Kurs administrowania systemem Linux

Zajęcia nr 14: Systemy plików

Instytut Informatyki Uniwersytetu Wrocławskiego

5 czerwca 2025

Wirtualizacja

Wirtualizacja hardware'u

- *Proces* — wirtualizacja procesora.
- *Plik* — wirtualizacja urządzenia pamięci masowej.

Historia

- CTSS — Compatible Time-Sharing System (MIT, 1961–1973)
- CTSS → ITSS (MIT, 1969), CTSS → Multics (MIT, 1969) → Unix (Bell Labs, 1969)

System plików

Rozwój

- Zbiór danych — plik.
- System plików pozwala umieścić wiele zbiorów danych na jednym nośniku.
- Katalogi (kartoteki, *directories*) — zbiory plików.
- Drzewiasta struktura katalogów.
- Ścieżka dostępu (*path*).

Konstrukcja

- Własności nośnika (sekwencyjny, częściowo sekwencyjny, o dostępie swobodnym).
- Potrzeby użytkownika.
- Metadane.

Narzędzia niskopoziomowe

System plików

- *on-disk format*
- implementacja

Narzędzia do przeglądania/edycji danych binarnych

- `hexdump`, `hd`, `od` plus `less`.
- `dhex`, `hexcompare`, `hexcurse`, `hexedit`, `hexer`, `lfhex`, `hexeditor` (ncurses-hexedit).
- `wxhexeditor` (WX), `ghex` (Gnome), `okteta` (KDE).
- Uwaga na rozmiar plików!

Przykład: tar

- Pamięć taśmowa (taśma magnetyczna $\frac{1}{2}$", długość rzędu 1 km).
- Dawniej np.: 9-ścieżkowa, 1600bpi, 64MB. (Obecnie: zapis serpentynowy lub poprzeczny, pojemności do 15TB.)
- Dostęp sekwencyjny. Zapis: bloki zawierające (domyślnie 20) rekordów po 512B (=10KiB). Przerwy międzyblokowe.
- Metadane pliku: *nagłówek* — jeden lub więcej rekordów umieszczonych przed treścią pliku.
- Brak „spisu treści" lub innych metadanych.
- Metadane: zapisywane tekstowo, napisy zakończone znakiem o kodzie 0, liczby zapisywane ósemkowo. Wypełnienie do granicy rekordu: znak o kodzie 0.

Nagłówek *(us-)tar*

Offset	Rozmiar	Zawartość
0	100	Ścieżka dostępu do pliku
100	8	Prawa dostępu (21 bitów ósemkowo)
108	8	Numer użytkownika
116	8	Numer grupy użytkownia
124	12	Rozmiar pliku w bajtach (ósemkowo)
136	12	Czas ostatniej modyfikacji (epoka Uniksa, ósemkowo)
148	7	Suma kontrolna nagłówka (18 bitów ósemkowo)
156	2	Typ pliku
157	100	Nazwa pliku wskazywanego przez link symboliczny
257	8	Ciąg znaków `ustar\x20\x20\0`
265	32	Nazwa użytkownika
297	32	Nazwa grupy użytkownika
329	8	Numer major urządzenia
337	8	Numer minor urządzenia
345	155	Prefiks nazwy pliku

Dyski

Urządzenia blokowe

- Zapis/odczyt *buforowany* w jądrze, z wykorzystaniem DMA.
- Jednostka adresowania: *sektor*, 512B lub 4KiB (*advanced format*). Adresy: LBA.
- Urządzenia blokowe *fizyczne* i *logiczne*: podział na partycje, LVM, dm, loop.

Systemy plików

- *Plik* — wirtualne urządzenie blokowe.
- Pliki mają *nazwy* i są pogrupowane w drzwiastą strukturę *katalogów*.
- Interfejs jądra: wywołania systemowe `open(2)`, `read(2)`, `write(2)`, `close(2)` i in.
- Interfejs wysokopoziomowy języka C (stdio, glibc): strumienie; `fopen(3)`, `fread(3)`, `fwrite(3)` i in. Dostęp *buforowany* na poziomie plików. Wiele udogodnień, np. `fprintf`, odporność na przerwanie operacji przez sygnały i in.

On-disk format systemu plików

Przechowywanie zawartości pliku na dysku

- Zawartość plików przechowywana w *blokach* (wielokrotność sektora).
- W ext234 rozmiar bloku $= 2^i$ KiB ($i = 1, \ldots, 6$), w niektórych architekturach nie więcej niż rozmiar strony pamięci wirtualnej. W x86: 1, 2 lub 4 KiB. W UFS niekoniecznie, zwykle większe.
- Blok należy w całości do jednego pliku.
  Wyjątek (np. reiserfs): *tail packing*.

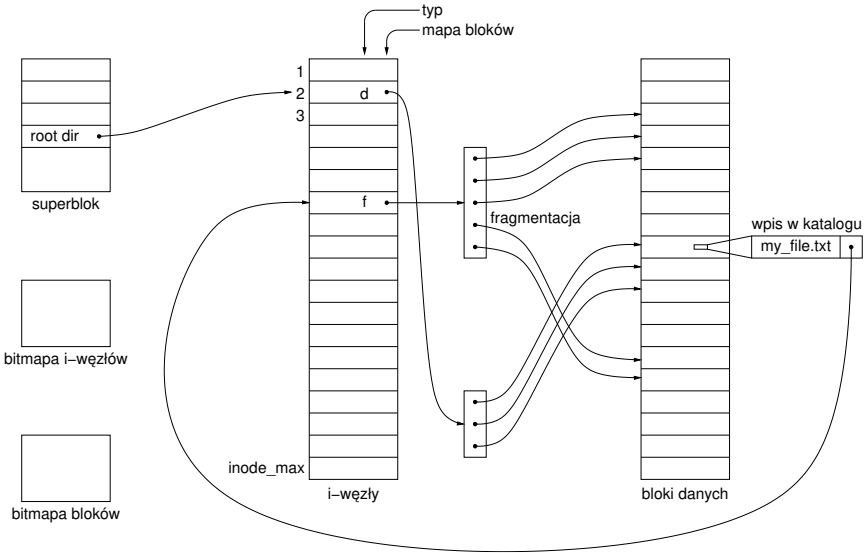
Metadane

- Blok zawierający informacje o całym systemie plików: *superblok*.
- Pliki są numerowane liczbami naturalnymi.
- Baza danych istniejących plików: tablica rekordów (i-węzłów) indeksowana numerami plików.
- I-węzeł zawiera m. in. *mapę bloków pliku*.
- Bitmapa zajętych i-węzłów.
- Bitmapa zajętych bloków.

Katalogi

- Katalog: plik zawierający zbiór nazw plików i przyporządkowanych im numerów plików (i-węzłów).
- Ponieważ katalog jest plikiem, to pojawia się stuktura rekurencyjna: drzewo katalogów.
- Ten sam i-węzeł może występować w wielu katalogach pod różnymi nazwami — dowiązania twarde (*hard links*).
- Potrzeba *link count* (zapisanego w i-węźle). Plik jest usuwany, gdy *link count* spadnie do zera.
- Dowiązania twarde zostały z czasem zabronione dla katalogów (katalogi tworzą *drzewo*).

Struktura systemu plików



Mapa bloków pliku

Wymagania

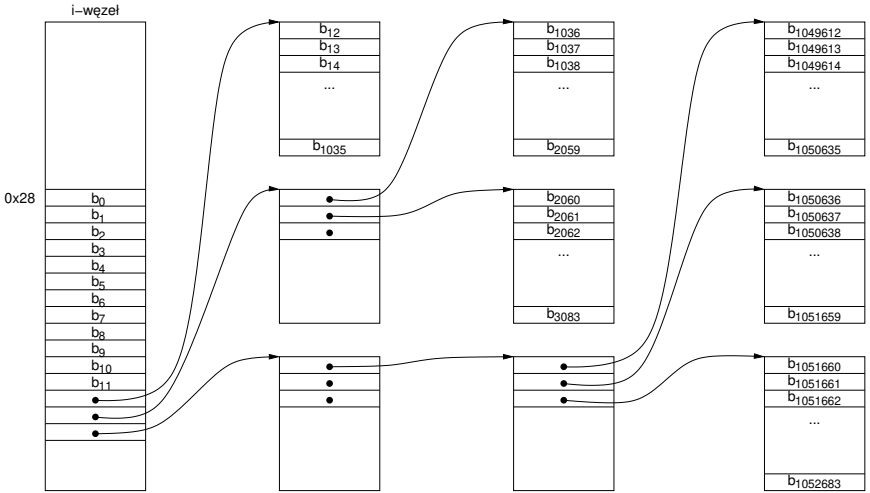
- Rozmiar pliku w bajtach jest zapisany w i-węźle, zatem wiadomo, ile bloków należy do pliku.
- Jednostką adresowania wolumenu jest teraz blok. Adresy bloków są 32-bitowe, chyba że system ext4 ma własność `INCOMPAT_64BIT`.
- Dla bloków 4 KiB i 32-bitowych numerów rozmiar wolumenu $\leq$ 16 TiB.
- *Seek* w pliku: aby przeczytać/zapisać `i`-ty bajt pliku, trzeba zlokalizować `j = i >> (10 + s_log_block_size)` blok tego pliku.
- Wyznaczenie fizycznego numeru bloku na podstawie `j` powinno być szybkie.
- Mapa bloków powinna zajmować niewiele miejsca.
- Adres bloku równy 0 warto zarezerwować na *dziury* (*holes*) w plikach.

Mapa bloków pliku

Klasyczne rozwiązanie (UFS, ext23)

- W i-węźle znajduje się tablica `B` numerów bloków rozmiaru 15.
- `B[0]` ... `B[11]` to numery pierwszych 12 bloków pliku.
- `B[12]` to numer bloku zawierającego tablicę kolejnych $\mathtt{N} = \mathtt{block\_size}/\mathtt{block\_addr\_size}$ numerów bloków pliku (zwykle 1024).
- `B[13]` to numer bloku zawierającego tablicę `N` numerów bloków, z których każdy zawiera tablicę kolejnych `N` numerów bloków pliku.
- `B[14]` to numer bloku zawierającego tablicę `N` numerów bloków, z których każdy zawiera tablicę `N` numerów bloków, z których każdy zawiera tablicę kolejnych `N` numerów bloków pliku.
- Plik może mieć co najwyżej $12 + \mathtt{N} + \mathtt{N}^2 + \mathtt{N}^3$ bloków. Dla bloków 4 KiB i 32-bitowych numerów bloków maksymalny rozmiar pliku to 1074791436 bloków, tj. około 4 TiB.

Mapa bloków pliku



Algorytm wyznaczania numeru bloku

Dane: numer `j` bloku pliku. Szukane: adres bloku `b` na dysku.

```
read_inode(I,n);                  /* wczytaj i-node n do bufora I */
B0 = I + 40;                        /* B0 wskazuje na mapę bloków */
if (j < 12) {
   b = B0[j];                                  /* direct block */
} else if ((j-=12) < N) {
   read_block(B1, B0[12]);
   b = B1[j];                                /* single indirect */
} else if ((j-=N) < N*N) {
   read_block(B1, B0[13]);
   read_block(B2, B1[j/N]);
   b = B2[j%N];                              /* double indirect */
} else {
   j -= N*N;
   read_block(B1, B0[14]);
   read_block(B2, B1[j/(N*N)]);
   j %= N*N;
   read_block(B3, B2[j/N]);
   b = B3[j%N];                              /* triple indirect */
}
```

Mapa bloków

Krytyka klasycznej mapy bloków

- Na każdy blok pliku przypada 32 lub 64 bity mapy bloków (plus adresy pośrednie). Dla bloków 4 KiB i adresów 32-bitowych mapa bloków zużywa ponad 1/1000 rozmiaru pliku.
- Długi czas przeszukiwania mapy (fsck).

Rozwiązanie: zakresy adresów bloków (*extents*)

- *Seek* w pliku musi być szybki!
- Zakresy są stałego rozmiaru, zapisanego w i-węźle (rozwiązanie w FFS, także własność `bigalloc` w ext4).
- Systemowe rozwiązanie w ext4: B+-drzewo zakresów.

Lokalność przestrzenna

Seek time

- Odczyt/zapis sekwencyjny nawet 100 razy szybszy niż przypadkowy.
- Dyski magnetyczne: przesunięcie głowicy + obrót plateru. Seek time rośnie proporcjonalnie do odległości (różnicy LBA).
- Dyski SSD: czas otwarcia bloku kasowania (rzędu 8 MB). Seek time rośnie skokowo po przekroczeniu rozmiaru bloku kasowania.

Rozwiązanie dobre dla dysków magnetycznych

- Kolejno zapisywane bloki powinny mieć bliskie adresy.
- System plików zapisuje na przemian bloki danych i metadanych.
- Wniosek: dane i metadane powinny być blisko siebie.
- Rozwiązanie: grupy cylindrów (UFS), grupy bloków (ext234).
- Bliskie dane względem czasu dostępu (dane i metadane jednego pliku, pliki z tego samego katalogu) umieszczać blisko siebie na dysku — w tej samej grupie bloków.
- Aby to umożliwić bez konieczności późniejszego przesuwania (defragmentacji) — nieskorelowane dane „rozrzucać" po różnych grupach bloków.

Grupy bloków

- Na całym dysku ciągła numeracja bloków począwszy od 0.
- Dysk podzielony na grupy bloków stałego rozmiaru (ostatnia być może niepełna).
- Geometria grup bloków jest opisana w tablicy deskryptorów grup.
- Alokator stara się przydzielić miejsce dla danych i metadanych w tej samej grupie bloków (o ile to możliwe).
- Oryginalnie w UFS: grupy cylindrów: wyrównanie grup bloków do granic cylindrów — minimalizuje konieczność ruchów głowicy.
- Od lat '90 geometria dysków nie jest znana, więc nie ma wyrównania do cylindrów.

Struktura grupy bloków ext234

- Tylko w grupie 0: 1024 bajty nieużywane.
- Superblok rozmiaru 1024 bajty (aktywny w grupie 0, w pozostałych: pasywny *backup*).
- Puste do końca bloku nr 0 (bloki $> 2\,\mathrm{KiB}$).
- Tablica deskryptorów grup.
- Dodatkowe miejsce zarezerwowane na tablicę deskryptorów grup (gdyby powiększono partycję: `e2resize`).
- Bitmapa zajętych bloków tej grupy: 1 blok.
- Bitmapa zajętych i-węzłów tej grupy: 1 blok.
- Tablica i-węzłów; rozmiar:
  `(s_inodes_per_group * s_inode_size) >> (10 + s_log_block_size)` bloków.
- Obszar bloków danych do końca grupy.

Rozmiar grupy bloków i *backupy* metadanych

- Bitmapa bloków zajmuje co najwyżej 1 blok. Stąd wynika maksymalny rozmiar grupy bloków: `(1 << 2 * (10 + s_log_block_size) + 3)`. Dla bloków 4 KiB $= 2^{12}$ B wynosi 2^{27} B $= 128$ MiB.
- Oryginalnie każda grupa zawierała *backup* superbloku i tablicy deskryptorów grup.
- Jeśli system ma własność `sparse_super`, to backupy są tylko w grupie 1 oraz tych, których numer dzieli się przez 3, 5 lub 7.
- Jeśli system ma własność `sparse_super2`, to *backupy* są tylko w grupie 1 i ostatniej.
- Można zrobić backup superbloku i BGDT w pliku na innym dysku.

Struktury danych

Zobacz dokumenty:

- Rémy Card, Theodore Ts'o, Stephen Tweedie, Design and Implementation of the Second Extended Filesystem
- Dave Poirier, The Second Extended File System Internal Layout
- Ext4 wiki: Ext4 Disk Layout
- Kernel wiki: ext4 Data Structures and Algorithms

Narzędzia: e2fsprogs

e2fsprogs

- `mke2fs`, `mklost+found`, `e2fsck`
- `filefrag`, `e2freefrag`, `e4defrag`
- `chattr`, `lsattr`
- `dumpe2fs`, `tune2fs`, `resize2fs`, `debugfs`, `e2image`, `e2undo`, `e2label`, `badblocks`

Inne

- `genext2fs` (systemy wbudowane)
- `fuseext2` (moduł ext2/3/4 dla FUSE)
- `ext3grep`, `extundelete`

Features

Ewolucja ext234 możliwa dzięki rozszerzeniom

- kompatybilne
- kompatybilne tylko do odczytu
- niekompatybilne

System plików ext234

- Jeden wspólny kod w jądrze.
- Rozróżnienie ext2/ext3/ext4: zdefiniowane zbiory własności.
- plik `/etc/mke2fs.conf`

Compatible features

`has_journal`	wykorzystuje księgowanie (i-węzeł 8).
`ext_attr`	udostępnia atrybuty roszerzone.
`resize_inode`	BGDT mogą być powiększane.
`dir_index`	używa H+-drzew do indeksowania katalogów.
`sparse_super2`	co najwyżej dwie kopie superbloku i BGDT.
`metadata_csum`	sumy kontrolne i-węzłów.

Read-only compatible features

`sparse_super` *backup* suerbloku w wybranych grupach.

`large_file` zawiera pliki większe niż 2 GiB.

`huge_file` rozmiary plików mierzone w blokach a nie sektorach (pozwala na pliki rozmiaru 2–16 TiB).

`uninit_bg` posiada sumy kontrolne BGDT.

`bigalloc` bitmapy bloków opisują klastry bloków (domyślnie po 16).

`dir_nlink` ignoruje licznik referencji dla katalogów (pozwala na katalogi zawierające więcej niż 32000 podkatalogów).

`extra_isize` rezerwuje dodatkowe miejsce w i-węźle na metadane.

`quota` ograniczanie przestrzeni zajętej przez użytkownika.

`project` graniczanie przestrzeni zajętej przez projekt.

Incompatible features

`64bit`	numery bloków mają 64 bity (system większy niż 16 TiB).
`ea_inode`	atrybuty rozszerzone mogą zajmować wiele i-węzłów.
`encrypt`	zawartość plików jest szyfrowana.
`extent`	używa B+-drzewa zakresów zamiast tradycyjnej mapy bloków.
`filetype`	typ pliku przechowywany także we wpisie w katalogu.
`flex_bg`	metadane grup bloków mogą być przechowywane w innych blokach.
`meta_bg`	metadane grup bloków mogą być powiększane.
`inline_data`	dane mogą być przechowywane w i-węźle.
`journal_dev`	księga jest przechowywana na osobnym urządzeniu.
`large_dir`	duże katalogi (wyższe H+-drzewo itp.)
`metadata_csum_seed`	zarodek jest przechowywany w superbloku. Pozwala na dynamiczną zmianę UUID systemu.
`mmp`	*multiple mount protection*.

mke2fs.conf

```
[defaults]
base_features = sparse_super,filetype,resize_inode,
   dir_index,ext_attr
default_mntopts = acl,user_xattr
enable_periodic_fsck = 0
blocksize = 4096
inode_size = 256
inode_ratio = 16384

[fs_types]
ext3 = {
   features = has_journal
}
ext4 = {
    features = has_journal,extent,huge_file,flex_bg,
       uninit_bg,dir_nlink,extra_isize
    auto_64-bit_support = 1
    inode_size = 256
}
```

Problem spójności danych i księgowanie

- ACID (Atomicity, Consistency, Isolation, Durability)
- Transakcje, które powinny być atomowe: alokacja i dealokacja bloku, dodanie i usunięcie linku w katalogu.
- Zob.: Gregory R. Ganger, Yale N. Patt, Soft Updates: A Solution to the Metadata Update Problem in File Systems
- *Journaling:* IBM JFS (1990), Microsoft NTFS (2000), ext3 (2001)
- Zob.: Stephen C. Tweedie, Journaling the Linux ext2fs Filesystem

Księgowanie w ext34

- Zadanie: ciąg bloków zostanie zapisany w całości (atomowo) lub wcale.
- Bloki są najpierw *w całości* zapisywane do księgi.
- Następnie są zapisywane we właściwe miejsca.
- Na koniec bloki w księdze są oznaczone jako przepisane.
- JFS, XFS, NTFS — transakcje: opis sposobu modyfikacji bloków.

Złożoność systemów plików

System	Liczba osobolat
ext4	8.5
XFS	17
ZFS	77

Złożoność systemów plików

System	Liczba osobolat
ext4	8.5
XFS	17
ZFS	77

I wrote FAT on an airplane, for heaven's sake.

Bill Gates (as quoted by Raymond Chen, 2013)

FAT

Struktura systemu na dysku

- Region zarezerwowany
- Tablica FAT
- Katalog główny (nie dotyczy FAT32)
- Region plików i katalogów

Przydział pamięci

- Jednostka adresowania: *klaster*. Numery klastrów są 8- (oryginalny system DOS), 12- (FAT12), 16- (FAT16) lub 28-bitowe (FAT32, zapisywane w 4 bajtach, 4 najbardziej znaczące bity zarezerwowane).
- Jeśli liczba klastrów jest mniejsza niż 4085, to mamy FAT12, jeśli większa lub równa 4085 i mniejsza od 65525, to FAT16, w p.p. FAT32.
- Rozmiar klastra: 1, 2, 4, 8, 16, 32, 64 lub 128 sektorów.
- Najmniejszy legalny rozmiar FAT32: 65525+1 sektorów.
- Maksymalna liczba klastrów w FAT32: $2^{28} - 2 = 268435454$. Dla klastrów 16-sektorowych rozmiar wolumenu ok. 2TB.

BIOS Parameter Block (Boot sector)

Offset	Rozmiar	Zawartość
0	3	0xEB*XX*90 (`jmp` *XX*)
3	8	Nazwa kreatora (oryg. `MSWIN4.1`)
11	2	Rozmiar sektora w B
13	1	Liczba sektorów na klaster
14	2	Rozmiar regionu zarezerwowanego w sektorach
16	1	Liczba tablic FAT (zwykle 2)
17	2	Liczba wpisów w katalogu głównym (FAT12 i 16)
19	2	Rozmiar wolumenu w sektorach (FAT12 i 16)
21	1	Typ nośnika
22	2	Rozmiar tablicy FAT w sektorach (FAT12 i 16)
24	2	Liczba sektorów na ścieżkę
26	2	Liczba głowic
28	4	Liczba ukrytych sektorów
32	4	Rozmiar wolumenu w sektorach

BPB, cd. dla FAT32

Offset	Rozmiar	Zawartość
36	4	Rozmiar tablicy FAT w sektorach
40	2	Konfiguracja tablic FAT
42	2	Wersja systemu plików
44	4	Adres katalogu głównego (zwykle 2 — pierwszy klaster)
48	2	FSINFO
50	2	Adres (w sektorach) backupu BPB
52	12	Zarezerwowane
64	1	Numer napędu
65	1	Zarezerwowane, wpisać 0
66	1	Rozszerzona sygntura 0x29
67	4	ID wolumenu
71	11	Etykieta wolumenu
82	8	Zawsze napis `FAT32\x20\x20\x20`

Tablica FAT

- Dla każdego klastra określa następny klaster w łańcuchu.
- Specjalne numery klastrów w FAT32: `0` — nieużywany, `0x0ffffff7` — uszkodzony, `0x0fffffff` — koniec łańcucha klastrów.
- Indeksy i wartości — numery klastrów.
- Zużyty obszar: *liczba klastrów* $\times$ *rozmiar numeru klastra*.
- Ułamek: *rozmiar numeru klastra*/*rozmiar klastra*, np. $4/8192 = 1/2048$ dla FAT32 i klastrów 16-sektorowych.
- Przykład: pendrive 16GB, klastry 8192 B: 2'097'152 klastrów. Rozmiar FAT: 8 MB.
- Problem: wyznaczenie liczby wolnych klastrów wymaga przejrzenia całej tablicy FAT (są rozszerzenia w FAT32).

Katalog w FAT32

Offset	Rozmiar	Zawartość
0	11	Nazwa (8+3)
11	1	Atrybuty (6 bitów)
12	1	Zarezerwowane, wpisać 0
13	1	1/10 sekundy czasu utworzenia (0–199)
14	2	Czas utworzenia
16	2	Data utworzenia
18	2	Data ostatniego dostępu
20	2	Dwa górne bajty numeru klastra
22	2	Czas ostatniej modyfikacji
24	2	Data ostatniej modyfikacji
26	2	Dwa dolne bajty numeru klastra
28	4	Rozmiar pliku w bajtach

Uwagi

- Rozdzielczość zapisu czasu: 2 sekundy.
- Maksymalny rozmiar pliku: $2^{32} - 1$ B, czyli 4 GB.

Nazwa pliku

Krótkie nazwy

- Małe i wielkie litery utożsamiane (są tylko wielkie).
- Format: 8+3, uzupełniane z prawej spacjami.
- Pierwszy znak `0xE5` — wolny wiersz w katalogu.
- Pierwszy znak `0x00` — wolny wiersz w katalogu, brak dalszych.
- Pierwszy znak `0x05` — znak `0xE5`.

Długie nazwy

- Konserwatywne rozszerzenie.
- Dodatkowe wpisy w katalogu.

Format nie definiuje procedur

Design space

- Wybór typu systemu (16–32) i jego parametrów (rozmiar klastra).
- Polityka alokacji klastrów — czy można zmniejszyć fragmentację?
- Prealokacja
- Proces defragmentacji
- Spójność wpisów w katalogach (problem długich nazw)

Naginanie formatu — czy można?

- *hard links*
- deduplikacja

Microsoft FAT32 File System Specification

The “fixed value” is simply a volume size that is the “FAT16 to FAT32 cutover value”. Any volume size smaller than this is FAT16 and any volume of this size or larger is FAT32. For Windows, this value is 512 MB. Any FAT volume smaller than 512 MB is FAT16, and any FAT volume of 512 MB or larger is FAT32. [. . .]
Do not spend too much time trying to figure out why this math works. The basis for the computation is complicated; the important point is that this is how Microsoft operating systems do it, and it works. Note, however, that this math does not work perfectly. It will occasionally set a FATSz that is up to 2 sectors too large for FAT16, and occasionally up to 8 sectors too large for FAT32. It will never compute a FATSz value that is too small, however. Because it is OK to have a FATSz that is too large, at the expense of wasting a few sectors, the fact that this computation is surprisingly simple more than makes up for it being off in a safe way in some cases.

Implementacja w Linuksie

Obsługa w jądrze

- Warstwa abstrakcji systemu plików (VFS) pozwala obsługiwać wiele różnych formatów.
- Moduły jądra `fat` i `vfat` — pełna obsługa FAT12, FAT16 i FAT32.

Narzędzia

- **dosfstools:** `mkfs.fat` (alias: `mkfs.vfat`, `mkfs.msdos`, `mkdosfs`), `fsck.fat` (alias: `fsck.vfat`, `fsck.msdos`, `dosfsck`), `fatlabel` (alias: `dosfslabel`).
- **fatcat** (odpowiednik `debugfs`).
- **testdisk** (odpowiednik `debugfs`).
- **fatsort** (sortuje wpisy w katalogach).
- **fusefat** (moduł FAT dla FUSE) i **mtools** (niezależna implementacja FAT w userspace).
- Inne: **fatattr**, **makebootfat**.