



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»

---

## ОТЧЕТ

по Лабораторной работе №5

по курсу «Операционные системы»

на тему: «Буферизованный и не буферизованный ввод-вывод»

Студент ИУ7-64Б  
(Группа)

\_\_\_\_\_  
(Подпись, дата)

Жабин Д. В.  
(И. О. Фамилия)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

Рязанова Н. Ю.  
(И. О. Фамилия)

2022 г.

# 1 Структура FILE

Описание структуры FILE приведено в Листинге 1.1

Листинг 1.1 – Файл /usr/include/bits/types/FILE.h

```
1 #ifndef __FILE_defined
2 #define __FILE_defined 1
3
4 struct _IO_FILE;
5
6 /* The opaque type of streams. This is the definition used elsewhere. */
7 typedef struct _IO_FILE FILE;
8
9 #endif
```

Описание структуры \_IO\_FILE приведено в Листингах 1.2 – 1.4

Листинг 1.2 – Файл /usr/include/bits/types/struct\_FILE.h, Часть 1

```
1 /* Copyright (C) 1991-2022 Free Software Foundation, Inc.
2    This file is part of the GNU C Library.
3    The GNU C Library is free software; you can redistribute it and/or
4    modify it under the terms of the GNU Lesser General Public
5    License as published by the Free Software Foundation; either
6    version 2.1 of the License, or (at your option) any later version.
7    The GNU C Library is distributed in the hope that it will be useful,
8    but WITHOUT ANY WARRANTY; without even the implied warranty of
9    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
10   Lesser General Public License for more details.
11   You should have received a copy of the GNU Lesser General Public
12   License along with the GNU C Library; if not, see
13   <https://www.gnu.org/licenses/>. */
14
15 #ifndef __struct_FILE_defined
16 #define __struct_FILE_defined 1
17
18 /* Caution: The contents of this file are not part of the official
19    stdio.h API. However, much of it is part of the official *binary*
20    interface, and therefore cannot be changed. */
21
22 #if defined _IO_USE_OLD_IO_FILE && !defined _LIBC
23 # error "_IO_USE_OLD_IO_FILE should only be defined when building libc itself"
24 #endif
25
26 #if defined _IO_lock_t_defined && !defined _LIBC
27 # error "_IO_lock_t_defined should only be defined when building libc itself"
28 #endif
29
30 #include <bits/types.h>
```

## Листинг 1.3 – Файл /usr/include/bits/types/struct\_FILE.h, Часть 2

```

33 struct _IO_marker;
34 struct _IO_codecvt;
35 struct _IO_wide_data;
36
37 /* During the build of glibc itself, _IO_lock_t will already have been
38    defined by internal headers.  */
39 #ifndef _IO_lock_t_defined
40 typedef void _IO_lock_t;
41 #endif
42
43 /* The tag name of this struct is _IO_FILE to preserve historic
44    C++ mangled names for functions taking FILE* arguments.
45    That name should not be used in new code.  */
46 struct _IO_FILE
47 {
48     int _flags;          /* High-order word is _IO_MAGIC; rest is flags.  */
49
50     /* The following pointers correspond to the C++ streambuf protocol.  */
51     char *_IO_read_ptr;  /* Current read pointer */
52     char *_IO_read_end;  /* End of get area.  */
53     char *_IO_read_base; /* Start of putback+get area.  */
54     char *_IO_write_base; /* Start of put area.  */
55     char *_IO_write_ptr; /* Current put pointer.  */
56     char *_IO_write_end; /* End of put area.  */
57     char *_IO_buf_base;  /* Start of reserve area.  */
58     char *_IO_buf_end;   /* End of reserve area.  */
59
60     /* The following fields are used to support backing up and undo.  */
61     char *_IO_save_base; /* Pointer to start of non-current get area.  */
62     char *_IO_backup_base; /* Pointer to first valid character of backup area */
63     char *_IO_save_end; /* Pointer to end of non-current get area.  */
64
65     struct _IO_marker *_markers;
66
67     struct _IO_FILE *_chain;
68
69     int _fileno;
70     int _flags2;
71     __off_t _old_offset; /* This used to be _offset but it's too small.  */
72
73     /* 1+column number of pbase(); 0 is unknown.  */
74     unsigned short _cur_column;
75     signed char _vtable_offset;
76     char _shortbuf[1];
77
78     _IO_lock_t *_lock;
79 #ifdef _IO_USE_OLD_IO_FILE
80 };
81
82 struct _IO_FILE_complete
83 {
84     struct _IO_FILE _file;
85 #endif
86     __off64_t _offset;
87     /* Wide character stream stuff.  */
88     struct _IO_codecvt *_codecvt;

```

Листинг 1.4 – Файл /usr/include/bits/types/struct\_FILE.h, Часть 3

```

89  struct _IO_wide_data *_wide_data;
90  struct _IO_FILE *_freeres_list;
91  void *_freeres_buf;
92  size_t __pad5;
93  int _mode;
94  /* Make sure we don't get into trouble again. */
95  char _unused2[15 * sizeof (int) - 4 * sizeof (void *) - sizeof (size_t)];
96 };
97
98 /* These macros are used by bits/stdio.h and internal headers. */
99 #define __getc_unlocked_body(_fp) \
100  (__glibc_unlikely ((_fp)->_IO_read_ptr >= (_fp)->_IO_read_end) \
101   ? __uflow (_fp) : *(unsigned char *) (_fp)->_IO_read_ptr++)
102
103 #define __putc_unlocked_body(_ch, _fp) \
104  (__glibc_unlikely ((_fp)->_IO_write_ptr >= (_fp)->_IO_write_end) \
105   ? __overflow (_fp, (unsigned char) (_ch)) \
106   : (unsigned char) (*( _fp)->_IO_write_ptr++ = (_ch)))
107
108 #define _IO_EOF_SEEN 0x0010
109 #define __feof_unlocked_body(_fp) (((_fp)->_flags & _IO_EOF_SEEN) != 0)
110
111 #define _IO_ERR_SEEN 0x0020
112 #define __ferror_unlocked_body(_fp) (((_fp)->_flags & _IO_ERR_SEEN) != 0)
113
114 #define _IO_USER_LOCK 0x8000
115 /* Many more flag bits are defined internally. */
116
117 #endif

```

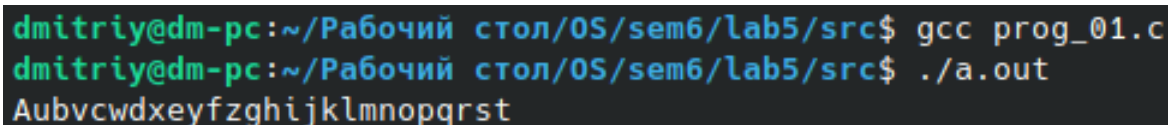
## 2 Реализация программ

### 2.1 Первая программа

#### 2.1.1 Коды программ

Листинг 2.1 – Открытие одного и того же файла несколько раз для чтения в одном потоке

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



```
dmitriy@dm-pc:~/Рабочий стол/OS/sem6/lab5/src$ gcc prog_01.c
dmitriy@dm-pc:~/Рабочий стол/OS/sem6/lab5/src$ ./a.out
Aubvcwdxeyfzghijklmnopqrst
```

Рисунок 2.1 – Программа с одним потоком

## Листинг 2.2 – Открытие одного и того же файла для чтения в двух потоках

```
1 #include <fcntl.h>
2 #include <pthread.h>
3 #include <stdio.h>
4 #include <unistd.h>
5
6 #define BUF_SIZE 20
7
8 void *read_buffer(void *args)
9 {
10     int fd = *(int*)args;
11     FILE *fs2 = fdopen(fd, "r");
12
13     char buff2[BUF_SIZE];
14     setvbuf(fs2, buff2, _IOFBF, BUF_SIZE);
15
16     int flag = 1;
17     char c;
18
19     while ((flag = fscanf(fs2, "%c", &c)) == 1)
20         fprintf(stdout, "%c", c);
21 }
22
23 int main(void)
24 {
25     setbuf(stdout, NULL);
26
27     pthread_t thread;
28     int fd = open("data.txt", O_RDONLY);
29
30     FILE *fs1 = fdopen(fd, "r");
31     char buff1[BUF_SIZE];
32     setvbuf(fs1, buff1, _IOFBF, BUF_SIZE);
33
34     pthread_create(&thread, NULL, read_buffer, (void *)&fd);
35     char c;
36     int flag;
37
38     while ((flag = fscanf(fs1, "%c", &c)) == 1)
39         fprintf(stdout, "%c", c);
40
41     pthread_join(thread, NULL);
42     fprintf(stdout, "\n");
43     return 0;
44 }
```

```
dmitriy@dm-pc:~/Рабочий стол/OS/sem6/lab5/src$ gcc prog_01_thread.c -lpthread
dmitriy@dm-pc:~/Рабочий стол/OS/sem6/lab5/src$ ./a.out
Abcdefghijklmnopqrstuvwxyz
```

Рисунок 2.2 – Программа с двумя потоками

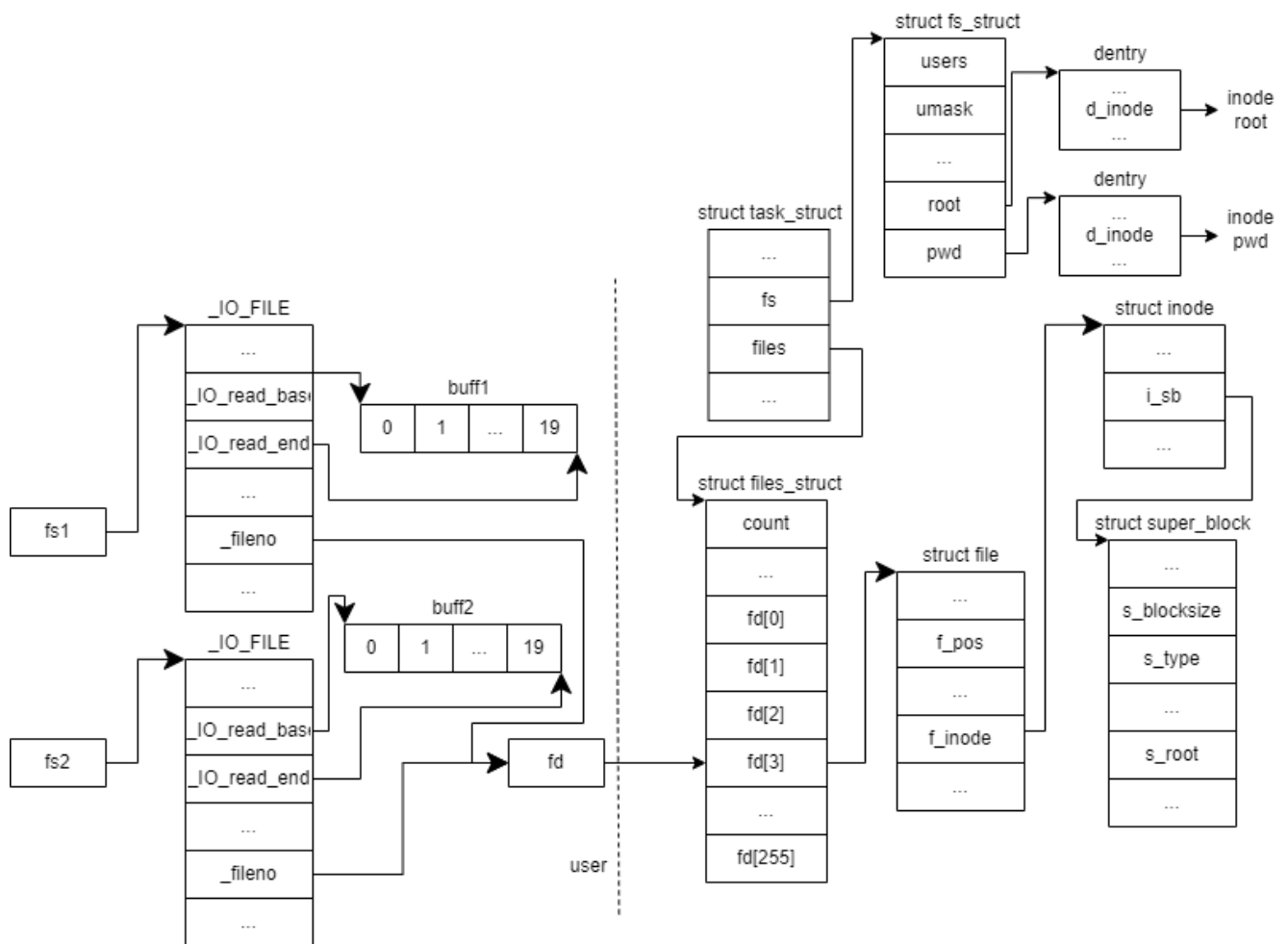


Рисунок 2.3 – Схема взаимодействия структур ядра

## 2.1.2 Схема взаимодействия структур ядра

### Выводы

- в результате вызова функции `open()` создается дескриптор файла в таблице открытых файлов процесса и дескриптор в системной таблице открытых файлов. В созданной структуре `struct file` хранится смещение в файле и флаги состояния файла;
- в результате двух вызовов функции `fdopen()` создаются два указателя на структуру `_IO_FILE`. Полю `_fileno` присваивается значение дескриптора, который вернула функция `open()`;
- функция `setvbuf()` явно задает размер буфера в 20 байт и меняет тип буферизации (для `fs1` и `fs2`) на полную;
- при первом вызове функции `fscanf()` в цикле (для `fs1`), `buff1` будет

заполнен первыми 20 символами (буквами алфавита). `f_pos` в структуре `struct file` открытого файла увеличится на 20;

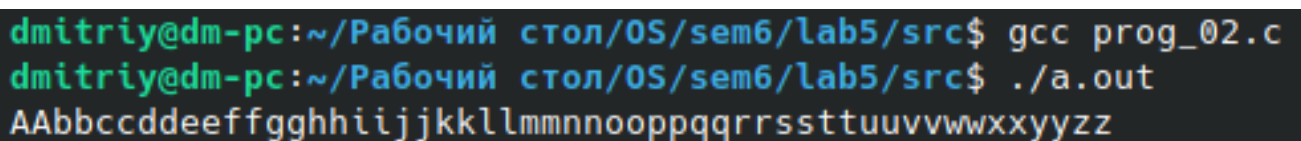
- при втором вызове `fscanf()` в цикле (для `fs2`) буфер `buff2` будет заполнен оставшимися 6 символами (начиная с `f_pos`);
- в цикле поочередно выводятся символы из `buff1` и `buff2`;
- в случае многопоточной реализации в потоке, который первым получит квант, первый вызов `fscanf()` заполнит буфер целиком и увеличит `f_pos` в структуре `struct file` открытого файла, а оставшиеся 6 символов будут записаны в буфер в другом потоке.

## 2.2 Вторая программа

### 2.2.1 Коды программ

Листинг 2.3 – Открытие одного и того же файла несколько раз для чтения в одном потоке

```
1 #include <fcntl.h>
2 #include <unistd.h>
3
4 int main(void)
5 {
6     int fd1 = open("data.txt", O_RDONLY);
7     int fd2 = open("data.txt", O_RDONLY);
8     int rc1, rc2 = 1;
9     char c;
10
11     while (rc1 == 1 || rc2 == 1)
12     {
13         if ((rc1 = read(fd1, &c, 1)) == 1) write(1, &c, 1);
14         if ((rc2 = read(fd2, &c, 1)) == 1) write(1, &c, 1);
15     }
16     c = '\n';
17     write(1, &c, 1);
18     return 0;
19 }
```



```
dmitriy@dm-pc:~/Рабочий стол/OS/sem6/lab5/src$ gcc prog_02.c
dmitriy@dm-pc:~/Рабочий стол/OS/sem6/lab5/src$ ./a.out
AAbbccddeeffgghhiijjkkllmmnnooppqrrssttuuvvwxxyyzz
```

Рисунок 2.4 – Программа с одним потоком



Листинг 2.4 – Открытие одного и того же файла несколько раз для чтения в двух потоках

```
1 #include <fcntl.h>
2 #include <pthread.h>
3 #include <stdio.h>
4 #include <unistd.h>
5
6 void *read_buffer(void *args)
7 {
8     int fd2 = open("data.txt", O_RDONLY);
9     char c;
10    int err;
11
12    while ((err = read(fd2, &c, 1)) == 1)
13        write(1, &c, 1);
14
15    return NULL;
16 }
17
18 int main(void)
19 {
20     int fd1 = open("data.txt", O_RDONLY);
21
22     pthread_t thread;
23     pthread_create(&thread, NULL, read_buffer, NULL);
24     char c;
25     int err;
26
27     while ((err = read(fd1, &c, 1)) == 1)
28         write(1, &c, 1);
29
30     pthread_join(thread, NULL);
31     c = '\n';
32     write(1, &c, 1);
33     return 0;
34 }
```

```
dmitriy@dm-pc:~/Рабочий стол/OS/sem6/lab5/src$ gcc prog_02_thread.c -lpthread
dmitriy@dm-pc:~/Рабочий стол/OS/sem6/lab5/src$ ./a.out
AbcdefghiAjbkclmdenfogphqirjskltumvnwoxpyqzrstuvwxyz
```

Рисунок 2.5 – Программа с двумя потоками

Листинг 2.5 – Открытие одного и того же файла несколько раз для чтения в двух потоках с использованием мьютекса

```
1 #include <fcntl.h>
2 #include <pthread.h>
3 #include <stdio.h>
4 #include <unistd.h>
5
6 pthread_mutex_t mutex;
7
8 void *read_buffer(void *args)
9 {
10     int fd2 = open("data.txt", O_RDONLY);
11     char c;
12     int err;
13
14     pthread_mutex_lock(&mutex);
15     while ((err = read(fd2, &c, 1)) == 1)
16         write(1, &c, 1);
17     pthread_mutex_unlock(&mutex);
18
19     return NULL;
20 }
21
22 int main(void)
23 {
24     int fd1 = open("data.txt", O_RDONLY);
25
26     pthread_t thread;
27     pthread_create(&thread, NULL, read_buffer, NULL);
28     char c;
29     int err;
30
31     pthread_mutex_lock(&mutex);
32     while ((err = read(fd1, &c, 1)) == 1)
33         write(1, &c, 1);
34     pthread_mutex_unlock(&mutex);
35
36     pthread_join(thread, NULL);
37     c = '\n';
38     write(1, &c, 1);
39     return 0;
40 }
```

```
dmitriy@dm-pc:~/Рабочий стол/OS/sem6/lab5/src$ gcc prog_02_mutex.c -lpthread
dmitriy@dm-pc:~/Рабочий стол/OS/sem6/lab5/src$ ./a.out
AbcdefghijklmnopqrstuvwxyzAbcdefghijklmnopqrstuvwxyz
```

Рисунок 2.6 – Программа с двумя потоками и мьютексом

## 2.2.2 Схема взаимодействия структур ядра

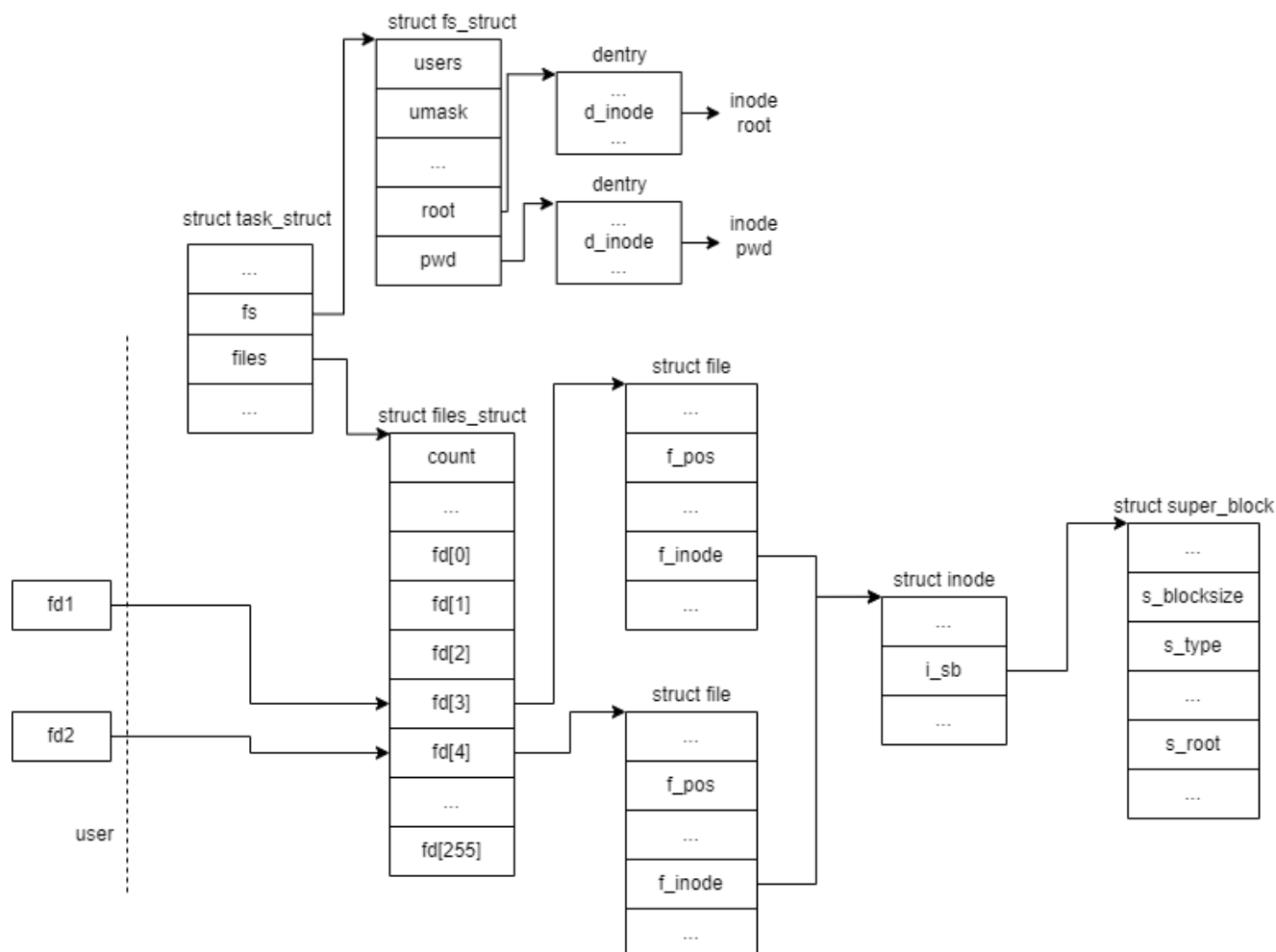


Рисунок 2.7 – Схема взаимодействия структур ядра

### Вывод

- в результате двух вызовов функции `open()` для одного и того же файла создаются два дескриптора в таблице открытых файлов процесса и два дескриптора в системной таблице открытых файлов, поэтому в программе существуют две различные `struct file`, но ссылающиеся на один и тот же `struct inode`;
- из-за того что структуры `struct file` разные и значения смещения в файле независимы, посимвольная печать просто дважды выведет содержимое файла в формате «AAbbсс...» (в случае однопоточной реализации);
- в случае многопоточной реализации потоки будут поочередно получать квант времени и печатать символы;

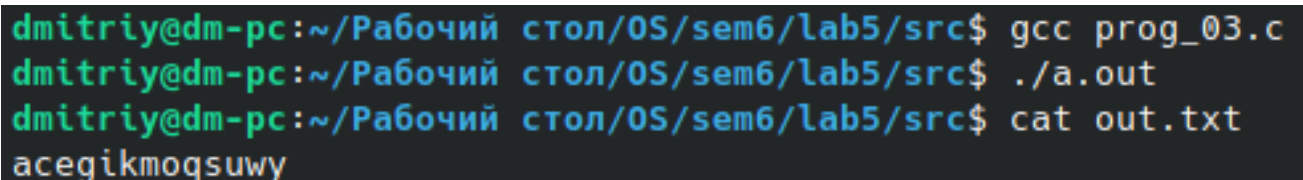
- использование мьютекса решает эту проблему: вывод второго потока начнется после окончания вывода первого потока.

## 2.3 Третья программа

### 2.3.1 Коды программ

Листинг 2.6 – Открытие одного и того же файла несколько раз для записи в одном потоке

```
1  #include <fcntl.h>
2  #include <stdio.h>
3  #include <unistd.h>
4
5  int main()
6  {
7      FILE *f1 = fopen("out.txt", "w");
8      FILE *f2 = fopen("out.txt", "w");
9
10     for (char c = 'a'; c <= 'z'; c++)
11         fprintf(c % 2 ? f1 : f2, "%c", c);
12
13     fclose(f2);
14     fprintf(f1, "\n");
15     fclose(f1);
16
17     return 0;
18 }
```



```
dmitriy@dm-pc:~/Рабочий стол/OS/sem6/lab5/src$ gcc prog_03.c
dmitriy@dm-pc:~/Рабочий стол/OS/sem6/lab5/src$ ./a.out
dmitriy@dm-pc:~/Рабочий стол/OS/sem6/lab5/src$ cat out.txt
acegikmoqsuw
```

Рисунок 2.8 – Программа с одним потоком

Листинг 2.7 – Открытие одного и того же файла несколько раз для записи в двух потоках

```
1 #include <fcntl.h>
2 #include <pthread.h>
3 #include <stdio.h>
4 #include <unistd.h>
5
6 void *write_buffer(void *args)
7 {
8     FILE *f = fopen("out.txt", "w");
9
10    for (char c = 'b'; c <= 'z'; c += 2)
11        fprintf(f, "%c", c);
12
13    fclose(f);
14    return NULL;
15 }
16
17 int main()
18 {
19     FILE *f1 = fopen("out.txt", "w");
20
21     pthread_t thread;
22     int rc = pthread_create(&thread, NULL, write_buffer, NULL);
23
24     for (char c = 'a'; c <= 'z'; c += 2)
25         fprintf(f1, "%c", c);
26
27     pthread_join(thread, NULL);
28     fprintf(f1, "\n");
29     fclose(f1);
30
31     return 0;
32 }
```

```
dmitriy@dm-pc:~/Рабочий стол/OS/sem6/lab5/src$ gcc prog_03_thread.c -lpthread
dmitriy@dm-pc:~/Рабочий стол/OS/sem6/lab5/src$ ./a.out
dmitriy@dm-pc:~/Рабочий стол/OS/sem6/lab5/src$ cat out.txt
acegikmoqsuwy
```

Рисунок 2.9 – Программа с двумя потоками

## 2.3.2 Схема взаимодействия структур ядра

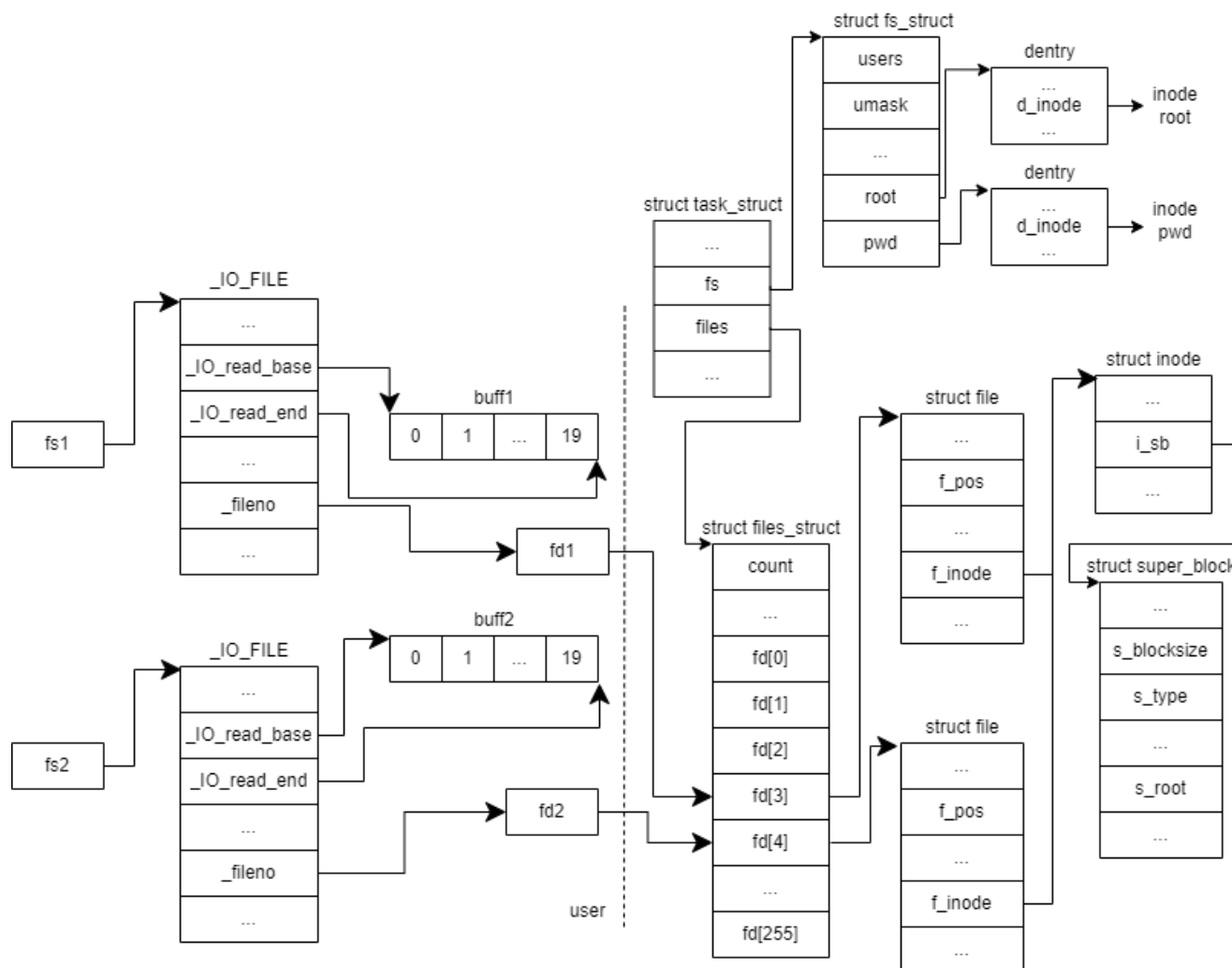


Рисунок 2.10 – Схема взаимодействия структур ядра

## Вывод

- файл открывается на запись два раза в результате двух вызовов функции `fopen()`;
- функция `fprintf()` предоставляет буферизованный вывод: изначально информация пишется в буфер, а из буфера в файл, если произошло одно из событий:

буфер полон;

вызвана функция `fclose()`;

вызвана функция `fflush()`;

- в данной программе информация запишется в файл в результате вызова функции `fclose()`;
- из-за того что `f_pos` независимы для каждого дескриптора файла, запись в файл будет производиться с его начала;
- таким образом, информация, записанная при первом вызове `fclose()`, будет потеряна в результате второго вызова `fclose()`.
- в многопоточной реализации результат аналогичен — с помощью `pthread_join` дожидаемся вызова `fclose()` для `f2` в отдельном потоке и далее вызываем `fclose()` для `f1`.

Для исключения проблемы потери данных нужно открывать файл в режиме добавления - `O_APPEND`. Тогда установка смещения в конец файла и операция записи в файл выполняются как атомарная операция.