Algorytm zwalniania bloku dyskowego ext2_free_blocks (jądro 2.2.17)
Marcin Stefaniak
index:181066
20 grudzień 2000

1 Wstęp

Przed opisem funkcji zwalniania bloku dyskowego w systemie plików Ext2 opiszę pobieżnie sam system i jego struktury niezbędne do zrozumienia algorytmu. W systemie Ext2 partycja dysku podzielona jest na wiele grup bloków. Na początku umieszczony jest blok startowy, a za nim znajdują się kolejne grupy bloków *(Tablica 1)*, przy czym każda grupa bloków zawiera kopię bloku specjalnego (super bloku) oraz bloki struktur i-węzłów i danych.

Grupy *(Tablica 2)* składają się z:

- super bloku
- deskryptora grupy
- bitmapy zajętości bloków danej grupy
- bitmapy zajętości struktur i-węzłów danej grupy
- tablicy struktur i-węzłów
- bloków danych

Tablica 1: Struktura partycji

Blok startowy	Grupa bloków 0	Grupa bloków 1	....	Grupa bloków n

Tablica 2: Grupy bloków

super blok	deskryptor grupy	bitmapy zajętości bloków danej grupy	bitmapy zajętości struktur i-węzłów danej grupy	tablica struktur i-węzłów	bloki danych

Fizyczny **super blok** zdefiniowany jest jako struktura *ext2_super_block*, zawiera ona między innymi:

- informacje kontrolne dotyczące systemu plików takie jak liczba struktur i-węzłów i bloków (w tym liczbę wolnych struktur)

- liczba struktur i-węzłów i bloków w grupie wraz z czasami ostatniej modyfikacji (w tym liczbę wolnych struktur)
- pozycja pierwszego bloku danych
- liczba bloków

Deskryptor udostępnia informacje o:

- bitmapie bloku
- bitmapie struktur i-węzłów
- tablicy struktur i-węzłów
- liczbie wolnych bloków w grupie
- liczbie wolnych struktur i-węzłów
- liczbie katalogów

Bitmapa zajętości bloku jest wielkości jednego bloku i zawiera bity mówiące czy dany blok (odpowiadający bitowi) jest zajęty czy wolny (podobnie jest z **bitmapą struktur i-węzłów**)

Tablica i-węzłów składa się z występujących po sobie bloków zawierających struktury i-węzłów (struktura i-węzła zawiera informacje o pliku oraz o blokach w których są zapisane dane z tego pliku)

Bloki danych są to kolejne bloki w których zapisane są dane odpowiednich plików.

2 Algorytm

2.1 Opis

Funkcja *ext_2_free_blocks* jest zdefiniowana w pliku *fs/ext2/balloc.c* Służy ona (jak sama nazwa wskazuje) do zwalniania bloków dyskowych w systemie plików ext2. Funkcja ta jest wywoływana, podczas kasowania plików oraz podczas zmiany (zmniejszenia) wielkości pliku. Wywoływana jest z:

- wskaźnikiem do i-węzła pliku, którego bloki ma zwolnić
- numerem (bezwzględnym) bloku od którego ma zacząć zwalniać
- liczbą bloków do zwolnienia.

Ze struktury i-węzła, którego wskaźnik dostaje jako parametr ustala wskaźnik do super bloku. Następnie zakłada blokadę na super bloku, aby zapewnić sobie wyłączność dostępu do niego. Sprawdza, czy bloki, które ma zwolnić, znajdują się w strefie bloków z danymi (czy znajdują się pomiędzy pierwszym, a ostatnim blokiem danych - wartości te są zapisane w każdym super bloku grupy).

Na podstawie numeru pierwszego bloku do zwolnienia (drugiego parametru funkcji) ustala numer grupy bloków. Później jeszcze raz sprawdzana jest poprawność argumentów wejściowych tj. czy bloki do zwolnienia znajdują się w jednej grupie. Następnie na podstawie super bloku i obliczonego wcześniej numeru grupy wczytuje bitmapę bloków grupy oraz ustala deskryptor grupy. Teraz następuje kolejne sprawdzenie bloków do zwolnienia, tym razem czy nie znajdują się w strefie bloków systemowych i jeśli to nie spowoduje przerwania funkcji to przechodzi ona do zwalniania bloków. Zaczynając od pierwszego zwalnia kolejne bloki następująco: sprawdza czy w mapie bitowej bloków dany bit odpowiadający zwalnianemu blokowi jest zajęty (czyli czy blok jest zajęty)

- jeśli tak to ustawia go na wolny (zmienia bit) aktualizując odpowiednie struktury: zwalnia bloki w i-węźle pliku, zwiększa o jeden liczbę wolnych bloków w super bloku grupy i deskryptorze grupy.
- jeśli nie to przechodzi do kolejnego bloku.

Następnie zaznacza wszystkie bufory, które zmieniła jako "brudne"(bufor z deskryptorem, super blokiem i bitmapą) oraz zaznacza, że zmieniła super blok. Kończąc funkcja zwalnia blokadę z super bloku.

2.2 Kod

Dokładniej można prześledzić algorytm na podstawie kodu funkcji:

```
void ext2_free_blocks (const struct inode * inode, unsigned long block,
     unsigned long count)
{
```

inode - wskaźnik do i-węzła którego bloki chcemy zwolnić
block - bezwzględny numer bloku, od którego chcemy zacząć zwalnianie
count - ilość bloków do zwolnienia
wskaźnik do bufora z bitmapą

```
      struct buffer_head * bh;
```

wskaźnik do bufora z deskryptorem grupy

```
      struct buffer_head * bh2;
```

numer grupy bloków

```
      unsigned long block_group;
```

numer bloku w grupie

```
      unsigned long bit;
      unsigned long i;
```

numer bitmapy w grupy w której bloki chcemy zwolnić

```
      int bitmap_nr;
      unsigned long overflow;
```

super blok w pamięci

```
        struct super_block * sb;
```

deskryptor grupy

```
        struct ext2_group_desc * gdp;
```

super blok grupy

```
        struct ext2_super_block * es;
```

ustalamy super blok, pobierając wskaźnik z przekazanego jako parametr i-węzła

```
        sb = inode->i_sb;
        if (!sb) {
                printk ("ext2_free_blocks: nonexistent device");
                return;
        }
```

zakładamy blokadę na super bloku

```
        lock_super (sb);
```

ustalamy wskaźnik do super bloku grupy

```
        es = sb->u.ext2_sb.s_es;
```

sprawdzamy czy bloki, które mamy zwolnić znajdują się w strefie bloków z danymi

```
        if (block < le32_to_cpu(es->s_first_data_block) ||
            (block + count) > le32_to_cpu(es->s_blocks_count)) {
                ext2_error (sb, "ext2_free_blocks",
                            "Freeing blocks not in datazone - "
                            "block = %lu, count = %lu", block, count);
                goto error_return;
        }

        ext2_debug ("freeing block %lu\n", block);

do_more:

        overflow = 0;
```

ustalamy numer grupy bloków do zwolnienia

```
        block_group = (block - le32_to_cpu(es->s_first_data_block)) /
                      EXT2_BLOCKS_PER_GROUP(sb);
```

ustalamy numer pierwszego bloku do zwolnienia względem grupy

```
        bit = (block - le32_to_cpu(es->s_first_data_block)) %
                      EXT2_BLOCKS_PER_GROUP(sb);
```

sprawdzamy, czy zwalniane bloki znajdują się w jednej grupie bloków

```
if (bit + count > EXT2_BLOCKS_PER_GROUP(sb)) {
        overflow = bit + count - EXT2_BLOCKS_PER_GROUP(sb);
        count -= overflow;
}
```

wczytujemy bitmapę bloków grupy

```
bitmap_nr = load_block_bitmap (sb, block_group);
if (bitmap_nr < 0)
        goto error_return;

bh = sb->u.ext2_sb.s_block_bitmap[bitmap_nr];
```

ustalamy deskryptor grupy i wskaźnik do bufora, który go zawiera

```
gdp = ext2_get_group_desc (sb, block_group, &bh2);
if (!gdp)
        goto error_return;
```

sprawdzamy czy zwalniane bloki nie znajdują się w strefie bloków systemowych

```
if (test_opt (sb, CHECK_STRICT) &&
    (in_range (le32_to_cpu(gdp->bg_block_bitmap), block, count) ||
     in_range (le32_to_cpu(gdp->bg_inode_bitmap), block, count) ||
     in_range (block, le32_to_cpu(gdp->bg_inode_table),
               sb->u.ext2_sb.s_itb_per_group) ||
     in_range (block + count - 1, le32_to_cpu(gdp->bg_inode_table),
               sb->u.ext2_sb.s_itb_per_group)))
        ext2_panic (sb, "ext2_free_blocks",
                    "Freeing blocks in system zones - "
                    "Block = %lu, count = %lu",
                    block, count);
```

przechodzimy kolejne bity bitmapy odpowiadające kolejnym zwalnianym blokom i sprawdzamy czy są zajęte

- *jeśli nie są zajęte to nie musimy nic aktualizować*
- *jeśli są zajęte to je zwalniamy aktualizując odpowiednie struktury: zwalniamy blok w i-węźle (DQUOT_FREE_BLOCK), podnosimy w deskryptorze grupy oraz super bloku licznik wolnych bloków*

```
for (i = 0; i < count; i++) {
        if (!ext2_clear_bit (bit + i, bh->b_data))
                ext2_warning (sb, "ext2_free_blocks",
                              "bit already cleared for block %lu",
                              block);
        else {
```

```
                        DQUOT_FREE_BLOCK(sb, inode, 1);
                        gdp->bg_free_blocks_count =
                                cpu_to_le16(le16_to_cpu(gdp->bg_free_blocks_count)+1);
                        es->s_free_blocks_count =
                                cpu_to_le32(le32_to_cpu(es->s_free_blocks_count)+1);
                }
        }
```

zaznaczamy jako 'brudne' bufory w których dokonaliśmy zmian (funkcja mark_ buffer_ dirty ustala odpowiedni bit w nagłówku bufora) i rozpoczynamy transmisję (funkcja 'll_ rw_ block'), w miarę możliwości asynchroniczną.

bufor z deskryptorem

```
        mark_buffer_dirty(bh2, 1);
```

bufor zawierający super blok grupy

```
        mark_buffer_dirty(sb->u.ext2_sb.s_sbh, 1);
```

bufor z bitmapą

```
        mark_buffer_dirty(bh, 1);
```

Jeżeli transmisja jest synchroniczna, czekamy na jej zakończenie (funkcja "wait_ on_ buffer").

```
        if (sb->s_flags & MS_SYNCHRONOUS) {
                ll_rw_block (WRITE, 1, &bh);
                wait_on_buffer (bh);
        }

        if (overflow) {
                block += count;
                count = overflow;
                goto do_more;
        }
```

zaznaczamy, że zmieniliśmy super blok

```
        sb->s_dirt = 1;

error_return:
```

zwalniamy blokadę super bloku

```
        unlock_super (sb);
        return;
}
```

Bibliografia

[1] Pliki źródłowe Linuxa (na stronie http://lxr.linux.no)

[2] Projekt Linux

[3] M.Beck - 'Linux Kernel Jądro Systemu '