Obsługa nagłówków buforów, list wolnych buforów i list LRU

Grzegorz Kozakiewicz

6 grudnia 2000

1 Wstęp

Podstawowe informacje o podręcznej pamięci buforowej linuksa:

- podręczna pamięć buforowa (ang. *buffer cache*) linuksa służy do zwiększenia wydajności operacji wejścia/wyjścia na urządzeniach blokowych (w szczególności dotyczy to twardych dysków)
- wybrane bloki urządzeń blokowych są odwzorowane w pewien fragment pamięci RAM, który to fragment nazywamy buforem
- dzięki użyciu podręcznej pamięci buforowej uzyskujemy szybsze czytanie i pisanie bloków, ponieważ niektóre odwołania do urządzeń blokowych mogą zostać wyeliminowane
- niestety buforowanie zapisu wnosi pewną wadę: gdy zostaje odcięty dopływ prądu, tracimy niezapisane dane
- podręczna pamięć buforowa jest wykorzystywana przez wiele funkcji systemowych (np. *read()*, *write()*) zapewniając ich izolację od rzeczywistej implementacji odczytu-zapisu dla urządzenia blokowego
- istnieje specjalny demon *bdflush*, który zajmuje się zapisywaniem zabrudzonych buforów
- pamięć buforowa jest globalna (tzn. wspólna dla wszystkich urządzeń blokowych)
- dostępne są różne rozmiary buforów

2 Nagłówki buforów

Nagłówek bufora (ang. *buffer head*) jest to struktura danych opisująca aktualny stan bufora. Struktura nagłówka bufora jest zdefiniowana w pliku *fs.h* [1].

```
struct buffer_head {
    struct buffer_head * b_next;
    unsigned long b_blocknr;
    unsigned long b_size;
    kdev_t b_dev;
    kdev_t b_rdev;
    unsigned long b_rsector;
    struct buffer_head * b_this_page;
    unsigned long b_state;
    struct buffer_head * b_next_free;
    unsigned int b_count;

    char * b_data;
    unsigned int b_list;
    unsigned long b_flushtime;
    struct wait_queue * b_wait;
    struct buffer_head ** b_pprev;
    struct buffer_head * b_prev_free;
    struct buffer_head * b_reqnext;

    void (*b_end_io)(struct buffer_head *bh, int uptodate);
    void *b_dev_id;
};
```

Flagi statusu bufora są zdefiniowane w następujący sposób (druga kolumna jest numerem bitu):

```
#define BH_Uptodate     0
#define BH_Dirty        1
#define BH_Lock         2
#define BH_Req          3
#define BH_Protected    6
```

A oto opis poszczególnych pól (kolejność jest taka jak w definicji struktury):

b_next wskaźnik do następnego nagłówka bufora w kolejce mieszającej

b_blocknr numer bloku

b_size rozmiar bloku

b_dev numer urządzenia (*B_FREE* jeśli bufor jest wolny)

b_rdev numer rzeczywistego urządzenia

b_rsector numer rzeczywistego bloku (numer sektora)

b_this_page cykliczna lista buforów na jednej stronie pamięci

b_state flagi statusu bufora

- **BH_Uptodate** bufor zawiera aktualne dane
- **BH_Dirty** bufor jest zabrudzony
- **BH_Lock** bufor jest zablokowany (trwa na nim operacja wejścia/wyjścia)
- **BH_Req** dane w buforze nie zostały unieważnione
- **BH_Protected** ten bufor jest chroniony

b_next_free wskaźnik do następnego nagłówka bufora w liście wolnych buforów lub LRU (obie listy używają tego samego pola

b_count liczba użytkowników korzystajżcych z tego bloku

b_data wskaźnik do bloku danych

b_list numer listy LRU na której znajduje się ten bufor

b_flushtime (dla zabrudzonych buforów) czas po którym bufor powinien zostać zapisany na dysk

b_wait kolejka procesów czekających na zwolnienie bufora

b_pprev wskaźnik do poprzedniego nagłówka bufora w kolejce mieszającej (właściwie to nie do samego nagłówka, tylko do jego pola *b_next*

b_prev_free wskaźnik do poprzedniego nagłówka bufora w liście wolnych buforów lub LRU (obie listy używają tego samego pola

b_reqnext wskaźnik do następnego nagłówka bufora w strukturze request

b_end_io używane do kończenia operacji wejścia/wyjścia

b_dev_id używane do kończenia operacji wejścia/wyjścia

Podsumowując, w nagłówku bufora przechowywana jest:

- lokalizacja bufora (*b_ dev, b_ blocknr, b_ rdev, b_ rsector*)
- rozmiar bufora (*b_ size*)
- informacje o lokalizacji w poszczególnych strukturach danych
  - wskaźniki używane do połączenia buforów w listy
    - kolejki mieszające (*b_ next, b_ pprev*)
    - listy wolnych buforów i LRU (*b_ next_free, b_ prev_free*)
    - lista buforów na tej samej stronie pamięci (*b_ this_page*)
    - lista buforów w strukturze request (*b_ reqnext*)
  - numer listy LRU (*b_ list*)
- wskaźnik na blok danych (*b_ data*)
- informacje dotyczące aktualnego stanu bufora (flagi statusu *b_ state, b_ count, b_ flushtime*)
- dowiązanie do kolejki procesów czekających (*b_ wait*)
- informacje używane do kończenia operacji wejścia/wyjścia (*b_ end_ io, b_ dev_ id*)

Struktura jest zoptymalizowana pod względem wydajności wyszukiwania (i dlatego kolejność pól w definicji struktury na pierwszy rzut oka wydaje się nielogiczna).

3 Kolejki mieszające

Problem: Jak znaleźć nagłówek bufora przyporządkowany określonemu urządzeniu i blokowi (lub stwierdzić jego brak)?

Rozwiązanie: Problem ten jest rozwiązany za pomocą tablicy kolejek mieszających. Jest to implementacja słownika przez mieszanie z problemem kolizji rozwiązanym metodą łańcuchową - listy będące łańcuchami to właśnie wspomniane kolejki mieszające.

Deklaracja tablicy kolejek mieszających (z pliku *buffer.c* [2]):

```
static struct buffer_head ** hash_table = NULL;
```

Pamięć jest alokowana przy inicjalizacji po ustaleniu wielkości tablicy. Pojedyncza kolejka mieszająca to dwukierunkowa (właściwie to prawie dwukierunkowa - nie ma wskaźnika do poprzedniego nagłówka tylko wskaźnik do pola *b_ next* tego nagłówka) lista prosta, wykorzystująca pola *b_ next* i *b_ pprev* nagłówka bufora. Bardzo ciekawa jest funkcja mieszająca:

```
#define _hashfn(dev,block) \
        ((((dev)<<(bh_hash_shift - 6)) ^ ((dev)<<(bh_hash_shift - 9))) ^ \
        (((block)<<(bh_hash_shift - 6)) ^ ((block) >> 13) ^ ((block) << \
        (bh_hash_shift - 12))))
```

Istnieje także makro zwracające dla zadanej pary (nr_urządzenia, nr_bloku) odpowiednią kolejkę mieszającą:

```
#define hash(dev,block) hash_table[_hashfn(dev,block) & bh_hash_mask]
```

4 Listy wolnych buforów

Bufory wolne (tzn. takie którym aktualnie nie jest przyporządkowany żaden blok) przetrzymywane są na listach buforów wolnych. Są to listy cykliczne i dwukierunkowe, wykorzystujące składowe *b_next_free* i *b_prev_free* nagłówka bufora. Jest jedna taka lista dla każdego możliwego rozmiaru bufora. Definicja odpowiedniej struktuty znajduje się w pliku *buffer.c* [2]:

```
#define NR_SIZES 7
static struct buffer_head * free_list[NR_SIZES] = {NULL, };
```

Jak widać aktualnie obsługiwanych jest 7 rozmiarów. Rozmiary te to kolejne potęgi dwójki od 512 do 32768. Istnieje nawet makro zamieniające rozmiar na właściwy indeks - nazywa się *BUFSIZE_INDEX*.

5 Listy LRU

Używane aktualnie bufory przechowywane są w listach LRU. Listy te stanowią implementację algorytmu LRU co pozwala na wybierać odpowiednie bufory do usunięcia gdy zachodzi taka potrzeba (np. potrzebne jest więcej pamięci). Definicja list LRU znajduje się w pliku *buffer.c* [2]:

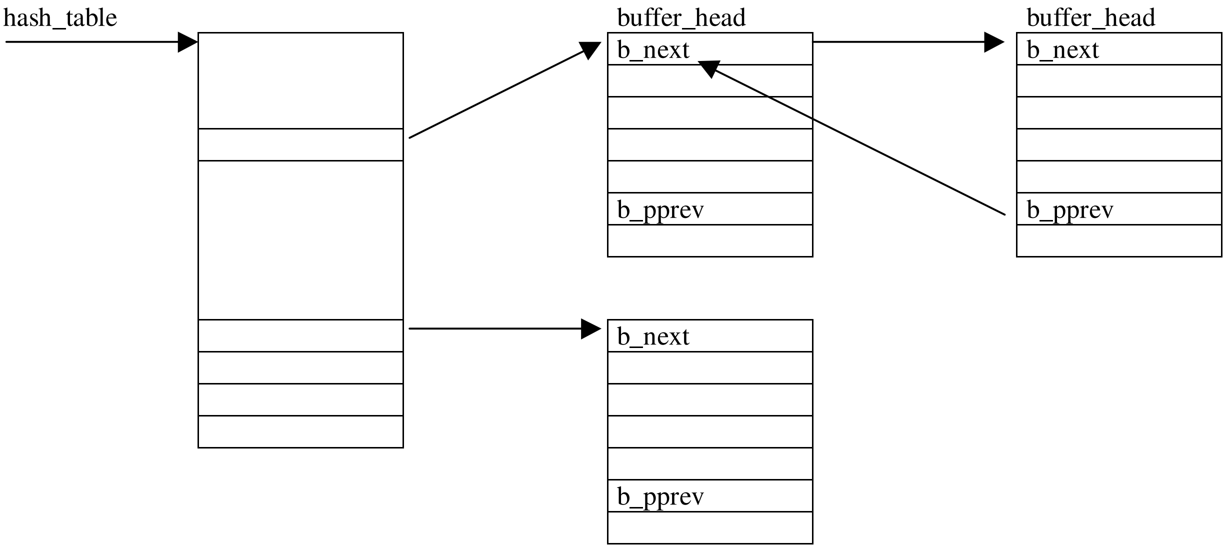
```
static struct buffer_head * lru_list[NR_LIST] = {NULL, };
```

natomiast pozostałe makra dotyczące list LRU są w pliku *fs.h* [1]:

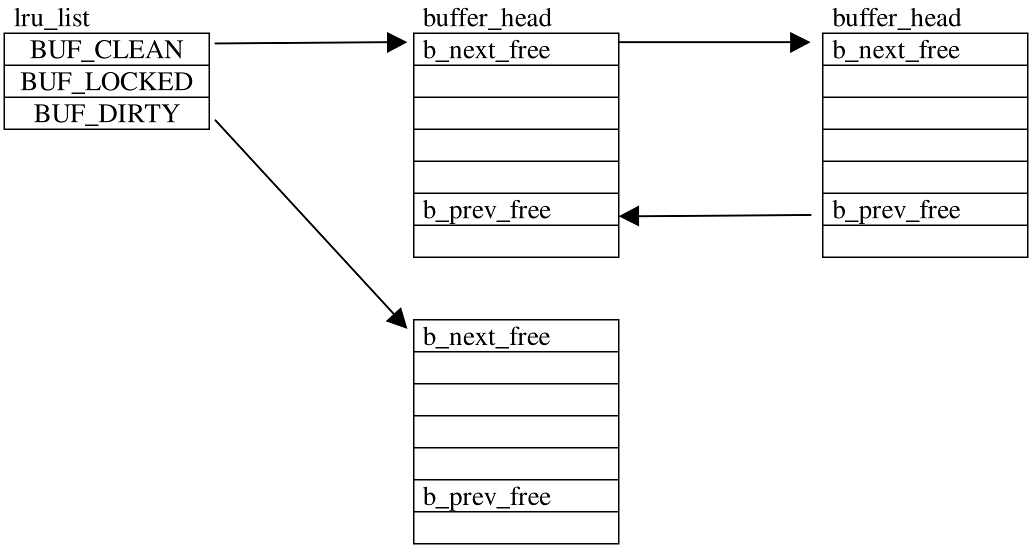
```
#define BUF_CLEAN      0        /* bufory czyste */
#define BUF_LOCKED     1        /* bufory zablokowane do zapisu */
#define BUF_DIRTY      2        /* bufory zabrudzone,
                                   które trzeba będzie kiedyś zapisać */
#define NR_LIST        3
```

Pojedyncza lista LRU ma budowę analogiczną do listy wolnych buforów, tzn. jest cykliczna i dwukierunkowa, oraz wykorzystuje składowe *b_next_free* i *b_prev_free* nagłówka bufora.

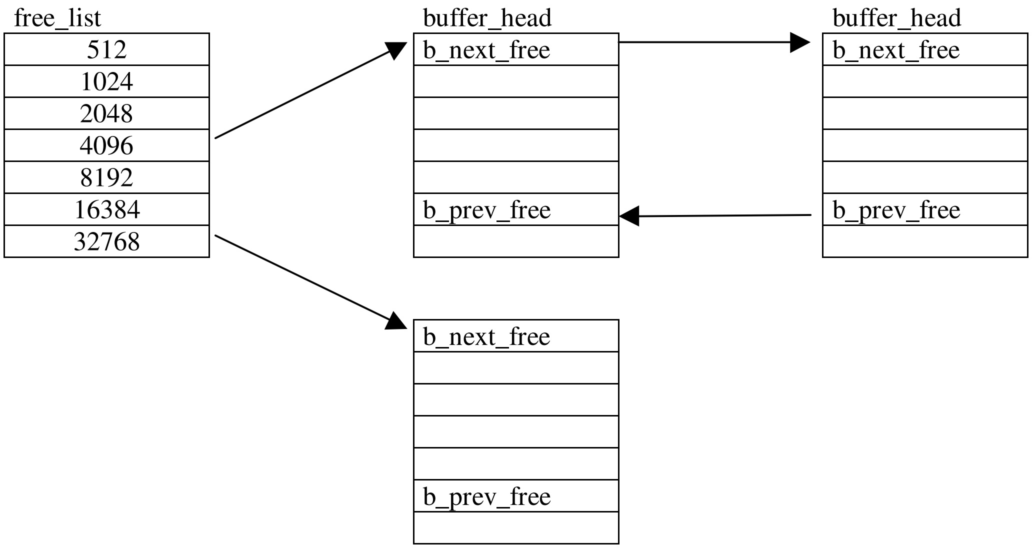
6 Porównanie struktur kolejek mieszających, list LRU i list wolnych buforów



Rysunek 1: Struktura kolejek mieszających



Rysunek 2: Struktura list LRU



Rysunek 3: Struktura list wolnych buforów

7 Informacje uzupełniające

- Jak widać wszystkie występujące tu listy (kolejki mieszające, listy buforów wolnych i LRU) są podwójne. Dzięki temu istnieje możliwość łatwego (w czasie stałym) usunięcia danego nagłówka bufora z dowolnej z tych list.
- Można wymusić zapis wszystkich zabrudzonych buforów danego urządzenia blokowego. Służą do tego funkcje *sync_dev()* i *fsync_dev()* (obie funkcje mają jeden parametr - numer urządzenia). Funkcja *fsync_dev()* czeka aż zapis się zakończy a *sync_dev()* nie czeka (tylko inicjuje).
- Oczywiście w implementacji pamięci buforowej (plik *buffer.c* [2]) jest spora liczba funkcji wykonujących typowe czynności na omawianych strukturach danych, np:

```
static inline void remove_from_hash_queue(struct buffer_head * bh)
static inline void remove_from_lru_list(struct buffer_head * bh)
static inline void remove_from_free_list(struct buffer_head * bh)
static void remove_from_queues(struct buffer_head * bh)
static void put_last_free(struct buffer_head * bh)
static void insert_into_queues(struct buffer_head * bh)
struct buffer_head * find_buffer(kdev_t dev, int block, int size)
```

- W jądrze 2.2.17 wprowadzono wiele zmian (dotyczacych przede wszystkim wydajnośći) w stosunku do wersji poprzednich, dlatego tez czytając dokumentację z poprzednich lat [3] należy sprawdzać w kodzie czy coś się nie zmieniło.
- Pozostałe tematy projektowe związane z pamięcią buforową to:
  - Algorytm czytania bloku *bread*
  - Algorytm czytania bloku z wyprzedzeniem *breada*
  - Algorytm znajdowania bufora z blokiem *getblk*
  - Algorytm zwalniania bufora z blokiem *brelse*
  - Administrowanie rozmiarem pamięci buforowej
  - Algorytm odzyskiwania wolnych buforów *refill_free_list*
  - Działanie demona *bd_flush*

Bibliografia

[1] plik źrodłowy jądra */include/linux/fs.h*

[2] plik źrodłowy jądra */include/fs/buffer.c*

[3] projekty z lat poprzednich