



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 №10 по курсу "Операционные системы"

Тема Буферизованный и небуферизованный ввод-вывод

Студент Гурова Н.А.

Группа ИУ7-64Б

Оценка (баллы) _____

Преподаватель Рязанова Н. Ю.

Москва — 2023 г.

1 Структура `_IO_FILE`

Листинг 1.1 – Структура `_IO_FILE`

```
1 // /usr/include/x86_64-linux-gnu/bits/types/FILE.h:
2
3 #ifndef __FILE_defined
4 #define __FILE_defined 1
5
6 struct _IO_FILE;
7
8 /* The opaque type of streams. This is the definition used elsewhere.
   */
9 typedef struct _IO_FILE FILE;
10
11 #endif
12
13
14 // /usr/include/x86_64-linux-gnu/bits/libio.h:
15 ...
16 struct _IO_FILE {
17     int _flags;          /* High-order word is _IO_MAGIC; rest is
18                          flags. */
19     #define _IO_file_flags _flags
20
21     /* The following pointers correspond to the C++ streambuf
22        protocol. */
23     /* Note: Tk uses the _IO_read_ptr and _IO_read_end fields
24        directly. */
25     char* _IO_read_ptr;  /* Current read pointer */
26     char* _IO_read_end;  /* End of get area. */
27     char* _IO_read_base; /* Start of putback+get area. */
28     char* _IO_write_base; /* Start of put area. */
29     char* _IO_write_ptr; /* Current put pointer. */
30     char* _IO_write_end; /* End of put area. */
31     char* _IO_buf_base;  /* Start of reserve area. */
32     char* _IO_buf_end;   /* End of reserve area. */
33     /* The following fields are used to support backing up and undo. */
```

```

31  char *_IO_save_base; /* Pointer to start of non-current get area.
    */
32  char *_IO_backup_base; /* Pointer to first valid character of
    backup area */
33  char *_IO_save_end; /* Pointer to end of non-current get area. */
34
35  struct _IO_marker *_markers;
36
37  struct _IO_FILE *_chain;
38
39  int _fileno;
40  #if 0
41  int _blksize;
42  #else
43  int _flags2;
44  #endif
45  _IO_off_t _old_offset; /* This used to be _offset but it's too
    small. */
46
47  #define __HAVE_COLUMN /* temporary */
48  /* 1+column number of pbase(); 0 is unknown. */
49  unsigned short _cur_column;
50  signed char _vtable_offset;
51  char _shortbuf[1];
52
53  /* char* _save_gptr; char* _save_egptr; */
54
55  _IO_lock_t *_lock;
56  #ifdef _IO_USE_OLD_IO_FILE
57 };
58 ...

```

Листинг 1.2 – Структура struct file

```

1  struct file {
2      union {
3          struct llist_node    fu_llist;
4          struct rcu_head      fu_rcuhead;
5      } f_u;

```

```

6   struct path      f_path;
7   struct inode      *f_inode;    /* cached value */
8   const struct file_operations *f_op;
9   /*
10    * Protects f_ep_links, f_flags.
11    * Must not be taken from IRQ context.
12    */
13   spinlock_t        f_lock;
14   enum rw_hint       f_write_hint;
15   atomic_long_t      f_count;
16   unsigned int       f_flags;
17   fmode_t            f_mode;
18   struct mutex        f_pos_lock;
19   loff_t              f_pos;
20   struct fown_struct  f_owner;
21   const struct cred   *f_cred;
22   struct file_ra_state f_ra;
23   u64                 f_version;
24 #ifdef CONFIG_SECURITY
25     void               *f_security;
26 #endif
27     /* needed for tty driver, and maybe others */
28     void               *private_data;
29 #ifdef CONFIG_EPOLL
30     /* Used by fs/eventpoll.c to link all the hooks to this file */
31     struct list_head    f_ep_links;
32     struct list_head    f_tfile_llink;
33 #endif /* #ifdef CONFIG_EPOLL */
34     struct address_space *f_mapping;
35     errseq_t            f_wb_err;
36 } __randomize_layout

```

2 Первая программа

Листинг 2.1 – Первая программа

```
1 //testC10.c
2 #include <stdio.h>
3 #include <fcntl.h>
4
5 /*
6  On my machine, a buffer size of 20 bytes
7  translated into a 12-character buffer.
8  Apparently 8 bytes were used up by the
9  stdio library for bookkeeping.
10 */
11
12 int main()
13 {
14     // have kernel open connection to file alphabet.txt
15     int fd = open("alphabet.txt", O_RDONLY);
16     // create two a C I/O buffered streams using the above connection
17     FILE *fs1 = fdopen(fd, "r");
18     char buff1[20];
19     setvbuf(fs1, buff1, _IOFBF, 20);
20
21     FILE *fs2 = fdopen(fd, "r");
22     char buff2[20];
23     setvbuf(fs2, buff2, _IOFBF, 20);
24
25     // read a char & write it alternately from fs1 and fs2
26     int flag1 = 1, flag2 = 2;
27     while(flag1 == 1 || flag2 == 1)
28     {
29         char c;
30         flag1 = fscanf(fs1, "%c", &c);
31         if (flag1 == 1)
32             fprintf(stdout, "%c", c);
33         flag2 = fscanf(fs2, "%c", &c);
34         if (flag2 == 1)
35             fprintf(stdout, "%c", c);
```

```
36     }
37     return 0;
38 }
```



```
● (base) i111034458: src $ gcc task_01_v1.c
● (base) i111034458: src $ ./a.out
○ Aubvcwdxeyfzghijklmnopqrst(base) i111034458: src $
```

Рисунок 2.1 – Результат работы первой программы

Листинг 2.2 – Первая программа (реализация с потоками)

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <pthread.h>
4 #include <stdlib.h>
5 #include <unistd.h>
6
7 void *thread_func(void *args)
8 {
9     int flag = 1;
10    FILE *fs = (FILE *)args;
11
12
13    while (flag == 1)
14    {
15        char c;
16        if ((flag = fscanf(fs, "%c", &c)) == 1)
17            fprintf(stdout, "thread 2 read: %c\n", c);
18    }
19 }
20 int main(void)
21 {
22     setbuf(stdout, NULL);
23     pthread_t thread;
24     int fd = open("alphabet.txt", O_RDONLY);
25
26     FILE *fs1 = fdopen(fd, "r");
27     char buff1[20];
```

```

28     setvbuf(fs1 , buff1 , _IOFBF, 20);
29
30     FILE *fs2 = fdopen(fd , "r");
31     char buff2[20];
32     setvbuf(fs2 , buff2 , _IOFBF, 20);
33
34     if (pthread_create(&thread , NULL, thread_func , (void *)fs2) != 0)
35     {
36         perror("Error in pthread_create\n");
37         exit(-1);
38     }
39
40     int flag = 1;
41     while (flag == 1)
42     {
43         char c;
44         if ((flag = fscanf(fs1 , "%c" , &c)) == 1)
45         {
46             fprintf(stdout , "thread 1 read: %c\n" , c);
47         }
48     }
49     pthread_join(thread , NULL);
50     close(fd);
51     return 0;
52 }

```

```

● (base) i111034458: src $ gcc task_01_v2.c
task_01_v2.c:19:1: warning: non-void function does not return
}
^
1 warning generated.
● (base) i111034458: src $ ./a.out
thread 1 read: A
thread 1 read: b
thread 1 read: c
thread 1 read: d
thread 1 read: e
thread 1 read: f
thread 1 read: g
thread 1 read: h
thread 1 read: i
thread 1 read: j
thread 1 read: k
thread 1 read: l
thread 1 read: m
thread 1 read: n
thread 1 read: o
thread 1 read: p
thread 1 read: q
thread 1 read: r
thread 1 read: s
thread 1 read: t
thread 2 read: u
thread 2 read: v
thread 2 read: w
thread 2 read: x
thread 2 read: y
thread 2 read: z
○ (base) i111034458: src $ █

```

Рисунок 2.2 – Результат работы первой программы (с потоками)

Системный вызов `open()` создает новый файловый дескриптор для открытого только для чтения файла "alphabet.txt" (который содержит символы `Abcdefghijklmnopqrstuvwxyz`). В системной таблице открытых файлов будет создан один дескриптор `struct file`.

Вызов `fdopen()` возвращает указатель на структуру типа `FILE` (`fs1` и `fs2`), которая ссылается на дескриптор открытого файла `fd` в таблице `struct files_struct`.

Вызов функции `setvbuf()` (для `fs1` и `fs2`) явно задает буффер и его размер (20 байт) и меняет тип буферизации на полную (`_IOFBF`).

Проблема: При первом вызове `fscanf()` (для `fs1`) буффер `buff1` полностью заполнился первыми 20 символами. Значение `f_pos` в структуре `struct_file` открытого файла увеличилось на 20. При следующем вызове `fscanf()` (для `fs2`) в `buff2` считались оставшиеся 6 символов, начиная с `f_pos` (`fs1` и `fs2` ссылаются на один и тот же дескриптор `fd`).

Затем в однопоточной программе в цикле поочередно выводятся символы из buff1 и buff2 (так как в buff2 записались лишь оставшиеся 6 символов, после 6 итерации цикла будут выводиться символы только из buff1). В двупоточной программе главный поток начинает вывод быстрее, так как для второго потока сначала затрачивается время на его создание, и только потом начинается вывод.

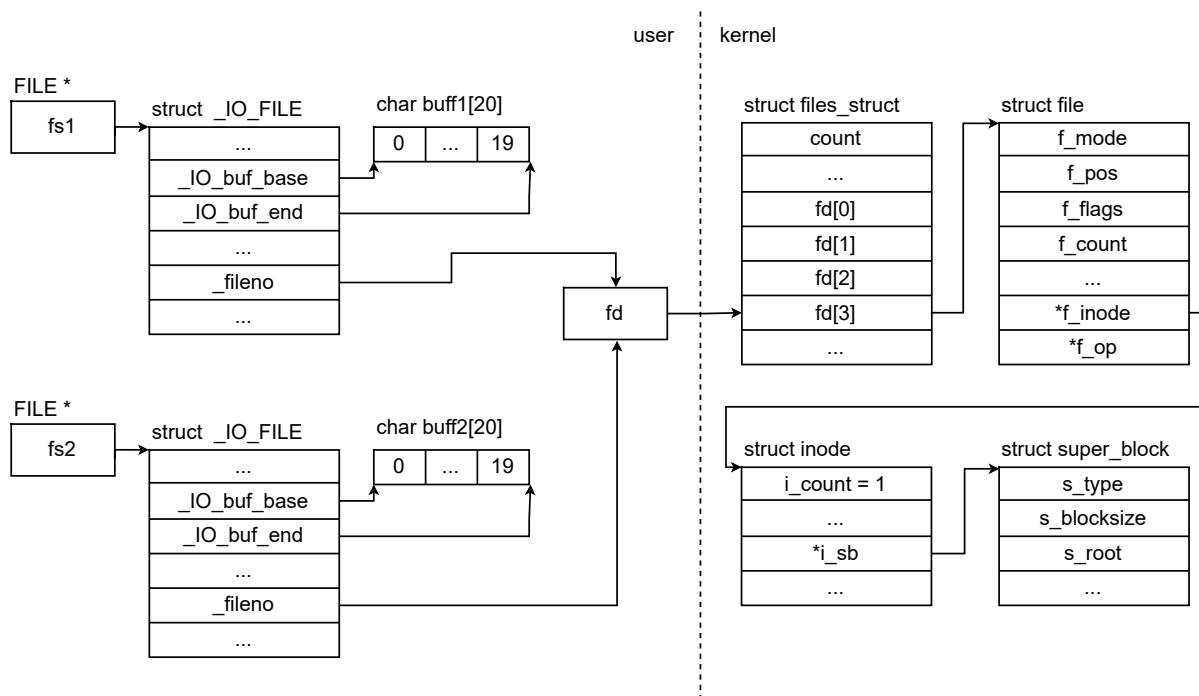
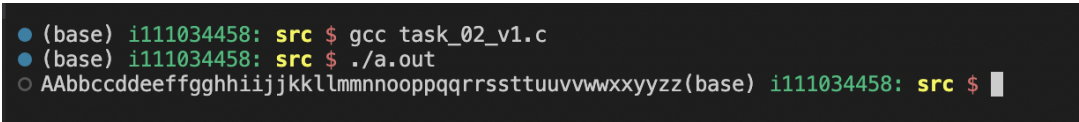


Рисунок 2.3 – Схема связей структур в первой программе

3 Вторая программа

Листинг 3.1 – Вторая программа 1 вариант

```
1 #include <fcntl.h>
2 #include <unistd.h>
3
4 int main(void)
5 {
6     int fd1 = open("alphabet.txt", O_RDONLY);
7     int fd2 = open("alphabet.txt", O_RDONLY);
8     int rc1 = 1, rc2 = 1;
9
10    while (rc1 == 1 && rc2 == 1)
11    {
12        char c;
13
14        rc1 = read(fd1, &c, 1);
15        if (rc1 == 1)
16        {
17            write(1, &c, 1);
18            rc2 = read(fd2, &c, 1);
19            if (rc2 == 1)
20            {
21                write(1, &c, 1);
22            }
23        }
24    }
25    close(fd1);
26    close(fd2);
27    return 0;
28 }
```



```
● (base) i111034458: src $ gcc task_02_v1.c
● (base) i111034458: src $ ./a.out
○ AAbbccddeeffgghhiijjkkllmmnnooppqrrssttuuvvwxxyyzz(base) i111034458: src $ █
```

Рисунок 3.1 – Результат работы второй программы 1 вариант

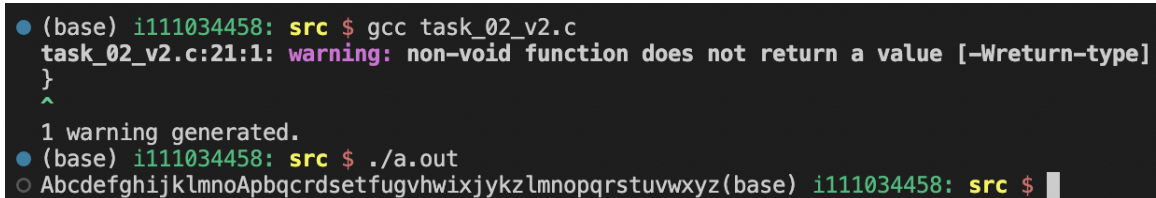
Листинг 3.2 – Вторая программа (с потоками) 1 вариант

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #include <pthread.h>
5
6 void *thread_func(void *args)
7 {
8     int fd2 = open("alphabet.txt", O_RDONLY);
9     int rc = 1;
10
11     while (rc == 1)
12     {
13         char c;
14         rc = read(fd2, &c, 1);
15         if (rc == 1)
16         {
17             write(1, &c, 1);
18         }
19     }
20     close(fd2);
21 }
22
23 int main(void)
24 {
25     int fd1 = open("alphabet.txt", O_RDONLY);
26
27     pthread_t thread;
28     if (pthread_create(&thread, NULL, thread_func, 0) != 0)
29     {
30         perror("error in pthread_create\n");
31         return -1;
32     }
33
34     int rc = 1;
35     while (rc == 1)
36     {
37         char c;
```

```

38         rc = read(fd1, &c, 1);
39         if (rc == 1)
40         {
41             write(1, &c, 1);
42         }
43     }
44
45     pthread_join(thread, NULL);
46     close(fd1);
47
48     return 0;
49 }

```



```

(base) i111034458: src $ gcc task_02_v2.c
task_02_v2.c:21:1: warning: non-void function does not return a value [-Wreturn-type]
}
^
1 warning generated.
(base) i111034458: src $ ./a.out
AbcdefghijklmnoApbqcrdsetfugvhwixjyzlmnopqrstuvwxyz(base) i111034458: src $

```

Рисунок 3.2 – Результат работы второй программы (с потоками) 1 вариант

Два системных вызова `open()` создают два независимых дескриптора открытого только для чтения файла, создавая две независимых записи в системной таблице открытых файлов.

Проблема: В программе существует две различные структуры `struct file`, которые при этом ссылаются на одну и ту же структуру `struct inode`. В каждой структуре свое поле `f_pos` (то есть смещения независимы), поэтому на экран каждый символ выводится дважды.

При этом в однопоточной программе в цикле каждый символ из файла выводится два раза подряд, а в двупоточной заранее предсказать порядок вывода символов невозможно, так как потоки выполняются параллельно (при этом дочерний поток начинает вывод позже, так как затрачивается время на его создание).

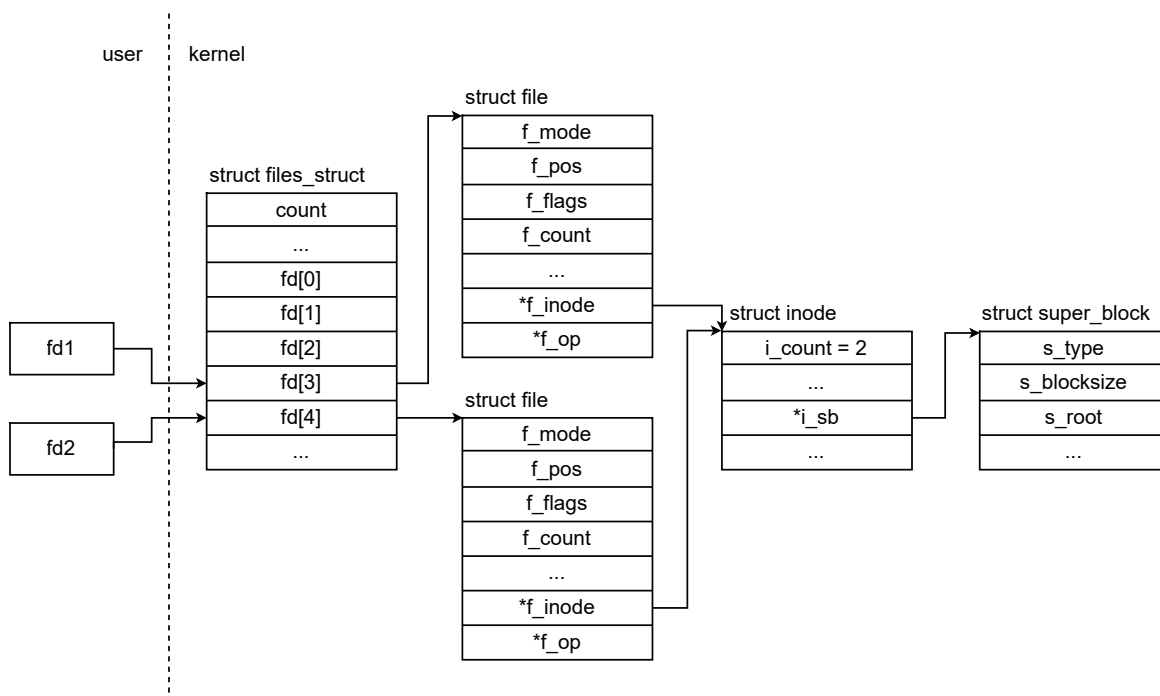


Рисунок 3.3 – Схема связей структур во второй программе

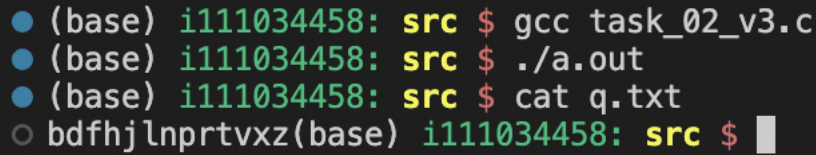
Листинг 3.3 – Вторая программа 2 вариант

```

1 #include <fcntl.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <stdio.h>
5 int main()
6 {
7     int fd1 = open("q.txt", O_RDWR);
8     int fd2 = open("q.txt", O_RDWR);
9     int curr = 0;
10    for(char c = 'a'; c <= 'z'; c++)
11    {
12        if (c%2){
13            write(fd1, &c, 1);
14        }
15        else{
16            write(fd2, &c, 1);
17        }
18    }
19    close(fd1);
20    close(fd2);

```

```
21     return 0;
22 }
```



A terminal window with a dark background and light-colored text. It shows four lines of output, each starting with a colored circle icon: a blue circle for the first three lines and a grey circle for the last line. The text is as follows:

```
● (base) i111034458: src $ gcc task_02_v3.c
● (base) i111034458: src $ ./a.out
● (base) i111034458: src $ cat q.txt
○ bdfhjlnprtvxz(base) i111034458: src $
```

Рисунок 3.4 – Результат работы второй программы 2 вариант

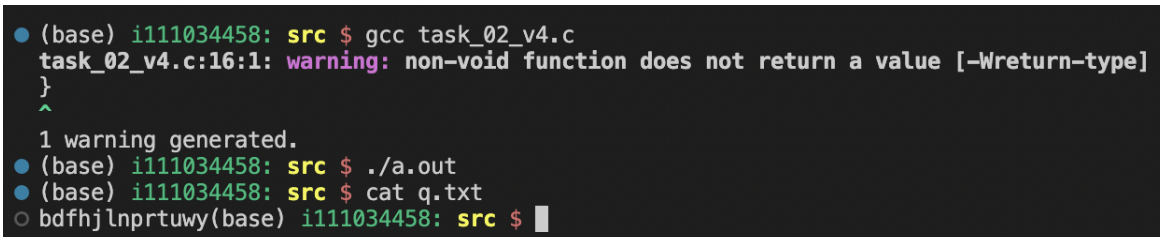
Листинг 3.4 – Вторая программа (с потоками) 2 вариант

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #include <pthread.h>
5
6 void *thread_func(void *args)
7 {
8     int fd2 = open("q.txt", O_RDWR);
9
10    for (char c = 'b'; c <= 'z'; c += 2)
11    {
12        write(fd2, &c, 1);
13    }
14
15    close(fd2);
16 }
17
18 int main(void)
19 {
20     int fd1 = open("q.txt", O_RDWR);
21
22     pthread_t thread;
23     if (pthread_create(&thread, NULL, thread_func, 0) != 0)
24     {
25         perror("error in pthread_create\n");
26         return -1;
27     }
28 }
```

```

27     }
28
29     for (char c = 'a'; c <= 'z'; c += 2)
30     {
31         write(fd1, &c, 1);
32     }
33
34     close(fd1);
35
36     return 0;
37 }

```



```

(base) i111034458: src $ gcc task_02_v4.c
task_02_v4.c:16:1: warning: non-void function does not return a value [-Wreturn-type]
}
^
1 warning generated.
(base) i111034458: src $ ./a.out
(base) i111034458: src $ cat q.txt
bdfhjlnprtuy(base) i111034458: src $

```

Рисунок 3.5 – Результат работы второй программы 2 вариант

Два системных вызова `open()` создают два независимых дескриптора открытого только для чтения файла, создавая две независимых записи в системной таблице открытых файлов.

Так как `f_pos` независимы для каждого дескриптора файла, запись в файл в каждом случае в данной программе производится с его начала.

Символы, имеющие четный код в таблице ASCII (`b`, `d`, ...) записываются в буфер, который относится к структуре, на которую указывает `fd2`, нечетный (`a`, `c`, ...) — к `fd1`.

Проблема: данные, которые были записаны после первого вызова `write` (для `fd1`), были потеряны в результате второго вызова `write` (для `fd2`), поэтому в файле `q.txt` записаны только символы `bdfhjlnprtvyxz`.

В двупоточной реализации принцип действий аналогичен, в файл будет записан тот символ, для которого `write()` вызовется последним.

4 Третья программа

Листинг 4.1 – Третья программа

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #include <sys/stat.h>
5
6 #define FILENAME "q.txt"
7
8 void print_file_info(char *message)
9 {
10     struct stat statbuf;
11     printf("%s", message);
12     if (stat(FILENAME, &statbuf) == 0)
13     {
14         printf("st_ino: %ld\n", statbuf.st_ino);
15         printf("st_size: %ld\n", statbuf.st_size);
16     }
17     else
18         printf("Error in stat\n\n");
19 }
20
21 int main()
22 {
23     print_file_info("Before first open\n");
24     FILE *f1 = fopen(FILENAME, "w");
25     print_file_info("After first open\n");
26     FILE *f2 = fopen(FILENAME, "w");
27     print_file_info("After second open\n");
28
29     for (char c = 'a'; c <= 'z'; c++)
30     {
31         if (c % 2)
32         {
33             fprintf(f1, "%c", c);
34         }
35         else
```



```

36     {
37         fprintf(f2, "%c", c);
38     }
39 }
40
41 print_file_info("Before first close\n");
42 fclose(f1);
43 print_file_info("After first close\n");
44 fclose(f2);
45 print_file_info("After second close\n");
46
47 return 0;
48 }

```

```

• (base) i111034458: src $ gcc task_03_v1.c
task_03_v1.c:14:33: warning: format specifies type 'long' but the argument
      printf("st_ino: %ld\n", statbuf.st_ino);
                        ~~~~ ^~~~~~
                        %llu
task_03_v1.c:15:34: warning: format specifies type 'long' but the argument
      printf("st_size: %ld\n", statbuf.st_size);
                        ~~~~ ^~~~~~
                        %lld

2 warnings generated.
• (base) i111034458: src $ ./a.out
Before first open
st_ino: 9610063
st_size: 13
After first open
st_ino: 9610063
st_size: 0
After second open
st_ino: 9610063
st_size: 0
Before first close
st_ino: 9610063
st_size: 0
After first close
st_ino: 9610063
st_size: 13
After second close
st_ino: 9610063
st_size: 13
○ (base) i111034458: src $ 

```

Рисунок 4.1 – Результат работы третьей программы

Содержимое файла q.txt: bdfhjlnprtvxz

Файл outfile.txt дважды открывается на запись с помощью функции `open()`. С помощью функции `fprintf()` стандартной библиотеки `stdio.h` выполняется буферизованный вывод. Буфер создается без явного вмешательства. Информация сначала пишется в буфер, а из буфера информация переписывается в файл, если произошло одно из 3 событий:

1. буфер заполнен;
2. вызвана функция `fclose()` (в данной программе именно эти события приводят к записи в файл);
3. вызвана функция `fflush()` (принудительная запись в файл).

Так как `f_pos` независимы для каждого дескриптора файла, запись в файл в каждом случае в данной программе производится с его начала.

Символы, имеющие четный код в таблице ASCII (`b`, `d`, ...) записываются в буфер, который относится к структуре, на которую указывает `f2`, нечетный (`a`, `c`, ...) – к `f1`.

Проблема: данные, которые были записаны после первого вызова `fclose` (для `f1`), были потеряны в результате второго вызова `fclose` (для `f2`), поэтому в файле outfile.txt записаны только символы `bdfhjlnprtvxz` (из буффера, относящегося к `f2`).

Если поменять вызовы `fclose` для `f1` и `f2` местами, то результат будет противоположным: `asegikmoqsuwy`

Листинг 4.2 – Третья программа (реализация с потоками)

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <pthread.h>
4 #include <unistd.h>
5 #include <sys/stat.h>
6
7 #define FILENAME "q.txt"
8
9 void print_file_info(char *message)
10 {
```

```

11     struct stat statbuf;
12     printf("%s", message);
13     if (stat(FILENAME , &statbuf) == 0)
14     {
15         printf("st_ino: %ld\n", statbuf.st_ino);
16         printf("st_size: %ld\n", statbuf.st_size);
17     }
18     else
19         printf("Error in stat\n\n");
20 }
21
22 void *thread_func(void *args)
23 {
24     FILE *f2 = fopen(FILENAME, "w");
25     print_file_info("After second open\n");
26
27     for (char c = 'b'; c <= 'z'; c += 2)
28     {
29         fprintf(f2, "%c", c);
30     }
31     print_file_info("Before first close\n");
32     fclose(f2);
33     print_file_info("After first close\n");
34 }
35
36 int main()
37 {
38     print_file_info("Before first open\n");
39     FILE *f1 = fopen(FILENAME, "w");
40     print_file_info("After first open\n");
41
42
43     pthread_t thread;
44     int rc = pthread_create(&thread , NULL, thread_func , NULL);
45
46     for (char c = 'a'; c <= 'z'; c += 2)
47     {
48         fprintf(f1, "%c", c);
49     }

```

```

50
51 pthread_join(thread, NULL);
52 fclose(f1);
53 print_file_info("After second close\n");
54
55
56 return 0;
57 }

```

```

● (base) i111034458: src $ gcc task_03_v2.c
task_03_v2.c:15:33: warning: format specifies type 'long' but the argument
printf("st_ino: %ld\n", statbuf.st_ino);
                    ~~~ ^~~~~~
                    %llu
task_03_v2.c:16:34: warning: format specifies type 'long' but the argument
printf("st_size: %ld\n", statbuf.st_size);
                    ~~~ ^~~~~~
                    %lld
task_03_v2.c:34:1: warning: non-void function does not return a value
}
^
3 warnings generated.
● (base) i111034458: src $ ./a.out
Before first open
st_ino: 9610063
st_size: 13
After first open
st_ino: 9610063
st_size: 0
After second open
st_ino: 9610063
st_size: 0
Before first close
st_ino: 9610063
st_size: 0
After first close
st_ino: 9610063
st_size: 13
After second close
st_ino: 9610063
st_size: 13
○ (base) i111034458: src $ █

```

Рисунок 4.2 – Результат работы третьей программы (с потоками)

Содержимое файла q.txt: acegikmoqsuwy

В двупоточной реализации принцип действий аналогичен (в данном случае теряются данные, связанные с f2, так как для него fclose вызывается раньше, чем для f1).

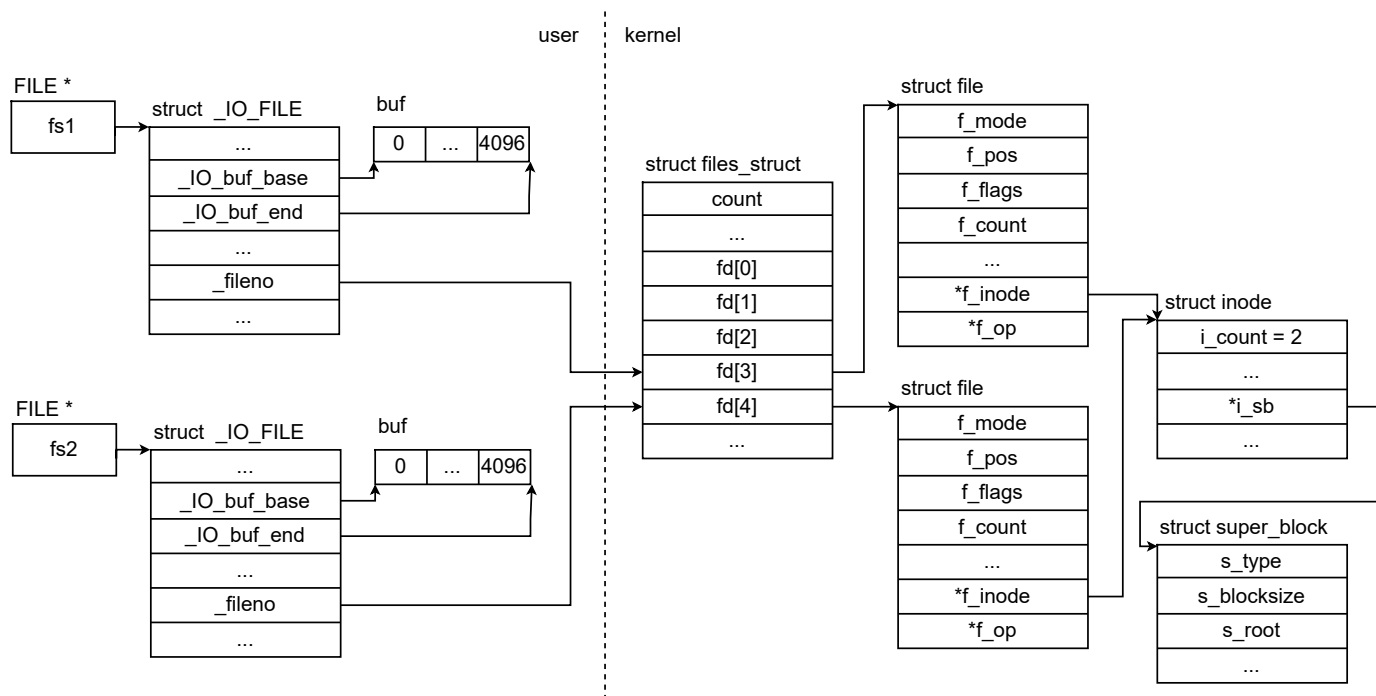


Рисунок 4.3 – Схема связей структур в третьей программе