Rozdział 1

Zwalnianie pamięci przez jądro - kfree

autor: **Jakub Królikowski** nr indeksu: 181057

Rozdział opisuje mechanizm zwalniania pamięci operacyjnej widzianej przez jądro, a także struktury danych, które służą do operacji na pamięci.

1.1 Wprowadzenie

Mechanizm przydzielania pamięci dla jądra 2.2.17 jest rozwiązaniem zaproponowanym najprawdopodobniej przez Sun Microsystems w systemie SunOS. Implementacji tej idei pod linuxem podjął się Mark Hemment. Różni się ona znacznie on mechanizmów w jądrach 2.0.x. Zastosowano alokator pamięci z buforowaniem obiektów (ang. *An Object Caching Kernel Memory Allocator*). Ograniczono w ten sposób fizyczne zwalnianie i przydzielanie pamięci. Polega to na zachowywaniu stanu obiektu pamięci pomiędzy kolejnymi użyciami i nie niszczeniu go i tworzeniu od nowa za każdym razem. W ten sposób pamięć jest przydzielana właściwie softwareowo - fizyczny przydział następuje tylko przy pierwszym tworzeniu obiektu. Zwalnianie pamięci przez jądro nie powoduje zatem niszczenia obiektów pamięci a jedynie 'odłożenie' ich do ponownego użycia w przyszłości. Alokator realizuje przydzielanie w wielowątkowym trybie pracy poprzez sysnchronizację semaforami.

1.2 Alokator

1.2.1 Operacje

Idea trzech podstawowych operacji na alokatorze jest następująca:

- przydzielanie obiektów:

  jeśli jest obiekt w cachu:
  ....weź go - konstrukcja niepotrzebna;
  w przeciwnym wypadku:

....zaalokuj pamięć fizycznie;
....skonstruuj obiekt;

- zwalnianie obiektów:

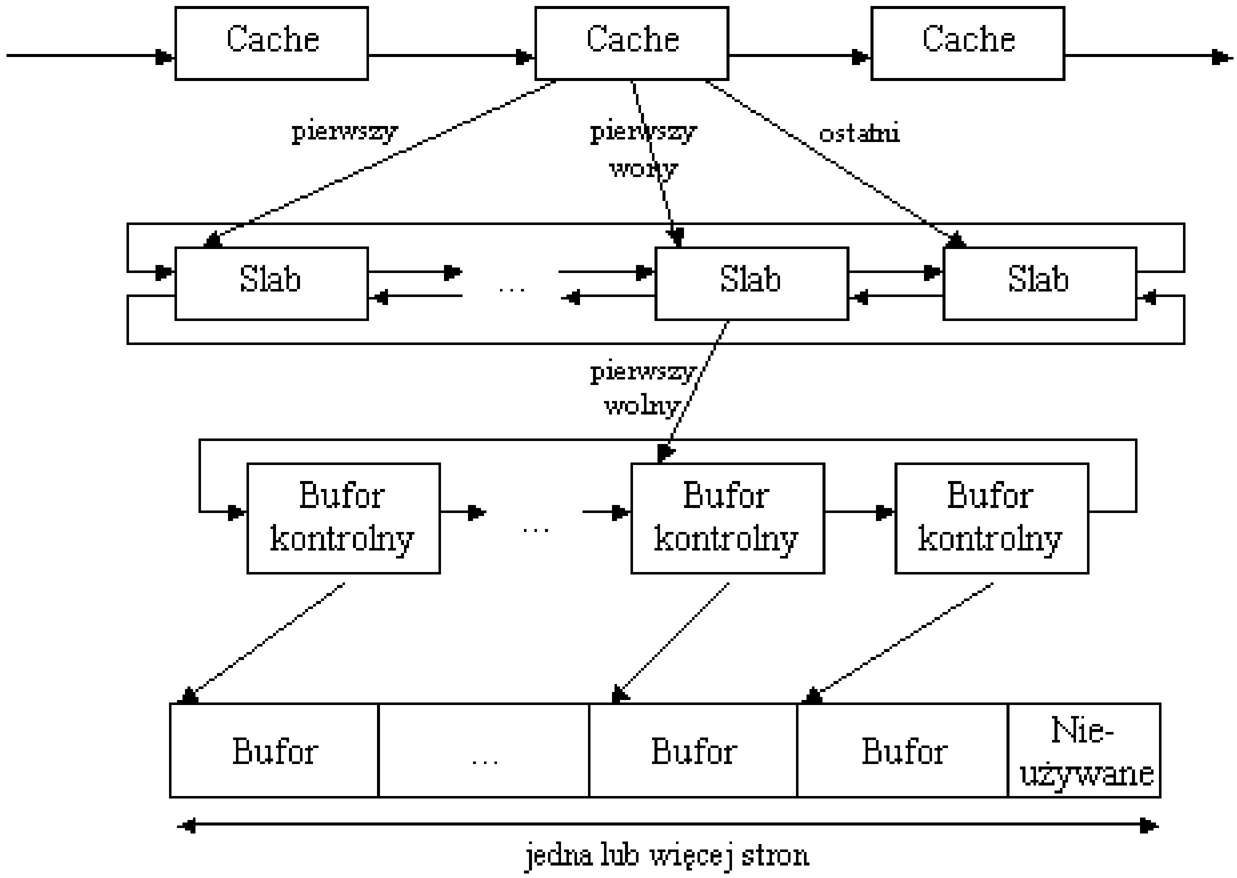
  zwróć obiekt do cache - nie niszcz go fizycznie;

- odzyskiwanie obiektów z cache:

  weź pewną liczbę obiektów z cache;
  zniszcz je;
  zwolnij przydzieloną im pamięć fizycznie;

1.2.2 Schemat

Oto schemat struktur danych alokatora:



Rysunek 1.1: Schemat struktur dla obiektów rozmiaru większego niż 1/8 strony

1.2.3 Struktury danych

- **cache**

  Cache jest jedyną strukturą logiczną mechanizmu alokacji widzianą przez jądro. Używa ona oczywiście pomocniczych, niebanalnych struktur (o tym za chwilę). Wszystkie obiekty w cachu mają ustaloną wielkość - rozmiary te są określone w tablicy `cache_sizes` używanej w definicji typu `cache_sizes`. Może to być 32, 64, 128, ..., 65536, 131072 bajtów. Cache ułożone są w listę jednokierunkową. Jej implementacja wraz z najważniejszymi polami:

```
struct kmem_cache_s(
    kmem_slab_t      *c_freep;          # lista wolnych slabów w cachu
    kmem_slab_t      *c_firstp;         # pierwszy slab w łańcuchu
    kmem_slab_t      *c_nextp;          # ostatni slab w łańcuchu
    size_t            c_org_size;       # rozmiar obiektów
    void (*c_ctor) ...                  # wskaźnik na konstruktor;
    void (*c_dtor) ...                  # wskaźnik na destrutor;
    const char       *name;             # nazwa
    kmem_cache_s      c_nextp;          # następny cache w liście
)
```

  W dalszej części pojawi się typ `kmem_cache_t`, który jest zdefiniowany po prostu jako:

  `typedef struct kmem_cache_s kmem_cache_t;`

  Funkcje interfejsu struktury

  – konstruowanie

```
kmem_cache_t * kmem_cache_create(...)   # utworzenie cachea
int            kmem_cache_shrink(...)   # ściśnięcie cache - usuwa tak
                                        # wiele slabów z cache jak to możliwe
int            kmem_cache_destroy(...)  # niszczy cache (o ile to możliwe)
```

  – zwalnianie/przydzielanie

```
void *  kmem_cache_alloc(
            kmem_cache_t *cachep,
            int flags)
```

  Ważna funkcja. Bierze obiekt z cache. Flagą może być KM_SLEEP lub KM_NOSLEEP co w przypadku braku pamięci oznacza odpowiednio oczekiwanie na zwolnienie jej przez inny proces lub nie oczekiwanie. W przypadku gdy pamięć w cachu jest za mało wywoływana jest funkcja `kmem_cache_grow`, która powoduje fizyczne skonstruowanie nowego obiektu. Funkcja jest wywoływana w algorytmie przydzielania pamięci dla jądra -)`kmalloc`.

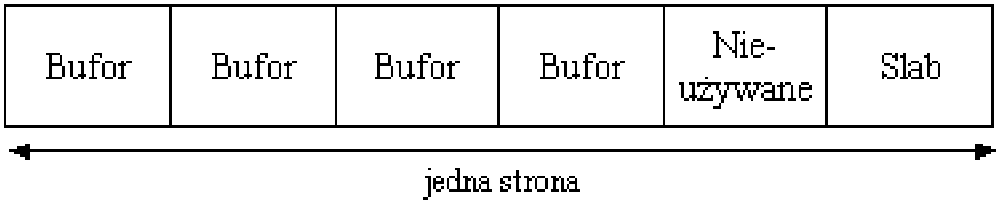
```
void    kmem_cache_free(
            kmem_cache_t *cachep,
            void *objp)
```

Funkcja zwraca obiekt `objp` do cache `cachep`. Funkcja ta jest opisana szczegółowo dalej. Jest ona bowiem wywoływana przez `kfree()`.

- **slab**

  Slab (w tłumaczeniu: tafla) jest podstawową jednostką alokatora. Powiększenie pamięci w cache (`grow`) oznacza w pierwszej kolejności zbudowanie slaba z obiektami. Podobnie odzyskiwanie pamięci (`shrink`) po pierwsze zwolni puste slaby. Slab zawiera jedną lub więcej stron pamięci podzielone na kawałki stałej długości oraz licznik zalokowanych kawałków. Jeśli licznik ten jest równy zero pamięć określaną przez slab można zwolnić fizycznie. I to jest najważniejsza korzyść z budowania tej struktury - zastępuje to skomplikowane algorytmy zwalniania zastosowane w innych alokatorach (zresztą także we wcześniejszych jądrach linuxa). Przydzielanie i zwalnianie pamięci dla jądra jest także błyskawiczne ponieważ polega tylko na usunięciu lub wstawieniu na listę wolnych obiektów i zaktualizowanie licznika.

  Slaby ułożone są w liste dwukierunkową cykliczną. Zawierają wskaźnik na listę wolnych obiektów. Slab widzi obiekty poprzez specjalne dodatkowe struktury **bufory kontrolne** (`kmem_bufctl_s`). Dotyczy to jednak obiektów o rozmiarze więszym niż 1/8 strony. Dla obiektów mniejszych slab wygląda nieco inaczej. Po pierwsze określa on dokładnie jedą stronę pamięci. Powstaje przy alokowaniu strony i jego dane są umieszczane na końcu. Reszta jest podzielona na równych rozmiarów kawałki - bufory. Każdy bufor jest swoim buforem kontrolnym. Linki listy wolnych obiektów znajdują się na końcu każdego bufora na co alokator dodaje kilka bajtów do obiektu. Takie rozwiązanie gwarantuje zoptymalizowanie alokacji dla małych obiektów - przecież takie są używane najczęściej. Tworzenie tu buforów kontrolnych mijało by się z celem, bowiem zajmowały by one prawie tyle pamięci co same obiekty. Oto



Rysunek 1.2: Schemat slaba dla obiektów rozmiaru mniejszego niż 1/8 strony

  implementacja slaba i najważniejszych pól:

```
typedef struct kmem_slab_s {
  struct kmem_bufctl_s  *s_freep;  # wkaźnik na pierwszy wolny obiekt
  unsigned long          s_inuse;  # licznik obiektów zajętych
  struct kmem_slab_s    *s_nextp;  # następny slab
  struct kmem_slab_s    *s_prevp;  # poprzedni slab
  void                  *s_mem;    # adres pierwszego obiektu slaba
} kmem_slab_t;
```

  Lista slabów jest posortowana partiami. Na jej początku znajdują się pełne slaby, potem częściowo zajęte, a na końcu całkowicie wolne (`s_inuse` = 0). Wskaźnik w cache na listę wolnych slabów wskazuje na pierwszy niepusty slab.

- **bufor kontrolny**

  Struktura zawiera link do bufora kontrolnego następnego wolnego obiektu, adres samego bufora (czyli właśnie wolnego obiektu) oraz wskaźnik slaba do którego jest przydzielona. Oto jej implementacja:

```
typedef struct kmem_bufctl_s {
  union {
    struct kmem_bufctl_s *buf_nextp; # następny bufor kontrolny wolnego obiektu
    kmem_slab_t          *buf_slabp; # slab dla obiektu
    void                *buf_objp;  # adres obiektu (bufora)
  } u;
} kmem_bufctl_t;
```

1.3 Zwalnianie pamięci przez jądro

1.3.1 Algorytm kfree()

Zwalnianie pamięci przez jądro jest trywialne. Polega ono na wstawieniu obiektu do listy wolnych obiektów. Deklaracja procedury kree jest następnująca (`objp` jest adresem zwalnianego obiektu):

```
void kfree(const void *objp) { ... }
```

Działanie samej tej procedury polega na, po pierwsze, wyliczeniu na podstawie adresu danego obiektu strony pamięci na której obiekt leży. Każda strona może być przydzielona do co najwyżej jednego cachea. Teraz więc następuje trywialne Odczytanie jego adresu. Umieszczenie obiektu na liście wolnych wykonuje się poprzez wywołanie funkcji `kmem_cache_free` z adres obiektu i cache jako parametrami.

Algorytm 1 algorytm kfree()

Dane wejściowe: adres obiektu do zwolnienia *objp

jeśli podany adres jest null **to**
    zakończ z błedem;
koniec jeśli
oblicz adres strony adresu *objp;
jeśli strona nie istnieje lub nie jest przydzielona alokatorowi **to**
    zakończ z błedem;
koniec jeśli
weź adres cache dla wyliczonej strony;
jeśli adres ten istnieje i obiekt objp jest w stanie używania **to**
    wywołaj procedurę aktualizacji `kmem_cache_free` z adresem cache i obiektu jako parametrami;
koniec jeśli

1.3.2 Procedura `kmem_cache_free()`

Procedura dostaje adres cache i obektu do zwolnienia. Najpierw odnajduje slaba który obsługuje dany obiet. Aktualizuje listę wolnych obiektów w slabie poprzez wstawienie zwalnianego **przed** pierwszy wolny. Zmniejsza też licznik używanych obiektów. Jeśli doszedł on do zera wcale nie niszczy slaba, a jedynie umieszcza go na koniec slabów, tam gdzie są puste slaby. Deklaracja procedury jest następująca:

```
static inline void
__kmem_cache_free(kmem_cache_t *cachep, const void *objp) { ... }
```

Literatura

[Bonwick] Jeff Bonwick: *The Slab Allocator: An Object-Caching Kernel Memory Allocator* http://www.nuclecu.unam.mx/ miguel/bonwick.ps.

[Hemmet97] Mark Hemmet: źródła linuxa - /usr/src/linux-2.2.17/mm/slab.c