Implementacja systemu plików

1 Struktura systemu plików

Dyski stanowią większość pamięci na której znajduje się system plików, ponieważ:

- mogą być odczytywane/modyfikowane
- mają bezpośredni dostęp do każdego zawieranego bloku

W celu zwiększenia wydajności, transfer między dyskiem a pamięcią odbywa się w blokach [block]. Każdy blok ma jeden lub więcej sektorów [sectors] (sektor: 32 do 4096 B, zwykle 512B).
System plików [file system] zapewnia wydajny i wygodny dostęp do dysku, pozwalając na łatwe przechowywanie, namierzanie i odczytywanie danych. W tym celu najczęściej składa się on z kilku warstw. W tym modelu, warstwy wyższe korzystają z niższych.

- Sterowanie wejściem-wyjściem [I/O control] składa się z modułu obsługi urządzenia [device driver] i obsługi przerywań. Zmienia wysokopoziomowe polecenia ("wczytaj blok 123") na niskopoziomowe polecenia wykonywane przez sterownik urządzenia [hardware controller]. Zazwyczaj zapisuje bity do wyznaczonych miejsc w pamięci sterownika.
- Podstawowy system plików [basic file system] zajmuje się wydawaniem ogólnych poleceń do odpowiedniego modułu sterowania, blokami, buforami, cache'ami różnych systemów plików. Na przykład jeśli bufor jest pełny, to w jego gestii należy znalezienie więcej pamięci bądź zwolnienie nieużywanej.
- Moduł organizacji pliku [file-organization module] wie o plikach i ich logicznych blokach, jak również fizycznych blokach. Znając sposób alokacji, potrafi przetłumaczyć adres bloku logicznego na adres bloku fizycznego. Kolejno numerowane bloki logiczne niekoniecznie odpowiadają fizycznym numerom bloków. Ponadto zarządza on wolnym miejscem.
- Poziom logiczny [logical file system] odpowiada za [metadata information], czyli wszelkie informacje o strukturze systemu plików poza faktycznymi danymi. Zarządza on katalogami oraz plikami - tymi ostatnimi przez blok kontrolny pliku [file-control block, FCB] (np. i-węzeł [inode] w UNIX - tu pojawiają się nieścisłości, związek nie jest dobrze wyjaśniony w książce) (zawiera on informacje o właścicielu, uprawnieniu i lokalizacji zawartości).

Należy pamiętać, że ta sama warstwa może być używana przez różne systemy plików.
Typowe systemy plików: Linux - extended file system (ext3, ext4), Windows - FAT, FAT32, NTFS.

2 Implementacja systemy plików

Na dysku, systemy plików mogą zawierać informacje o tym jak uruchomić system operacyjny, liczbę bloków, strukturę katalogów, itd.
Krótki opis:

- Blok sterujący rozruchem [boot control block] zawiera informacje potrzebne do uruchomienia systemu z tego urządzenia. W NTFS: [partition boot sector], w UFS (Unix file system): [boot block].
- Blok sterujący partycji [volume control block] zawiera informacje o partycji, takie jak liczba bloków, wskaźniki na FCB itd. W UFS: [superblock], w NTFS: [master file table].
- Struktura katalogów, używana do organizacji plików.
- FCB dla każdego pliku

Ponadto w pamięci głównej również przechowuje się informacje potrzebne do działania systemu plików (ładowane podczas montowania, aktualizowane podczas wykonywania operacji, usuwane przy odmontowaniu):

- Tablica montaży [mount table] zawiera informacje o zamontowanych partycjach
- Cache struktury katalogów
- Ogólnosystemowa tablica otwartych plików [system-wide open-file table] zawiera kopię FCB dla każdego otwartego pliku i wiele innych informacji
- Procesowa tablica otwartych plików [per-process open-file table] zawiera wskaźniki do odpowiednich pól w tablicy wyżej
- Bufory bloków

Oto dość wymowny schemat opisujący otwieranie i odczytywanie pliku.

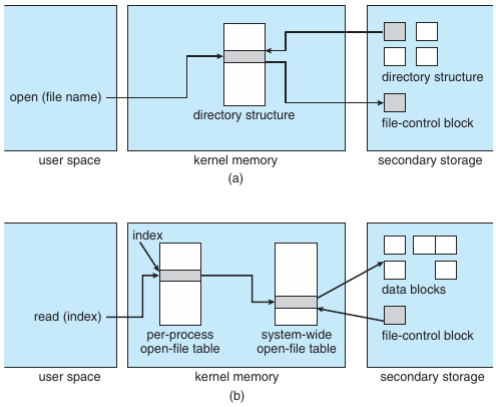

Figure 12.3 In-memory file-system structures. (a) File open. (b) File read.

Operacje te odbywają się przez wskaźnik zwany w UNIXie deskryptorem pliku [file descriptor], w Windowsie uchwytem plikowym [file handle].
Strefy [partition] dzielimy na surowe [raw] i przyrządzone [ćooked"], w zależności od tego, czy (nie)znajduje się na nich jakiś system plików. Czasem surowe strefy są pożądane, np. są używane przez bazy danych albo w RAIDach.
Informacje o rozruchu mogą być przechowywane jak to opisano wyżej. Uruchamianie zaczyna się od ustalonej lokacji, np. pierwszego bajtu. Tak uruchomiony [boot loader] zna już strukturę systemu plików i wie jak wczytać jądro (rozruch dwustopniowy). [boot loader] może znać więcej niż jeden system operacyjny czy system plików, np. GRUB.
Strefa główna [root partition], zawierająca jądro, jest montowana podczas rozruchu. Pozostałe partycje mogą być automatycznie zamontowane podczas rozruchu, lub ręcznie później, zależnie od systemu operacyjnego.
Wirtualny system plików (virtual file system, VFS) to warstwa abstrakcji która

- oddziela ogólne operacje systemów plików od ich implementacji. Tych implementacji może być kilka, dzięki czemu różne systemy plików mają ten sam interfejs
- dostarcza mechanizm reprezentowania pliku przez sieć w unikalny sposób. Bazuje na strukturze zwanej [vnode], która zawiera numer unikalnego pliku w sieci (i-węzły są unikalne tylko lokalnie).
  Np. VFS w Linuksie składa się z:
  - i-węzłów reprezentujących pojedyncze pliki
  - [file object], reprezentującego otwarte pliki
  - [superblock object] reprezentującego cały system plików
  - [dentry object] reprezentującego pojedyncze katalogi

  VFS udostępnia API w postaci wywołań open, close, read, itd.
  Strukturę katalogów możemy przechowywać w sposób liniowy (kosztowne przeszukiwanie, można się wspomóc drzewami) lub przy użyciu hashów (sztywne).

3 Metody przydziału

- Przydział ciągły [contiguous allocation] zakłada, że każdy plik obejmuje zbiór sąsiadujących ciągle bloków. Fajne, bo głowica mało musi pracować i łatwo dostać się do pliku. Niefajne, bo trzeba szacować znajdowanie miejsca dla nowych plików (ile wolnego zostawić?) i powstaje problem [external fragmentation]. Aby trochę ulżyć wprowadza się porcje nadmiarowe [extents]. Tj. po zainicjowaniu bloków pliku dodaje się (jeśli miejsce pozwala) puste bloki ńa przyszłość".

- Przydział listowy [linked allocation] zakłada, że plik jest listą bloków, tj każdy blok pliku zawiera wskaźnik na następny blok. Zalety - rozwiązuje problemy przydziału ciągłego. Wady - załóżmy, że chcemy zdobyć informację z n-tego bloku. Wtedy musimy przelecieć listę bloków od początku. Ponadto marnuje się trochę przestrzeni na wskaźniki. Aby sobie z tym radzić wprowadza się [clusters], czyli umawiamy się, że jednostką będzie [cluster] składający się z np. czterech ciągłych bloków. Dzięki temu mamy mniej wskaźników i skakania, ale większą fragmentację. Dodatkowym problemem jest ryzyko przerwania listy w razie zniszczenia jednego bloku. Tablica rozmieszczenia plików (FAT) jest wariacją tej metody, w której rezerwujemy część dysku (np początek) na tablicę numerowaną numerami bloków, gdzie wartość elementu tablicy oznacza albo blok następny, albo koniec pliku, albo puste miejsce.

- Przydział indeksowy [indexed allocation] rozwiązuje te problemy poprzez [index block]. Każdy plik posiada własny [index block], który jest tablicą adresów bloków. Wtedy katalog posiada wskaźniki na [index block] pliku. Kiedy tworzymy plik, wskaźniki są ustalane na null. To rozwiązanie też ma wady. Po pierwsze wiąże się z większym narzutem wskaźników. Po drugie pojawia się problem, jak duży powinien być [index block]. Radzimy sobie z tym poprzez szereg technik. Po pierwsze ([linked scheme]) możemy łączyć [index block] w listy. Po drugie ([multilevel index]) zakłada, by [index block] był adresowany z innego [index block]. Po trzecie - schemat kombinowany [combined scheme] - (używane w uniksie) jest rozwiązanie pośrednim. I-węzeł zawiera 15 wskaźników. Pierwsze 12 są bezpośrednio do bloków. Kolejny wskazuje na [index block] wskazujący na bloki. Następny na [index block] wskazujący na [index block], a ten na bloki. Ostatni wskaźnik dodaje kolejną warstwę w wyliczance ([single/double/triple indirect block]).

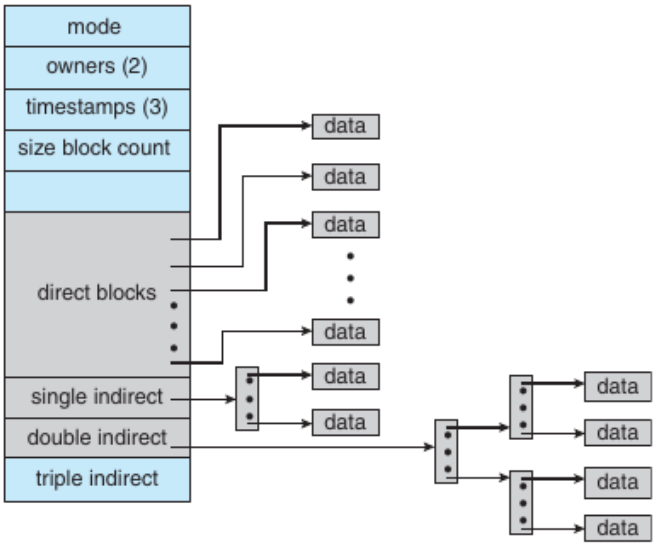

Figure 12.9 The UNIX inode.

4 Zarządzanie wolną przestrzenią

W celu zarządzania wolnym miejscem przechowuje się listy wolnych obszarów [free-space list], które niekoniecznie muszą być implementowane jako listy:

- Mapa bitowa [bit map/bit vector] każdy blok jest reprezentowany przez bit - 1 gdy wolny, 0, gdy zaalokowany. Zaleta: prostota. Wada: rozmiar.
- Lista [linked link] polega na połączeniu wszystkich wolnych bloków w listę z wskaźnikiem na pierwszy z nich.

5 Wydajność

W celu zwiększenia wydajności pojawia się szereg technik. Pamięć podręczna buforów [buffer cache] przechowuje bloki zakładając, że wkrótce zostaną użyte ponownie. Inne podejście prezentuje pamięć podręczna stron [page cache]. Używa ona technik pamięci wirtualnej, by cache'ować dane pliku jako strony zamiast bloków. Kiedy używamy pamięci podręcznej stron do stron procesów i plików, mówimy o ujednoliconej pamięci wirtualnej [unified virtual memory]. Niektóre wersje Linuksa i Uniksa dostarczają ujednoliconej pamięci podręcznej buforów [unified buffer cache]. Rozważmy sytuację, gdy chcemy wczytać plik z dysku do pamięci. Najpierw idzie on przez [buffer cache] (ten od komend read(), write()), a następnie przez [page cache] (ten od [memory mapped IO]), mamy więc podwójne cache'owanie. Ujednolicona pamięć podręczna buforów rozwiązuje ten problem, poprzez połączenie pamięci pomocniczych.
Kolejnym zagadnieniem jest [asynchronous write / synchronous write]. Zazwyczaj stosowane jest to pierwsze polegające na zapisywaniu zmian do cache'u i

zwrotu kontroli sterowania. W drugim przypadku należy czekać, aż zmiany dotrą na urządzenie.
Inne techniki. [Free-behind] usuwa stronę z bufora jak tylko kolejna strona jest żądana. [Read-ahead] wraz z żądaną stroną wczytujemy i cache'ujemy kilka kolejnych.

6 Rekonstrukcja

[Consistency checker] to program (np. fsck w Linuksie), który porównuje dane z struktury katalogów i bloki na dysku, szukając wszelkich różnic.
Niektóre systemy są ze strukturą dziennika (systemy transakcyjne) [log-based transaction-oriented / journaling]. W nich zmiany metadata są zapisywane w logach. Każdy zestaw operacji wykonywany w danym zadaniu to transakcja [transaction]. Wraz z zapisem uważa się je za dokonane, i kontrola wraca do procesu. W międzyczasie, polecenia są faktycznie wykonywane. Transakcje są przechowywane w [circular buffer], co pozwala pozwala odtworzyć zadania po usterce.
Jeszcze innym sposobem jest niezapisywanie zmian bezpośrednio do wyjściowych bloków, ale do nowych bloków i przepięcie wskaźników. Jeśli nie usuniemy starych bloków powstanie migawka [snapshot] - czyli stan plików przed wykonaniem operacji.

7 NFS

NFS network file system] służą do uzyskiwania dostępu do plików w sieciach LAN (a nawet WAN). NFS jest zbiorem połączonych niezależnych maszyn z niezależnymi systemami plików. Idea polega na umożliwieniu pewnego stopnia dzielenia się w ramach relacji klient-serwer (można być oboma).
Protokół montowania [mount protocol] NFS. Aby korzystać z udostępnionych plików, najpierw musi wystąpić montowanie. Potrzebne jest do tego: nazwa zdalnego katalogu i serwera na którym jest on przechowywany. Serwery przechowują listę eksportową [export list], w której wyszczególnione są katalogi do montowania wraz z nazwami maszyn, które mogą je montować. Do tego dochodzą prawa dostępu itd. Ponadto przechowywana jest lista klientów wraz z obecnie zamontowanymi katalogami.
Protokól NFS. Poza standardowymi rozwiązaniami (odczyt, zapis, itd.) rzuca się w oczy brak operacji open() i close(). Jest to celowy zabieg, ponieważ w NFS serwery są bezstanowe, tzn. nie przechowują informacji o klientach pomiędzy kolejnymi dostępami. Dzięki temu łatwiej rekonstruować dane po awarii.
Jeszcze mapka jak wygląda obsługa poleceń - bez niespodzianek:

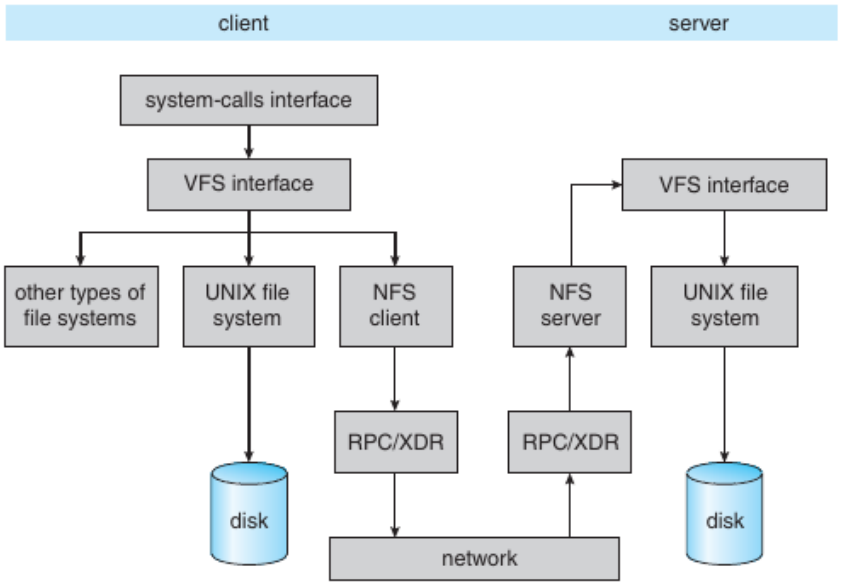

Figure 12.15 Schematic view of the NFS architecture.

Tłumaczenie nazw [path-name translation]. Ścieżka pliku jest rozbijana na składowe (np. /usr/local/dir1/file.txt -¿ usr, local, dir1) i wykonuje się lookup() dla każdej pary składowa/v-węzeł katalogu.

8 System WAFL

Można sobie poczytać jak to działa, ale chyba nie warto pisać.