



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

по курсу «Операционные системы»

на тему:

«Буферизованный и небуферизованный ввод-вывод»

Студент группы ИУ7-66Б

(Подпись, дата)

В. М. Мансуров

(И.О. Фамилия)

Преподаватель

(Подпись, дата)

Н. Ю. Рязанова

(И.О. Фамилия)

2023 г.

1 Структура FILE

Версия ядра: 6.3.2

Листинг 1 – Описание структуры FILE

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

Листинг 2 – Описание структуры _IO_FILE

```
1 struct _IO_FILE {
2     int _flags;          /* High-order word is _IO_MAGIC; rest is flags. */
3     #define _IO_file_flags _flags
4
5     /* The following pointers correspond to the C++ streambuf protocol. */
6     /* Note: Tk uses the _IO_read_ptr and _IO_read_end fields directly. */
7     char* _IO_read_ptr;  /* Current read pointer */
8     char* _IO_read_end;  /* End of get area. */
9     char* _IO_read_base; /* Start of putback+get area. */
10    char* _IO_write_base; /* Start of put area. */
11    char* _IO_write_ptr;  /* Current put pointer. */
12    char* _IO_write_end;  /* End of put area. */
13    char* _IO_buf_base;   /* Start of reserve area. */
14    char* _IO_buf_end;    /* End of reserve area. */
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                           */
19    char *_IO_save_end; /* Pointer to end of non-current get area. */
20
21    struct _IO_marker *_markers;
22
23    struct _IO_FILE *_chain;
24
25    int _fileno;
26    #if 0
27    int _blksize;
28    #else
29    int _flags2;
30    #endif
31    _IO_off_t _old_offset; /* This used to be _offset but it's too small. */
```

```

31
32  #define __HAVE_COLUMN /* temporary */
33  /* 1+column number of pbase(); 0 is unknown. */
34  unsigned short _cur_column;
35  signed char _vtable_offset;
36  char _shortbuf[1];
37
38  /* char* _save_gptr; char* _save_egptr; */
39
40  _IO_lock_t *_lock;
41  #ifdef _IO_USE_OLD_IO_FILE
42  };

```

2 Первая программа

Листинг 3 – Первая программа

```

1  #include <stdio.h>
2  #include <fcntl.h>
3
4  int main()
5  {
6      int fd = open("alphabet.txt", O_RDONLY);
7      FILE *fs1 = fdopen(fd, "r");
8      char buff1[20];
9      setvbuf(fs1, buff1, _IOFBF, 20);
10     FILE *fs2 = fdopen(fd, "r");
11     char buff2[20];
12     setvbuf(fs2, buff2, _IOFBF, 20);
13     int flag1 = 1, flag2 = 2;
14     while(flag1 == 1 || flag2 == 1)
15     {
16         char c;
17         flag1 = fscanf(fs1, "%c", &c);
18         if (flag1 == 1)
19         {
20             fprintf(stdout, "%c", c);
21         }
22         flag2 = fscanf(fs2, "%c", &c);
23         if (flag2 == 1)

```

```

24     {
25         fprintf(stdout, "%c", c);
26     }
27 }
28 printf("\n");
29 return 0;
30 }

```

```

vladislav@linux-hp-pavilion:~/OS/6sem/lab_10/src$ gcc -Wall -Werror testCIO.c -o testCIO
vladislav@linux-hp-pavilion:~/OS/6sem/lab_10/src$ ./testCIO
Aubvcwdxyfzghijklmnopqrst
vladislav@linux-hp-pavilion:~/OS/6sem/lab_10/src$ █

```

Рисунок 1 – Результат работы первой программы

Листинг 4 – Первая программа с 2-мя дополнительными потоками

```

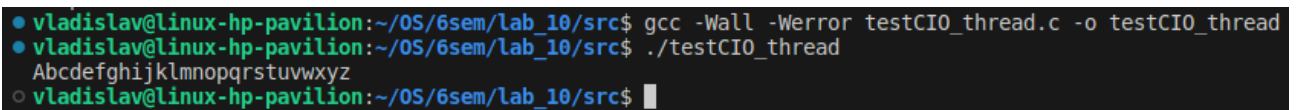
1  #include <stdio.h>
2  #include <fcntl.h>
3  #include <pthread.h>
4
5  struct args_struct
6  {
7      FILE* fs;
8  };
9
10 void *read_buf(void *args)
11 {
12     struct args_struct *cur_args = (struct args_struct *) args;
13     FILE *fs = cur_args->fs;
14     int flag = 1;
15
16     while (flag == 1)
17     {
18         char c;
19         if ((flag = fscanf(fs, "%c", &c)) == 1)
20             fprintf(stdout, "%c", c);
21     }
22     return NULL;
23 }
24
25 int main(void)

```

```

26 {
27     int fd = open("alphabet.txt", O_RDONLY);
28
29     FILE *fs1 = fdopen(fd, "r");
30     char buff1[20];
31     setvbuf(fs1, buff1, _IOFBF, 20);
32     struct args_struct args1 = { .fs = fs1 };
33
34     FILE *fs2 = fdopen(fd, "r");
35     char buff2[20];
36     setvbuf(fs2, buff2, _IOFBF, 20);
37     struct args_struct args2 = { .fs = fs2 };
38
39     pthread_t td1, td2;
40     pthread_create(&td1, NULL, read_buf, &args2);
41     pthread_create(&td2, NULL, read_buf, &args1);
42     pthread_join(td1, NULL);
43     pthread_join(td2, NULL);
44     puts("");
45     return 0;
46 }

```



```

● vladislav@linux-hp-pavilion:~/OS/6sem/lab_10/src$ gcc -Wall -Werror testCIO_thread.c -o testCIO_thread
● vladislav@linux-hp-pavilion:~/OS/6sem/lab_10/src$ ./testCIO_thread
Abcdefghijklmnopqrstuvwxyz
○ vladislav@linux-hp-pavilion:~/OS/6sem/lab_10/src$

```

Рисунок 2 – Результат работы первой программы с 2-мя дополнительными потоками

2.1 Анализ результата

Системный вызов `open()` открывает файл «alphabet.txt» только для чтения (установлен флаг `O_RDONLY`), создает дескриптор открытого файла, соответствующий индексу в таблице дескрипторов файлов, открытых процессом. В данном случае файловому дескриптору присваивается значение 3, так как значения 0, 1, 2 заняты стандартными потоками (`stdin`, `stdout`, `stderr`),

а другие файлы процессом не открывались. Поле `fd_array[3]` указывает на `struct file`, связанную со структурой `struct inode`, соответствующую файлу «alphabet.txt».

Два вызова `fdopen()` стандартной библиотеки создают структуры `FILE` (`fs1, fs2`), поле `_fileno` которых инициализируется дескриптором открытого файла.

Вызов функции `setvbuf()` явно задает буферы для каждой из структур `FILE` и их размеры (20 байт), задавая указатели на начало и конец буфера. В данном случае устанавливается буферизация (`_IOFBF`).

В цикле поочередно для `fs1` и `fs2` вызывается функция `fscanf()` стандартной библиотеки. Так как была установлена буферизация, при первом вызове `fscanf()` буфер структуры `fs1` будет полностью заполнен, то есть в него сразу запишутся первые 20 символов (буквы 'A' – '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. Вызывая `fscanf()` для `fs2`, в буфер `fs2` запишутся оставшиеся символы (от 'u' до 'z'), так как `fs2` ссылается на тот же дескриптор открытого файла, что и структура `fs1`, а поле `f_pos` соответствующей структуры `struct file` было изменено. В переменную `c` запишется символ 'u'.

При последующих 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.

В случае многопоточной реализации возможны различные варианты вывода, в зависимости от способа планирования потоков. Возможна ситуация аналогичная однопоточному варианту, с той поправкой, что порядок вывода символов разными потоками может отличаться. Возможна также ситуация, при которой главный поток начинает вывод быстрее, так как второму потоку необходимо время на его создание.

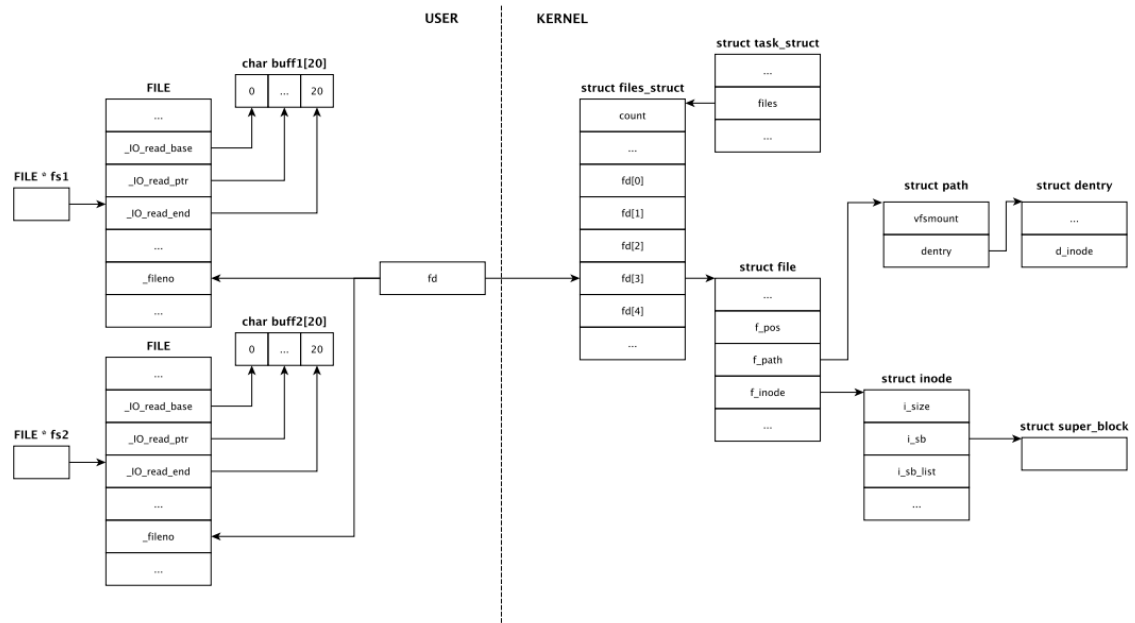


Рисунок 3 – Схема связи структур, используемых в первой программе

3 Вторая программа

Листинг 5 – Вторая программа

```

1 #include <fcntl.h>
2 #include <unistd.h>
3 #include <stdio.h>
4
5 int main()
6 {
7     char c;
8     int fd1 = open("alphabet.txt", O_RDONLY);
9     int fd2 = open("alphabet.txt", O_RDONLY);
10    int flag1 = 1, flag2 = 1;
11    while(flag1 == 1 || flag2 == 1)
12    {
13        if ((flag1 = read(fd1, &c, 1)) == 1)
14            printf("%c", c);
15        if ((flag2 = read(fd2, &c, 1)) == 1)
16            printf("%c", c);
17    }
18    puts("");

```

```
19     return 0;
20 }
```

```
● vladislav@linux-hp-pavilion:~/OS/6sem/lab_10/src$ gcc -Wall -Werror testKernelIO.c -o testKernelIO
● vladislav@linux-hp-pavilion:~/OS/6sem/lab_10/src$ ./testKernelIO
AAbbccddeeffgghhiijjkkllmmnnnooppqrrssttuuvvwwxxyyzz
○ vladislav@linux-hp-pavilion:~/OS/6sem/lab_10/src$
```

Рисунок 4 – Результат работы второй программы

Листинг 6 – Вторая программа с дополнительным потоком

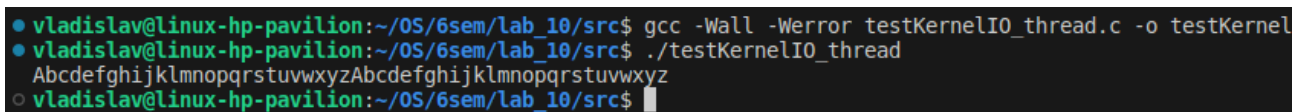
```
1 #include <fcntl.h>
2 #include <unistd.h>
3 #include <stdio.h>
4 #include <pthread.h>
5
6 pthread_mutex_t mutex;
7
8 struct args_struct
9 {
10     int fd;
11 };
12
13 void *read_buf(void *args)
14 {
15     struct args_struct *cur_args = (struct args_struct *) args;
16     int fd = cur_args->fd;
17     int flag = 1;
18     pthread_mutex_lock(&mutex);
19     while (flag == 1)
20     {
21         char c;
22         if ((flag = read(fd, &c, 1)) == 1)
23             printf("%c", c);
24     }
25     pthread_mutex_unlock(&mutex);
26
27     return NULL;
28 }
29
30 int main()
```



```

31 {
32     int fd1 = open("alphabet.txt", O_RDONLY);
33     struct args_struct args1 = { .fd = fd1 };
34
35     int fd2 = open("alphabet.txt", O_RDONLY);
36     struct args_struct args2 = { .fd = fd2 };
37
38     pthread_t td1, td2;
39     pthread_create(&td1, NULL, read_buf, &args1);
40     pthread_create(&td2, NULL, read_buf, &args2);
41     pthread_join(td1, NULL);
42     pthread_join(td2, NULL);
43     puts("");
44     return 0;
45 }

```



```

vladislav@linux-hp-pavilion:~/OS/6sem/lab_10/src$ gcc -Wall -Werror testKernelIO_thread.c -o testKernel
vladislav@linux-hp-pavilion:~/OS/6sem/lab_10/src$ ./testKernelIO_thread
AbcdefghijklmnopqrstuvwxyzAbcdefghijklmnopqrstuvwxyz
vladislav@linux-hp-pavilion:~/OS/6sem/lab_10/src$

```

Рисунок 5 – Результат работы второй программы с 2-мя дополнительными потоками

3.1 Анализ результата

Во второй программе посредством двух вызовов `open()` файл «alphabet.txt» дважды открывается только для чтения (`O_RDONLY`), создаются два дескриптора открытого файла (им присваиваются значения 3 и 4). Вместе с этим, 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 по два раза.

При многопоточной реализации алфавит также будет выведен два раза,

однако порядок вывода символов заранее предсказать невозможно.

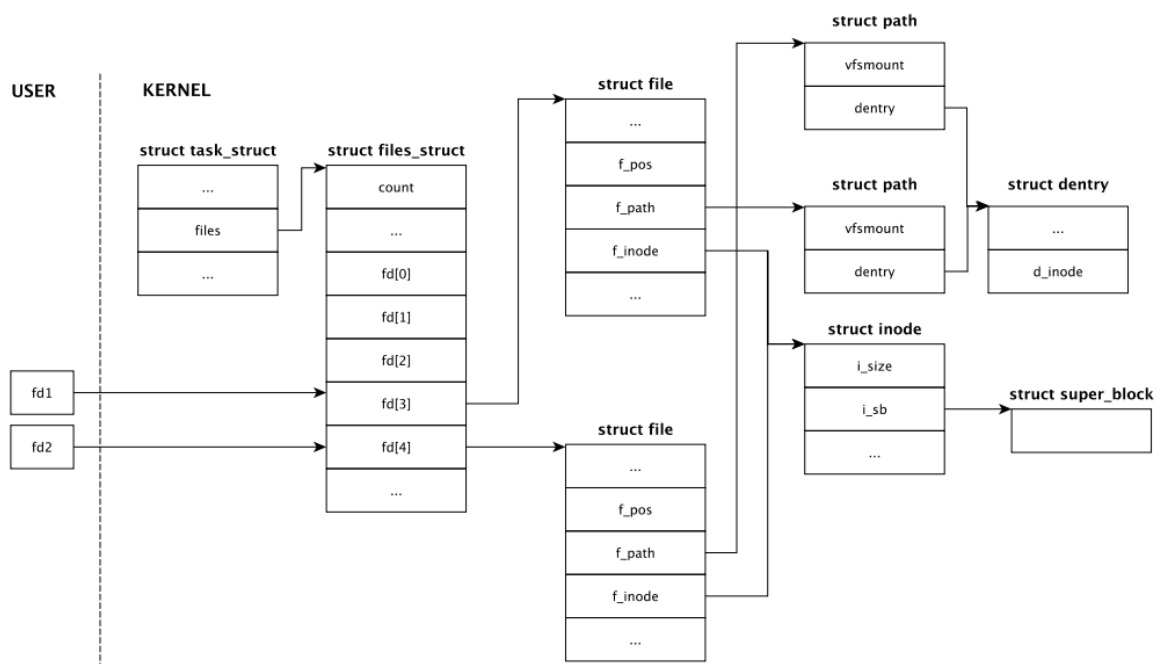


Рисунок 6 – Схема связи структур, используемых во второй программе

4 Третья программа

Листинг 7 – Третья программа с fopen

```
1 #include <stdio.h>
2 #include <sys/stat.h>
3
4 void file_info(FILE *fs)
5 {
6     struct stat statbuf;
7     stat("result.txt", &statbuf);
8     printf("st_ino: %ld\n", statbuf.st_ino);
9     printf("st_size: %ld\n", statbuf.st_size);
10    printf("pos: %ld\n", ftell(fs));
11 }
12
13 int main(void)
14 {
15     FILE *fs1 = fopen("result.txt", "w");
16     file_info(fs1);
17
18     FILE *fs2 = fopen("result.txt", "w");
19     file_info(fs2);
20
21     for (char ch = 'a'; ch <= 'z'; ++ch)
22         fprintf(ch % 2 ? fs1 : fs2, "%c", ch);
23
24     file_info(fs1);
25     fclose(fs1);
26
27     file_info(fs2);
28     fclose(fs2);
29
30     return 0;
31 }
```

Листинг 8 – Результат работы третьей программы с open

```
1 $ gcc -Wall -Werror test_fopen.c -o test_fopen
2 $ ./test_fopen
3 st_ino: 1526963
4 st_size: 0
5 pos: 0
6 st_ino: 1526963
7 st_size: 0
8 pos: 0
9 st_ino: 1526963
10 st_size: 0
11 pos: 13
12 st_ino: 1526963
13 st_size: 13
14 pos: 13
15
16 $ cat result.txt
17 bdfhjlnprtvxz
```

Листинг 9 – Третья программа с open

```
1 #include <sys/stat.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #include <stdio.h>
5
6 void file_info()
7 {
8     struct stat statbuf;
9     stat("result.txt", &statbuf);
10    printf("st_ino: %ld\n", statbuf.st_ino);
11    printf("st_size: %ld\n", statbuf.st_size);
12 }
13
14 int main(void)
15 {
16     int fd1 = open("result.txt", O_WRONLY, 0777);
17     int fd2 = open("result.txt", O_WRONLY, 0777);
18
19     for (char ch = 'a'; ch <= 'z'; ++ch)
20     {
21         char buf[2];
22         snprintf(buf, 2, "%c", ch);
23         if (ch % 2) {
24             write(fd1, buf, 1);
25             file_info();
26         }
27         else
28         {
29             write(fd2, buf, 1);
30             file_info();
31         }
32     }
33
34     return 0;
35 }
```

Листинг 10 – Результат работы третьей программы с open

```
1 $ gcc -Wall -Werror testopen.c -o test_open
2 $ ./test_open
3 st_ino: 1526963
4 st_size: 1
5 st_ino: 1526963
6 st_size: 1
7 st_ino: 1526963
8 st_size: 2
9 st_ino: 1526963
10 st_size: 2
11 st_ino: 1526963
12 st_size: 3
13 st_ino: 1526963
14 st_size: 3
15 st_ino: 1526963
16 st_size: 4
17 st_ino: 1526963
18 st_size: 4
19 st_ino: 1526963
20 st_size: 5
21 st_ino: 1526963
22 st_size: 5
23 st_ino: 1526963
24 st_size: 6
25 st_ino: 1526963
26 st_size: 6
27 st_ino: 1526963
28 st_size: 7
29 st_ino: 1526963
30 st_size: 7
31 st_ino: 1526963
32 st_size: 8
33 st_ino: 1526963
34 st_size: 8
35 st_ino: 1526963
36 st_size: 9
37 st_ino: 1526963
38 st_size: 9
39 st_ino: 1526963
```

```
40 st_size: 10
41 st_ino: 1526963
42 st_size: 10
43 st_ino: 1526963
44 st_size: 11
45 st_ino: 1526963
46 st_size: 11
47 st_ino: 1526963
48 st_size: 12
49 st_ino: 1526963
50 st_size: 12
51 st_ino: 1526963
52 st_size: 13
53 st_ino: 1526963
54 st_size: 13
55
56 $ cat result.txt
57 bdfhjlnprtvxz
```

4.1 Анализ результатов

В третьей программе с использованием `fopen()` файл дважды открывается для записи (опция «w»). 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 буферизация, при которой запись в файл из буфера происходит либо когда буфер заполнен, либо когда вызывается `fflush`, либо `fclose`.

В данном случае, с помощью вывода полей структуры `struct stat`, можно наблюдать, что запись происходит при вызове `fclose()`. До вызовов `fclose()` в цикле в файл записываются буквы латинского алфавита с помощью поочередной передачи функции `fprintf()` дескрипторов.

При вызове `fclose(fs1)` нечетные буквы алфавита записываются в файл. При вызове `fclose(fs2)`, так как поле `f_pos` соответствующей струк-

туры `struct file` не изменялось, запись в файл произойдет с начала файла, и записанные ранее символы перезапишутся четными буквами алфавита. Так как буферы содержали одинаковое количество символов, данные первой записи утеряны.

В третьей программе с использованием `open()` результат аналогичный, поскольку установлен флаг `O_WRONLY`. Схема связи структур, используемых в третьей программе с `open()` совпадает с рисунком 6.

Схема связи структур, используемых в третьей программе с `foren()`, представлена на рисунке 9.

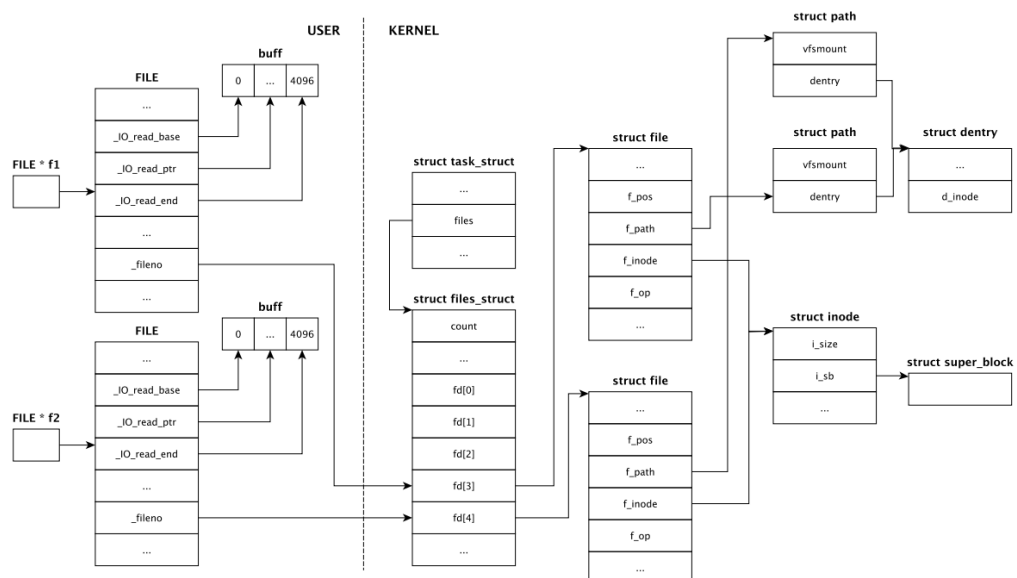


Рисунок 7 – Схема связи структур, используемых в третьей программе с `foren`

Проблема решается добавлением флага `O_APPEND` в программу с `open()` или изменением опции «w» на опцию «a» в программе с `foren()`.

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.