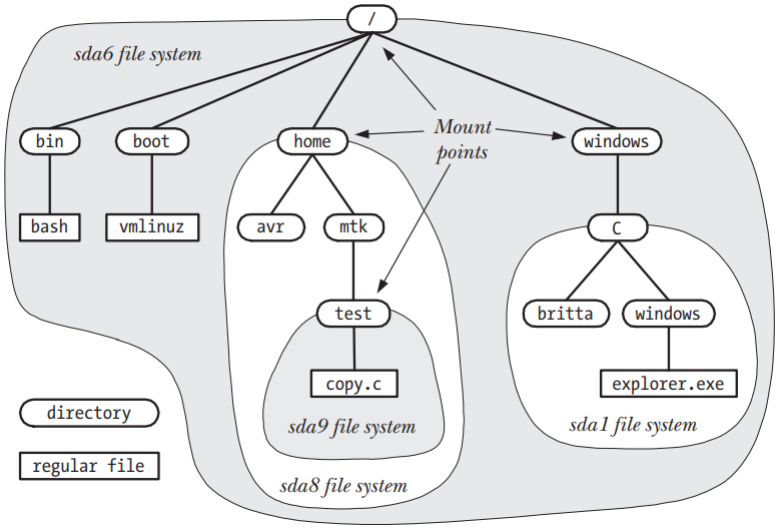
Lista 11

tags: SO

Zadanie 1

Zadanie 1. Wyjaśnij czym są **punkty montażowe**, a następnie wyświetl listę zam plików i wyjaśnij co znajduje się w poszczególnych kolumnach wydruku. Które z punk dostęp do instancji **pseudo systemów plików**? Na podstawie `mount(8)` wyjaśnij z atrybutów punktów montażowych: «noatime», «noexec» i «sync», a następnie pod ich zastosowanie jest pożądane.

Wskazówka: Rozważ semantykę wymienionych atrybutów w kontekście systemu plików na przenoś



::: info **punkty montażowe** – A mount point is a directory or file at which a new file system, directory, or file is made accessible. To mount a file system or a directory, the mount point must be a directory; and to mount a file, the mount point must be a file.

Typically, a file system, directory, or file is mounted over an empty mount point, but that is not required. If the file or directory that serves as the mount point contains any data, that data is not accessible while it is mounted over by another file or directory. In effect, the mounted file or directory covers what was previously in that directory. The original directory or file that has been mounted over is accessible again once the mount over it is undone. (source)

Można je zobaczyć poleceniem `mount` lub (w piękniejszej wersji) `findmnt` (source)

- SOURCE -- urządzenie źródłowe (partycja, katalog itd.)
- FSTYPE -- typ systemu plików (np. btrfs)
- OPTIONS (najpopularniejsze)
  - `nosuid` – Do not honor set-user-ID and set-group-ID bits or file capabilities when executing programs from this filesystem.
  - `rw` / `ro` – read-write / read-only
  - `noexec` – Do not permit direct execution of any binaries on the mounted filesystem.
  - `nodev` – Do not interpret character or block special devices on the filesystem.
  - `seclabel` – (???)
  - `relatime` – Update inode access times relative to modify or change time.

(szczegóły wydruku)

pseudo system plików – is a hierarchical interface to non-file objects that appear as if they were regular files in the tree of a disk-based or long-term-storage file system. :::

Które z punktów montażowych dają dostęp do pseudo systemów plików?

/proc, /sys

Na podstawie mount(8) wyjaśnij znaczenie następujących atrybutów punktów montażowych

```
noatime
              Do not update inode access times  on  this  filesystem  (e.g.  for
              faster  access  on the news spool to speed up news servers).  This
              works for  all  inode  types  (directories  too),  so  it  implies
              nodiratime.
```

Scenariusz: Przyspieszenie dostępu do danych (np. dla przesyłu z USB o dużej liczbie plików)

```
noexec
              Do  not  permit  direct  execution  of any binaries on the mounted
              filesystem.
```

Scenariusz: Bezpieczeństwo plików systemu (jeśli wiemy, że system plików ma tylko rpzechowywać pliki to lepiej nie umożliwiać ich uruchamiania)

```
sync          All I/O to the filesystem should be done  synchronously.   In  the
              case  of  media  with  a limited number of write cycles (e.g. some
              flash drives), sync may cause life-cycle shortening.

Meaning that all changes to the according
filesystem are immediately flushed to disk; the respective
write operations are being waited for.
```

Scenariusz: Chcemy zapobiec utracie danych (np. poprzez chwilową przerwę zasilania)

Zadanie 2

Zadanie 2. Korzystając z pól **superbloku** (ang. *superblock*) podaj wzór na wylicz
bloku, liczby i-węzłów i bloków przechowywanych w **grupie bloków** (ang. *block group*
deskryptorów grup bloków (ang. *block group descriptor table*). Wymień składowe na
oraz podaj ich rozmiar w blokach. Które grupy bloków przechowują kopie zapasow
deskryptorów grup bloków?

::: info (source)

Blok - Partycja, dysk, plik lub urządzenie blokowe jest podzielone na bloki (zbiory sektorów).

Grupa bloków - bloki są grupowane między innymi w celu zmniejszenia fragmentacji.

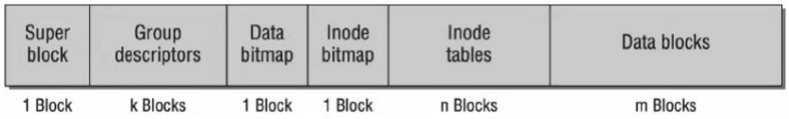
Tablica deskryptorów grup bloków - przechowuje się w niej informacje o każdej grupie bloków, takie jak lokalizacje mapy bitowej i tablicy i-węzłów, mapy bitowej bloku, liczby wolnych bloków i i-węzłów. Tablica deskryptorów jest przechowywana bezpośrednio po superbloku.

Superblok - przechowuje metadane dotyczące zamontowanego systemu plików ::: spoiler -- liczba i-węzłów -- liczba wszystkich bloków systemu plików -- liczba bloków zarezerwowana dla roota -- liczba wolnych bloków -- liczba wolnych i-węzłów -- s_log_block_size (do wyliczania rozmiaru bloku) -- s_log_frag_size (do wyliczania rozmiaru fragmentu) -- liczba bloków na grupę bloków -- liczba fragmentów na grupę bloków -- liczba i-węzłów na grupę bloków -- uniksowy czas ostatniego zamontowania systemu plików -- uniksowy czas ostatniego zapisu do systemu plików -- liczba zamontowań od ostatniej weryfikacji -- największa możliwa liczba zamontowań przed koniecznością wykonania pełnego sprawdzenia systemu -- (magic) wartość identyfikująca typ systemu plików -- stan systemu (zamontowany lub nie) służący do sprawdzenia czy system został poprawnie odmontowany i nie zawiera potencjalnych błędów. :::

Oblicz

- **rozmiar bloku** = 1024 << *s_log_block_size*
- **liczba i-węzłów w grupie bloków** = *s_inodes_per_group*
- **liczba bloków w grupie bloków** = *s_blocks_per_group*
- **deskryptory** = $\lceil$ *s_blocks_count / s_blocks_per_group* $\rceil$

Wymień składowe należące do grupy bloków oraz podaj ich rozmiar w blokach.



$$ n = \frac{s_inodes_per_group}{blksz / s_inode_size} $$ $$ k = \lceil deskryptory \cdot \frac{sizeof(group_desc)}{blksz} \rceil $$

(source) **Które grupy bloków przechowują kopie zapasową superbloku i tablicy deskryptorów grup bloków?**

The first version of ext2 (revision 0) stores a copy at the start of every block group, along with backups of the group descriptor block(s). Because this can consume a considerable amount of space for large filesystems, later revisions can optionally reduce the number of backup copies by only putting backups in specific groups (this is the sparse superblock feature). The groups chosen are 0, 1 and powers of 3, 5 and 7.

Zadanie 3

Zadanie 3. Podstawowymi operacjami na systemie plików są: wyzeruj lub zapal b
albo bloków, wczytaj / zapisz i-węzeł albo **blok pośredni** (ang. *indirect block*) albo b
kroków niezbędnych do realizacji funkcji dopisującej n bloków na koniec pliku. Zakła
kroki funkcji są zawsze wdrażane **synchronicznie**. Zadbaj o to by funkcje nie narusz
plików w przypadku awarii zasilania. Dopuszczamy powstawanie wycieków pamięci.

::: info **Blok pośredni** (*indirect block*) -- w inode jest używany przy dużych plikach. Zawiera wskaźniki na inne bloki pośrednie lub na bloki danych.

Zapisy synchroniczne -- wymuszamy czekanie na zakończenie operacji wejścia-wyjścia. Szybkość operacji ograniczona przepustowością dysku.

Spójność systemu plików -- zgodność faktycznego stanu struktur w systemie plików z ich metadanymi. :::

Dopisanie n bajtów na koniec pliku

1. odszukujemy wolne bloki łącznie mogące pomieścić n bajtów
   - interesują nas grupy, gdzie są jeszcze wolne bloki (tj. *bg_free_blocks_count* > 0)
   - dekrementujemy *s_free_blocks_count* superbloku i *bg_free_blocks_count* grup wybranych bloków
2. ustawiamy odpowiadające im bity w *data bitmap* (bloki są teraz w użyciu)
3. aktualizujemy i-węzeł o nowo dodane bloki (tj. uzupełniamy odpowiednio jego strukturę drzewiastą i metadane)

Zadanie 4

Zadanie 4. Przy pomocy wywołania systemowego `rename(2)` można przenieść **ato**
znajdującego się w obrębie tego samego systemu plików. Czemu «`rename`» zakończy się
próbujemy przenieść plik do innego systemu plików? Powtórz polecenia z zadania 3 c
plik między dwoma różnymi katalogami w obrębie tego samego systemu plików. Zak
docelowym jest wystarczająco dużo miejsca na dodanie wpisu. Pamiętaj, że wpis katal
granicy między blokami!

::: info **atomowość** -- własność operacji odpornej na przerwanie, w tym przypadku oznacza, że plik przenoszony jest w sposób bezpieczny, tj. nie ryzykujemy pozostawienia *stanu pośredniego* przenoszonego pliku.

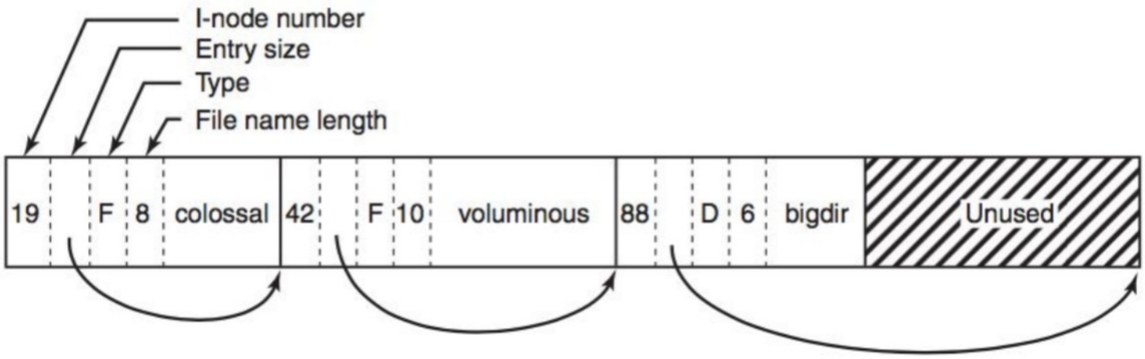
EXDEV (Invalid cross-device link) -- cross-device link not permitted. :::

Dlaczego *rename* zwraca błąd przy przenoszeniu między systemami?

Rename działa atomowo, ponieważ manipuluje tylko wpisamy katalogów. Przerzucenie pliku między systemami plików wymagałoby manipulacji i-węzłami.

Przenoszenie między katalogami

Reprezentacja katalogów



1. Korzystając z tablicy i-węzłów oraz mijanych katalogów odnajdujemy i-węzeł I zadany pierwszym argumentem *rename*.
2. Odszukujemy katalog docelowy:
   - jeśli plik pod tą nazwą już istnieje podmieniamy jego i-węzeł na I
   - wpp. próbujemy wypełnić nieużytek katalogu lub rozszerzamy listę wiązaną katalogu o nowy wpis (tj. przestawiamy wskaźnik poprzedniego wpisu, dbając by nowy wpis nie przeciął granicy między blokami)
3. zwolnij wpis w poprzednim katalogu

Zadanie 5

Zadanie 5. Przy pomocy wywołania systemowego `unlink(2)` można usunąć plik
Powtórz polecenia z zadania 3 dla funkcji usuwającej plik zwykły z katalogu. Kiedy r
operacji usunięcia pliku tj. odkasowania (ang. *undelete*)? Zauważ, że usunięcie plik
możliwości czytania jego zawartości, o ile go otworzyły przed wywołaniem `unlink(2`
plik zostanie faktycznie usunięty z dysku?

Usuwanie

1. usuwamy wpis w katalogu
   - jeśli to pierwszy wpis tego katalogu to zerujemy numer i-węzła
   - wpp. zwiększamy rozmiar poprzedniego wpisu. Wpis, który chcemy usunąć stanie się nieużytkiem poprzedniego wpisu
2. dekrementujemy *referance count*
3. jeśli *referance count* == 0 i nie ma otwartych deskryptorów:
   - zwalniamy bloki w i-węźle (aktualizujemy liczby wolnych blokół w superblokach i grupach zwalnianych bloków, gasimy bity w odpowiednich mapach bitowych)
   - zwalniamy i-węzeł w tablicy i-węzłów

Kiedy możliwe jest przywrócenie pliku?

Jeśli i-węzeł nie został wyczyszczony???

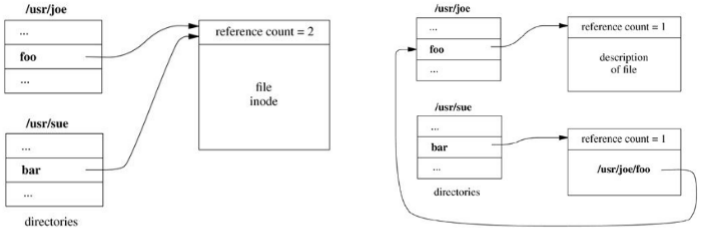
Kiedy plik jest usuwany

Plik zostanie usunięty z dysku, gdy *reference count* jego i-węzła spadnie do zera oraz nie będzie otwarty w żadnym programie, wtedy struktura i-węzła jest zwalniana i nie mamy sposobu odszukiwania poszczególnych bloków danych, z których składał się ten plik.

Zadanie 6

Zadanie 6. Wyjaśnij co robi system plików `ext2` przy tworzeniu **dowiązania twa**
i **symbolicznego** (ang. *symbolic link*). Gdzie jest przechowywana zawartość dowiąza
za pomocą dowiązania symbolicznego stworzyć w systemie plików pętlę? Kiedy jądrc
ją wykryje i zwróci błąd «ELOOP»? Czemu pętli nie da się zrobić z użyciem dowiązar

Dowiązanie symboliczne vs. twarde



Dowiązania twarde to wskaźniki na i-węzły (licznik referencji!) plików → różne nazwy tego samego pliku w obrębie jednego systemu plików.

Dowiązania symboliczne kodują ścieżkę do której należy przekierować algorytm rozwiązywania nazw.

Co robi ext2 przy tworzeniu dowiązania twardego i symbolicznego?

Utworzenie dowiązania twardego to po prostu dodanie wpisu do katalogu z numerem istniejącego już i-węzła i podbicie jego licznika referencji.

Utworzenie dowiązania symbolicznego to z kolei utworzenie nowego i-węzła przechowującego ścieżkę do pliku docelowego i dodanie stosownego wpisu do katalogu.

Gdzie jest przechowywana zawartość dowiązania symbolicznego W i-węźle w miejscu normalnie okupowanym przez listę adresów bloków (o ile ścieżka nie jest na to za długa) lub w sposób klasyczny w blokach. (source)

Jak za pomocą dowiązania symbolicznego stworzyć w systemie plików pętlę? `ln -s . loop` `find -L loop`

> jest to możliwe, ponieważ system nie sprawdza poprawności symlinków w żaden sposób

Kiedy jądro systemu operacyjnego ją wykryje i zwróci błąd «ELOOP»? Istnieje twardy limit na liczbę dowiązań symbolicznych w ścieżce do pliku, po którego przekroczeniu zwracany jest błąd *ELOOP*.

Czemu pętli nie da się zrobić z użyciem dowiązania twardego Zabrania się tworzenia dowiązań twardych dla katalogów. Pętla po twardych dowiązaniach byłaby wyjątkowo trudna do wykrycia oraz musielibyśmy zrezygnować z bardzo wygodnej struktury drzewiastej dla hierarchii katalogów.

Zadanie 7

Zadanie 7. Czemu **fragmentacja systemu plików** jest szkodliwym zjawiskiem? Zr
ext4 filesystem: current status and future plans[2]. Opisz w jaki sposób **odrocz**
(ang. *delayed allocation*) [§3.2] zapobiega powstawaniu fragmentacji. Wytłumacz jak
[§2.2] pomagają w ograniczaniu rozmiaru metadanych przechowujących adresy bloków
pliku. Czy po defragmentacji systemu plików ext4 liczba wolnych bloków może
wyglądać najprostszy algorytm defragmentacji [§3.3]?

::: info **Fragmentacja systemu plików** - dane plików są na dysku ułożone w nieciągły sposób. Dostęp do nieciągłych zasobów jest wolniejszy. :::

Zakresy

Optymalizacja wykorzystująca fakt, że plik może składać się z fragmentów ciągłej pamięci. Ciągły przedział pamięci wyrażany jest w strukturze danych jako zakres (ang. *extend*). Możemy zatem reprezentować plik jako zbiór zakresów, tym sposobem oszczędzamy pamięć, która normalnie byłaby lokowana w niepotrzebnych wskaźnikach na poszczególne bloki pliku i strukturach danych. Kolejną zmianą poprawa jakości diagnostyki poprzez wyposażenie każdego węzła drzewa zakresów w header z metadanymi.

Odroczony przydział bloków

Żądania o alokacje bloków możemy buforować (do momnetu wykonania *page flush*) co daje nam wgląd w prawdopodobny rozmiar pliku. Możemy wtedy zrealizować bardziej doinformowaną strategię rozmieszczania bloków w pamięci redukując fragmentację. Takie rozwiązanie pozwala zaoszczędzić liczbę zawołań do VFS i zrezygnować z alokacji bloków tymczasowych.

Czy ilość wolnych bloków może wzrosnąć po defragmentacji?

Tak, bo użyta zostanie mniejsza liczba zakresów przez co część bloków wykorzystywana przez drzewo zakresów może zostać zwolniona.

Jak mógłby wyglądać najprostszy algorytm defragmentacji?

1. Utwórz tymczasowy i-węzeł I
2. Zaalokuj ciągły zakres dla I
3. Skopiuj dane z pierwotnego i-węzła do I
4. Ustaw wskaźniki w pierwotnym i-węźle na zaalokowany zakres
5. Zwolnij skopiowane bloki oraz węzeł I

Zadanie 8

Zadanie 8. Przy użyciu programu `debugfs(8)` dla wybranej instancji systemu plik
przechowująca główny system plików Twojej instalacji systemu Linux) pokaż:

- fragmentację systemu plików (`freefrag`) i informacje o grupach bloków (sta
- zakresy bloków z których składa się wybrany duży plik (`extents`),
- że dowiązanie symboliczne może być przechowywane w i-węźle (`idump`),
- do jakiego pliku należy wybrany blok (`blocks`, `icheck`, `ncheck`),
- reprezentację liniową małego katalogu (`bdump`).

Ostrzeżenie! Narzędzie `debugfs` działa domyślnie w trybie tylko do odczytu, więc możesz go bez
komputerze. Trybu do odczytu i zapisu używasz na własną odpowiedzialność!

Fragmentacja systemu plików

```
 debugfs:  freefrag

Device: ext4.fs
Blocksize: 1024 bytes
Total blocks: 102400
Free blocks: 93504 (91.3%)

Min. free extent: 4096 KB
Max. free extent: 28413 KB
Avg. free extent: 13357 KB
Num. free extent: 7

HISTOGRAM OF FREE EXTENT SIZES:
Extent Size Range :  Free extents   Free Blocks  Percent
    4M...    8M-  :             3         16713   17.87%
    8M...   16M-  :             3         48378   51.74%
   16M...   32M-  :             1         28413   30.39%
```

Informacje o grupach bloków

```
debugfs:  stats

Filesystem volume name:   testfs
Last mounted on:          <not available>
Filesystem UUID:          f8c7ce16-e81a-4df4-8e60-8e183ba1da1e
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal ext_attr resize_inode dir_index filetype extent 64bit flex_bg sp
arse_super large_file huge_file dir_nlink extra_isize metadata_csum
Filesystem flags:         signed_directory_hash
Default mount options:    user_xattr acl
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              25688
Block count:              102400
Reserved block count:     5120
Free blocks:              93504
Free inodes:              25677
First block:              1
Block size:               1024
Fragment size:            1024
Group descriptor size:    64
Reserved GDT blocks:      256
Blocks per group:         8192
Fragments per group:      8192
Inodes per group:         1976
Inode blocks per group:   247
Flex block group size:    16
Filesystem created:       Tue Jan 11 23:50:18 2022
Last mount time:          n/a
Last write time:          Tue Jan 11 23:50:18 2022
Mount count:              0
Maximum mount count:      -1
Last checked:             Tue Jan 11 23:50:18 2022
Check interval:           0 (<none>)
Lifetime writes:          278 kB
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)
First inode:              11
Inode size:               128
Journal inode:            8
Default directory hash:   half_md4
Directory Hash Seed:      6d845172-58f4-4370-958e-eb89e1e019df
Journal backup:           inode blocks
Checksum type:            crc32c
Checksum:                 0x17465728
Directories:              2

 Group  0: block bitmap at 259, inode bitmap at 272, inode table at 285
           4683 free blocks, 1965 free inodes, 2 used directories, 1965 unused inodes
           [Checksum 0x0ac3]
 Group  1: block bitmap at 260, inode bitmap at 273, inode table at 532
           7934 free blocks, 1976 free inodes, 0 used directories, 1976 unused inodes
           [Inode not init, Block not init, Checksum 0xdfa3]
 Group  2: block bitmap at 261, inode bitmap at 274, inode table at 779
           8192 free blocks, 1976 free inodes, 0 used directories, 1976 unused inodes
           [Inode not init, Block not init, Checksum 0x3c24]
 Group  3: block bitmap at 262, inode bitmap at 275, inode table at 1026
           7934 free blocks, 1976 free inodes, 0 used directories, 1976 unused inodes
           [Inode not init, Block not init, Checksum 0xadb7]
 Group  4: block bitmap at 263, inode bitmap at 276, inode table at 1273
           8192 free blocks, 1976 free inodes, 0 used directories, 1976 unused inodes
           [Inode not init, Block not init, Checksum 0x4a85]
 Group  5: block bitmap at 264, inode bitmap at 277, inode table at 1520
           7934 free blocks, 1976 free inodes, 0 used directories, 1976 unused inodes
           [Inode not init, Block not init, Checksum 0x17c6]
 Group  6: block bitmap at 265, inode bitmap at 278, inode table at 1767
           4096 free blocks, 1976 free inodes, 0 used directories, 1976 unused inodes
           [Inode not init, Checksum 0xe071]
 Group  7: block bitmap at 266, inode bitmap at 279, inode table at 2014
           7934 free blocks, 1976 free inodes, 0 used directories, 1976 unused inodes
           [Inode not init, Block not init, Checksum 0x904f]
 Group  8: block bitmap at 267, inode bitmap at 280, inode table at 2261
           8192 free blocks, 1976 free inodes, 0 used directories, 1976 unused inodes
           [Inode not init, Block not init, Checksum 0xd5bd]
 Group  9: block bitmap at 268, inode bitmap at 281, inode table at 2508
           7934 free blocks, 1976 free inodes, 0 used directories, 1976 unused inodes
           [Inode not init, Block not init, Checksum 0x7504]
 Group 10: block bitmap at 269, inode bitmap at 282, inode table at 2755
           8192 free blocks, 1976 free inodes, 0 used directories, 1976 unused inodes
           [Inode not init, Block not init, Checksum 0xf933]
 Group 11: block bitmap at 270, inode bitmap at 283, inode table at 3002
           8192 free blocks, 1976 free inodes, 0 used directories, 1976 unused inodes
           [Inode not init, Block not init, Checksum 0x9778]
 Group 12: block bitmap at 271, inode bitmap at 284, inode table at 3249
           4095 free blocks, 1976 free inodes, 0 used directories, 1976 unused inodes
           [Inode not init, Checksum 0xc922]
```

Zakresy pliku

```
debugfs: extents big_cats.png

Level Entries       Logical        Physical Length Flags
 0/ 0   1/  2     0 -  4682   3510 -   8192   4683
 0/ 0   2/  2  4683 - 19314   8451 -  23082  14632
```

i-węzeł dowiązania symbolicznego

```
debugfs:  inode_dump symbolic_cats

0000  ffa1 0000 0c00 0000 5815 de61 5815 de61  ........X..aX..a
0020  5815 de61 0000 0000 0000 0100 0000 0000  X..a............
0040  0000 0000 0000 0000 6269 675f 6361 7473  ........big_cats
0060  2e70 6e67 0000 0000 0000 0000 0000 0000  .png............
0100  0000 0000 0000 0000 0000 0000 0000 0000  ................
*
0160  0000 0000 0000 0000 0000 0000 3c30 0000  ............<0..
```

Do jakiego pliku należy blok

W tym celu wybierzmy jakiś blok poleceniem `blocks` dla pliku może to być np. ostatni z bloków:

```
<wiele bloków później> 42914
```

Sprawdzając numer i-węzła dla tego bloku poleceniem `icheck` dowiadujemy się, że jest on równy 12. Teraz wystarczy odszukać ten i-węzeł:

```
debugfs:  ncheck 15

Inode Pathname
15 /cats/big_cats.png
```

Reprezentacja liniowa małego katalogu

Blok katalogu:

```
debugfs:  blocks cats
23083
```

Reprezentacja:

```
debugfs:  bdump 23083
0000  0e00 0000 0c00 0102 2e00 0000 0200 0000  ................
0020  0c00 0202 2e2e 0000 0f00 0000 1400 0c01  ................
0040  6269 675f 6361 7473 2e70 6e67 1000 0000  big_cats.png....
0060  c803 0d07 7379 6d62 6f6c 6963 5f63 6174  ....symbolic_cat
0100  7300 0000 0000 0000 0000 0000 0000 0000  s...............
0120  0000 0000 0000 0000 0000 0000 0000 0000  ................
*
1760  0000 0000 0000 0000 0c00 00de 74ce 1fd8  ............t...
```

Zadanie 9

Zadanie 9 (bonus). Na podstawie §3 artykułu A Directory Index for Ext2[3] opisz s
i operację wyszukiwania wpisu katalogu o zadanej nazwie. Następnie wyświetl repre
katalogu, np. `/var/lib/dpkg/info`, używając polecenia `htree` programu `debugfs`

Zadanie 10

Zadanie 10 (bonus). Czym różni się **księgowanie metadanych** od **księgowania**
wydruku polecenia `logdump` programu `debugfs` i opisu struktur dyskowych `ext4`[4] o
Opisz znaczenie poszczególnych bloków z których może składać się pojedyncza **trans**
składowane w dzienniku muszą być **idempotentne**?