



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»

---

**ОТЧЕТ**  
по Лабораторной работе  
по дисциплине «Операционные системы»  
на тему: «Буферизованный и небуферизованный ввод–вывод»

Студент группы ИУ7-63Б

\_\_\_\_\_  
(Подпись, дата)

Фролова Л. В.  
(Фамилия И.О.)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

Рязанова Н. Ю.  
(Фамилия И.О.)

Москва — 2025 г.

# 1 Структуры

Версия ядра: 6.5.0-32-generic.

Листинг 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 flags. */
6
7     /* The following pointers correspond to the C++ streambuf 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 undo. */
18    char *_IO_save_base; /* Pointer to start of non-current get area. */
19    char *_IO_backup_base; /* Pointer to first valid character of backup
20                           area */
21
22    char *_IO_save_end; /* Pointer to end of non-current get area. */
23
24    struct _IO_marker *_markers;
25
26    struct _IO_FILE *_chain;
27
28    int _fileno;
29    int _flags2;
30    __off_t _old_offset; /* This used to be _offset but it's too small. */
31
32    /* 1+column number of pbase(); 0 is unknown. */
33    unsigned short _cur_column;
34    signed char _vtable_offset;
35    char _shortbuf[1];
36
37    _IO_lock_t *_lock;
38 #ifdef _IO_USE_OLD_IO_FILE
39 };
40 #endif
```

## Листинг 2 — Структура struct stat

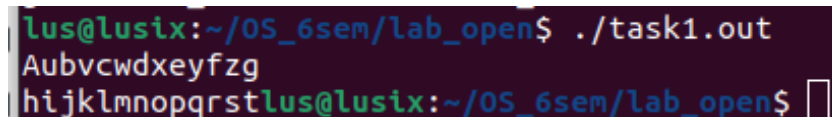
```
1 struct stat {
2     unsigned long st_dev;      /* Device. */
3     unsigned long st_ino;      /* File serial number. */
4     unsigned int  st_mode;     /* File mode. */
5     unsigned int  st_nlink;    /* Link count. */
6     unsigned int  st_uid;      /* User ID of the file's owner. */
7     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 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
11     FILE *fs2 = fdopen(fd, "r");
12     char buff2[20];
13     setvbuf(fs2, buff2, _IOFBF, 20);
14     int flag1 = 1, flag2 = 1;
15
16     while (flag1 == 1 || flag2 == 1)
17     {
18         char c;
19         flag1 = fscanf(fs1, "%c", &c);
20         if (flag1 == 1)
21             fprintf(stdout, "%c", c);
22         flag2 = fscanf(fs2, "%c", &c);
23         if (flag2 == 1)
24             fprintf(stdout, "%c", c);
25     }
26
27     return 0;
28 }
```



```
lus@lusix:~/OS_6sem/lab_open$ ./task1.out
Aubvcwdxeyfzg
hijklmnopqrstlus@lusix:~/OS_6sem/lab_open$
```

Рисунок 1 — Результат выполнения программы

Файл "alphabet.txt" открывается только для чтения (O\_RDONLY) 1 раз системным вызовом `open()`, который создает дескриптор открытого файла и возвращает индекс элемента в массиве `fd_array` структуры `files_struct`. Индекс равен 3, поскольку элементы массива `fd_array` с индексами 0, 1, 2 инициализированы стандартными потоками `stdin`, `stdout`, `stderr`.

Библиотечная функция `fdopen()` возвращает указатели `fs1` и `fs2` на структуры `struct FILE`, поля которых указывают на дескриптор `fd`, созданный системным вызовом `open()`. Создаются буферы `buff1`, `buff2` размером 20 байт. Функция `setvbuf()` для дескрипторов `fs1`, `fs2` задает буферы `buff1` и `buff2` с типом буферизации `_IOFBF` (Fully Buffered – полная буферизация).

При первом вызове `fscanf()` для `fs1` в буфер `buff1` считываются первые 20 символов, то есть буфер `buff1` становится заполненным. Значение `f_pos` в структуре `struct_file` открытого файла увеличивается на 20, в переменную `s` записывается символ 'а' и значение этой переменной выводится на экран функцией `fprintf()`. При первом вызове `fscanf()` для `fs2` в буфер `buff2` считываются оставшиеся в файле 6 символов, в переменную `s` записывается символ 'u' и значение переменной `s` выводится на экран функцией `fprintf()`. В цикле символы из `buff1` и `buff2` будут поочередно выводиться на экран до тех пор, пока один из буферов не станет пуст. В таком случае оставшиеся символы из второго буфера будут последовательно выведены на экран.

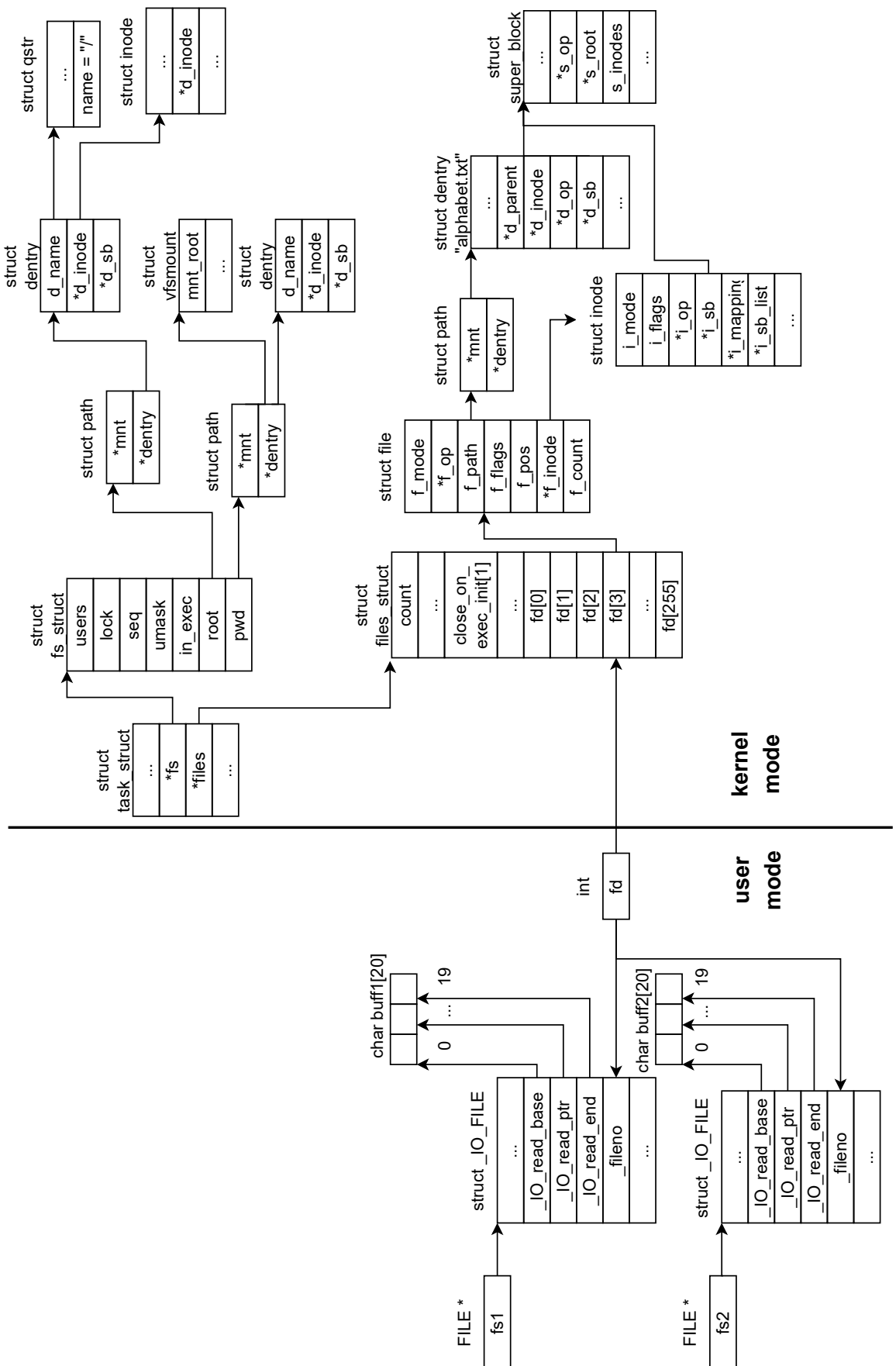


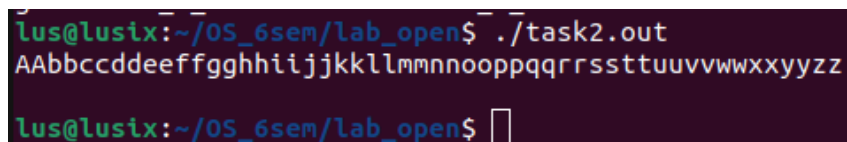
Рисунок 2 — Связи структур в первой программе

## 2.2 Вторая программа – первый вариант

### 2.2.1 Без использования дополнительных потоков

Листинг 4 — Вторая программа, первый вариант

```
1 #include <fcntl.h>
2 #include <unistd.h>
3
4 int main()
5 {
6     char c;
7     int fd1 = open("alphabet.txt", O_RDONLY);
8     int fd2 = open("alphabet.txt", O_RDONLY);
9     int flag1 = 1, flag2 = 1;
10    while ((flag1 == 1) && (flag2 == 1))
11    {
12        if (1 == (flag1 = read(fd1, &c, 1)))
13        {
14            write(1, &c, 1);
15            if (1 == (flag2 = read(fd2, &c, 1)))
16                write(1, &c, 1);
17        }
18    }
19    return 0;
20 }
```



```
lus@lusix:~/OS_6sem/lab_open$ ./task2.out
AAbbccddeeffgghhiijjkkllmmnnnooppqrrssttuuvvwxxyzz
lus@lusix:~/OS_6sem/lab_open$ █
```

Рисунок 3 — Результат выполнения программы

Один и тот же файл "alphabet.txt" открывается два раза только для чтения (O\_RDONLY) системным вызовом `open()`, который создает дескриптор открытого файла в таблице открытых файлов процесса и запись в системной таблице открытых файлов. Поскольку файл был открыт дважды, в системной таблице создается два дескриптора `struct file`, каждый из которых имеет собственный указатель `f_pos`. В данной программе чтение из файла является независимым, то есть при поочередном вызове `read()` для каждого из дескрипторов соответствующие указатели `f_pos` проходят по всем позициям файла, каждый символ

считывается и выводится на экран два раза.

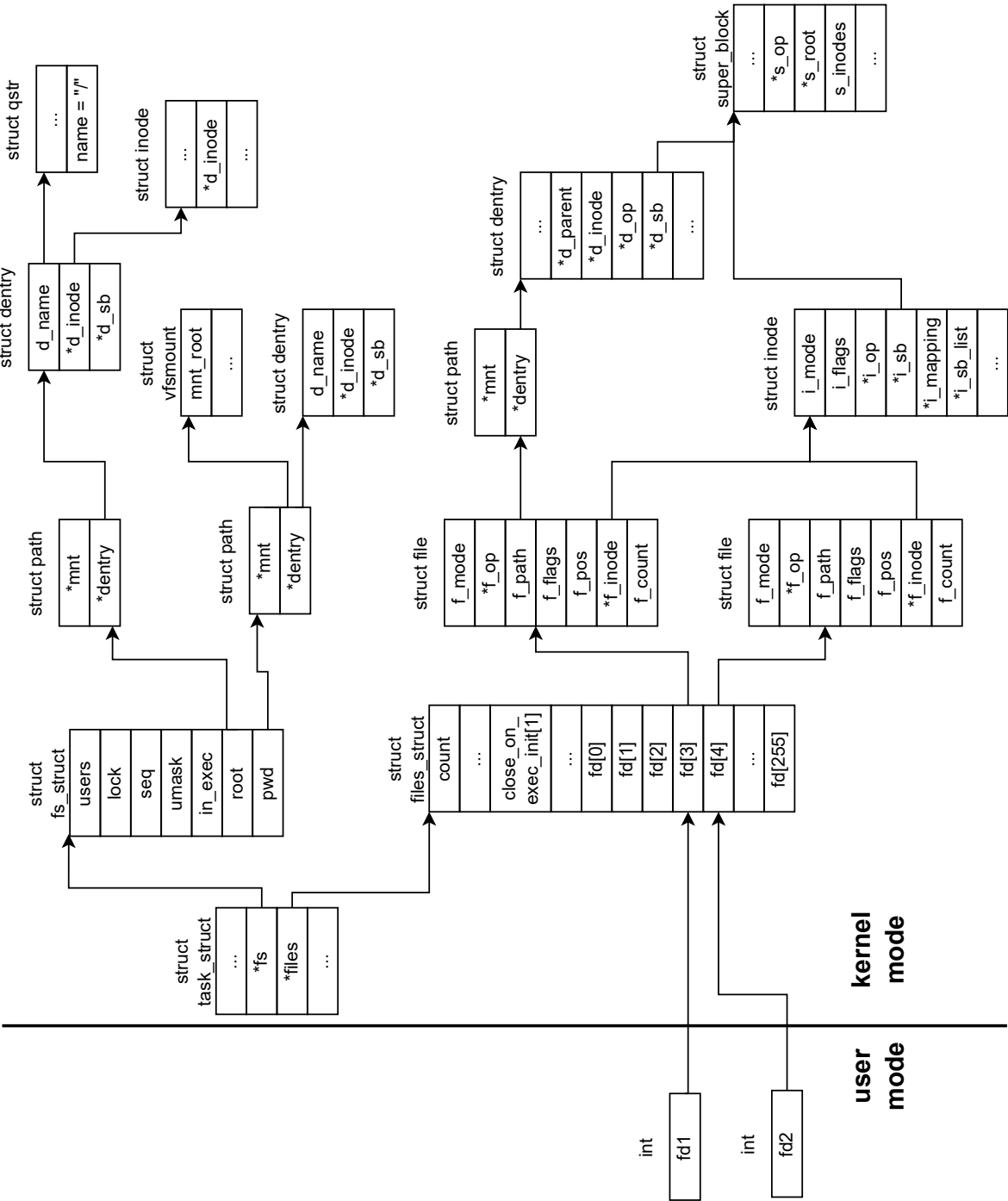


Рисунок 4 — Связи структур во второй программе



## 2.2.2 С использованием двух дополнительных потоков

Листинг 5 — Вторая программа, первый вариант(два дополнительных потока)

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <fcntl.h>
4 #include <unistd.h>
5
6 void *func_thread(void *arg)
7 {
8     int *fd = arg;
9     char c;
10    while (read(*fd, &c, 1))
11        write(1, &c, 1);
12    return NULL;
13 }
14
15 int main()
16 {
17     int fd[2] = {open("alphabet.txt", O_RDONLY),
18                 open("alphabet.txt", O_RDONLY)};
19     pthread_t thr[2];
20     for (int i = 0; i < 2; i++)
21     {
22         if (pthread_create(&thr[i], NULL, func_thread, &fd[i]))
23         {
24             perror("pthread_create");
25             return 1;
26         }
27     }
28     for (int i = 0; i < 2; i++)
29     {
30         if (pthread_join(thr[i], NULL))
31         {
32             perror("pthread_join");
33             return 1;
34         }
35     }
36     close(fd[0]);
37     close(fd[1]);
38
39     return 0;
40 }
```

```
lus@lusix:~/OS_6sem/lab_open$ ./task2_thread.out
AbcdefghijAkblcmdneofpgqhrisjtkulvmwxyz
nopqrstuvwxyz
lus@lusix:~/OS_6sem/lab_open$
```

Рисунок 5 — Результат выполнения программы

Порядок вывода символов в многопоточной версии программы не определен, поскольку потоки выполняются асинхронно.

## 2.3 Вторая программа – второй вариант

### 2.3.1 Без использования дополнительных потоков

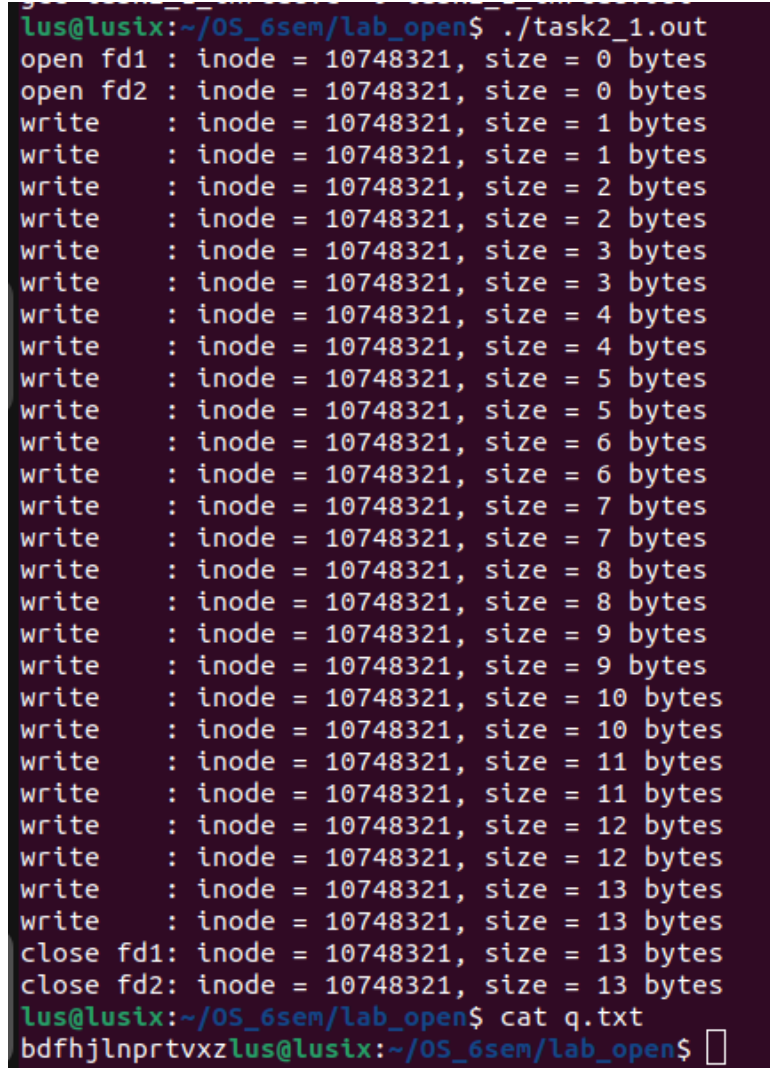
Листинг 6 — Вторая программа, второй вариант

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

```

30 }
31 close(fd1);
32 print_stat("close fd1");
33 close(fd2);
34 print_stat("close fd2");
35 return 0;
36 }

```



```

lus@lusix:~/OS_6sem/lab_open$ ./task2_1.out
open fd1 : inode = 10748321, size = 0 bytes
open fd2 : inode = 10748321, size = 0 bytes
write  : inode = 10748321, size = 1 bytes
write  : inode = 10748321, size = 1 bytes
write  : inode = 10748321, size = 2 bytes
write  : inode = 10748321, size = 2 bytes
write  : inode = 10748321, size = 3 bytes
write  : inode = 10748321, size = 3 bytes
write  : inode = 10748321, size = 4 bytes
write  : inode = 10748321, size = 4 bytes
write  : inode = 10748321, size = 5 bytes
write  : inode = 10748321, size = 5 bytes
write  : inode = 10748321, size = 6 bytes
write  : inode = 10748321, size = 6 bytes
write  : inode = 10748321, size = 7 bytes
write  : inode = 10748321, size = 7 bytes
write  : inode = 10748321, size = 8 bytes
write  : inode = 10748321, size = 8 bytes
write  : inode = 10748321, size = 9 bytes
write  : inode = 10748321, size = 9 bytes
write  : inode = 10748321, size = 10 bytes
write  : inode = 10748321, size = 10 bytes
write  : inode = 10748321, size = 11 bytes
write  : inode = 10748321, size = 11 bytes
write  : inode = 10748321, size = 12 bytes
write  : inode = 10748321, size = 12 bytes
write  : inode = 10748321, size = 13 bytes
write  : inode = 10748321, size = 13 bytes
close fd1: inode = 10748321, size = 13 bytes
close fd2: inode = 10748321, size = 13 bytes
lus@lusix:~/OS_6sem/lab_open$ cat q.txt
bdfhjlnprtvxz
lus@lusix:~/OS_6sem/lab_open$

```

Рисунок 6 — Результат выполнения программы

В данной программе один и тот же файл "q.txt" открывается дважды для чтения и записи (O\_RDWR) системным вызовом `open()`, который 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 нулевую позицию и соответствующий указатель

f\_pos инкрементируется. При первом вызове write() для fd2 символ 'b' также записывается в файл на нулевую позицию и соответствующий указатель f\_pos инкрементируется.

В результате, при поочередной записи символов в файл он будет содержать только символы, которые записывались через fd2, а данные, записанные через fd1, будут потеряны.

Избежать потери данных можно дважды открыв файл "q.txt" с использованием флагов O\_RDWR | O\_APPEND (открыть для чтения, записи, добавления записи в конец файла). В таком случае, 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.

### 2.3.2 С использованием двух дополнительных потоков

Листинг 7 — 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
7 struct stat statbuf;
8 void print_stat(const char* action, int i)
9 {
10     stat("q.txt", &statbuf);
11     fprintf(stdout, "%s %d: inode = %ld, size = %ld bytes\n",
12             action, i, statbuf.st_ino, statbuf.st_size);
13 }
14 struct thread_arg {
15     int fd;
16     int i;
17 };
18 void *func_thread(void *arg)
19 {
20     struct thread_arg *targ = arg;
21     for (char c = 'a'; c <= 'z'; c++)
22         if (c % 2 == targ->i)
23             {
```

```

24     write(targ->fd, &c, 1);
25     print_stat("write", targ->i);
26 }
27 return NULL;
28 }
29
30 int main()
31 {
32     int fd[2] = {open("q.txt", O_RDWR),
33                 open("q.txt", O_RDWR)};
34     pthread_t thr[2];
35     struct thread_arg targ[2];
36     for (int i = 0; i < 2; i++)
37     {
38         targ[i].fd = fd[i];
39         targ[i].i = i;
40         if (pthread_create(&thr[i], NULL, func_thread, &targ[i]))
41         {
42             perror("pthread_create");
43             return 1;
44         }
45     }
46     for (int i = 0; i < 2; i++)
47         if (pthread_join(thr[i], NULL))
48         {
49             perror("pthread_join");
50             return 1;
51         }
52     close(fd[0]);
53     close(fd[1]);
54     return 0;
55 }

```

```
lus@lusix:~/OS_6sem/lab_open_result/lst$ ./task2_1_thread.out
write 0: inode = 10753659, size = 1 bytes
write 0: inode = 10753659, size = 2 bytes
write 1: inode = 10753659, size = 1 bytes
write 0: inode = 10753659, size = 3 bytes
write 1: inode = 10753659, size = 3 bytes
write 0: inode = 10753659, size = 4 bytes
write 1: inode = 10753659, size = 5 bytes
write 0: inode = 10753659, size = 5 bytes
write 1: inode = 10753659, size = 6 bytes
write 0: inode = 10753659, size = 6 bytes
write 1: inode = 10753659, size = 7 bytes
write 0: inode = 10753659, size = 7 bytes
write 1: inode = 10753659, size = 8 bytes
write 0: inode = 10753659, size = 8 bytes
write 1: inode = 10753659, size = 8 bytes
write 0: inode = 10753659, size = 9 bytes
write 1: inode = 10753659, size = 9 bytes
write 0: inode = 10753659, size = 10 bytes
write 1: inode = 10753659, size = 10 bytes
write 0: inode = 10753659, size = 11 bytes
write 1: inode = 10753659, size = 11 bytes
write 0: inode = 10753659, size = 12 bytes
write 1: inode = 10753659, size = 13 bytes
write 0: inode = 10753659, size = 13 bytes
write 1: inode = 10753659, size = 13 bytes
lus@lusix:~/OS_6sem/lab_open_result/lst$
```

Рисунок 7 — Результат выполнения программы

Порядок записи символов в многопоточной версии программы не определен, поскольку потоки выполняются асинхронно. При первом вызове `write()` для `fd[0]` (первый поток) символ 'a' записывается в файл на нулевую позицию и соответствующий указатель `f_pos` инкрементируется. При первом вызове `write()` для `fd[1]` (второй поток) символ 'b' также записывается в файл на нулевую позицию и соответствующий указатель `f_pos` инкрементируется.

В результате, при поочередной записи символов в файл он будет содержать только символы, которые записывались вторым потоком.

Избежать потери данных можно дважды открыв файл "q.txt" с использованием флагов `O_RDWR | O_APPEND` (открыть для чтения, записи, добавления записи в конец файла). В таком случае для каждого из потоков символ будет записываться на последнюю позицию в файле, поэтому в конце работы программы файл будет содержать все символы алфавита.

## 2.4 Третья программа

Листинг 8 — Третья программа

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <sys/stat.h>
4
5 struct stat statbuf;
6 void print_stat(const char *action)
7 {
8     struct stat statbuf;
9     if (stat("q.txt", &statbuf) == 0)
10     {
11         fprintf(stdout, "%s: inode = %ld, size = %ld bytes\n",
12                 action, statbuf.st_ino, statbuf.st_size);
13     }
14     else
15         perror("stat");
16 }
17 int main()
18 {
19     FILE *fs1 = fopen("q.txt", "w");
20     print_stat("fopen fs1 ");
21     FILE *fs2 = fopen("q.txt", "w");
22     print_stat("fopen fs2 ");
23     for (char c = 'a'; c <= 'z'; c++)
24     {
25         if (c % 2)
26             fprintf(fs1, "%c", c);
27         else
28             fprintf(fs2, "%c", c);
29         print_stat("fprintf ");
30     }
31     fclose(fs1);
32     print_stat("fclose fs1");
33     fclose(fs2);
34     print_stat("fclose fs2");
35     return 0;
36 }
```

```

lus@lusix:~/OS_6sem/lab_open$ ./task3.out
fopen fs1 : inode = 10748321, size = 0 bytes
fopen fs2 : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fprintf : inode = 10748321, size = 0 bytes
fclose fs1: inode = 10748321, size = 13 bytes
fclose fs2: inode = 10748321, size = 13 bytes
lus@lusix:~/OS_6sem/lab_open$ cat q.txt
bdfhjlnprtvxz
lus@lusix:~/OS_6sem/lab_open$ 

```

Рисунок 8 — Результат выполнения программы

Третья программа отличается от второй программы тем, что в ней файл "q.txt" дважды открывается на чтение и запись функцией `fopen()` из библиотеки буферизованного ввода–вывода `stdio.h`. В результате выполнения вызова функции `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 данных в файл);
3. вызвана функция `close()` / `fclose()` (закрытие файла).

В третьей программе 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` содержимое файла очищается и буфер для `fs2` записывается в файл. В итоге, происходит потеря данных, поскольку в файле будет находиться только содержимое буфера для `fs2`.

Избежать потери данных можно указав при втором открытии файла режим записи в конец файла ("a"). Таким образом, при вызове `fclose()` для `fs2` содержимое файла не будет очищено, а содержимое буфера для `fs2` будет добавлено в конец файла.

