



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное 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:  
Буферизованный и небуферизованный ввод–вывод  
по курсу «Операционные системы»

Студент Дмитриенко А.В.

Группа ИУ7-63Б

Преподаватель Рязанова Н. Ю.

# 1 Структуры

Версия ядра: 6.1.138-1.

Листинг 1.1 – Структура struct `_IO_FILE`

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

```

29
30  /* 1+column number of pbase(); 0 is unknown. */
31  unsigned short _cur_column;
32  signed char _vtable_offset;
33  char _shortbuf[1];
34
35  _IO_lock_t *_lock;
36 #ifdef _IO_USE_OLD_IO_FILE
37 };

```

Листинг 1.2 – Структура struct stat

```

1  struct stat {
2      unsigned long st_dev;      /* Device. */
3      unsigned long st_ino;      /* File serial number. */
4      unsigned int st_mode;      /* File mode. */
5      unsigned int st_nlink;     /* Link count. */
6      unsigned int st_uid;       /* User ID of the file's owner. */
7      unsigned int st_gid;       /* Group ID of the file's group. */
8      unsigned long st_rdev;     /* Device number, if device. */
9      unsigned long __pad1;
10     long st_size;      /* Size of file, in bytes. */
11     int st_blksize;    /* Optimal block size for I/O. */
12     int __pad2;
13     long st_blocks;    /* Number 512-byte blocks allocated. */
14     long st_atime;     /* Time of last access. */
15     unsigned long st_atime_nsec;
16     long st_mtime;     /* Time of last modification. */
17     unsigned long st_mtime_nsec;
18     long st_ctime;     /* Time of last status change. */
19     unsigned long st_ctime_nsec;
20     unsigned int __unused4;
21     unsigned int __unused5;
22 };

```

## 2 Программы

### 2.1 Первая программа

Листинг 2.1 – 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 = 1;
14    while (flag1 == 1 || flag2 == 1)
15    {
16        char c;
17        flag1 = fscanf(fs1, "%c", &c);
18        if (flag1 == 1)
19            fprintf(stdout, "%c", c);
20        flag2 = fscanf(fs2, "%c", &c);
21        if (flag2 == 1)
22            fprintf(stdout, "%c", c);
23    }
24    return 0;
25 }
```

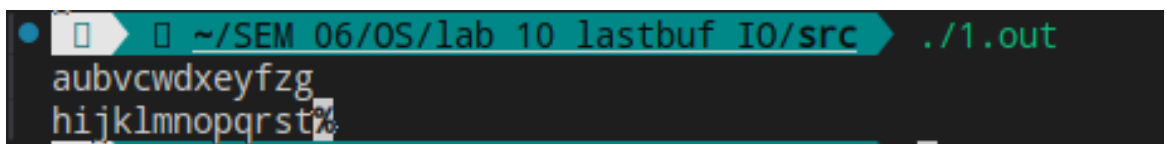


Рисунок 2.1 — Результат выполнения программы

В первой программе файл открывается только для чтения (O\_RDONLY). Системный вызов `open()` создает дескриптор открытого файла. Системный вызов `open()` возвращает индекс элемента в массиве `fd_array` структуры

files\_struct. Индекс равен 3, так как элементы массива fd\_array с индексами 0, 1, 2 инициализированы стандартными потоками stdin, stdout, stderr. Библиотечная функция fdopen() получает файловый дескриптор fd и создает экземпляр FILE (fs1, fs2), поля которых указывают на дескриптор fd, созданный системным вызовом open(). Буферы buff1, buff2 размером 20 байт. Функция setvbuf() для дескрипторов fs1, fs2 задает буферы buff1, buff2 с типом буферизации \_IOFBF—полная буферизация ввода/вывода.

При первом вызове fscanf() для fs1 в буфер buff1 считываются первые 20 символов. Значение f\_pos в структуре struct\_file открытого файла увеличивается на 20. В переменную c записывается символ 'a', значение переменной c выводится на экран функцией fprintf(). При первом вызове fscanf() для fs2 в буфер buff2 считываются оставшиеся в файле символы.

В цикле символы из buff1, buff2 будут поочередно выводиться на экран, пока один из буферов не будет пуст. В этом случае оставшиеся символы из второго буфера будут последовательно выведены на экран.

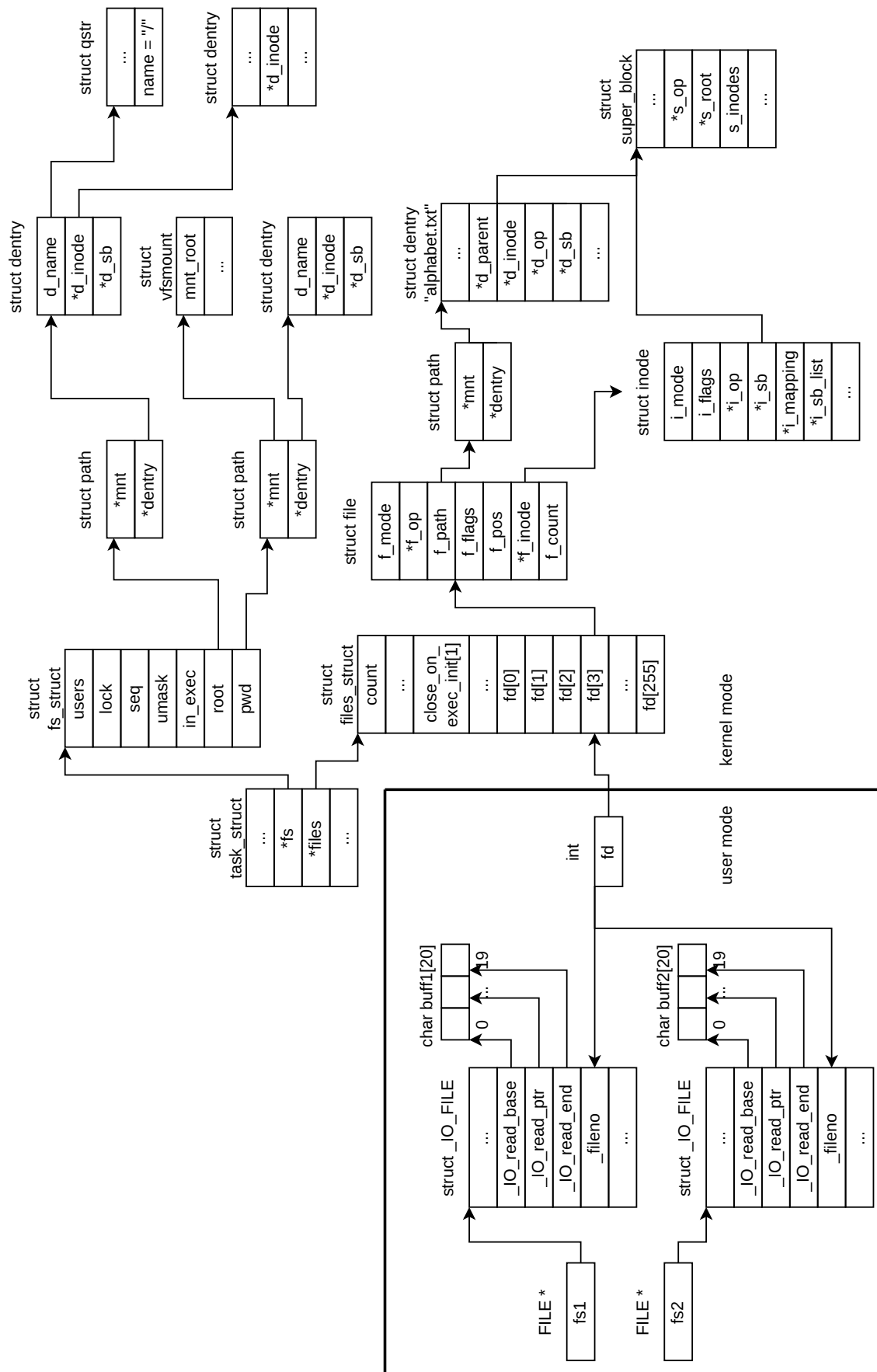


Рисунок 2.2 — Связи структур в первой программе

## 2.2 Вторая программа, первый вариант

### 2.2.1 Без использования потоков

Листинг 2.2 – Вторая программа, первый вариант

```
1 #include <fcntl.h>
2 #include <unistd.h>
3 int main()
4 {
5     char c;
6     int fd1 = open("alphabet.txt", O_RDONLY);
7     int fd2 = open("alphabet.txt", O_RDONLY);
8     int flag1 = 1, flag2 = 1;
9     while ((flag1 == 1) && (flag2 == 1))
10    {
11        if (1 == (flag1 = read(fd1, &c, 1)))
12        {
13            write(1, &c, 1);
14            if (1 == (flag2 = read(fd2, &c, 1)))
15                write(1, &c, 1);
16        }
17    }
18    return 0;
19 }
```

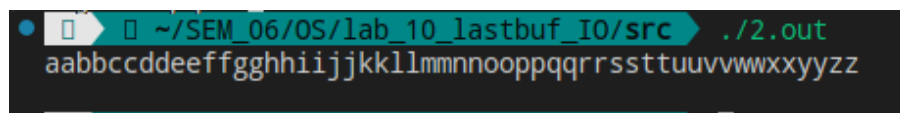


Рисунок 2.3 — Результат выполнения программы

Во второй программе один и тот же файл открывается два раза только для чтения (`O_RDONLY`). Системный вызов `open()` создает дескриптор открытого файла в таблице открытых файлов процесса и запись в системной таблице открытых файлов. Так как файл открывается дважды, то в системной таблице создается два дескриптора `struct file`, каждый из которых имеет собственное поле `f_pos`. Так как `f_pos` изменяется при поочередном вызове `read()` для каждого дескриптора соответствующее поле `f_pos` проходят по всем позициям файла, каждый символ выводится по два раза.

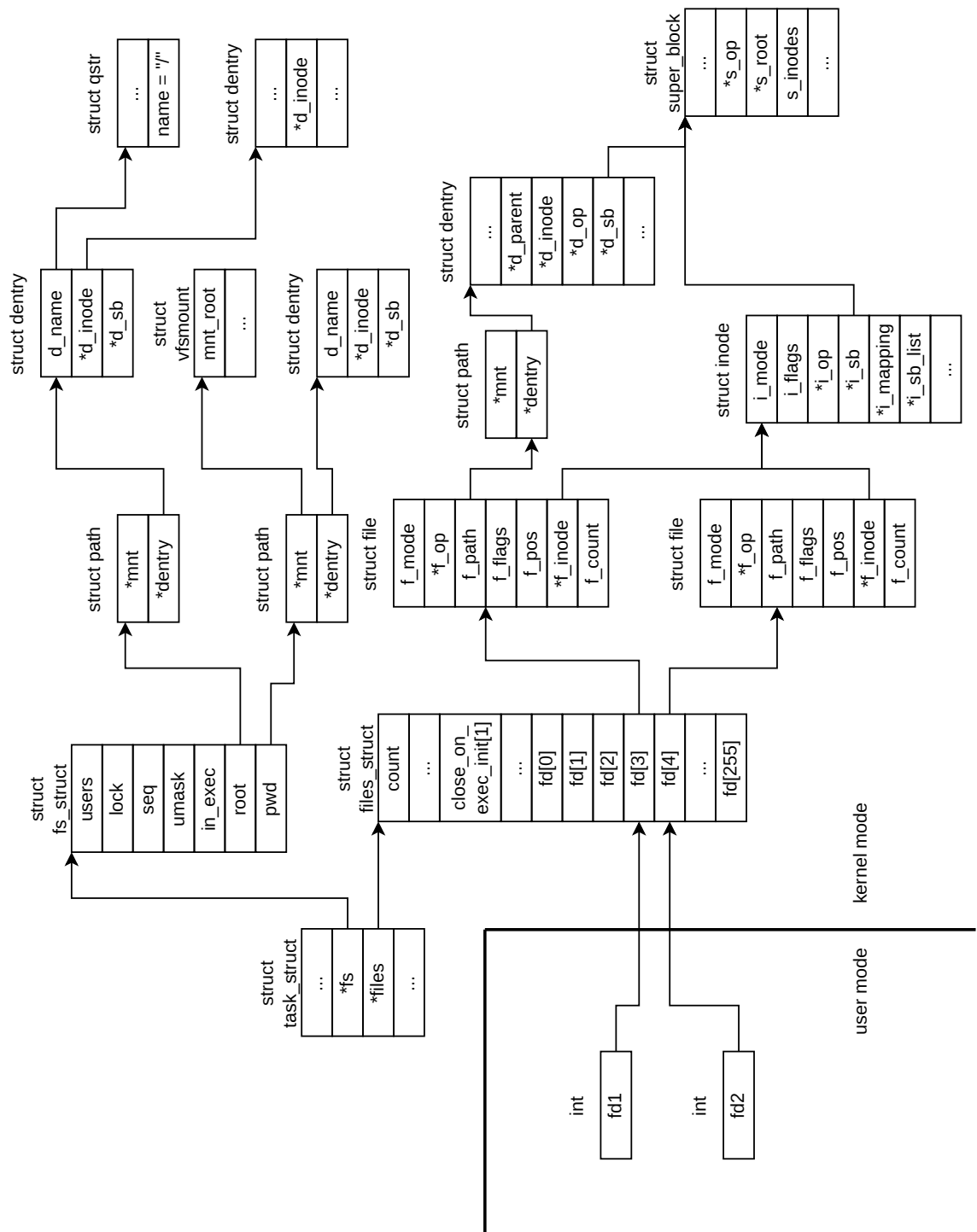


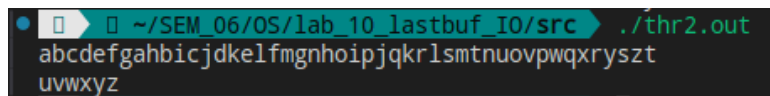
Рисунок 2.4 — Связи структур во второй программе



## 2.2.2 С использованием двух дополнительных потоков

Листинг 2.3 – Вторая программа, первый вариант (два дополнительных потока)

```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <fcntl.h>
4  #include <unistd.h>
5  void *thread_start(void *arg)
6  {
7      int *fd = arg;
8      char c;
9      while (read(*fd, &c, 1))
10         write(1, &c, 1);
11     return NULL;
12 }
13 int main()
14 {
15     int fd[2] = {open("alphabet.txt", O_RDONLY),
16                 open("alphabet.txt", O_RDONLY)};
17     pthread_t thr[2];
18     for (int i = 0; i < 2; i++)
19         if (pthread_create(&thr[i], NULL, thread_start, &fd[i]))
20         {
21             perror("pthread_create");
22             return 1;
23         }
24     close(fd[0]);
25     close(fd[1]);
26     return 0;
27 }
```

A terminal window with a dark background and light green text. The prompt is a blue square icon followed by the path ~/SEM\_06/OS/lab\_10\_lastbuf\_IO/src. The command ./thr2.out is entered. The output consists of two lines of lowercase letters: the first line contains 'abcdefghijklmnopqrstu' and the second line contains 'vwxyz'.

```
~/SEM_06/OS/lab_10_lastbuf_IO/src ./thr2.out  
abcdefghijklmnopqrstu  
vwxyz
```

Рисунок 2.5 — Результат выполнения программы

В многопоточной версии программы порядок вывода символов не определен, так как потоки выполняются параллельно (асинхронно). В случае с главным и дополнительным потоками дополнительный поток начинает выполнение позже главного, так как на его создание затрачивается время.

## 2.3 Вторая программа, второй вариант

### 2.3.1 Без использования потоков

Листинг 2.4 – Вторая программа, второй вариант

```
1  #include <fcntl.h>
2  #include <stdio.h>
3  #include <sys/stat.h>
4  #include <unistd.h>
5
6  struct stat statbuf;
7
8  int main()
9  {
10     int fd1 = open("q.txt", O_RDWR);
11     stat("q.txt", &statbuf);
12     fprintf(stdout, "open fd1: inode = %ld, size = %ld bytes\n",
13             statbuf.st_ino, statbuf.st_size);
14     int fd2 = open("q.txt", O_RDWR);
15     stat("q.txt", &statbuf);
16     fprintf(stdout, "open fd2: inode = %ld, size = %ld bytes\n",
17             statbuf.st_ino, statbuf.st_size);
18     for (char c = 'a'; c <= 'z'; c++) {
19         if (c % 2)
20             write(fd1, &c, 1);
21         else
22             write(fd2, &c, 1);
23         stat("q.txt", &statbuf);
24         fprintf(stdout, "write: inode = %ld, size = %ld bytes\n",
25                 statbuf.st_ino, statbuf.st_size);
26     }
27     close(fd1);
28     stat("q.txt", &statbuf);
29     fprintf(stdout, "close fd1: inode = %ld, size = %ld bytes\n",
30             statbuf.st_ino, statbuf.st_size);
31     close(fd2);
32     stat("q.txt", &statbuf);
33     fprintf(stdout, "close fd2: inode = %ld, size = %ld bytes\n",
34             statbuf.st_ino, statbuf.st_size);
35     return 0;
36 }
```

```
• [ ] [ ] ~/SEM_06/OS/lab_10_lastbuf_IO/src ./2_1.out
open fd1: inode = 44522035, size = 0 bytes
open fd2: inode = 44522035, size = 0 bytes
write: inode = 44522035, size = 1 bytes
write: inode = 44522035, size = 1 bytes
write: inode = 44522035, size = 2 bytes
write: inode = 44522035, size = 2 bytes
write: inode = 44522035, size = 3 bytes
write: inode = 44522035, size = 3 bytes
write: inode = 44522035, size = 4 bytes
write: inode = 44522035, size = 4 bytes
write: inode = 44522035, size = 5 bytes
write: inode = 44522035, size = 5 bytes
write: inode = 44522035, size = 6 bytes
write: inode = 44522035, size = 6 bytes
write: inode = 44522035, size = 7 bytes
write: inode = 44522035, size = 7 bytes
write: inode = 44522035, size = 8 bytes
write: inode = 44522035, size = 8 bytes
write: inode = 44522035, size = 9 bytes
write: inode = 44522035, size = 9 bytes
write: inode = 44522035, size = 10 bytes
write: inode = 44522035, size = 10 bytes
write: inode = 44522035, size = 11 bytes
write: inode = 44522035, size = 11 bytes
write: inode = 44522035, size = 12 bytes
write: inode = 44522035, size = 12 bytes
write: inode = 44522035, size = 13 bytes
write: inode = 44522035, size = 13 bytes
close fd1: inode = 44522035, size = 13 bytes
close fd2: inode = 44522035, size = 13 bytes
```

Рисунок 2.6 — Результат выполнения программы

В программе один и тот же файл ("q.txt") открывается дважды для чтения и записи (O\_RDWR). Системный вызов open() создает дескриптор открытого файла в таблице открытых файлов процесса и запись в системной таблице открытых файлов. Так как файл открывается дважды, то в системной таблице создается два дескриптора struct file, каждый из которых имеет собственный указатель f\_pos. При первом вызове write() для fd1 символ 'a' записывается в файл на 0 позицию и соответствующий указатель f\_pos увеличивается на 1. При первом вызове write() для fd2 символ 'b' также записывается в файл на 0 позицию и соответствующий указатель f\_pos увеличивается на 1. Таким образом, при поочередной записи символов в файл он будет содержать только символы, которые записывались через fd2. Произошла потеря данных.

Чтобы избежать потерю данных, файл открывается с флагом O\_APPEND. При первом вызове write() для fd1 символ 'a' записывается в файл на последнюю позицию (0). При первом вызове write() для fd2 символ 'b' записывается в файл на последнюю позицию (1). При поочередной записи символов в файл он будет содержать все символы алфавита.

## 2.3.2 С использованием двух дополнительных потоков

Листинг 2.5 – Вторая программа, второй вариант (два дополнительных потока)

```
1  #include <fcntl.h>
2  #include <pthread.h>
3  #include <stdio.h>
4  #include <sys/stat.h>
5  #include <unistd.h>
6
7  struct stat statbuf;
8
9  struct thread_arg {
10     int fd;
11     int i;
12 };
13
14 void* thread_start(void* arg)
15 {
16     struct thread_arg* targ = arg;
17
18     for (char c = 'a'; c <= 'z'; c++)
19         if (c % 2 == targ->i) {
20             write(targ->fd, &c, 1);
21             fstat(targ->fd, &statbuf);
22             off_t size = lseek(targ->fd, 0, SEEK_CUR);
23             fprintf(stdout, "write %d: inode number = %ld, size
24                         = %ld bytes\n",
25                         targ->i, statbuf.st_ino, size);
26         }
27     return NULL;
28 }
29
30 int main()
31 {
32     int fd[2] = {
33         open("q.txt", O_RDWR),
34         open("q.txt", O_RDWR)
35     };
```

```

36     pthread_t thr[2];
37     struct thread_arg targ[2];
38
39     for (int i = 0; i < 2; i++)
40     {
41         targ[i].fd = fd[i];
42         targ[i].i = i;
43
44         if (pthread_create(&thr[i], NULL, thread_start,
45                             &targ[i]))
46         {
47             perror("pthread_create");
48             return 1;
49         }
50
51         for (int i = 0; i < 2; i++)
52             if (pthread_join(thr[i], NULL)) {
53                 perror("pthread_join");
54                 return 1;
55             }
56
57         close(fd[0]);
58         close(fd[1]);
59
60         return 0;
61     }

```

В многопоточной версии программы потоки выполняются параллельно (асинхронно). Однако потеря данных полностью аналогична той, что однопоточной версии программы: при первом вызове `write()` для `fd[0]` (первый поток) символ 'a' записывается в файл на 0 позицию и соответствующий указатель `f_pos` увеличивается на 1; при первом вызове `write()` для `fd[1]` (второй поток) символ 'b' также записывается в файл на 0 позицию и соответствующий указатель `f_pos` увеличивается на 1. Таким образом, при поочередной записи символов в файл он будет содержать только символы, которые записывались вторым потоком.

Чтобы избежать потерю данных, файл можно дважды открыть для чтения, записи и записи в конец (`O_RDWR`). При первом вызове `write()`

```
• [ ] ~/SEM_06/OS/lab_10_lastbuf_IO/src ./2_1_thread.out
write 0: inode number = 44522035, size = 1 bytes
write 0: inode number = 44522035, size = 2 bytes
write 1: inode number = 44522035, size = 1 bytes
write 1: inode number = 44522035, size = 2 bytes
write 0: inode number = 44522035, size = 3 bytes
write 1: inode number = 44522035, size = 3 bytes
write 0: inode number = 44522035, size = 4 bytes
write 1: inode number = 44522035, size = 4 bytes
write 0: inode number = 44522035, size = 5 bytes
write 0: inode number = 44522035, size = 6 bytes
write 1: inode number = 44522035, size = 5 bytes
write 1: inode number = 44522035, size = 6 bytes
write 1: inode number = 44522035, size = 7 bytes
write 1: inode number = 44522035, size = 8 bytes
write 1: inode number = 44522035, size = 9 bytes
write 1: inode number = 44522035, size = 10 bytes
write 1: inode number = 44522035, size = 11 bytes
write 1: inode number = 44522035, size = 12 bytes
write 1: inode number = 44522035, size = 13 bytes
write 0: inode number = 44522035, size = 7 bytes
write 0: inode number = 44522035, size = 8 bytes
write 0: inode number = 44522035, size = 9 bytes
write 0: inode number = 44522035, size = 10 bytes
write 0: inode number = 44522035, size = 11 bytes
write 0: inode number = 44522035, size = 12 bytes
write 0: inode number = 44522035, size = 13 bytes
```

Рисунок 2.7 — Результат выполнения программы

для `fd[0]` (первый поток) символ 'а' записывается в файл на последнюю позицию (0). При первом вызове `write()` для `fd[1]` (второй поток) символ 'b' также записывается в файл на последнюю позицию (1). Таким образом, при поочередной записи символов в файл он будет содержать все символы алфавита. Порядок символов не определен, так как потоки выполняются параллельно (асинхронно).

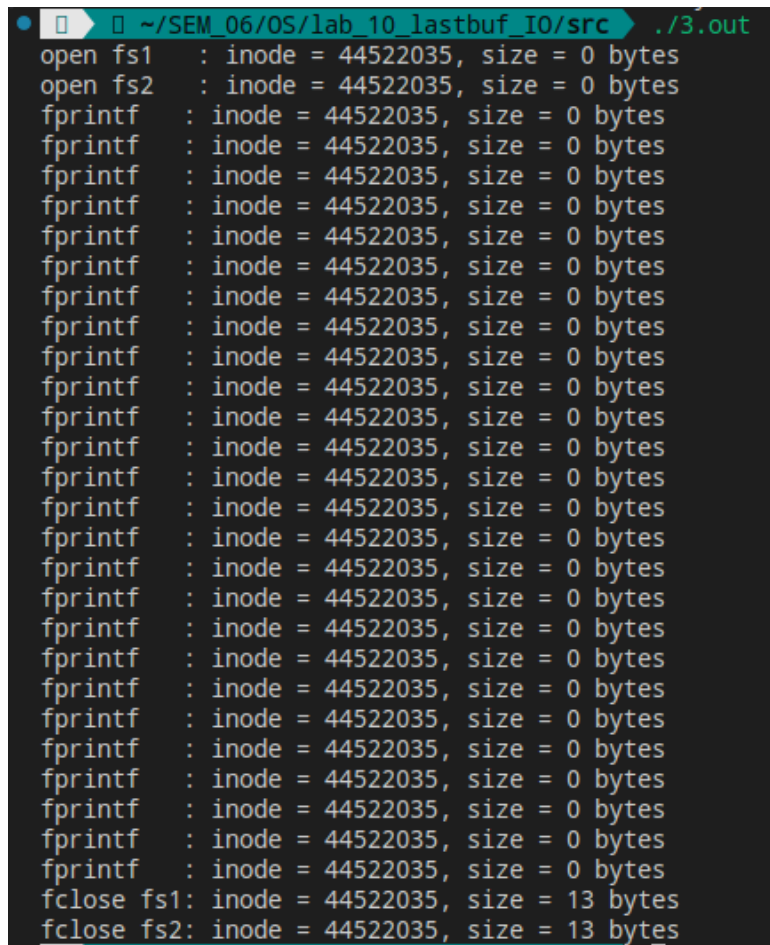
## 2.4 Третья программа

Листинг 2.6 – Третья программа

```
1 #include <fcntl.h>
2 #include <stdio.h>
3 #include <sys/stat.h>
4
5 struct stat statbuf;
6
7 int main()
8 {
9     FILE* fs1 = fopen("q.txt", "w");
10    stat("q.txt", &statbuf);
11    fprintf(stdout, "fopen fs1: inode = %ld, size = %ld bytes\n",
12            statbuf.st_ino, statbuf.st_size);
13    FILE* fs2 = fopen("q.txt", "w");
14    stat("q.txt", &statbuf);
15    fprintf(stdout, "fopen fs2: inode = %ld, size = %ld bytes\n",
16            statbuf.st_ino, statbuf.st_size);
17
18    for (char c = 'a'; c <= 'z'; c++) {
19        if (c % 2)
20            fprintf(fs1, "%c", c);
21        else
22            fprintf(fs2, "%c", c);
23        stat("q.txt", &statbuf);
24        fprintf(stdout, "fprintf: inode = %ld, size = %ld
25                bytes\n",
26                statbuf.st_ino, statbuf.st_size);
27    }
28
29    fclose(fs1);
30    stat("q.txt", &statbuf);
31    fprintf(stdout, "fclose fs1: inode = %ld, size = %ld
32            bytes\n",
33            statbuf.st_ino, statbuf.st_size);
34    fclose(fs2);
35    stat("q.txt", &statbuf);
36    fprintf(stdout, "fclose fs2: inode = %ld, size = %ld
37            bytes\n",
38            statbuf.st_ino, statbuf.st_size);
```



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



```
• [ ] [ ] ~/.SEM_06/OS/lab_10_lastbuf_IO/src ./3.out
open fs1 : inode = 44522035, size = 0 bytes
open fs2 : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fprintf : inode = 44522035, size = 0 bytes
fclose fs1: inode = 44522035, size = 13 bytes
fclose fs2: inode = 44522035, size = 13 bytes
```

Рисунок 2.8 — Результат выполнения программы

В третьей программе файл дважды открывается на чтение и запись функцией `fopen()` из библиотеки буферизованного ввода-вывода `stdio.h`. В результате выполнения `fopen()` в системной таблице открытых файлов создаются два дескриптора `struct file`, каждый из которых имеет собственный указатель `f_pos`, при этом оба дескриптора ссылаются на один и тот же `inode`. Библиотечная функция `fprintf()` выполняет буферизованный вывод. При этом буфер создается неявно. Данные из буфера записываются в файл по трем причинам:

- 1) буфер заполнен,
- 2) вызвана функция `fflush()` (принудительная запись),
- 3) вызвана функция `close()` / `fclose()`.

В данном случае запись в файл происходит в результате вызова функции `fclose()`. При вызове `fclose()` для `fs1` буфер для `fs1` записывается в файл. При вызове `fclose()` для `fs2` все содержимое файла очищается и буфер для `fs2` записывается в файл. Таким образом, произошла потеря данных, так как в файле находится только содержимое буфера для `fs2`.

Чтобы избежать потерю данных, при втором открытии файла можно указать режим записи в конец файла ("a"). В таком случае содержимое при вызове `fclose()` для `fs2` содержимое файла не будет очищаться, а содержимое буфера для `fs2` будет добавлено в конец файла.

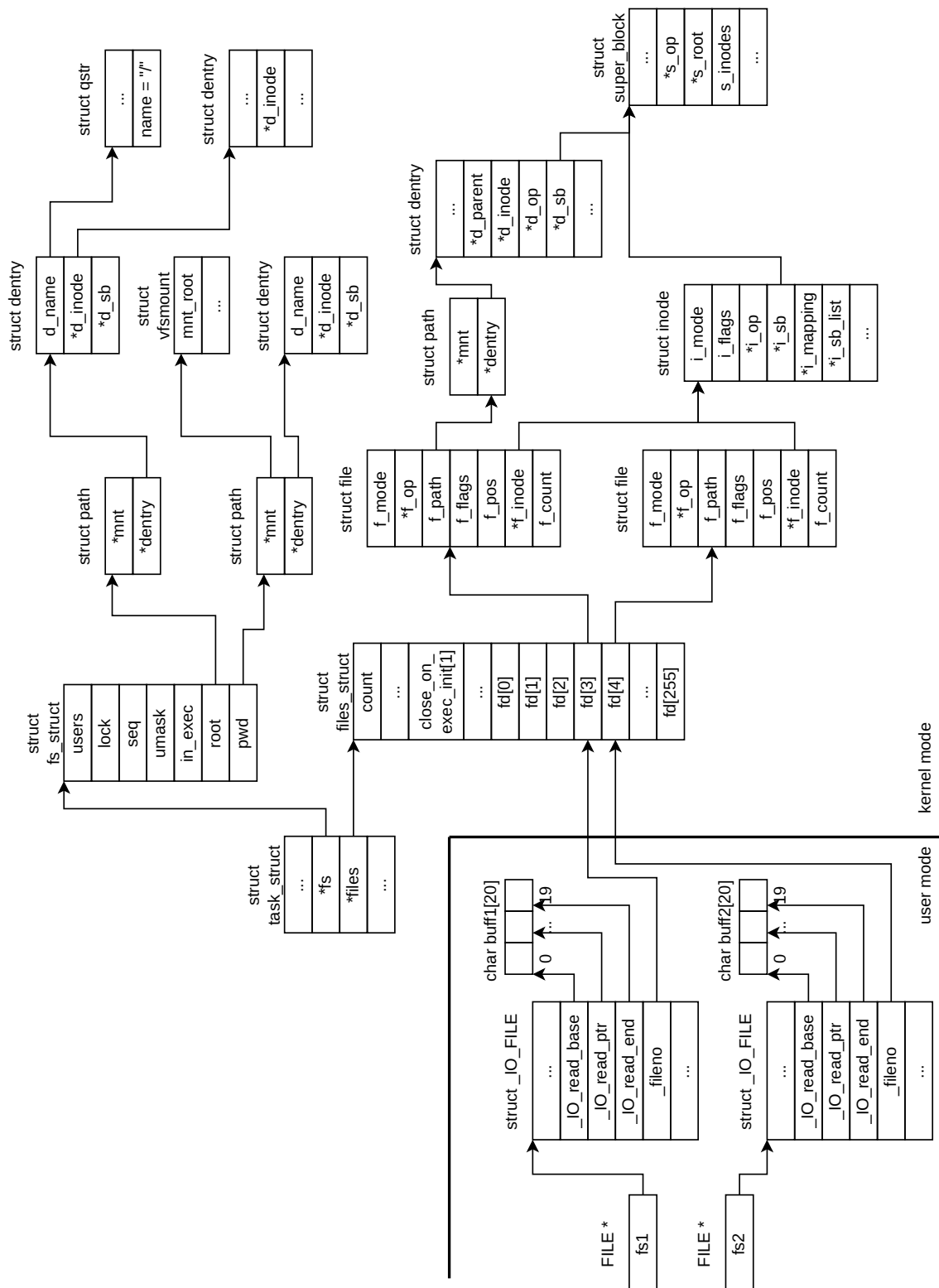


Рисунок 2.9 — Связи структур в третьей программе