



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»

«Буферизованный и не буферизованный ввод-вывод»

Студент _____ Маслова Марина Дмитриевна

Группа _____ ИУ7-63Б

Оценка (баллы) _____

Преподаватель _____ Рязанова Наталья Юрьевна

2022 г.

1 Структура FILE

Листинг 1.1 – Описание структуры FILE в файле /usr/include/bits/types/FILE.h

```
1 typedef struct _IO_FILE FILE;
```

Листинг 1.2 – Описание структуры _IO_FILE в файле /usr/include/bits/types/struct_FILE.h

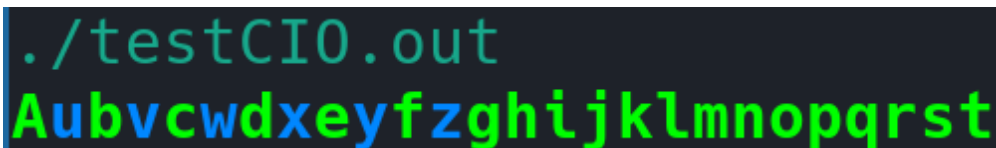
```
1 struct _IO_FILE
2 {
3     int _flags;          /* High-order word is _IO_MAGIC; rest is flags. */
4
5     /* The following pointers correspond to the C++ streambuf protocol. */
6     char *_IO_read_ptr;  /* Current read pointer */
7     char *_IO_read_end;  /* End of get area. */
8     char *_IO_read_base;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

```
40 #endif
41 __off64_t _offset;
42 /* Wide character stream stuff. */
43 struct _IO_codecvt *_codecvt;
44 struct _IO_wide_data *_wide_data;
45 struct _IO_FILE *_freeres_list;
46 void *_freeres_buf;
47 size_t __pad5;
48 int _mode;
49 /* Make sure we don't get into trouble again. */
50 char _unused2[15 * sizeof (int) - 4 * sizeof (void *) - sizeof (size_t)];
51 };
```

2 Первая программа

Листинг 2.1 – 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 }
```



```
./testCI0.out
Aubvcwdxeyfzghijklmnopqrst
```

Рисунок 2.1 – Результат работы первой программы. Один поток

Листинг 2.2 – 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     {
20         char c;
21         if ((flag = fscanf(fs, "%c", &c)) == 1)
22             fprintf(stdout, "%s%c" CLEAR, color, c);
23     }
24     return NULL;
25 }
26
27 int main(void)
28 {
29     int fd = open("alphabet.txt", O_RDONLY);
30
31     FILE *fs1 = fdopen(fd, "r");
32     char buff1[20];
33     setvbuf(fs1, buff1, _IOFBF, 20);
34     struct args_struct args1 = { .fs = fs1, .color = GREEN };
35
36     FILE *fs2 = fdopen(fd, "r");
37     char buff2[20];
38     setvbuf(fs2, buff2, _IOFBF, 20);
39     struct args_struct args2 = { .fs = fs2, .color = BLUE };
40
41     pthread_t td;
42     pthread_create(&td, NULL, read_buf, &args2);
43     read_buf(&args1);
44     pthread_join(td, NULL);
45     puts("");
46     return 0;
47 }
```

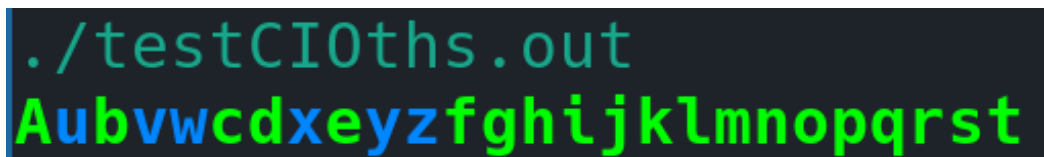


Рисунок 2.2 – Результат работы первой программы. Два потока

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 `s` 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 `s` запишется символ 'u'.

При следующих вызовах `fscanf()` переменной `s` 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 производиться не будет, так как файл был полностью прочитан при первом заполнении буфера `fs2`, что представлено на рисунке 2.1.

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.

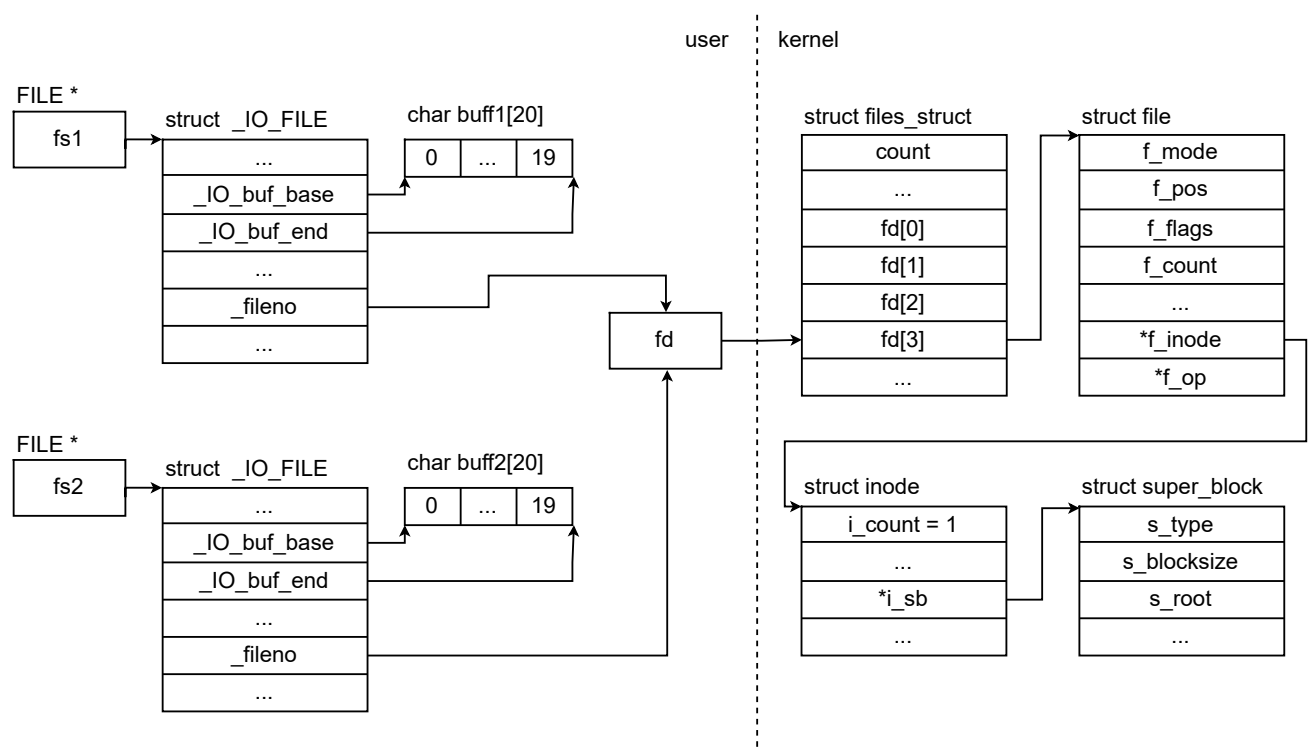
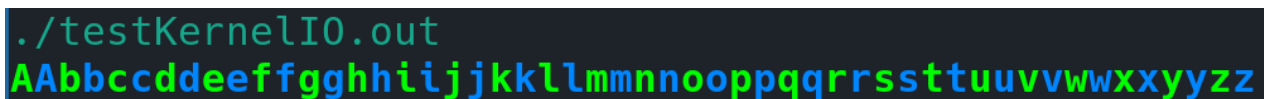


Рисунок 2.3 – Схема связей структур в первой программе

3 Вторая программа

Листинг 3.1 – 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 3.1 – Результат работы второй программы. Один поток

Листинг 3.2 – 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 struct args_struct { int fd; char * color; };
11
12 void *read_buf(void *args)
13 {
14     struct args_struct *cur_args = (struct args_struct *) args;
15     int fd = cur_args->fd;
16     char *color = cur_args->color;
17
18     int flag = 1;
19
20     while (flag == 1)
21     {
22         char c;
23         if ((flag = read(fd, &c, 1)) == 1)
24             printf("%s%c" CLEAR, color, c);
25     }
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
AAbbccddeffeghfigjkhiljkmnlmnoopqrpsqtuvrwsxtzyuvwxyz
```

Рисунок 3.2 – 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.

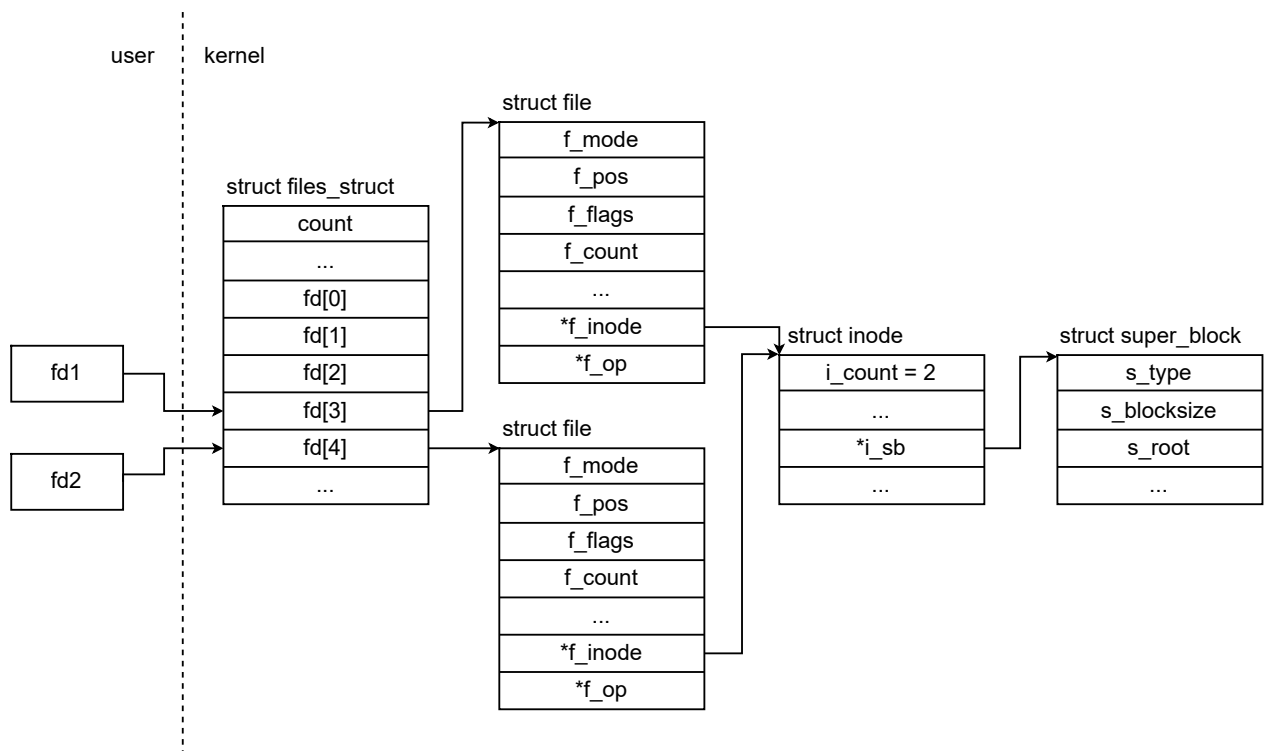
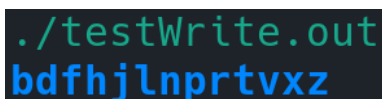


Рисунок 3.3 – Схема связей структур в второй программе

4 Третья программа

Листинг 4.1 – 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("Общий размер в байтах: %ld\n", statbuf.st_size);
12    printf("Текущая позиция: %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
26     fileInfo(fs1);
27     fclose(fs1);
28     fileInfo(fs1);
29
30     fileInfo(fs2);
31     fclose(fs2);
32     fileInfo(fs2);
33
34     return 0;
35 }
```



./testWrite.out
bdfhjlnprtvxz

Рисунок 4.1 – Результат работы третьей программы. fs2 закрывается последним

```
./testWrite.out  
acegikmoqsuw
```

Рисунок 4.2 – 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 4.3 – Информация о состоянии открытых файлов

Анализ результата

В данной программе с помощью функции `fopen()` стандартной библиотеки файл дважды открывается для записи. Так же как и во второй 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 файла, и записанные ранее символы перезапишутся новыми данными (четными буквами алфавита), что и показано на рисунке 4.1. 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 4.2).

Листинг 4.2 – 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", "w");
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 4.4 – Результат работы третьей программы.
Последним вызывается `fclose` в вспомогательном потоке

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

Рисунок 4.5 – Результат работы третьей программы.
Последним вызывается `fclose` в главном потоке

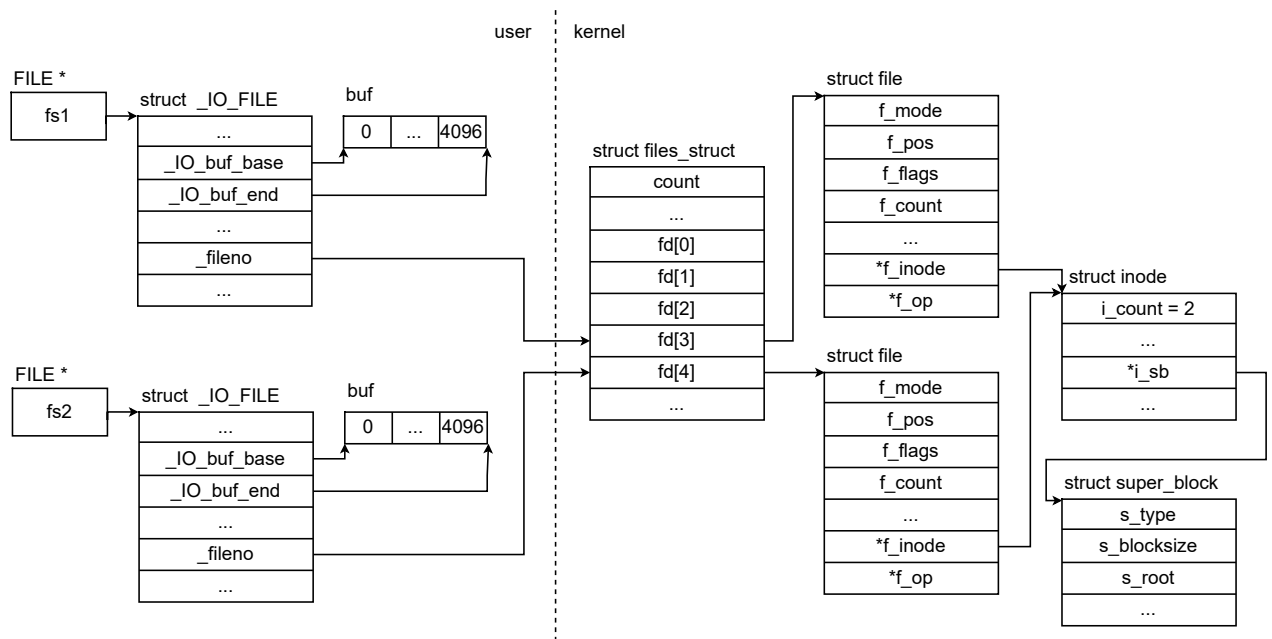


Рисунок 4.6 – Схема связей структур в третьей программе

5 Вывод

В данной лабораторной работе демонстрируется ряд проблем при работе с вводом-выводом.

В первой программе предполагалось, что символы из файла будут читаться по очереди каждым из потоков и выведутся в том порядке, в котором они расположены в файле, однако включение буферизации, как было показано, привело к другому результату (проблема буферизации).

Во второй программе в силу наличия двух файловых дескрипторов, связанных с одним `inode`, чтение файла и вывод информации из него происходит дважды. При необходимости решения этой проблемы можно создать разделяемую область памяти для отслеживания позиции в файле и использовать `mutex` для доступа к ней.

В третьей программе, так же как и в первой, показывается проблема буферизации, из-за которой в данной программе запись в файл происходит только при вызове `fclose()`, данные в файле перезаписываются, возникает потеря информации. Решением проблемы является открытие файла в режиме добавления `O_APPEND`. В этом случае операция записи в файл атомарна, а перед каждым вызовом функции записи смещение в файле будет устанавливаться на его конец.