Rozdział 1

Algorytm zwalniania i-węzła na dysku ifree

Paweł Dariusz Mogielnicki

Algorytm zwalniania i-węzła na dysku jest używany do zaznaczenia i-węzła na dysku jako nieużywanego. Jest on stosowany przez algorytm iput (jeżeli kasowane jest ostatnie dowiązanie tego i-węzła). Najkrócej rzecz biorąc algorytm zwiększa liczbę wolnych i-węzłów w systemie plików. Dalej rozpatruje dwa warianty. Jeżeli blok identyfikacyjny jest zablokowany, to algorytm kończy działanie. W przeciwnym razie jądro dopisuj zwolniony numer i-węzła do listy wolnych i-węzłów, chyba że na liście nie ma wolnego miejsca. W takim przypadku porównuje numer zwalnianego i-węzła z największym numerem i-węzła na liście (ten numer jest zapisany w wyszczególnionym polu, więc dostęp do niego jest błyskawiczny). Jeżeli zwalniany i-węzeł ma mniejszy numer to jest wpisywany na miejsce dotychczasowego i-węzła o maksymalnym numerze. W ten sposób zachowana jest zasada, że wszystkie wolne i-węzły o numerach mniejszych od największego numeru spośród i-węzłów na liście też są na tej liście. Dzięki temu jeżeli zabraknie wolnych i-węzłów na liście wystarczy przeszukać system plików w poszukiwaniu tylko tych ewentualnie wolnych i-węzłów o numerach wyższych od maksymalnego na liście.

1.1 Implementacja algorytmu w jądrze linuxa 2.2.17 na podstawie systemu plików ext2

Poniżej przedstawiam sposób implementacji interesującego nas algorytmu, jaki zastosowano w jądrze linuxa w wersji 2.2.17. Warto zauważyć, że obecna implementacja zawiera jedynie kosmetyczne zmiany w stosunku do implementacji z jądra 1.0.9 (napisanej w 1993 r. przez Remy Carda (card@masi.ibp.fr, z Laboratoire MASI - Institut Blaise Pascal na Uniwersytecie Pierre'a i Marie Curie w Paryżu) i Stephena Tweedie (sct@dcs.ed.ac.uk). Obecnie obowiązująca implementacja została po raz pierwszy zastosowana w 1995 r. w jądrze 2.2.15, a zawiera poprawki wprowadzone przez Davida S. Millera (davem@caip.rutgers.edu). Ten sam kod został zastosowany w beta-wersjach jądra 2.4.0.

1.1.1 Struktury

W tym rozdziale przedstawiam jedynie struktury przydatne do analizy kodu samej funkcji zwalniającej i-węzeł.

Struktura i-węzła

Deklaracja struktury inode występuje w pliku include/linux/fs.h w linii 335. Oto najważniejsze (wybrane!) pola tej struktury, które będą dla nas interesujące w kontekście omawianego algorytmu.

```
struct inode {
        unsigned long           i_ino;
```

W tym elemencie będzie przechowywany numer i-węzła.

```
        unsigned int            i_count;
        umode_t                 i_mode;
```

W tym elemencie będzie przechowywana informacja o tym, czy i-węzeł należy do katalogu czy do pliku.

```
        nlink_t                 i_nlink;
```

W tym elemencie będzie przechowywana liczba dowiązań i-węzła.

```
        struct super_block      *i_sb;
```

W tym elemencie będzie przechowywana informacja nt. superbloku, do którego należy i-węzeł.

```
};
```

Struktura superbloku

Deklaracja struktury super_block występuje w pliku include/linux/fs.h w linii 521.

```
struct super_block {
        unsigned char           s_dirt;
```

W tym elemencie będzie przechowywana informacja, czy superblok był modyfikowany (tzw. bit zabrudzenia).

```
        unsigned long           s_flags;
        union {
                struct ext2_sb_info     ext2_sb;
```

W tym elemencie będą przechowywane różne informacje o superbloku. Szczegóły poniżej.

```
        } u;
};
```

Struktura deskryptora grupy

Deklaracja struktury ext2_group_desc występuje w pliku include/linux/ext2_fs.h w linii 148.

```
struct ext2_group_desc
{
        __u16   bg_free_inodes_count;   /* Free inodes count */
        __u16   bg_used_dirs_count;     /* Directories count */
```

W tych elementach będą przechowywane informacje o ilości wolnych i-węzłów i zajętych katalogów w deskryptorze grupy.

```
};
```

Struktura superbloku ext2

Deklaracja struktury ext2_super_block występuje w pliku include/linux/ext2_fs.h w linii 335.

```
struct ext2_super_block {
        __u32   s_inodes_count;         /* Inodes count */
        __u32   s_free_inodes_count;    /* Free inodes count */
```

W tych elementach będą przechowywane informacje o ilości i-węzłów w superbloku i ilości wolnych i-węzłów w tymże.

```
};
```

Struktura informacji o superbloku ext2

Deklaracja struktury ext2_sb_info występuje w pliku include/linux/ext2_fs_sb.h w linii 32.

```
struct ext2_sb_info {
        struct buffer_head * s_sbh;     /* Buffer containing the super block */
```

W tym elemencie będzie przechowywany wskaźnik do bufora, w którym znajduje się superblok. Potrzebujemy tej informacji, żeby po modyfikacji superbloku móc łatwo zaznaczyć modyfikację bufora, w którym on się znajduje.

```
        struct ext2_super_block * s_es; /* Pointer to the super block in the buffer */
        struct buffer_head * s_inode_bitmap[EXT2_MAX_GROUP_LOADED];
```

W tym elemencie będą przechowywane wskaźniki do buforów w których znajdują się poszczególne mapy bitowe zajętości i-węzłów w grupach.

```
};
```

1.1.2 Funkcja ext2_free_inode

Deklaracja funkcji ext2_free_inode występuje w pliku fs/ext2/ialloc.c w linii 180. Jej prototyp znajduje się w pliku include/linux/ext2_fs.h w linii 555.

```
/*
 * NOTE! When we get the inode, we're the only people
 * that have access to it, and as such there are no
 * race conditions we have to worry about. The inode
 * is not on the hash-lists, and it cannot be reached
 * through the filesystem because the directory entry
 * has been deleted earlier.
 *
 * HOWEVER: we must make sure that we get no aliases,
 * which means that we have to call "clear_inode()"
 * _before_ we mark the inode not in use in the inode
 * bitmaps. Otherwise a newly created file might use
 * the same inode number (not actually the same pointer
 * though), and then we'd have two inodes sharing the
 * same inode number and space on the harddisk.
 */
void ext2_free_inode (struct inode * inode)
{
        struct super_block * sb = inode->i_sb;
```

W tej zmiennej będzie przechowywany wskaźnik do superbloku właściwego dla zwalnianego i-węzła.

```
        int is_directory;
```

W tej zmiennej będzie informacja, czy zwalniany i-węzeł należy do katalogu.

```
unsigned long ino;
```

W tej zmiennej będzie przechowywany numer i-węzła.

```
struct buffer_head * bh;
```

W tej zmiennej będzie przechowywany bufor z mapą zajętości i-węzłów grupie.

```
struct buffer_head * bh2;
```

W tej zmiennej będzie przechowywany bufor z informacjami o danej grupie i-węzłów.

```
unsigned long block_group;
```

W tej zmiennej będzie przechowywany numer grupy i-węzłów.

```
unsigned long bit;
```

W tej zmiennej będzie przechowywany numer (pozycja) i-węzła w grupie.

```
int bitmap_nr;
```

W tej zmiennej będzie przechowywana pozycja mapy bitowej zajętości i-węzłów w grupie w tablicy bitmap dla danego superbloku.

```
struct ext2_group_desc * gdp;
```

W tej zmiennej będzie przechowywany deskryptor grupy, a więc właściwe informacje o danej grupie i-węzłów.

```
struct ext2_super_block * es;
```

W tej zmiennej będzie przechowywany wskaźnik do superbloku.

```
        if (!inode->i_dev) {
                printk ("ext2_free_inode: inode has no device\n");
                return;
        }
        if (inode->i_count > 1) {
                printk ("ext2_free_inode: inode has count=%d\n", inode->i_count);
                return;
        }
        if (inode->i_nlink) {
                printk ("ext2_free_inode: inode has nlink=%d\n",
                        inode->i_nlink);
                return;
        }
        if (!sb) {
                printk("ext2_free_inode: inode on nonexistent device\n");
                return;
        }
```

Analiza ewentualnych sytuacji błędnych.

```
        ino = inode->i_ino;
        ext2_debug ("freeing inode %lu\n", ino);
```

Pobranie numeru i-węzła.

```
        /*
         * Note: we must free any quota before locking the superblock,
         * as writing the quota to disk may need the lock as well.
         */
        DQUOT_FREE_INODE(sb, inode);
        DQUOT_DROP(inode);

        lock_super (sb);
```

Tu następuje blokada superbloku.

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

Wskaźnik do superbloku w buforze.

```
        if (ino < EXT2_FIRST_INO(sb) ||
            ino > le32_to_cpu(es->s_inodes_count)) {
```

Funkcja le32_to_cpu odwraca kolejność bajtów w liczbie.

```
                ext2_error (sb, "free_inode",
                            "reserved inode or nonexistent inode");
                goto error_return;
```

Próba zwolnienia i-węzła o numerze nie znajdującym się w dowzolonym przedziale.

```
        }
        block_group = (ino - 1) / EXT2_INODES_PER_GROUP(sb);
```

Wylicza numer grupy, do której należy zwalnianiy i-węzeł.

```
        bit = (ino - 1) % EXT2_INODES_PER_GROUP(sb);
```

Wylicza numer (pozycję) i-węzła w tej grupie.

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

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

Wczytuje mapę bitową zajętości i-węzłów w owej grupie do bufora bh.

```
        is_directory = S_ISDIR(inode->i_mode);

        /* Do this BEFORE marking the inode not in use */
        clear_inode (inode);
```

Wyczyszczenia i-węzła należy dokonać przed odznaczeniem jego zajętości, bowiem w przeciwnym razie inny proces mógłby użyć tego i-węzła.

```
        /* Ok, now we can actually update the inode bitmaps.. */
        if (!ext2_clear_bit (bit, bh->b_data))
```

Czyści wskaźnik do danych.

```
                ext2_warning (sb, "ext2_free_inode",
                              "bit already cleared for inode %lu", ino);
        else {
                gdp = ext2_get_group_desc (sb, block_group, &bh2);
```

Powyższa funkcja zwraca wskaźnik do danych o grupie zawartych w buforze, zaś wskaźnik do tego bufora ustawia w zmiennej bh2. Żądany bufor odnajdujemy obliczając numer deskryptora grupy i pozycję w tym deskryptorze właściwą dla podanej grupy i odwołując się do informacji zawartej w suberbloku.

```
                if (gdp) {
                        gdp->bg_free_inodes_count =
                                cpu_to_le16(le16_to_cpu(gdp->bg_free_inodes_count) + 1);
                        if (is_directory)
                                gdp->bg_used_dirs_count =
                                        cpu_to_le16(le16_to_cpu(gdp->bg_used_dirs_count) - 1);
                }
```

Jeżeli wskaźnik ten nie był wcześniej wyczyszczony, to w deskryptorze grupy zwiększa liczbę wolnych i-węzłów o jeden i w razie gdyby zwalniany i-węzeł należał do katalogu zmniejsza o jeden liczbę używanych katalogów w tym deskryptorze grupy.

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

Zaznacza bit modyfikacji dla nagłówka bufora, w którym jest zapisany deskryptor grupy, w której dokonywaliśmy zmian.

```
                es->s_free_inodes_count =
                        cpu_to_le32(le32_to_cpu(es->s_free_inodes_count) + 1);
```

Zwiększa liczbę wolnych i-węzłów o jeden w superbloku.

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

Zaznacza bit modyfikacji dla nagłówka bufora zawierającego superblok.

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

Zaznacza bit modyfikacji dla nagłówka bufora, w którym zawarta była modyfikowana mapa bitowa zajętości i-węzłów w grupie.

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

Zaznacza modyfikację całego superbloku.

```
error_return:
        unlock_super (sb);
```

Odblokowuje superblok.

```
}
```

1.2 Zastosowanie algorytmu

Algorytm ifree jest wykorzystywany w jądrze linuxa tylko przez algorytm iput, jeżeli kasowane jest ostatnie dowiązanie danego i-węzła. W omawianym przez nas systemie plików ext2 jako operacja kasowania i-węzła (w superoperacjach dla systemu plików) jest ustawiona (w pliku fs/ext2/super.c) funkcja ext2_delete_inode, która wywołuje (w pliku fs/ext2/inode.c) w odpowiednim przypadku funkcję ext2_free_inode czyli właściwy algorytm ifree.

Jedynym zaś miejscem wywołania superoperacji delete_inode (w naszym przypadku ext2_delete_inode) jest funkcja iput() w pliku fs/inode.c:

```
if (op && op->delete_inode) {
        void (*delete)(struct inode *) = op->delete_inode;
        spin_unlock(&inode_lock);
        delete(inode);
        spin_lock(&inode_lock);
}
```

1.3 Bibliografia

- http://rainbow.mimuw.edu.pl/SO/LinuxPrezentacje/PLIKI/P1/sld021.htm
- http://lxr.linux.no
- Maurice J. Bach „Budowa systemu operacyjnego UNIX"
- http://www.mogiel.net/ifree.ps
- http://www.mogiel.net/ifree-zrodla.ps
- źródła linuxa (wersja jądra 2.2.17) - pliki:
  - include/linux/fs.h
  - include/linux/ext2_fs.h
  - include/linux/ext2_fs_sb.h
  - fs/ext2/ialloc.c
  - fs/ext2/inode.c
  - fs/ext2/super.c
  - fs/inode.c

Przydatne deklaracje i wystąpienia struktur i funkcji w kodzie jądra linuxa w wersji 2.2.17:

- Deklaracje struktur
  - inode (linia 335, include/linux/fs.h)
  - super_block (linia 521, include/linux/fs.h)
  - buffer_head (linia 201, include/linux/fs.h)
  - ext2_group_desc (linia 148, include/linux/ext2_fs.h)
  - ext2_super_block (linia 335, include/linux/ext2_fs.h)
  - ext2_sb_info (linia 32, include/linux/ext2_fs_sb.h)
- Deklaracje funkcji
  - ext2_free_inode (linia 180, fs/ext2/ialloc.c)
  - ext2_delete_inode (linia 48, fs/ext2/inode.c)
- Wystąpienia funkcji
  - ext2_free_inode (linia 63, fs/ext2/inode.c)
  - ext2_delete_inode (linia 136, fs/ext2/super.c)
  - delete_inode (linie 777 i 778, fs/inode.c)