40 Algorytm czytania bloków z wyprzedzeniem `breada`

Franciszek Grzegorek

15 grudnia 2000

1 Wprowadzenie

Czytanie bloków z wyprzedzeniem (*blocks read ahead*) zostało wprowadzone w Linuxie, aby przyspieszyć odczyt z dysku. Występuje ono w formie odczytywania kilku (kilkunastu) bloków następujących po bloku, który należy w danym momencie odczytać, o ile należą one do tego samego pliku. Działa ono dlatego, że gdy jest żądanie na dany blok dyskowy, z dużym prawdopodobieństwem w niedalekiej przyszłości pojawią się żądania na następujące bezpośrednio po nim bloki, zwłaszcza w przypadku, gdy należą one do tego smego pliku.

Algorytm czytania bloków z wyprzedzeniem jest zrealizowany w funkcji `breada` umieszczonej w pliku `fs/buffer.c` w źródłach Linuksa.

Będę omawiał wersję 2.2.17

2 Struktura `buffer_head`

Jest to struktura, która przechowuje nagłówek bufora. Wiele, z wykonywanych w funkcji `breada` operacji, odnosi się do całej struktury; nie ma tam odniesień do poszczególnych pól tej struktury. Warto jednak zobaczyć kilka pól, żeby dowiedzieć się, jakie informacje przechowuje, ułatwi to zrozumienie algorytmu.

- `struct buffer_head * b_next` - wskaźnik na następny nagłówek na liście mieszającej (*hash queue list*)
- `unsigned long b_blocknr` - numer bloku, który ten bufor powinien zawierać (blok może nie być jeszcze odczytany)
- `unsigned long b_size` - rozmiar bufora
- `kdev_t b_dev` - urządzenie związane z tym blokiem (`B_FREE` = wolny)
- `unsigned long b_state` - stan bufora
- `unsigned int b_count` - liczba użytkowników w danym momencie (`getblk` zwiększa, a `brelse` zmniejsza)
- `char * b_data` - wskaźnik do bloku danych

3 Funkcja `breada`

3.1 Parametry wywołania

W parametrach określa się informacje związane z żądanym do odczytu blokiem, oraz plikiem w którym się ten blok znajduje. Ilość bloków do przeczytania z wyprzedzeniem funkcja oblicza sama, kierując się tymi parametrami, ale nie tylko nimi.

- `kdev_t dev` - numer systemu plików (numer urządzenia z którego ma być odczyt)
- `int block` - numer bloku, który chcemy przeczytać z urządzenia danego parametrem `dev` (zwany dalej żądanym blokiem)
- `int bufsize` - rozmiar bufora
- `unsigned int pos` - pozycja bloku do przeczytania w pliku
- `unsigned int filesize` - rozmiar pliku

3.2 Wynik działania

Wynikiem działania funkcji jest:

- `NULL` - gdy zakończy się błędem,
- `bh` - czyli wskaźnik do nagłówka pierwszego wczytanego bufora (tego, który zawiera żądany blok) w przypadku powodzenia.

3.3 Ważniejsze zmienne

- `bhlist` - lista, a właściwie tablica o rozmiarze `NBUF`, wskaźników do nagłówków buforów (tych, do których mają być zapisywane kolejne, odczytywane z pliku, bloki)
- `blocks` - służy do obliczenia, a później do przechowywania, liczby buforów do odczytania
- `bh` - wskaźnik do nagłówka żądanego bufora, zwracany jako wynik funkcji w przypadku powodzenia
- `index` - indeks rozmiaru bufora, przyjmie wartość makra `BUFSIZE_INDEX` dla rozmiaru bufora podzielonego przez 512 (wszystkie rozmiary buforów są potęgami 2 począwszy od 512, stąd często występuje prawy shift o 9.) Makro to jest zdefiniowane jako logarytm przy podstawie 2 (zrealizowane przy pomocy tablicy `buffersize_index`, w celu przyspieszenia działania)
- `j` - zlicza ilość bloków do przeczytania (można by nadać jakąś bardziej znaczącą nazwę, a tak, w jednej pętli, pojawia się nic nie mówiące porównanie `i<j`)

3.4 Algorytm

Funkcja rozpoczyna działanie od sprawdzenia poprawności parametrów wejściowych (tzn. czy odczyt z takimi parametrami ma sens.)

Gdy wskaźnik pozycji jest na końcu pliku, lub poza nim (`pos >= filesize`) (żądanie czytania poza końcem pliku) funkcja kończy działanie z błędem. (`NULL`)

Gdy numer bloku jest ujemny (`block < 0`) kończy się także z wartością `NULL`.

Następnie, korzystając z funkcji `getblk`, próbuje odszukać bufor z odczytywanym blokiem w tablicy mieszającej (`bh = getblk(dev,block,bufsize)`).

Gdy `getblk` go tam znajdzie i bufor ten zawiera aktualne dane (zajdzie wtedy `buffer_uptodate(bh)`), funkcja `breada` kończy działanie, zwracając wskaźnik do nagłowka znalezionego bufora. Nie czyta więc nic z wyprzedzeniem. Zakłada, że skoro jest ten blok, to prawdopodobnie jest też kilka następnych. Gdy jednak nie ma ani jednego (ten blok był ostatnim, który został przeczytany (np. z wyprzedzeniem) przy wcześniejszym odczycie jakiegoś bloku), to program żądający następnego bloku będzie musiał czekać na jego odczyt z dysku. Jednak sprawdzanie przy każdym odczycie bloku, czy wczytana jest odpowiednia liczba następnych bloków, mogło by znacznie spowolnić pracę i spowodować wiele niepotrzebnych odczytów z dysku, a także zwiększyłoby zajętość buforów.

Gdy zwrócony bufor nie będzie zawierał aktualnych danych (był w tablicy mieszającej, ale nieaktualny, bądź też go nie było - `getblk` zajeło pusty bufor z kolejki buforów wolnych (gdy nie było wolnych, odzyskuje się wolne bufory za pomocą funkcji `refill_free_list`)) funkcja zainicjuje odczytanie do tego bufora żądanego bloku z dysku (`ll_rw_block(READ, 1, &bh)`)

Kolejną czynnością jest ustalenie liczby bloków, które mają być odczytane z wyprzedzeniem. W tym celu, w kodzie, tuż przed definicją funkcji `breada`, zdefiniowana jest stała `NBUF` przyjmująca wartość 16. (Od wersji 1.2.x (na pewno od 1.2.13) nie jest zmieniana, być może, z powodu zwiększenia szybkości dysków i procesorów, jak i zwiększenia używanych plików, opłacało by się ją zwiększyć (chociaż równie dobrze na szybszych dyskach wielkość bloku może być większa.)) Najpierw liczba bloków do przeczytania wprzód (`blocks`) przyjmuje wartość: liczba bloków do końca pliku (`filesize - pos` >> `(9+index)`). Potem, przyjmuje większą z wartości: `read_ahead[MAJOR(dev)]` >> `index` (ilość sektorów czytana z wyprzedzeniem dla danego urządzenia w przeliczeniu na bloki) i swojej dotychczasowej wartości. Na koniec, gdy jest większa od `NBUF`, jest obniżana do `NBUF`. Obliczona w ten sposób liczba jest liczbą wszystkich przeczytanych bloków, zatem liczba czytanych bloków z wyprzedzeniem jest o jeden mniejsza, w szczególności nigdy nie jest większa od `NBUF - 1` (czyli teraz najwyżej 15).

Następnie są przydzielane bufory funkcją `getblk`, do momentu, aż liczba przydzielonych buforów będzie równa obliczonej w poprzednim kroku (`blocks`), albo natrafi się na bufor z aktualnymi danymi (który się od razu zwalnia funkcją `brelse`). Liczba tak przydzielonych buforów jest zapamiętana w zmiennej `j`. Bufory (wskaźniki ich nagłówków) są zapamiętywane w tablicy `bhlist`.

Następnie inicjowane jest czytanie z wyprzedzeniem do wszystkich `j` buforów poza pierwszym: `ll_rw_block(READA, (j-1), bhlist+1)`. Pierwszy parametr oznacza, że będzie czytanie z wyprzedzeniem, drugi określa liczbę czytanych bloków z wyprzedzeniem, a trzeci wskazuje na drugi element listy, gdyż wskaźnik w pierwszym wskazuje na nagłówek żądanego bufora, z czytaniem inicjowanym wcześniej.

Zwalniane są (także funkcją `brelse`) wszytkie przydzielone wcześniej bufory, ewentualnie poza ostatnim aktualnym (gdy na taki się natrafiło, który był natychmiast zwalniany).

Funkcja czeka na zakończenie odczytu żądanego bloku: `wait_on_buffer(bh)`.

Gdy bufor zawiera aktualne dane, czyli odczyt się powiódł, kończy działanie zwracając wskaźnik do jego nagłówka, jako wynik.

Gdy nie zawiera aktualnych danych, to zwalnia ten bufor i kończy się z błędem (zwraca `NULL`).

4 Przykładowe zadania

4.1 Zadanie 1

Dlaczego czytanie z wyprzedzeniem może czasami spowolnić działanie?

4.2 Zadanie 2

Podać przykłady sytuacji, w której funkcja `breada` spowoduje przeczytanie mniej niż `NBUF` bloków.

4.3 Zadanie 3

Podać przykład sytuacji, w której funkcja `breada` nie spowoduje żadnego dostępu do dysku, pomimo tego, że parametry będą poprawne (sensowne).

5 Odpowiedzi do przykładowych zadań

5.1 Odpowiedź do zadania 1

Możliwe jest, że jakiś proces odczytuje plik w jakimś miejscu i później czyta ten plik zupełnie gdzie indziej (lub zaprzestaje czytania). Może to wystąpić np. przy wyszukiwaniu binarnym w wielkim pliku. Bufory odczytane z wyprzedzeniem są wtedy najprawdopodobniej czytane całkowicie na próżno.

Spowolnienie występuje wtedy z kilku powodów:

- wykonywanie większej ilości instrukcji,
- zwiększenie zajętości buforów i co za tym idzie, możliwość zmienienia zawartości buforów (zapisanie w buforze innego bloku), niektóre potrzebne bloki mogą więc wymagać ponownego odczytu z dysku (bez czytania z wyprzedzeniem nadal byłyby w buforze)

- Zwiększenie ilości żądań do dysku, co spowoduje spowolnienie wszystkich odczytów (i zapisów) do (z) dysku.

5.2 Odpowiedź do zadania 2

Przykładowe sytuacje:

- do końca pliku jest mniej niż `NBUF` bloków, albo
- któryś z `NBUF - 1` bloków, począwszy od żądanego, jest już w pamięci (istnieje bufor z tym blokiem.)

5.3 Odpowiedź do zadania 3

Będzie tak, gdy żądany blok jest już w którymś z buforów (i jest aktualny).

Bibliografia

[1] Projekt Linux

[2] M. J. Bach "Budowa systemu operacyjnego Unix", WNT 1995

[3] Cross-Referencing Linux `http://lxr.linux.no`