



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 не буферизованный ввод-вывод»

Студент ИУ7-63Б
(Группа)

(Подпись, дата)

Авдейкина В. П.
(И. О. Фамилия)

Преподаватель

(Подпись, дата)

Рязанова Н. Ю.
(И. О. Фамилия)

2024 г.

1 Структуры

Версия ядра: 6.5.0-32-generic.

Листинг 1.1 – Структура struct _IO_FILE

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

```

31     char _shortbuf[1];
32
33     _IO_lock_t *_lock;
34 #ifdef _IO_USE_OLD_IO_FILE
35 };

```

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 };

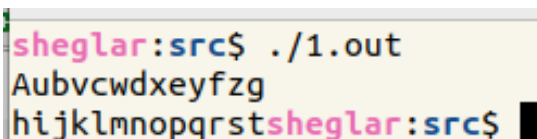
```

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 }
```



```
sheglar:src$ ./1.out
Aubvcwdxeyfzg
hijklmnopqrstsheglar:src$
```

Рисунок 2.1 — Результат выполнения программы

В первой программе файл открывается только для чтения (O_RDONLY). С помощью системного вызова `open()` создается 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()` возвращает указатели на структуры `struct FILE` (`fs1`, `fs2`), поля которых указывают на дескриптор `fd`, созданный системным вызовом `open()`. Создаются буферы `buff1`, `buff2` размером 20 байт. С помощью функции `setvbuf()` для дескрипторов `fs1`, `fs2` задаются буферы `buff1`, `buff2` с типом буферизации `_IOFBF`.

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 `s` записывается символ `'a'`, значение переменной `s` выводится на экран с помощью функции `fprintf()`. При первом вызове `fscanf()` для `fs2` в буфер `buff2` считываются оставшиеся в файле символы. В переменную `s` записывается символ `'u'`, значение переменной `s` выводится на экран с помощью функции `fprintf()`.

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.

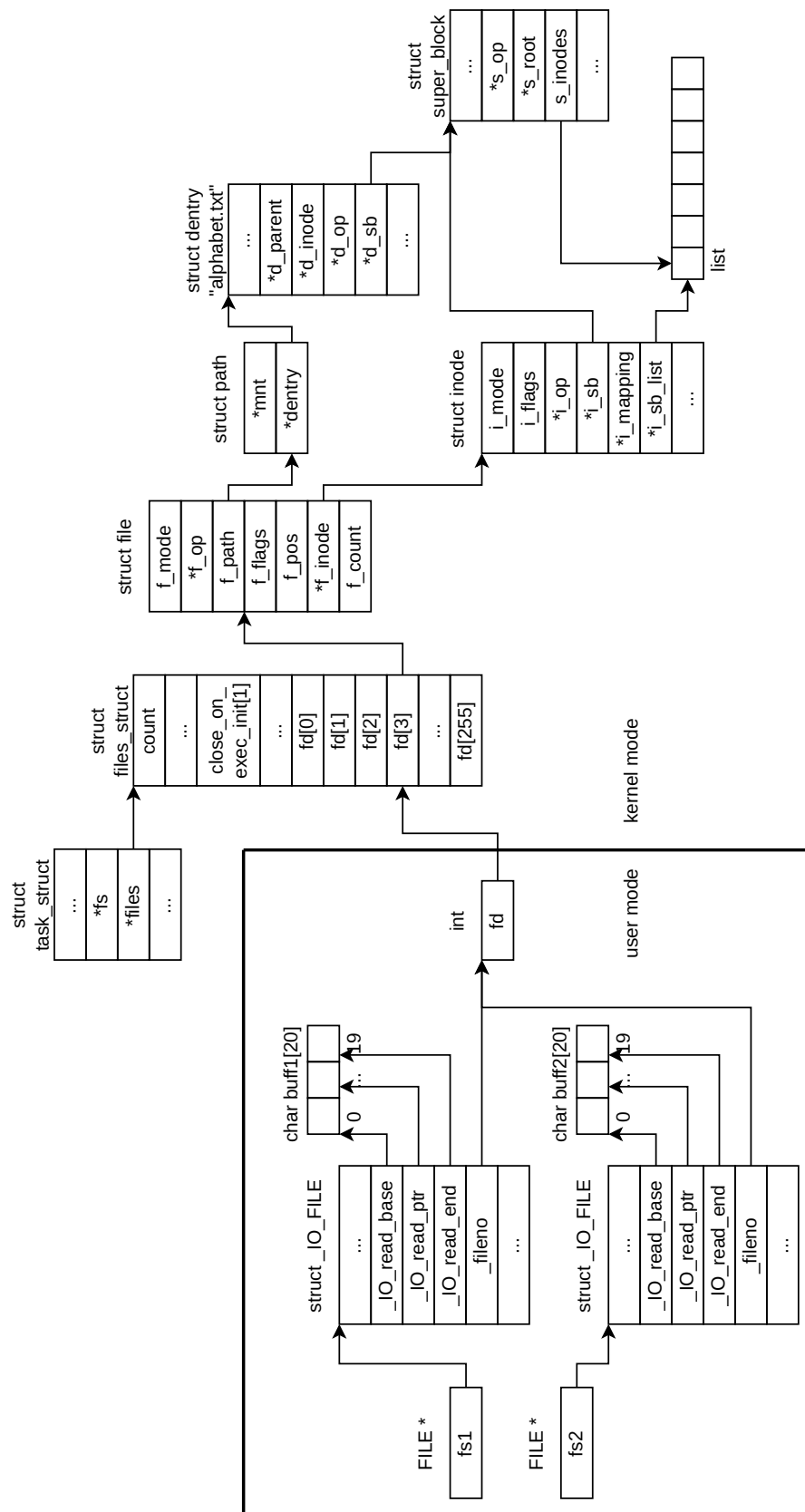


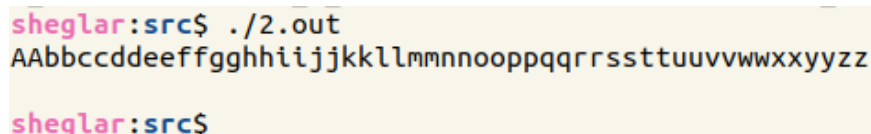
Рисунок 2.2 — Связи структур в первой программе

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 }
```



```
sheglar:src$ ./2.out
AAbbccddeeffgghhiijjkkllmmnnoopppqqrrssttuuvvwwxxyyzz
sheglar:src$
```

Рисунок 2.3 — Результат выполнения программы

Во второй программе один и тот же файл открывается два раза только для чтения (`O_RDONLY`). С помощью системного вызова `open()` создается 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`. Таким образом, в данном случае чтение из файла является независимым: при поочередном вызове `read()` для каждого дескриптора соответствующие указатели `f_pos` проходят по всем позициям файла, каждый символ считывается и выводится по два раза.

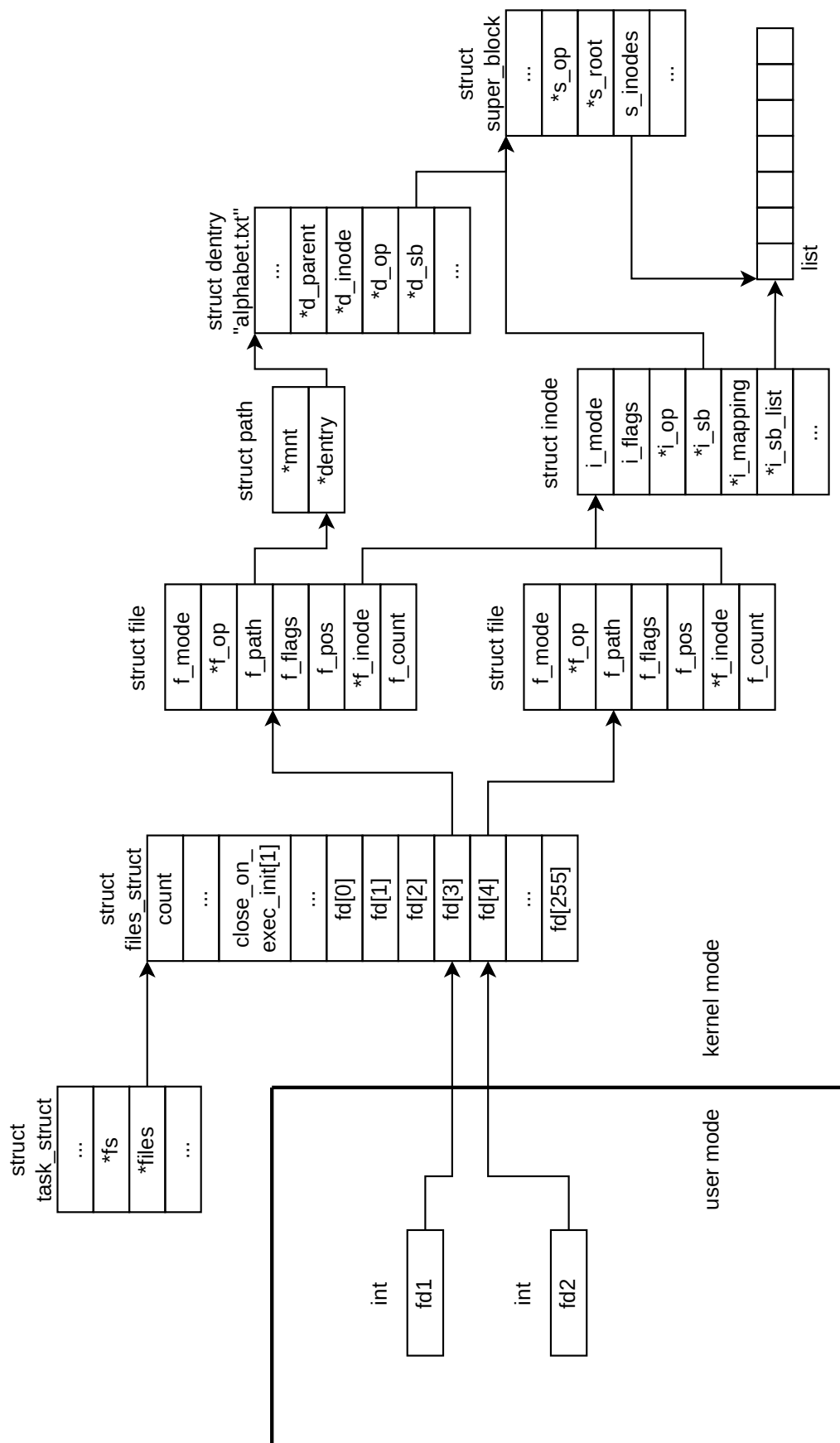


Рисунок 2.4 — Связи структур во второй программе

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     for (int i = 0; i < 2; i++)
25         if (pthread_join(thr[i], NULL))
26         {
27             perror("pthread_join");
28             return 1;
29         }
30     close(fd[0]);
31     close(fd[1]);
32     return 0;
33 }
```

```
sheglar:src$ ./2_thread.out  
AbcdefghijklmnAobpcqdrstguhviwxyzl  
mnopqrstuvwxyz  
sheglar:src$ █
```

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.

При создании дополнительных потоков связи структур не меняются, так как ресурсами, в том числе открытыми файлами, владеет процесс.

2.3 Вторая программа, второй вариант

2.3.1 Без использования потоков

Листинг 2.4 – Вторая программа, второй вариант

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

```

sheglar:src$ ./2_1.out
open fd1 : inode = 5153037, size = 1 bytes
open fd2 : inode = 5153037, size = 1 bytes
write  : inode = 5153037, size = 1 bytes
write  : inode = 5153037, size = 1 bytes
write  : inode = 5153037, size = 2 bytes
write  : inode = 5153037, size = 2 bytes
write  : inode = 5153037, size = 3 bytes
write  : inode = 5153037, size = 3 bytes
write  : inode = 5153037, size = 4 bytes
write  : inode = 5153037, size = 4 bytes
write  : inode = 5153037, size = 5 bytes
write  : inode = 5153037, size = 5 bytes
write  : inode = 5153037, size = 6 bytes
write  : inode = 5153037, size = 6 bytes
write  : inode = 5153037, size = 7 bytes
write  : inode = 5153037, size = 7 bytes
write  : inode = 5153037, size = 8 bytes
write  : inode = 5153037, size = 8 bytes
write  : inode = 5153037, size = 9 bytes
write  : inode = 5153037, size = 9 bytes
write  : inode = 5153037, size = 10 bytes
write  : inode = 5153037, size = 10 bytes
write  : inode = 5153037, size = 11 bytes
write  : inode = 5153037, size = 11 bytes
write  : inode = 5153037, size = 12 bytes
write  : inode = 5153037, size = 12 bytes
write  : inode = 5153037, size = 13 bytes
write  : inode = 5153037, size = 13 bytes
close fd1: inode = 5153037, size = 13 bytes
close fd2: inode = 5153037, size = 13 bytes
sheglar:src$ cat q.txt
bdfhjlnprtvxzsheglar:src$

```

Рисунок 2.6 — Результат выполнения программы

В программе один и тот же файл ("q.txt") открывается дважды для чтения и записи (O_RDWR). С помощью системного вызова open() создается 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 с помощью fd2. Произошла потеря данных.

Чтобы избежать потерю данных, файл можно дважды открыть для чтения, записи и добавления записи в конец (O_RDWR | 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' также записывается в файл на последнюю позицию (1). Таким образом, при поочередной записи символов в файл он будет содержать все символы алфавита.

Для второго варианта второй программы связи структур не меняются.

2.3.2 С использованием двух дополнительных потоков

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

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

```

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

```

```

sheglar:src$ ./2_1_thread.out
write 0: inode number = 5153037, size = 1 bytes
write 0: inode number = 5153037, size = 2 bytes
write 0: inode number = 5153037, size = 3 bytes
write 0: inode number = 5153037, size = 4 bytes
write 1: inode number = 5153037, size = 1 bytes
write 0: inode number = 5153037, size = 5 bytes
write 1: inode number = 5153037, size = 5 bytes
write 0: inode number = 5153037, size = 6 bytes
write 1: inode number = 5153037, size = 6 bytes
write 0: inode number = 5153037, size = 7 bytes
write 0: inode number = 5153037, size = 8 bytes
write 1: inode number = 5153037, size = 7 bytes
write 0: inode number = 5153037, size = 9 bytes
write 1: inode number = 5153037, size = 9 bytes
write 0: inode number = 5153037, size = 10 bytes
write 1: inode number = 5153037, size = 10 bytes
write 1: inode number = 5153037, size = 11 bytes
write 1: inode number = 5153037, size = 11 bytes
write 0: inode number = 5153037, size = 11 bytes
write 1: inode number = 5153037, size = 11 bytes
write 0: inode number = 5153037, size = 12 bytes
write 1: inode number = 5153037, size = 12 bytes
write 0: inode number = 5153037, size = 13 bytes
write 1: inode number = 5153037, size = 13 bytes
write 1: inode number = 5153037, size = 13 bytes
write 1: inode number = 5153037, size = 13 bytes
sheglar:src$ cat q.txt
acegikmoqsuwysheglar:src$ █

```

Рисунок 2.7 — Результат выполнения программы

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 | O_APPEND`). При первом вызове `write()` 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).

При создании дополнительных потоков связи структур не меняются, так как ресурсами, в том числе открытыми файлами, владеет процесс.

2.4 Третья программа

Листинг 2.6 – Третья программа

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <sys/stat.h>
4 #define PRINT_STAT(action) \
5     do { \
6         stat("q.txt", &statbuf); \
7         fprintf(stdout, action ": inode = %ld, size = %ld bytes\n", \
8             statbuf.st_ino, statbuf.st_size); \
9     } while (0);
10 struct stat statbuf;
11 int main()
12 {
13     FILE *fs1 = fopen("q.txt", "w");
14     PRINT_STAT("open fs1");
15     FILE *fs2 = fopen("q.txt", "w");
16     PRINT_STAT("open fs2");
17     for (char c = 'a'; c <= 'z'; c++)
18     {
19         if (c % 2)
20             fprintf(fs1, "%c", c);
21         else
22             fprintf(fs2, "%c", c);
23         PRINT_STAT("fprintf");
24     }
25     fclose(fs1);
26     PRINT_STAT("fclose fs1");
27     fclose(fs2);
28     PRINT_STAT("fclose fs2");
29     return 0;
30 }
```

В третьей программе файл дважды открывается на чтение и запись с помощью функции `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`. С помощью библиотечной функции `fprintf()` выполняется

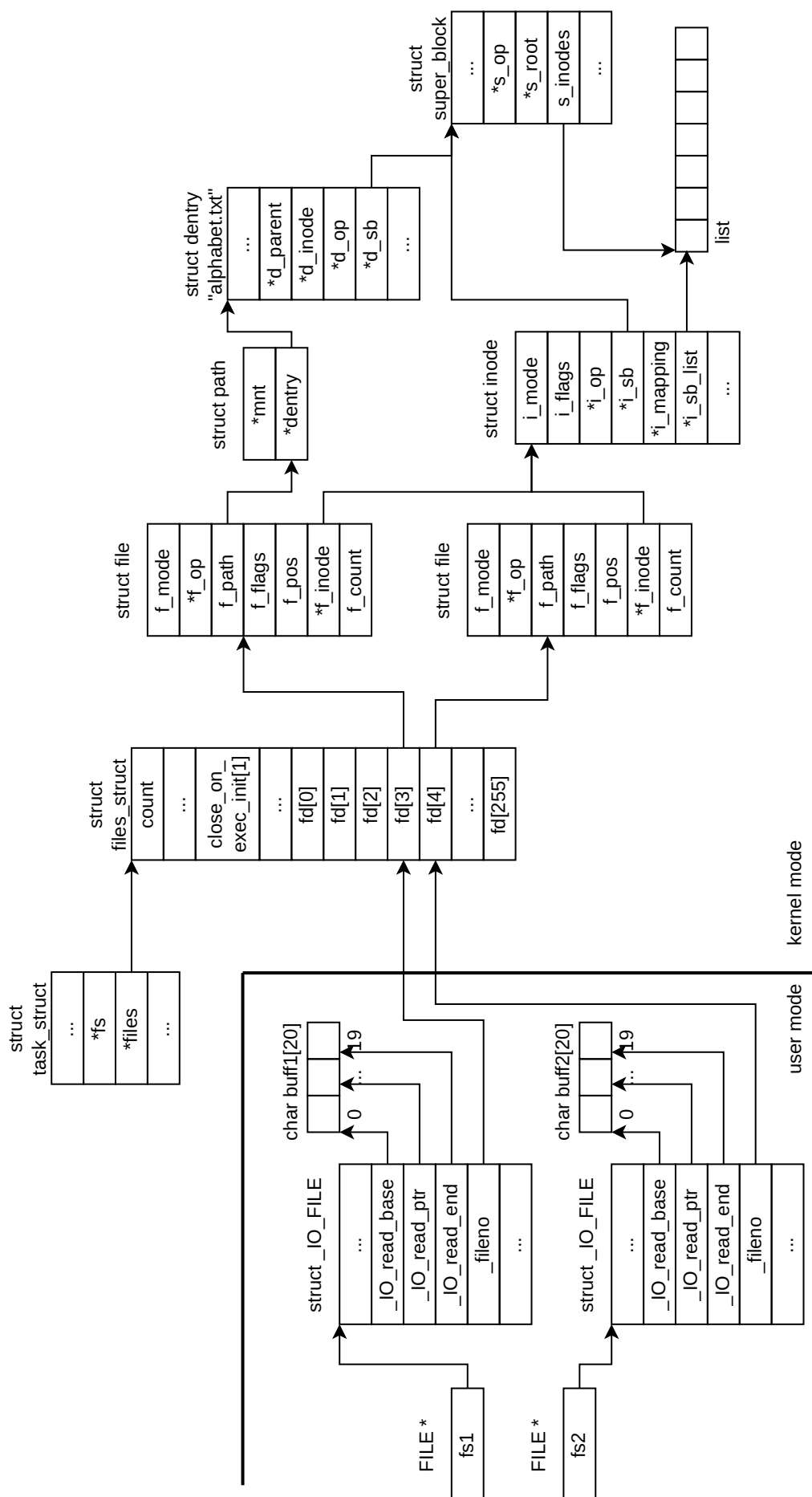


Рисунок 2.9 — Связи структур в третьей программе