Algorytm block_write
referat

Bartosz Tomecki

20 grudnia 2000

Spis treści

1 Wstęp

Referat ten powstał jako indywidualna praca studencka w ramach zajęć z „Systemów Operacyjnych” prowadzonych na wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego w roku akademickim 2000/2001.
Referat ten ma na calu przedstawienie algorytmu funkcji `block_write` z jądra systemu Linux. Powstał on w oparciu o jądro systemu Linux w wersji 2.2.17 i zawarte w nim informacje dotyczą jedynie tej wersji jądra systemu.

2 Algorytm block_write

Dane wejściowe:
- `struct file * filp`,
- `const char * buf`,
- `size_t count`,
- `loff_t * ppos`

Wynik funkcji:
- wartość typu `ssize_t` - ilość zapisanych bajtów

Algorytm 1 block_write - kontrola parametrów wywołania i ustalenie warunków działania

1: ustalamy numer urządzenia, na którym będziemy pracować
2: **jeśli** urządzenie tylko do odczytu **to** zwracamy błąd `EPERM`
3: ustalamy rozmiar bloku `block_size` dla urządzenia, blok `block` i pozycje w bloku `offset` na podstawie pozycji `ppos` oraz rozmiar urządzenia `size`

Algorytm 2 block_write - pętla główna

```
dopóki zostało coś do zapisania count > 0 wykonuj
  jeśli nie można zapisać więcej block >= size to zwracamy ilość zapisanych bajtów lub błąd ENOSPC jeśli jeszcze nic nie zapisaliśmy
  ustalamy rozmiar następnej porcji danych do zapisu chars i pobieramy nowy bufor bh na dane
  jeśli nie udało się pobrać bufora to zwracamy ilość zapisanych bajtów lub kod błędu EIO jeśli jeszcze nic nie zapisaliśmy
  jeśli bufor nie zawiera aktualnych danych to
    jeśli ilość danych do zapisania jest równa rozmiarowi bloku to
      czekamy na aktualizacje bufora
    w przeciwnym przypadku
      dopisujemy bufor bh do listy buforów
      jeśli nie możemy odczytywać z wyprzedzeniem to
        ustawiamy liczbe bloków do odczytania blocks na 1 (wymuszamy odczytanie bloku)
      w przeciwnym przypadku
        odczytujemy naprzód tyle bloków ile się da, jeśli przy odczycie wystąpi błąd to zwracamy ilość zapisanych do tej pory bajtów writen lub kod błędu EIO jeśli jeszcze nic nie zapisaliśmy
      koniec jeśli
      żądamy odczytu buforów z listy buforów bufferlist
      dla każdego bufora z listy buforów bufferlist wykonuj zwolnij bufor
      bf koniec dla każdego
      czekamy na zakończenie operacji na buforze bh
      jeśli bufor nie zaktualizowany to zwalniamy bufor i zwracamy ilość zapisanych bajtów lub błąd EIO jeśli jeszcze nic nie zapisaliśmy
    koniec jeśli
  koniec jeśli
  przesuwamy wskaźnik p do miejsca, od którego mamy pisać, przesuwamy się do następnego bloku i ustawiam pozycję w bloku na 0, uaktualniamy pozycje ppos, ilość zapisanych bajtów writen i ilość bajtów do zapisania count, zamazujemy starą zawartość bufora nowymi danymi, uaktualniamy wskaźniki do miejsc gdzie piszemy i skąd piszemy, modyfikujemy flagi dirty i uptodate bufora
  jeśli plik ma ustawioną flage O_SYNC to dopisujemy bufor do listy buforów w przeciwnym przypadku zwalniamy bufor koniec jeśli
  jeśli osiągneliśmy koniec tablicy buforów bufferlist to
    zapisujemy bufory z tablicy buforów bufferlist
    dla każdego bufora z listy buforów wykonuj
      czekamy na zakończenie operacji wejścia/wyjścia dla bufora
      jeśli bufor nie zaktualizowany to błąd, który zwrócimy później
      zwalniamy bufor
    koniec dla każdego
  koniec jeśli
  jeśli wystąpił błąd przy zapisie to przerywamy pętle
koniec dopóki
```

Algorytm 3 block_write - zakończenie

jeśli do tablicy buforów `bufferlist` coś zostało wpisane, `buffercount > 0` **to**
  zapisujemy bufory z tablicy buforów
  dla każdego bufora z listy buforów **wykonuj**
    czekamy na zakończenie operacji wejścia/wyjścia dla bufora
    jeśli bufor nie zaktualizowany **to** błąd, który zwrócimy później
    zwalniamy bufor
  koniec dla każdego
koniec jeśli
jeśli wystąpił wcześniej błąd przy zapisie **to** teraz go zwracamy
zwracamy ilość zapisanych bajtów, SUKCES !!!

3 Źródła

3.1 deklaracja funkcji block_write

funkcja `block_write` jest zadeklarowana w pliku include/linux/fs.h

```
#ifndef _LINUX_FS_H
...
#ifdef __KERNEL__
...
extern ssize_t block_write(struct file *, const char *, size_t, loff_t *);
...
```

3.2 definicja funkcji block_write

funkcja `block_write`[1] jest zdefiniowana w linux/fs/block_dev.c

```
ssize_t block_write(struct file * filp, const char * buf,
                    size_t count, loff_t *ppos)
{
/* ustalamy i-wezel pliku */
        struct inode * inode = filp->f_dentry->d_inode;
        ssize_t blocksize, blocksize_bits, i, buffercount, write_error;
        ssize_t block, blocks;
        loff_t offset;
        ssize_t chars;
        ssize_t written = 0;
        struct buffer_head * bhlist[NBUF];
        size_t size;
        kdev_t dev;
        struct buffer_head * bh, *bufferlist[NBUF];
        register char * p;

        write_error = buffercount = 0;
/* ustalamy numer urzadzenia na ktorym bedziemy pracowac */
        dev = inode->i_rdev;
```

[1] wersja oryginalna oczywiście nie posiada komentarzy po polsku, ona nie posiada ich prawie wcale

```
/* sprawdzamy czy nie jest to urzadzenie tylko do odczytu, jesli tak to blad dostepu */
        if ( is_read_only( inode->i_rdev ))
                return -EPERM;
/* BLOCK_SIZE zdefiniowane w include/linux/fs.h na 1024 */
        blocksize = BLOCK_SIZE;
/* sprawdzamy czy urzadzenie dev ma zdefiniwany rozmiar bloku  */
        if (blksize_size[MAJOR(dev)] && blksize_size[MAJOR(dev)][MINOR(dev)])
                blocksize = blksize_size[MAJOR(dev)][MINOR(dev)];
/* liczymy ilosc bitow na blok blocksize_bits=lg2(blocksize) */
        i = blocksize;
        blocksize_bits = 0;
        while(i != 1) {
                blocksize_bits++;
                i >>= 1;
        }
/* ustalamy blok i offset na podstawie pozycji */
        block = *ppos >> blocksize_bits;
        offset = *ppos & (blocksize-1);
/* ustalamy rozmiar urzadzenia jesli zdefiniowany jesli nie to ustalamy go na INT_MAX */
        if (blk_size[MAJOR(dev)])
                size = ((loff_t) blk_size[MAJOR(dev)][MINOR(dev)] << BLOCK_SIZE_BITS) >> blocksize_bits;
        else
                size = INT_MAX;
/* GŁÓWNA PĘTLA ZAPISU */
        while (count>0) {
                if (block >= size)
                        return written ? written : -ENOSPC;
/* ustalamy dlugosc nastepnej porcji danych do zapisu
i jesli jest za duza to ja obcinamy - reszta nastepnym razem */
                chars = blocksize - offset;
                if (chars > count)
                        chars=count;
#if 0
                /* get the buffer head */
                {
                        struct buffer_head * (*fn)(kdev_t, int, int) = getblk;
                        if (chars != blocksize)
                                fn = bread;
                        bh = fn(dev, block, blocksize);
                        if (!bh)
                                return written ? written : -EIO;
                        if (!buffer_uptodate(bh))
                                wait_on_buffer(bh);
                }
#else
/* pobieramy nowy bufor na dane */
                bh = getblk(dev, block, blocksize);
                if (!bh)
                        return written ? written : -EIO;
```

```
                if (!buffer_uptodate(bh))
                {
                  if (chars == blocksize)
                    wait_on_buffer(bh);
                  else
                  {
                    bhlist[0] = bh;
/* sprawdzamy czy możemy odczytywać z wyprzedzeniem */
                    if (!filp->f_reada || !read_ahead[MAJOR(dev)]) {
                      /* We do this to force the read of a single buffer */
/* jeśli nie to wymuszamy odczyt jednego bufora */
                      blocks = 1;
                    } else {
/* odczyt całej listy buforów */
                      /* Read-ahead before write */
                      blocks = read_ahead[MAJOR(dev)] / (blocksize >> 9) / 2;
                      if (block + blocks > size) blocks = size - block;
                      if (blocks > NBUF) blocks=NBUF;
                      if (!blocks) blocks = 1;
                      for(i=1; i<blocks; i++)
                      {
                        bhlist[i] = getblk (dev, block+i, blocksize);
                        if (!bhlist[i])
                        {
                          while(i >= 0) brelse(bhlist[i--]);
                          return written ? written : -EIO;
                        }
                      }
                    }
                    ll_rw_block(READ, blocks, bhlist);
                    for(i=1; i<blocks; i++) brelse(bhlist[i]);
                    wait_on_buffer(bh);
                    if (!buffer_uptodate(bh)) {
                          brelse(bh);
                          return written ? written : -EIO;
                    }
                  };
                };
#endif
                block++;
                p = offset + bh->b_data;
                offset = 0;
                *ppos += chars;
                written += chars;
                count -= chars;
/* zamazanie starej zawartości nową */
                copy_from_user(p,buf,chars);
                p += chars;
                buf += chars;
                mark_buffer_uptodate(bh, 1);
```

```
                mark_buffer_dirty(bh, 0);
                if (filp->f_flags & O_SYNC)
                        bufferlist[buffercount++] = bh;
                else
                        brelse(bh);
/* jeśli tablica buforów pełna */
                if (buffercount == NBUF){
/* żądamy zapisu tablicy buforów */
                        ll_rw_block(WRITE, buffercount, bufferlist);
/* dla każdego bufora czekamy na zakończenie operacji i zwalniamy bufor
jeśli wystąpił błąd to zwrucimy go później */
                        for(i=0; i<buffercount; i++){
                                wait_on_buffer(bufferlist[i]);
                                if (!buffer_uptodate(bufferlist[i]))
                                        write_error=1;
                                brelse(bufferlist[i]);
                        }
                        buffercount=0;
                }
/*  jeśli wystąpił błąd to przerywamy pętle */
                if(write_error)
                        break;
        }
/* KONIEC GŁÓWNEJ PĘTLI ZAPISU */

/* jeśli do tablicy bufferlist coś zostało wpisane, buffercount > 0 */
        if ( buffercount ){
/* to zapisujemy to */
                ll_rw_block(WRITE, buffercount, bufferlist);
/* czekamy na zakończenie operacji wejścia/wyjścia dla każdego bufora */
                for(i=0; i<buffercount; i++){
                        wait_on_buffer(bufferlist[i]);
/* jesli bufor niezaktualizowany to blad, który zwrócimy później */
                        if (!buffer_uptodate(bufferlist[i]))
                                write_error=1;
/* zwalniamy bufor, bezwzględnie */
                        brelse(bufferlist[i]);
                }
        }
/*  */
        filp->f_reada = 1;
/* jezeli wystąpił błąd to, teraz go zwracamy */
        if(write_error)
                return -EIO;
/* zwracamy ilosc zapisanych bajtów */
        return written;
}
```

3.3 typy

- `ssize_t`
  include/linux/types.h

  ```
  #ifndef _LINUX_TYPES_H
  ...
  #ifndef __KERNEL_STRICT_NAMES
  ...
  #ifndef _SSIZE_T
  #define _SSIZE_T
  typedef __kernel_ssize_t ssize_t;
  #endif
  ...
  ```

- `__kernel_ssize_t`
  include/asm-i386/posix_types.h

  ```
  #ifndef __ARCH_I386_POSIX_TYPES_H
  ...
  typedef int __kernel_ssize_t
  ...
  #endif
  ```

- `struct file`
  include/linux/fs.h

  ```
  #ifndef _LINUX_FS_H
  ...
  #ifdef __KERNEL__
  ...
  struct file {
      struct file             *f_next, **f_pprev;
      struct dentry           *f_dentry;
      struct file_operations  *f_op;
      mode_t                  f_mode;
      loff_t                  f_pos;
      unsigned int            f_count, f_flags;
      unsigned long           f_reada, f_ramax, f_raend, f_ralen, f_rawin;
      struct fown_struct      f_owner;
      unsigned int            f_uid, f_gid;
      int                     f_error;
      unsigned long           f_version;
      /* needed for tty driver, and maybe others */
      void                    *private_data;
  };
  ...
  ```

- `size_t`
  include/linux/types.h

```
#ifndef _LINUX_TYPES_H
...
#ifndef __KERNEL_STRICT_NAMES
...
#ifndef _SIZE_T
#define _SIZE_T
typedef __kernel_size_t size_t;
#endif
...
```

- `__kernel_size_t`
  include/asm-i386/posix_types.h

```
#ifndef __ARCH_I386_POSIX_TYPES_H
...
typedef unsigned int __kernel_size_t
...
#endif
```

- `loff_t`
  include/linux/types.h

```
#ifndef _LINUX_TYPES_H
...
#ifndef __KERNEL_STRICT_NAMES
...
#if defined(__GNUC__) && ! defined(__STRICT_ANSI__)
typedef __kernel_loff_t loff_t
#endif
...
```

- `__kernel_loff_t`
  include/asm-i386/posix_types.h

```
#ifndef __ARCH_I386_POSIX_TYPES_H
...
#ifdef __GNUC__
typedef long long __kernel_loff_t
#endif
...
```