezerwowane dla systemu

9.3.4 Pozycja Stat

Pozycję *Stat* można uznać za odpowiednik struktury *inode* z innych systemów plików. Różnicą jest to, że nie zawiera ona wskaźnika do danych. Jej pola to:

sd_mode typ pliku oraz uprawnienia

sd_nlink liczba twardych dowiązań

sd_gid, sd_uid właściciel i jego grupa

sd_size rozmiar

sd_rdev urządzenie, na którym znajduje się plik

sd_atime, sd_ctime, sd_mtime odpowiednio, czas ostatniego: dostępu, modyfikacji węzła oraz modyfikacji pliku

sd_first_direct_byte zawiera:

-1 dla katalogów oraz dużych plików, które nie zawierają pozycji bezpośrednich (zobacz 9.2.1)

1 dla małych plików, które są umieszczone w pozycjach bezpośrednich (zobacz 9.2.1)

>1 dla dużych plików, które mają zarówno pozycje pośrednie jak i bezpośrednie

przesunięcie od początku pliku dla bezpośrednich pozycji pliku

9.3.5 Nagłówek deHead – struktura reiserfs_de_head

Służy do indentyfikowania zawartości katalogu, posiada następujące pola:

deh_offset przesunięcie do wpisu

deh_dir_id identyfikator obiektu katalogu nadrzędnego

deh_objectid identyfikator obiektu, do którego odnosi się wpis

deh_location lokalizacja nazwy wewnątrz pozycji

deh_state na przyszłość, aby umieszczać pozycje Stat we wpisie (optymalizowanie wydajności)

9.4 Mechanizm księgowania

W chwili obecnej Reiserfs wspiera tylko księgowanie *meta-danych*. Trzy ciekawsze struktury, używane do obsługiwania księgowania są opisane poniżej.

Ciekawostka – w Reiserfs operujemy na pojęciach *zatwierdzeń* (commit), a nie *tranzakcji* (jak jest z reguły).

Stała *JOURNAL_TRANS_HALF* wynosi 1018 (dla bloków 4kB).

9.4.1 reiserfs_journal_desc

Pierwszy blok zapisywany podczas operacji zatwierdzania (commit). Jego pola to:

j_trans_id identyfikator zatwierdzenia

j_len długość zatwierdzenia (j_len + 1 to adres bloku zatwierdzenia)

j_mount_id ?

j_realblockJOURNAL_TRANS_HALF | adresy poszczególnych bloków

j_magic12 | ?

9.4.2 reiserfs_journal_commit

Ostatni blok zatwierdzenia. Pola:

j_trans_id musi być równe polu z *reiserfs_journal_desc* dla tego zatwierdzenia

len długość zatwierdzenia

j_realblockJOURNAL_TRANS_HALF | adresy poszczególnych bloków

j_digest16 | suma kontrolna; nieużywane

9.4.3 reiserfs_journal_header

Blok zapisywany, gdy tranzakcja jest uznana za całkowicie zapisaną (flushed) na dysk oraz jest młodsza od ostatnio zapisanej. Oznacza to, że zarówno dane księgowane jak i dane rzeczywiste zostały zapisane, a tranzakcja nie jest potrzebna. Pola:

j_last_flush_trans_id poprzednia zapisana tranzakcja

j_first_unflushed_offset przesunięcie, od którego należy zacząć odtwarzanie w przypadku awarii systemu

j_mount_id ?

9.5 Zastosowanie

Reiserfs jest systemem szybkim (szybszym od Ext2 i Ext3 w większości zastosowań). W szczególności, bardzo dobrze sprawdza się przy operacjach na małych plikach. Mechanizm księgowania pozwala stosować go dla serwerów (wstawanie systemu po awarii nie będzie trwało zbyt długo).

Rozdział 10

Reiser 4 – plany na przyszłość

Rozwój nowej wersji systemu ReiserFS jest wspierany przez DARPA (Defence Advanced Research Projects Agency). Celem jest stworzenie szybkiego oraz bezpiecznego (odpornego na ataki) systemu plików dla Linuksa.

System ma się ukazać na początku 2003. roku, dlatego opis dotyczy różnic pomiędzy obietnicami autorów a wersją 3.6.

10.1 Klucze

Do identyfikacji obiektów służą klucze (całe klucze, a nie same identyfikatory obiektów – zobacz 9.3.2). Jednak klucze nie muszą być unikalne (rodzaj haszowania). Dzięki mniejszej puli kluczy, mogą one być krótsze. Zmniejsza to zużycie pamięci przy pewnym narzucie czasowym, związanym z wyszukiwaniem liniowym na liście obiektów o tej samej wartości klucza.

10.2 Warstwy

System ReiserFS 4 jest zaprojektowany w czytelny i przejrzysty sposób. Służy do tego rozdzielenie warstwy *semantycznej* (wyższy poziom abstrakcji) oraz *przechowującej* (niższy poziom).

10.3 Wtyczki

Do obsługi obiektów systemu służą wtyczki. Wtyczka definiuje zestaw metod, służących do obsługi obiektu. Przewidywanych jest 8 standardowych wtyczek, kolejne mogą być dodawane przez programistów, ale wymaga to rekompilacji całego jądra.

wtyczka plików pomiędzy zewnętrznym interfejsem dostępu do plików a systemem ReiserFS

wtyczka katalogów w najbliższym czasie jedynie „standardowa" obsługa katalogów

wtyczka hash używana w szczególności do szybkiej współpracy z NFS

wtyczka bezpieczeństwa obsługa uprawnień oraz przezroczystego kodowania oraz kompresowania plików

wtyczka pozycji wyodrębniona, aby kod służący do operacji na drzewach (m.in. balansowania) był zakapsułkowany; dzięki temu będzie można dodawać kolejne pozycje (w szczególności dotyczące mechanizmów bezpieczeństwa) bez konieczności zmieniania całego kodu systemu

wtyczka przypisywania klucza klucze nie muszą być unikalne (zobacz 10.1), dlatego duże znaczenie dla wydajności systemu ma algorytm przydzielania kluczy jak też możliwość zamiany go na inny

wtyczki wyszukiwania węzłów i pozycji

10.4 Struktura

Najważniejszą zmianą w strukturze systemu jest dodanie nowego typu węzła (zobacz niżej), co umożliwiło efektywniejszą obsługę dużych plików.

10.4.1 twig node

Gałązka (*twig node*) to węzeł umieszczany na drugim poziomie (bezpośrednio nad liściami). Gałązka posiada dwa rodzaje wskaźników – do liści z pozycjami (tak jak w poprzedniej wersji – zobacz 9.2.1) oraz ze wskaźnikami do BLOBów (dużych plików). Różnicę ilustrują rysunki 9.2 (w wersji 3. systemu) oraz 10.1 (nowa wersja).

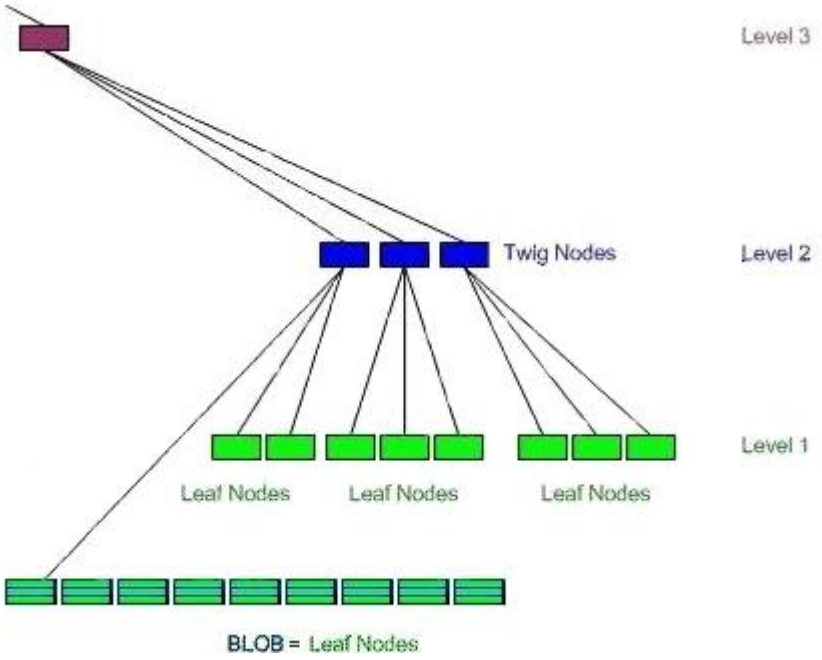
Przeniesienie BLOBów z poziomu 0, gdy BLOBy były wskazywane z liści, na poziom 1 , czyli na poziom liści, istotnie skróciło ścieżkę do BLOBa w B+drzewie a w rezultacie poprawiło wydajność obsługi dużych plików.

Dla przypomnienia dodam, że standardowe ograniczenie na wysokość drzewa to 5.

10.4.2 Pozycje

Dodano dodatkowe pole do nagłówka pozycji, określające wtyczkę, której należy użyć (*Item_Plugin_id*).

Zmiana kolejności w liściu – najpierw pozycje (item), potem nagłówki (zobacz rysunek 10.2).



Rysunek 10.1: Dostęp do dużych plików w ReiserFS 4.

Block Head	Item 0	...	Item N	Puste	Item Head N	...	Item Head 0

Rysunek 10.2: Nowa struktura liścia w ReiserFS 4

10.5 Księgowanie – *Wandering logs*

Wprowadzono mechanizm *wandering logs*, przyspieszający działanie księgowania.

W sytemie ReiserFS 4 księgowane dane będą zapisywane tylko raz, w przeciwieństwie do dwóch zapisów (do logu i w docelowe miejsce) w innych systemach.

W ReiserFS 4 *log* nie posiada przypisanych na stałe logów (poza blokami służącymi do przechowywania informacji o samym logu), więc bloki z logami mogą *wędrować* (być przypisywanymi do innych obiektów) w systemie plików.

W zamierzeniu autorów ten system powinien znacznie zredukować narzut związany z księgowaniem.

10.6 Zastosowanie

Przyspieszenie obsługi dużych plików, szybkie księgowanie oraz ulepszone mechanizmy bezpieczeństwa sprawiają, że ReiserFS 4 stanie się prawdopodobnie bardzo popularnym systemem, zarówno dla serwerów jak i stacji roboczych.

Rozdział 11

Materiały

FAT, FAT32

- http://www.pcguide.com/ref/hdd/file/fat.htm
- http://www.project9.com/fat32/

NTFS

- http://www.ntfs.com
- http://www.pcguide.com/red/hdd/file/ntfs.htm
- http://linux-ntfs.sourceforge.net/ntfs/index.html

ISO 9660

- "Introduction to ISO 9660", Disc Manufacturing Inc.
- http://www.angelfire.com/pa2/mpx/iso9660.html
- http://www.alumni.caltech.edu/∼pje/iso9660.html

UDF

- http://trylinux.com/projects/udf

JFS

- http://www-106.ibm.com/developerworks

XFS

- http://oss.sgi.com/projects/xfs/papers/xfs_usenix/index.html

Ext2

- materiały do wykładu SO, prezentacje z poprzednich lat

- ”Design and implementation of the Second Extended Filesystem” (http://web.mit.edu/tytso/www/linux/extintro.html)
- Ext2fs homepage (http://e2fsprogs.sourceforge.net/ext2.html)

Ext3

- zródła jądra 2.4.18 i 2.5.*
- http://www-106.ibm.com/developerworks

ReiserFS 3.6 i 4

- http://www.reiserfs.org
- zródła jądra

inne

- www.linuxgazette.com/issue55/florido.html

Indeks