



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский государственный технический университет имени
Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт по лабораторной работе 10 по курсу «Операционные системы»

Тема Буферизованный и небуферизованный ввод-вывод

Студент Морозов Д.В.

Группа ИУ7-62Б

Преподаватель Рязанова Н.Ю.

Москва — 2023 г.

1 Структуры

Листинг 1 – Структура FILE

```
1 typedef struct _IO_FILE FILE;
2 struct _IO_FILE
3 {
4     int _flags;      /* High-order word is _IO_MAGIC; rest is flags. */
5
6     /* The following pointers correspond to the C++ streambuf protocol.
7      */
8     char *_IO_read_ptr; /* Current read pointer */
9     char *_IO_read_end; /* End of get area. */
10    char *_IO_read_base; /* Start of putback+get area. */
11    char *_IO_write_base; /* Start of put area. */
12    char *_IO_write_ptr; /* Current put pointer. */
13    char *_IO_write_end; /* End of put area. */
14    char *_IO_buf_base; /* Start of reserve area. */
15    char *_IO_buf_end; /* End of reserve area. */
16
17    /* The following fields are used to support backing up and undo. */
18    char *_IO_save_base; /* Pointer to start of non-current get area. */
19    char *_IO_backup_base; /* Pointer to first valid character of
20        backup area */
21    char *_IO_save_end; /* Pointer to end of non-current get area. */
22
23    struct _IO_marker *_markers;
24
25    struct _IO_FILE *_chain;
26
27    int _fileno;
28    int _flags2;
29    __off_t _old_offset; /* This used to be _offset but it's too small.
30        */
31
32    /* 1+column number of pbase(); 0 is unknown. */
33    unsigned short _cur_column;
34    signed char _vtable_offset;
35    char _shortbuf[1];
36
37    _IO_lock_t *_lock;
38    #ifdef _IO_USE_OLD_IO_FILE
39
40    #endif
41 };
```

Листинг 2 – Структура stat

```
1 struct stat
2 {
3     /* These are the members that POSIX.1 requires. */
4
5     __mode_t st_mode;        /* File mode. */
6     #ifndef __USE_FILE_OFFSET64
7     __ino_t st_ino;          /* File serial number. */
8     #else
9     __ino64_t st_ino;         /* File serial number. */
10    #endif
11    __dev_t st_dev;           /* Device containing the file. */
12    __nlink_t st_nlink;       /* Link count. */
13
14    __uid_t st_uid;           /* User ID of the file's owner. */
15    __gid_t st_gid;           /* Group ID of the file's group. */
16    #ifndef __USE_FILE_OFFSET64
17    __off_t st_size;           /* Size of file, in bytes. */
18    #else
19    __off64_t st_size;         /* Size of file, in bytes. */
20    #endif
21
22    __time_t st_atime;         /* Time of last access. */
23    __time_t st_mtime;         /* Time of last modification. */
24    __time_t st_ctime;         /* Time of last status change. */
25
26    /* This should be defined if there is a 'st_blksize' member. */
27    #undef _STATBUF_ST_BLKSIZE
28 };
```

2 Первая программа

Листинг 3 – Код первой программы. Один поток

```
1 #include <stdio.h>
2 #include <fcntl.h>
3
4 #define GREEN "\033[01;38;05;46m"
5 #define BLUE  "\033[01;38;05;33m"
6 #define CLEAR "\033[0m"
7
8 int main(void)
9 {
10     int fd = open("alphabet.txt", O_RDONLY);
11
12     FILE *fs1 = fdopen(fd, "r");
13     char buff1[20];
14     setvbuf(fs1, buff1, _IOFBF, 20);
15
16     FILE *fs2 = fdopen(fd, "r");
17     char buff2[20];
18     setvbuf(fs2, buff2, _IOFBF, 20);
19
20     int flag1 = 1, flag2 = 1;
21
22     while (flag1 == 1 || flag2 == 1)
23     {
24         char c;
25
26         if ((flag1 = fscanf(fs1, "%c", &c)) == 1)
27             fprintf(stdout, GREEN "%c" CLEAR, c);
28
29         if ((flag2 = fscanf(fs2, "%c", &c)) == 1)
30             fprintf(stdout, BLUE "%c" CLEAR, c);
31     }
32
33     fprintf(stdout, "\n");
34     return 0;
35 }
```

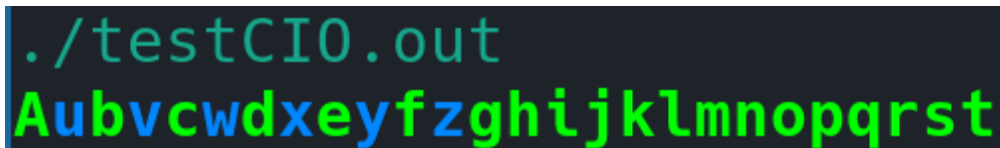


Рисунок 1 – Результат работы первой программы. Один поток

Листинг 4 – Код первой программы. Два потока

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <pthread.h>
4
5 #define GREEN "\033[01;38;05;46m"
6 #define BLUE  "\033[01;38;05;33m"
7 #define CLEAR "\033[0m"
8
9 struct args_struct { FILE * fs; char * color; };
10
11 void *read_buf(void *args)
12 {
13     struct args_struct *cur_args = (struct args_struct *) args;
14     FILE *fs = cur_args->fs;
15     char *color = cur_args->color;
16     int flag = 1;
17
18     while (flag == 1) {
19         char c;
20         if ((flag = fscanf(fs, "%c", &c)) == 1)
21             fprintf(stdout, "%s%c" CLEAR, color, c);
22     }
23     return NULL;
24 }
25
26 int main(void)
27 {
28     int fd = open("alphabet.txt", O_RDONLY);
29
30     FILE *fs1 = fdopen(fd, "r");
31     char buff1[20];
32     setvbuf(fs1, buff1, _IOFBF, 20);
33     struct args_struct args1 = { .fs = fs1, .color = GREEN };
34
35     FILE *fs2 = fdopen(fd, "r");
36     char buff2[20];
37     setvbuf(fs2, buff2, _IOFBF, 20);
38     struct args_struct args2 = { .fs = fs2, .color = BLUE };
39
40     pthread_t td;
41     pthread_create(&td, NULL, read_buf, &args2);
42     read_buf(&args1);
43     pthread_join(td, NULL);
44     puts("");
45     return 0;
46 }
```

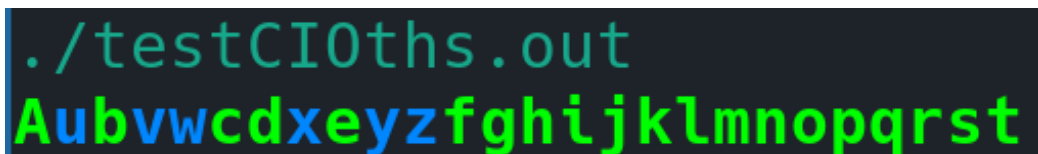


Рисунок 2 – Результат работы первой программы. Два потока

Анализ результата

Системный вызов `open()` открывает файл "alphabet.txt" только для чтения (`O_RDONLY`), создает дескриптор открытого файла, соответствующий индексу в таблице дескрипторов файлов, открытых процессом (массиве `fd_array` структуры `files_struct`). В данном случае файловому дескриптору присваивается значение 3, так как значения 0, 1, 2 заняты стандартными потоками ввода-вывода (`stdin`, `stdout`, `stderr`), а другие файлы процессом не открывались. Поле `fd_array[3]` указывает на `struct file`, связанную с `struct inode`, соответствующую файлу "alphabet.txt".

Два вызова `fdopen()` стандартной библиотеки создают структуры `FILE` (`fs1`, `fs2`), поле `_fileno` которых инициализируется файловым дескриптором, то есть значением 3.

Далее с помощью функции `setvbuf()` устанавливаются буферы для каждой из структур `FILE`, задавая указатели на начало и конец буфера (указатель на конец буфера рассчитывается через начало и размер буфера (20 байт), передаваемые в качестве параметров в функцию `setvbuf()`) и тип буферизации (в данном случае устанавливается полная буферизация).

В цикле поочередно для `fs1` и `fs2` вызывается функция `fscanf()` стандартной библиотеки. Так как была установлена полная буферизация, при первом вызове `fscanf()` буфер структуры `fs1` будет полностью заполнен, то есть в него сразу запишутся 20 символов (буквы от 'A' до 't'). При этом поле `f_pos` структуры `struct file` установится на следующий символ ('u'), а в переменную `c` запишется символ 'A', который и выведется на экран. При вызове `fscanf()` для `fs2` в ее буфер запишутся оставшиеся символы (от 'u' до 'z'), так как `fs2` ссылается на тот же дескриптор, что и структура `fs1`, а поле `f_pos` соответствующей структуры `struct file` было изменено. В переменную `c` запишется символ 'u'.

При следующих вызовах `fscanf()` переменной `c` будут присваиваться значения символов из буферов, и попеременно будут выводиться значения

из каждого буфера. Когда символы в одном из буферов кончатся, продолжится вывод символов только из одного буфера, а повторных заполнений производится не будет, так как файл был полностью прочитан при первом заполнении буфера `fs2`, что представлено на рисунке 13.

В случае многопоточной реализации возможны различные варианты вывода, в зависимости от того, какой поток первым вызовет `fscanf`. Возможна ситуация аналогичная однопоточному варианту, с той поправкой, что порядок вывода символов разными потоками может отличаться от запуска к запуску. Также возможно, что в одном из потоков дважды произойдет заполнение буфера и второй поток ничего не выведет.

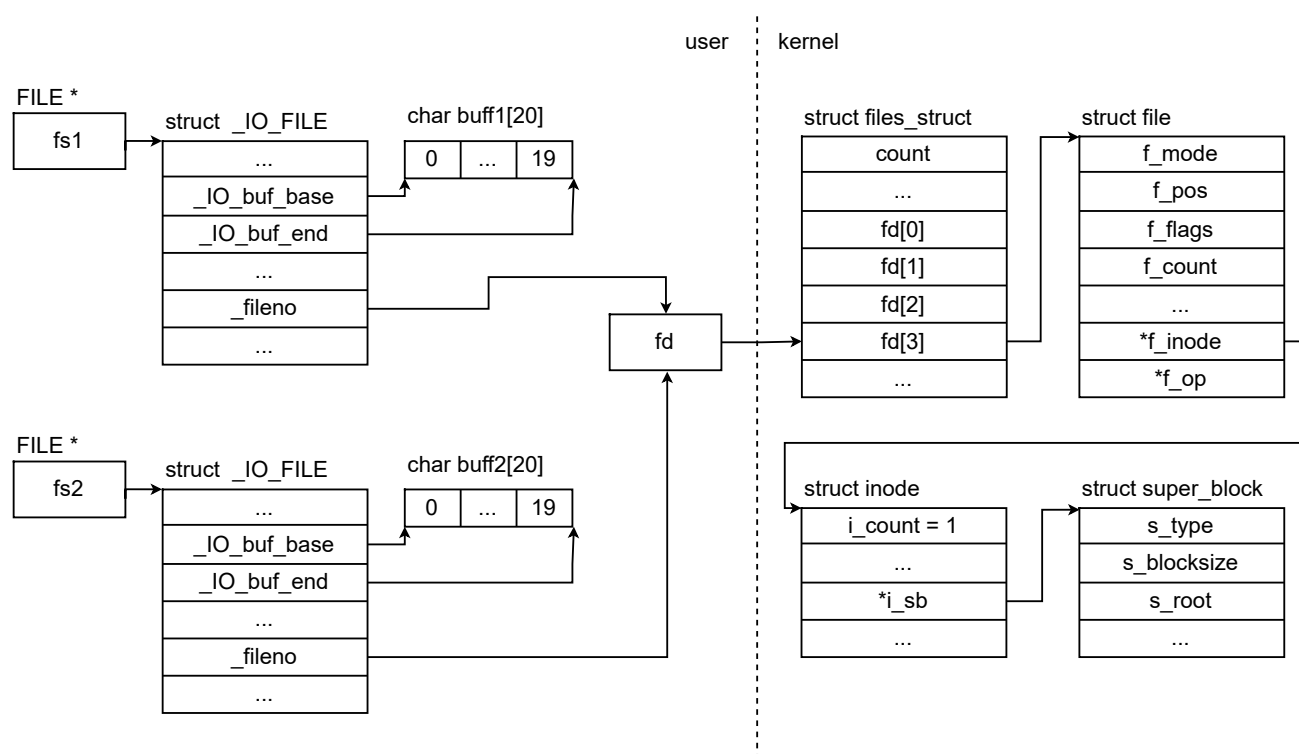
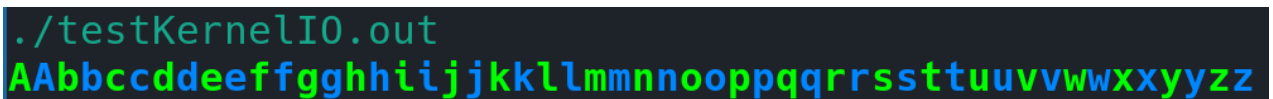


Рисунок 3 – Схема связей структур в первой программе

3 Вторая программа

Листинг 5 – Код второй программы. Один поток

```
1 #include <fcntl.h>
2 #include <unistd.h>
3 #include <stdio.h>
4
5 #define GREEN "\033[01;38;05;46m"
6 #define BLUE  "\033[01;38;05;33m"
7 #define CLEAR "\033[0m"
8
9 int main()
10 {
11     char c;
12     int fd1 = open("alphabet.txt", O_RDONLY);
13     int fd2 = open("alphabet.txt", O_RDONLY);
14     int flag1 = 1, flag2 = 1;
15
16     while(flag1 == 1 || flag2 == 1)
17     {
18         if ((flag1 = read(fd1, &c, 1)) == 1)
19             printf(GREEN "%c" CLEAR, c);
20
21         if ((flag2 = read(fd2, &c, 1)) == 1)
22             printf(BLUE "%c" CLEAR, c);
23     }
24
25     puts("");
26     return 0;
27 }
```



```
./testKernelIO.out
AAbbccddeeffgghhiijjkkllmmnnooppqrrssttuuvvwwxxyyzz
```

Рисунок 4 – Результат работы второй программы. Один поток

Листинг 6 – Код второй программы. Два потока

```
1 #include <fcntl.h>
2 #include <unistd.h>
3 #include <stdio.h>
4 #include <pthread.h>
5
6 #define GREEN "\033[01;38;05;46m"
7 #define BLUE  "\033[01;38;05;33m"
8 #define CLEAR "\033[0m"
9
10 pthread_mutex_t mutex;
11
12 struct args_struct { int fd; char * color; };
13
14 void *read_buf(void *args)
15 {
16     struct args_struct *cur_args = (struct args_struct *) args;
17     int fd = cur_args->fd;
18     char *color = cur_args->color;
19     int flag = 1;
20
21     while (flag == 1)
22     {
23         char c;
24         if ((flag = read(fd, &c, 1)) == 1)
25             printf("%s%c" CLEAR, color, c);
26     }
27     return NULL;
28 }
29
30 int main()
31 {
32     char c;
33     int fd1 = open("alphabet.txt", O_RDONLY);
34     struct args_struct args1 = { .fd = fd1, .color = GREEN };
35
36     int fd2 = open("alphabet.txt", O_RDONLY);
37     struct args_struct args2 = { .fd = fd2, .color = BLUE };
38
39     pthread_t td;
40     pthread_create(&td, NULL, read_buf, &args2);
41
42     read_buf(&args1);
43
44     pthread_join(td, NULL);
45     puts("");
46     return 0;
47 }
```

```
./testKernelIOths.out
AAbbccddeffeghfifgikhiljkmnlmnoopqrpsqtuvrwsxtzyuvwxyz
```

Рисунок 5 – Результат работы второй программы. Два потока

Анализ результата

В данной программе с помощью двух вызовов `open()` файл 'alphabet.txt' дважды открывается только для чтения, и создаются два дескриптора открытого файла (им присваиваются значения 3 и 4). При этом создаются две различные структуры `struct file`, ссылающиеся на одну и ту же структуру `struct inode`. Так как структуры `struct file` разные и их поля `f_pos` изменяются независимо, то для каждого файлового дескриптора произойдет полное чтение файла и каждый символ будет выведен два раза.

При многопоточной реализации алфавит также будет выведен два раза, однако порядок вывода символов при этом неизвестен.

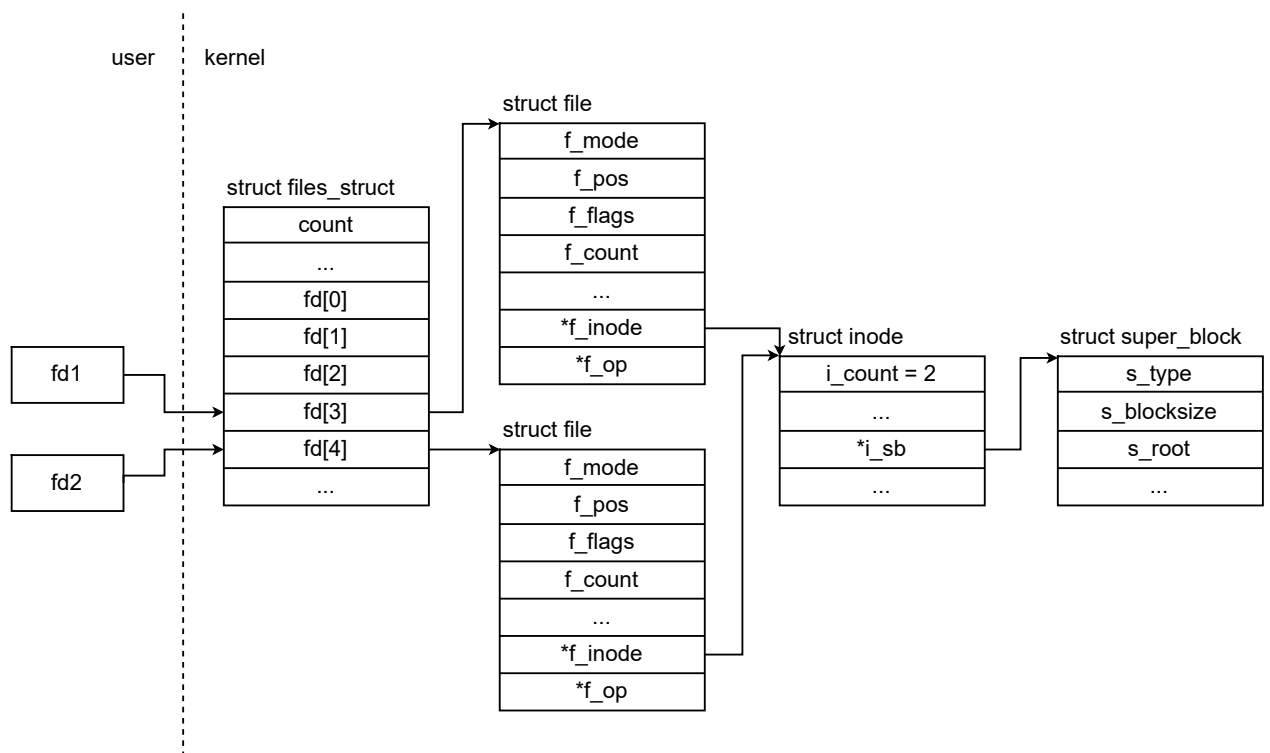


Рисунок 6 – Схема связей структур в второй программе

4 Третья программа

Листинг 7 – Код третьей программы. Один поток

```
1 #include <stdio.h>
2 #include <sys/stat.h>
3
4 #define CLEAR "\033[0m"
5
6 void fileInfo(FILE *fs)
7 {
8     struct stat statbuf;
9     stat("result.txt", &statbuf);
10    printf("\033[38;05;214minode:_%ld\n", statbuf.st_ino);
11    printf("Total_size:_%ld\n", statbuf.st_size);
12    printf("Current_pos:_%ld\n\n" CLEAR, ftell(fs));
13 }
14
15 int main(void)
16 {
17     FILE *fs1 = fopen("result.txt", "w");
18     fileInfo(fs1);
19
20     FILE *fs2 = fopen("result.txt", "w");
21     fileInfo(fs2);
22
23     for (char ch = 'a'; ch <= 'z'; ++ch) {
24         fprintf(ch % 2 ? fs1 : fs2, "%c", ch);
25         printf("%d\n", (int) ch);
26     }
27
28
29     fileInfo(fs1);
30     fclose(fs1);
31     fileInfo(fs1);
32
33     fileInfo(fs2); //26
34     fclose(fs2);
35     fileInfo(fs2);
36
37     return 0;
38 }
```

Анализ результата

В данной программе с помощью функции `fopen()` стандартной библиотеки файл дважды открывается для записи. Так же как и во второй

```
./testWrite.out  
bdfhjlnprtvxz
```

Рисунок 7 – Результат работы третьей программы. fs2 закрывается последним

```
./testWrite.out  
acegikmoqsuwy
```

Рисунок 8 – Результат работы третьей программы. fs1 закрывается последним

```
inode: 4196184  
Общий размер в байтах: 0  
Текущая позиция: 0  
  
inode: 4196184  
Общий размер в байтах: 0  
Текущая позиция: 0  
  
inode: 4196184  
Общий размер в байтах: 0  
Текущая позиция: 13  
  
inode: 4196184  
Общий размер в байтах: 13  
Текущая позиция: -1  
  
inode: 4196184  
Общий размер в байтах: 13  
Текущая позиция: 13  
  
inode: 4196184  
Общий размер в байтах: 13  
Текущая позиция: -1
```

Рисунок 9 – Информация о состоянии открытых файлов

программе создаются два файловых дескриптора (со значениями 3 и 4), на которые ссылаются структуры `FILE` (`fs1`, `fs2`). По умолчанию используется полная буферизация, при которой запись в файл из буфера происходит либо когда буфер заполнен, либо когда вызывается `fflush` или `fclose`.

В данном случае запись в файл происходит при вызове `fclose()`. До вызовов `fclose()` в цикле в файл записываются буквы латинского алфавита с помощью передачи функции `fprintf()` то одного дескриптора, то другого.

При вызове `fclose(fs1)` нечетные буквы алфавита записываются в файл. При вызове `fclose(fs2)`, так как поле `f_pos` соответствующей структуры `struct file` не изменялось, запись в файл произойдет с начала фай-

ла, и записанные ранее символы перезапишутся новыми данными (четными буквами алфавита), что и показано на рисунке 7. Так как буферы содержали одинаковое количество символов, данные первой записи были полностью утеряны. Если бы количество символов при второй записи было меньше, чем при первой, то последние символы первой записи сохранились бы.

Если бы сначала был вызов `fclose(fs2)`, а потом `fclose(fs1)`, то в файл записались нечетные буквы алфавита (рисунок 8).

Листинг 8 – Код третьей программы. Два потока

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <sys/stat.h>
4
5 #define GREEN "\033[01;38;05;46m"
6 #define BLUE  "\033[01;38;05;33m"
7 #define CLEAR "\033[0m"
8
9 struct args_struct { char begin; char * color; };
10
11 void *write_syms(void *args)
12 {
13     FILE *fs = fopen("resultths.txt", "a");
14
15     struct args_struct *cur_args = (struct args_struct *) args;
16     char begin = cur_args->begin;
17     char *color = cur_args->color;
18
19     for (char ch = begin; ch <= 'z'; ch += 2)
20         fprintf(fs, "%s%c" CLEAR, color, ch);
21
22     fclose(fs);
23     return NULL;
24 }
25
26 int main(void)
27 {
28     struct args_struct args1 = { .begin = 'a', .color = GREEN };
29     struct args_struct args2 = { .begin = 'b', .color = BLUE };
30
31     pthread_t td;
32     pthread_create(&td, NULL, write_syms, &args2);
33
34     write_syms(&args1);
35 }
```

```

36     pthread_join(td, NULL);
37     return 0;
38 }

```

```

./testWrite.out
bdfhjlnprtvxz

```

Рисунок 10 – Результат работы третьей программы.
Последним вызывается fclose в вспомогательном потоке

```

./testWrite.out
acegikmoqsuwy

```

Рисунок 11 – Результат работы третьей программы.
Последним вызывается fclose в главном потоке

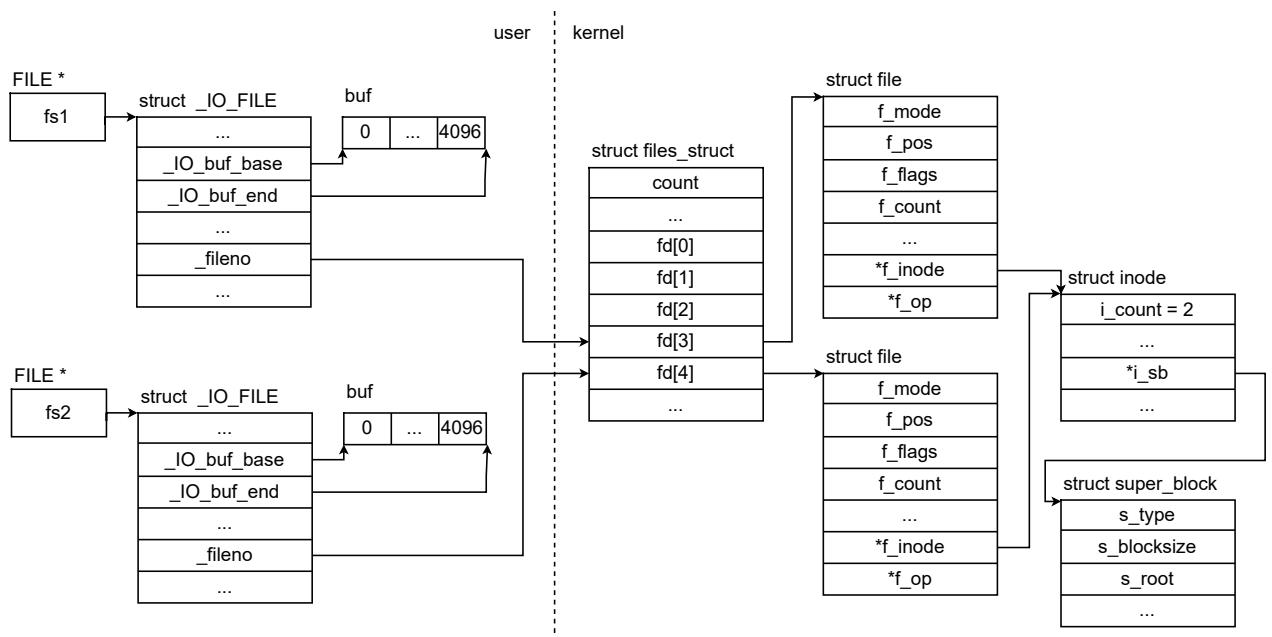


Рисунок 12 – Схема связей структур в третьей программе

5 Четвертая программа

Листинг 9 – Код четвертой программы. Один поток

```

1  #include <fcntl.h>
2  #include <stdio.h>
3  #include <sys/stat.h>
4
5  struct stat statbuf;

```

```

6
7 #define PRINT_STAT(action) \
8     do { \
9         stat("q.txt", &statbuf); \
10        fprintf(stdout, action ":\tinode\tnumber\t\t%ld,\t\
11        size\t\t%ld\tbytes,\tblksize\t\t%ld\n", \
12            statbuf.st_ino, statbuf.st_size, \
13            statbuf.st_blksize); \
14    } while (0)
15
16 int main()
17 {
18     int fd1 = open("q.txt", O_CREAT | O_WRONLY);
19     PRINT_STAT("open\tfd1");
20     int fd2 = open("q.txt", O_CREAT | O_WRONLY);
21     PRINT_STAT("open\tfd2");
22     for (char c = 'a'; c <= 'z'; c++)
23     {
24         c % 2 ? write(fd1, &c, 1) : write(fd2, &c, 1);
25         PRINT_STAT("write");
26     }
27     close(fd1);
28     PRINT_STAT("close\tfd1");
29     close(fd2);
30     PRINT_STAT("close\tfd2");
31     return 0;
32 }

```

В программе файл дважды открывается на запись функцией `open()`. В системной таблице открытых файлов создаётся два дескриптора `struct file`, каждый из которых имеет собственный указатель `f_pos`, но оба ссылаются на один и тот же `inode`. С помощью системного вызова `write()` выполняется небуферизованный вывод. При изменении порядка вызова функций `close()` вывод программы не изменяется, так как вывод не буферизуется.

```

dmitriy@VivoBook:~/bmstu/SEM_6/OS/bmstu-sem6-os/lab_10_io/src$ cat q.txt
bdfhjlnprtvxzndmitriy@VivoBook:~/bmstu/SEM_6/OS/bmstu-sem6-os/lab_10_io/src$ ./testWriteK.out
open fd1: inode number = 54669156,          size = 0 bytes, blksize = 4096
open fd2: inode number = 54669156,          size = 0 bytes, blksize = 4096
write: inode number = 54669156,             size = 1 bytes, blksize = 4096
write: inode number = 54669156,             size = 1 bytes, blksize = 4096
write: inode number = 54669156,             size = 2 bytes, blksize = 4096
write: inode number = 54669156,             size = 2 bytes, blksize = 4096
write: inode number = 54669156,             size = 3 bytes, blksize = 4096
write: inode number = 54669156,             size = 3 bytes, blksize = 4096
write: inode number = 54669156,             size = 4 bytes, blksize = 4096
write: inode number = 54669156,             size = 4 bytes, blksize = 4096
write: inode number = 54669156,             size = 5 bytes, blksize = 4096
write: inode number = 54669156,             size = 5 bytes, blksize = 4096
write: inode number = 54669156,             size = 6 bytes, blksize = 4096
write: inode number = 54669156,             size = 6 bytes, blksize = 4096
write: inode number = 54669156,             size = 7 bytes, blksize = 4096
write: inode number = 54669156,             size = 7 bytes, blksize = 4096
write: inode number = 54669156,             size = 8 bytes, blksize = 4096
write: inode number = 54669156,             size = 8 bytes, blksize = 4096
write: inode number = 54669156,             size = 9 bytes, blksize = 4096
write: inode number = 54669156,             size = 9 bytes, blksize = 4096
write: inode number = 54669156,             size = 10 bytes, blksize = 4096
write: inode number = 54669156,             size = 10 bytes, blksize = 4096
write: inode number = 54669156,             size = 11 bytes, blksize = 4096
write: inode number = 54669156,             size = 11 bytes, blksize = 4096
write: inode number = 54669156,             size = 12 bytes, blksize = 4096
write: inode number = 54669156,             size = 12 bytes, blksize = 4096
write: inode number = 54669156,             size = 13 bytes, blksize = 4096
write: inode number = 54669156,             size = 13 bytes, blksize = 4096
close fd1: inode number = 54669156,         size = 13 bytes, blksize = 4096
close fd2: inode number = 54669156,         size = 13 bytes, blksize = 4096
dmitriy@VivoBook:~/bmstu/SEM_6/OS/bmstu-sem6-os/lab_10_io/src$ cat q.txt
bdfhjlnprtvxzndmitriy@VivoBook:~/bmstu/SEM_6/OS/bmstu-sem6-os/lab_10_io/src$ █

```

Рисунок 13 – Результат работы четвертой программы. Один поток

Листинг 10 – Код четвертой программы. Два потока

```

1 #include <stdlib.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #include <pthread.h>
5 #include <stdio.h>
6 #include <sys/stat.h>
7
8 struct stat statbuf;
9
10 #define PRINT_STAT(action) \
11     do { \
12         stat("q.txt", &statbuf); \
13         fprintf(stdout, action ":\tinode\tnumber\t\t%ld,\t\
14         size\t\t%ld\tbytes,\t\tblksize\t\t%ld\n", \
15             statbuf.st_ino, statbuf.st_size, \
16             statbuf.st_blksize); \
17     } while (0)
18
19 void *thread_func(void *fd)
20 {
21     for (char c = 'a'; c <= 'm'; c++)
22     {
23         write((int)fd, &c, 1);
24         PRINT_STAT("write");
25     }
26
27     return NULL;
28 }
29
30 void *thread_func2(void *fd)
31 {
32     for (char c = 'n'; c <= 'z'; c++)
33     {
34         write((int)fd, &c, 1);
35         PRINT_STAT("write");
36     }
37
38     return NULL;
39 }
40
41 int main(void)
42 {
43     int fd[2] = {open("q.txt", O_CREAT | O_WRONLY),
44                 open("q.txt", O_CREAT | O_WRONLY)};
45     pthread_t threads[2];
46
47     void *(*thread_funcs[2])(void *)

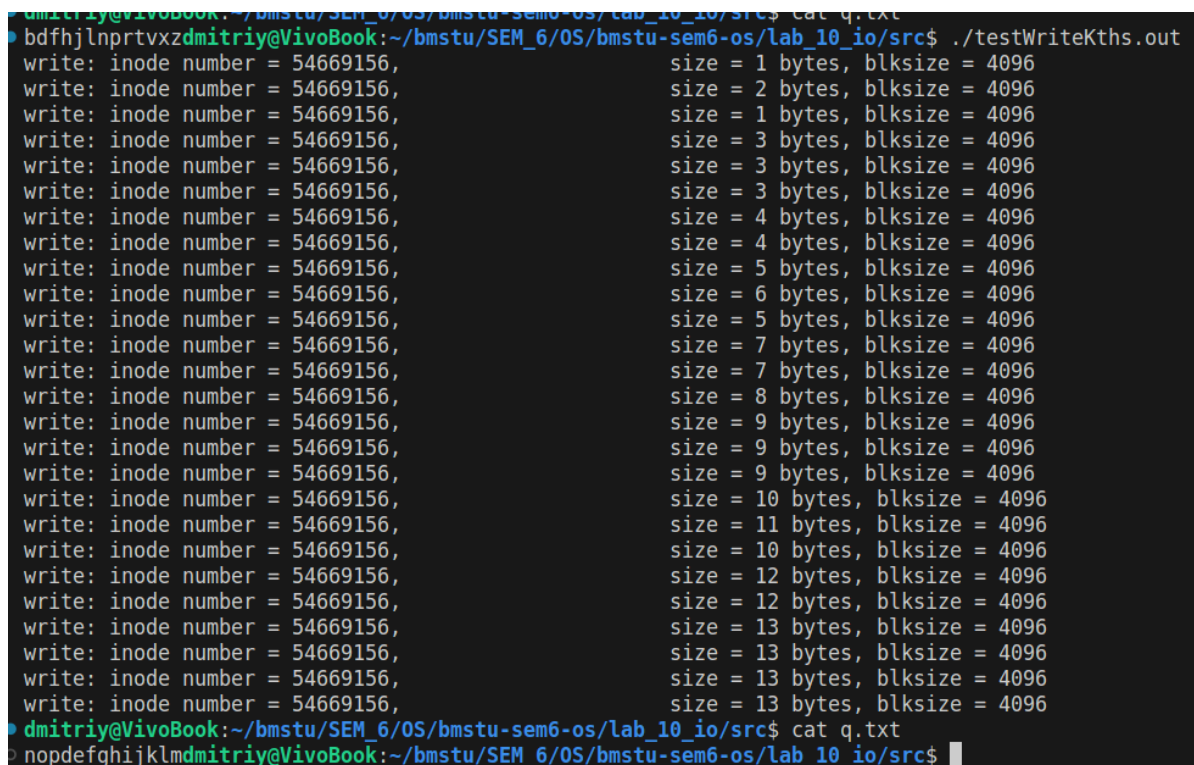
```

```

48         = {thread_func, thread_func2};
49
50     for (size_t i = 0; i < 2; i++)
51         if (pthread_create(&threads[i], NULL,
52             thread_funcs[i], fd[i]) != 0)
53         {
54             perror("pthread_create\n");
55             exit(1);
56         }
57
58     for (size_t i = 0; i < 2; i++)
59         if (pthread_join(threads[i], NULL) != 0)
60         {
61             perror("pthread_join\n");
62             exit(1);
63         }
64
65     close(fd[0]);
66     close(fd[1]);
67
68     return 0;
69 }

```

При двух потоках ситуация аналогична предыдущей программе, только состав символов может меняться от запуска к запуску.



```

dmitriy@VivoBook:~/bmstu/SEM_6/OS/bmstu-sem6-os/lab_10_io/src$ cat q.txt
• bdfhjlnprtvxz
dmitriy@VivoBook:~/bmstu/SEM_6/OS/bmstu-sem6-os/lab_10_io/src$ ./testWriteKths.out
write: inode number = 54669156, size = 1 bytes, blksize = 4096
write: inode number = 54669156, size = 2 bytes, blksize = 4096
write: inode number = 54669156, size = 1 bytes, blksize = 4096
write: inode number = 54669156, size = 3 bytes, blksize = 4096
write: inode number = 54669156, size = 3 bytes, blksize = 4096
write: inode number = 54669156, size = 3 bytes, blksize = 4096
write: inode number = 54669156, size = 4 bytes, blksize = 4096
write: inode number = 54669156, size = 4 bytes, blksize = 4096
write: inode number = 54669156, size = 5 bytes, blksize = 4096
write: inode number = 54669156, size = 6 bytes, blksize = 4096
write: inode number = 54669156, size = 5 bytes, blksize = 4096
write: inode number = 54669156, size = 7 bytes, blksize = 4096
write: inode number = 54669156, size = 7 bytes, blksize = 4096
write: inode number = 54669156, size = 8 bytes, blksize = 4096
write: inode number = 54669156, size = 9 bytes, blksize = 4096
write: inode number = 54669156, size = 9 bytes, blksize = 4096
write: inode number = 54669156, size = 9 bytes, blksize = 4096
write: inode number = 54669156, size = 10 bytes, blksize = 4096
write: inode number = 54669156, size = 11 bytes, blksize = 4096
write: inode number = 54669156, size = 10 bytes, blksize = 4096
write: inode number = 54669156, size = 12 bytes, blksize = 4096
write: inode number = 54669156, size = 12 bytes, blksize = 4096
write: inode number = 54669156, size = 13 bytes, blksize = 4096
write: inode number = 54669156, size = 13 bytes, blksize = 4096
write: inode number = 54669156, size = 13 bytes, blksize = 4096
write: inode number = 54669156, size = 13 bytes, blksize = 4096
dmitriy@VivoBook:~/bmstu/SEM_6/OS/bmstu-sem6-os/lab_10_io/src$ cat q.txt
• nopdefghijklm
dmitriy@VivoBook:~/bmstu/SEM_6/OS/bmstu-sem6-os/lab_10_io/src$

```

Рисунок 14 – Результат работы четвертой программы. Два потока

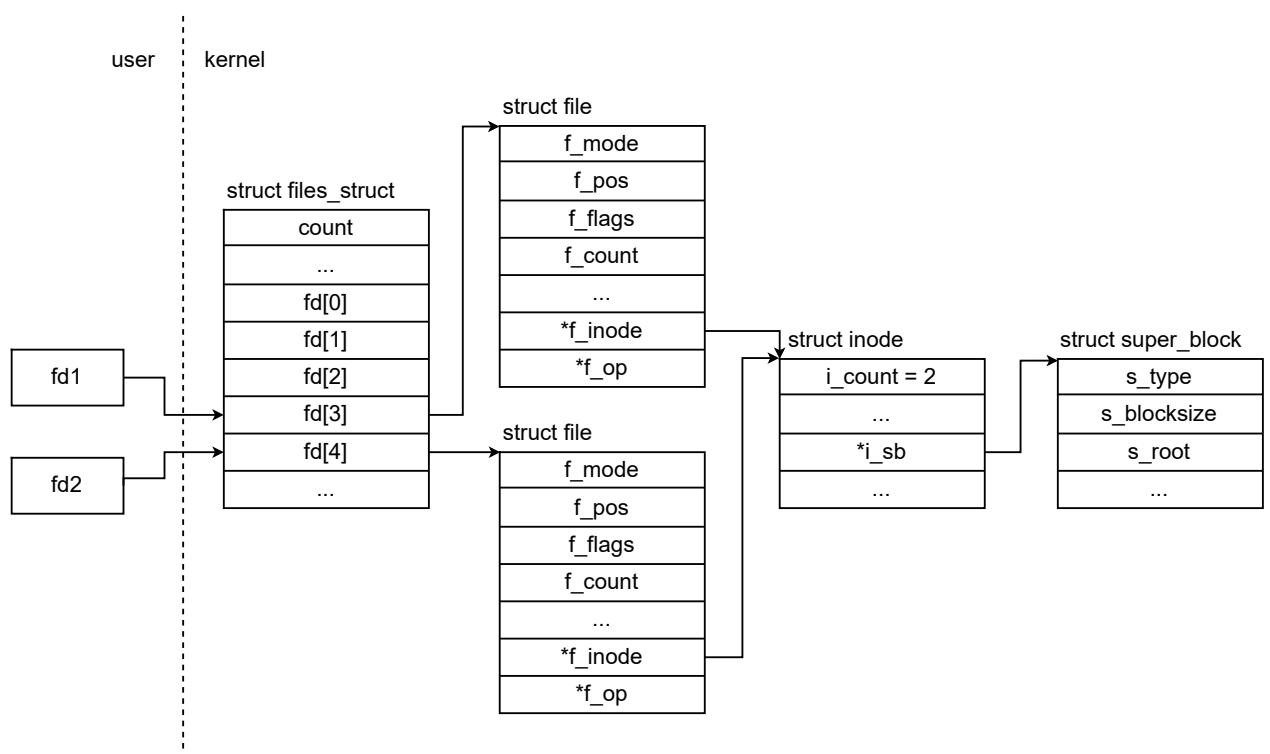


Рисунок 15 – Схема связей структур в четвертой программе

Заключение

Таким образом, при буферизованном вводе-выводе все данные и при чтении, и при записи пишутся сначала в буфер, а только после этого в файл. При отсутствии буферизации данные сразу пишутся в/читаются из файла.

В программе с *fopen()*, *fprintf()* (с буферизацией) информация записывается из буфера в файл при вызове *fclose()*. Поэтому размер файла до вызова *fclose()* по данным из *stat()* равен 0.

В программе с *open()*, *write()* (без буферизации) информация записывается в файл при вызове *write()*. Поэтому размер файла до вызова *fclose()* по данным из *stat()* равен 13 — столько символов в итоге останется в файле.