Algorytm zwalniania wolnych buforów

Przemysław K. Rekucki Jarosław Wypychowski

2000

Spis treści

1 Wstęp. **2**

2 Struktury i zmienne **2**

3 Algorytm zarządzania buforami w pigułce **4**

3.1 Alokacja buforów 4
3.2 Synchronizacja buforów 4
3.3 Zwalnianie buforów 5
3.4 Funkcje wymagane od alokatora pamięci jądra 5
3.5 Interfejs wymagany od modułu osługi urządzenia 5

4 Zarządzanie buforami **5**

4.1 Tworzenie nowych buforów 6
4.2 Synchronizacja buforów z urządzeniem blokowym 6
4.3 Uzyskiwanie wolnych buforów 7
4.4 bdflush 8
4.5 kupdate 8

5 Udostępniany interfejs **8**

1 Wstęp.

W celu zminimalizowani ilości bezpośrednich odwołań do urządzeń blokowych oraz w celu dodania jednolitego interfejsu odwołań do plików na różnych systemach plików wprowadzono system pamięci podręcznej. Wszystkie żądania pisania i czytania odnośnie plików wymagają pośrednictwa buforów. Niniejszy referat opisuje algorytmy i struktury danych związane ze zwalnianiem buforów w systemie pamięci podręcznej.

2 Struktury i zmienne

bdflush_param
: Jest to unia struktury i tablicy intów, tablice dodano po by łatwo było napisać interfejs konfiguracyjny. Parametry te można czytać z pliku */proc/vm/bdfulsh*.

buffer_head
: Jest to typ opisu nagłówków buforów. Używany on jest jako argument w operacjach na urządzeniach blokowych. Używane bufory zanajdują się w tablicy haszującej po identyfikatorze urządzenia i numerze bloku. W przypadku odwołania do danego bloku na danym urządzeniu następuje sprawdzenie czy takiego bufora już niema. Wtedy wykorzystywana jest właśnie ta struktura. W jej implementacji stosuje się adresowanie liniowe ze stałą wiekością tablicy. Wielkość dobiera się przy starcie systemu na podstawie ilości wolnej pamięci,alokuje się tyle stron wskaźników do nagłówkow ile wynosi najminiejsza liczba potęga dwójki większa od liczby megabajtów dostępnej pamięci.

  Inne struktury w których mogą zanajdować się dane typu `buffer_head` jest lista wolnych nagłówków. Dokładniej są to dwie listy:

  - `unused_list` są to wolne nagłówki gotowe do ponownego użycia, trzeba jednak zaalokować dla nich bloki. Zajmuje się tym funkcja `create_buffers`.
  - Do `reuse_list` trafiają bufory tymczasowe po zakończonych operacjach. Nie mogą one trafić do `unused_list` ponieważ mogą nastąpić odwołania do niego z starownika urządzeń. Zostanie później przełożony przez `recover_reusable_buffer_heads`.

NAZWA	OPIS	Domyślnie	MIN	MAX
`nfract`	procent buforów typu *dirty* przy którym aktywuje się `bdflush`	40%	0%	100%
`ndirty`	maksymalna liczba buforów do zsynchronizowania na jedno obudzenie wątku `bdflush`	500	10	5000
`nrefill`	ilość wolnych buforów do uzyskania przez algorytm refill	64	4	2000
`nref_dirt`	Jedyne gdzie występuje ten identyfikator to deklaracja w `buffer.c` oraz opisy w katalogu `Documentation`. Teoretycznie miało to być ograniczenie na ilość nie zsynchronizowanych buforów do aktywowania `bdflush` podczas odzyskiwania buforów.	256	25	2000
`interval`	Czas jaki ma spać `kupdate` w przejściu pentli	5s	0s	60s
`age_buffer`	Czas po jakim bufor powinien być zapisany.	30s	1s	600s
`age_super`	Czas po jakim bufor superbloku powinien być zapisany	5s	1s	600s

Tablica 1: struktura bdf_prm.b_un

- `free_list` jest tablicą list wolnych buforów o danej wielkości. Wielkości buforów mogą być tylko potęgą 2 oraz muszą być większe równe 512. List jest 7 więc największy bufor ma 64kb.
- `lru_list` jest tablicą 3 list: buforów czystych(`BUF_CLEAN`), zajętych (`BUF_LOCKED`), brudnych czyli wymagajacych zapisu (`BUF_DIRTY`). wkładanie do tych list jest na koniec więc na początku znajdują się bufory będące tam najdłużej.

nr_buffers ilość buforów

nr_buffers_type tablica ilości buforów danej grupie, indeks odpowiada temu z `lru_list`

nr_buffer_heads liczba nagłówków.

nr_hashed_buffers liczba buforów w tablicy.

Oprócz tego jest jszcze szereg kolejek procesów które będą opisane przy opisie algorytmu.

3 Algorytm zarządzania buforami w pigułce

3.1 Alokacja buforów

Nowe bufory są alokowane w wypadku żądania odczytu z lub zapisu do pliku na urządzeniu blokowym.

3.2 Synchronizacja buforów

Synchronizacja (zapisanie bufora na dysk) zachodzi w kilku przypadkach:

- Na żądanie.

  Proces może zarządać zapisu bufora na dysk poprzez wywłania systemowe.

- Gdy brakuje wolnych buforów.

  Na poziomie jądra działa demon `bdflush`, którego zadaniem jest synchronizowanie w każdym wywałaniu określonej ilości buforów (domyślnie 500). Jeśli nie uda się zaalokować nowego bufora to `bdflush` jest budzony tyle razy, aby uzyskać odpowiednią ilość wolych buforów.

- Co pewien czas

  Dodatkowo na poziomie jądra działa jeszcze jeden demon: `kupdate`. Jego zadaniem jest co pewien określony czas wywoływać `sync_old_buffers`. Domyślnie dzieje się to co 5 sekund.

3.3 Zwalnianie buforów

Po zsynchronizowaniu z urządzeniem bufor jest zaznaczany jako czysty. I jako taki może być przypisany do innego pliku. Jeśli wszystkie bufory na pewnej stronie są czyste, to bufory te się kasuje z systemu, a stronę zwalnia.

3.4 Funkcje wymagane od alokatora pamięci jądra

Alokacja i dealokacja stron pamięci wymaga istnienia odpowiednich funkcji w menedżerze pamięci jądra. Problem polega na tym, że bufory nie mogą być swapowane (dopuszczało by to bowiem zablokowanie algorytmu gdy próba zapisu bufora na dysk w momencie braku wolnych buforów wymagałaby stworzenia nowego bufora). W jędrze 2.2.17 alokator pamięci kernela udostępnia tryb BUFFER co oznacza właśnie pamięć nieswapowaną.

3.5 Interfejs wymagany od modułu osługi urządzenia

W momencie inicjalizacji bufora moduł musi podać swoją funkcję zapisu i odczytu z urządzenia. Informacja ta jest przechowywana w nagłówku bufora.

```
struct buffer_head
{
...
  void(*b_end_io)(sruct buffer_head *bh, int uptodate);
...
}
```

Jak więc widać jest to wskaźnik do funkcji.

4 Zarządzanie buforami

Opisanie algorytmu zwalniania buforów wymaga opisania kilku pomniejszych algorytmów, które przy okazji są wykorzystywane.

4.1 Tworzenie nowych buforów

Ten algorytm jest rozdzielony pomiędzy kilka funkcji.

1. alokacja pamięci na bufory Zajmuje się tym funkcja `grow_buffers` Algorytm wygląda następująco:

   ```
   1 jeśli nie udało sie zaalokować nowej strony to wyjdź
   2 stwórz na zaalokowanej stronie nowe nagłówki buforów
   3 uaktualnij wszystkie listy
   ```

   Przy czym w drugiej lini wywoływana jest funkcja `create_buffers`, której opis znajduje się poniżej.

2. tworzenie nagłówków buforów Jest to umieszczeone w funkcji `create_buffers`. Parametry wejściowe: strona pamięci, wielkość bufora.

   ```
   dopóki starczy miejsca na stronie
   jeśli nie znalazł nieużywanego nagłówku bufora
       zwolnij wszystkie użyte wcześniej nagłówki
       zwolnij stronę
       wyjdź z błędem
   koniec jeśli
   przypisz mu kolejny kawałek strony.
   dołącz go do odpowiedniej listy
   zainicjuj nagłówek
   koniec dopóki
   zwróć ostatni bufor
   ```

   Podany powyżej algorytm jest przepisanym do postaci strukturalnej algorytmem rzeczywistym. Oryginał posługuje się instrukcjami **goto** zamiast **if**

4.2 Synchronizacja buforów z urządzeniem blokowym

Kod odpowiedzialny za ten algorytm jest dosyć rozrzucony. Podstawową funkcją jest jednak `sync_buffers`, do której koniec końców trafiają wszystkie pozostałe wywołania synchronizujące (za wyjątkiem wywałań demona *bdflush*). Ten algorytm jest następujący: Dane wejściowe: urządzenie synchronizowane, czas oczekiwania.

```
dla wszystkich buforów brudnych
   jeśli bufor należy do urządzenia
jeśli jest zablokowany
    jeśli czas_oczekiwania=0 to kontynuuj
    w przeciwnym przypadku
czekaj na bufor
    koniec jeśli
w przeciwnym przypadku //nie jest zablokowany
    jeśli czas_oczekiwania=0 to
         zapisz bufor na dysk
przenieś bufor do kolejki buforów czystych
    w przeciwnym przypadku
ustaw czas zapisu bufora na czas_oczekiwania
    koniec jeśli
koniec jeśli
   koniec jeśli
koniec dla

dla wszystkich zablokowanych buforów
   jeśli czas_oczekiwania=0 to kontynuuj
   w przeciwnym przypadku
czekaj na bufor
   koniec jeśli
koniec dla
```

Wyjaśnienia wymagaja pojęcie bufora zablokowanego: Ponieważ "jednocześnie"dany bufor może być synchronizowany przez *bdfush* oraz powyższą funkcję więc status zablokowanego bufora jest ustalany przez demona *bdflush*, kiedy ten synchronizuje bufor

4.3 Uzyskiwanie wolnych buforów

To jest właśnie główny algorytm całego referatu - uzyskuje on w puli wszystkich buforów bufory wolne. Pełen kod jest zawarty w funkcji `refill_freelist`. Treść jest następująca:

```
jeśli udało się zaalokować nowe bufory to wyjdź
w przeciwnym przypadku
    obudź bdflush'a aby ten zwolnił kilka buforów
    przekaż sterowanie do schedulera
koniec jeśli
```

Nowe bufory alokuje funkcja `grow_buffers`. A *bdflush*'a budzi funkcja `wakeup_bdflush`

4.4 bdflush

`kflushd` jest wątkiem jądra, inicjowanym przez pierwszy proces w systemie. Jego zadaniem jest zapisywanie pewnej liczby zmodyfikowanych buforów do urządzenia blokowego. Algorytm przechodzi po kolejkach `lru_list` dla buforów zablokowanych do transmisji a później po buforach do typu `BUF_DIRTY`. Najpierw sprawdza się czy dany bufor nie jest we właściwej liście, jesli nie to przekłada się bufor do właściwej i przechdzi się do następnego. Jednak jeśli bufor jest na swoim miejscu oraz jest nie zablokowany i brudny to próbuje on wykonać operacje zapisu. Zapis odbywa się zazwyczaj synchronicznie. Wyjątek stanowią bloki dla *loopback device* które mogą do zapisu wymagać zaalokowania buforów a to może spowodować że przydzielanie zawiesi się czekanjąc na `bdflush` który czaka na koniec zapisu. By tego uniknąć zapis do tego typu urządeń wykonuje nie czekając na jego zakończenie. Jeśli po przejściu wszyskich buforów żadnego nie zwolniono to zanczy że były do urządzeń typu `loopback` to powtarza się proces z tym wyjątkiem że pierwszy zapis do `loopback` wykonuje sie blokująco a następne tak jak zwykle. Dodatkowo za kożdym razem ustawia się czas `b_flushtime` na 0. Spowoduje to zapisanie bufora podczas cyklu demona `kupdate`.

Po poprawnym wykonaniu cyklu demon budzi czekające na niego procesy na kolejce `bdfulsh_done`. Po zwolnieniu odpowiedniej liczby buforów demon zasypia na kolejce `bdflush_wait` czekając aż dostanie sygnał do dalszej pracy. Ze względu na uporządkowanie listy (FIFO) zapisane zostaną bufory najdłurzej będące w kolejce.

4.5 kupdate

Jest to bardzo prosty demon, który budzi się w równych odstępach czasu `bdflush_param.b_un.interval`, wołając funkcje `sync_old_buffers`. Funkcja ta działa niemal identycznie z `bdflush`. Jedyną istotną różnicą jest to że tu zapis jest zawsze synchroniczny i tylko gdy minoł czas przez jaki bufor mógł być niezapisywany.

5 Udostępniany interfejs

Nagłówki funkcji, które są dostępne publicznie są umieszczone w pliku *linux/fs.h*. Krótki spis znajduje się poniżej.

- Inicjujące

  ```
  void buffer_init(unsigned long)
  void init_buffer(struct buffer_head *bhm kdev_t dev, int block,
  bh_end_io_t *handler, void *de_id)
  ```

  W powyższym kodzie handler jest funkcją, którą musi udostępnić moduł obsługi urządzania - służy ona do zapisu/odczytu z urządzenia.

- Operacje na buforach

  ```
  void refile_buffer(struct buffer_head *buf)
  void try_to_free_buffer(struct page *, int wait)
  void set_writetime(struct buffer_head *buf, int flag)
  ```

  Pierwsza funkcja wstawia bufor do odpowiedniej dla niego listy. Druga zwalnia całą stronę pamięci (o ile to możliwe). Trzecia ustawia dla konkretnego bufora po jakim czasie ma zostać zsynchronizowany z urządzeniem.

- Synchronizujące

  ```
  void sync_dev(kdev_t dev)
  void fsync_dev(kdev_t dev)
  ```

  Obie synchronizują bufory urządzenia z urządzeniem.

- Zmienne globalne

  ```
  int nr_buffers
  ulong buffermem
  int nr_buffer_heads
  ```

  Ich nazwy są wystarczającym wyjaśnieniem.