



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"

Тема Буферизованный и не буферизованный ввод-вывод

---

Студент Хамзина Р. Р.

---

Группа ИУ7-63Б

---

Оценка (баллы) \_\_\_\_\_

---

Преподаватель Рязанова Н. Ю.

---

Москва — 2022 г.

# 1 Первая программа

## Коды программы

Листинг 1.1 – Один поток

```
1 #include <stdio.h>
2 #include <fcntl.h>
3
4 #define RESET    "\033[0m"
5 #define RED      "\033[1;31m"
6 #define BLUE     "\033[1;34m"
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 = 2;
21
22     while (flag1 == 1 || flag2 == 1)
23     {
24         char c;
25         flag1 = fscanf(fs1, "%c", &c);
26
27         if (flag1 == 1)
28         {
29             fprintf(stdout, RED "%c" RESET, c);
30         }
31
32         flag2 = fscanf(fs2, "%c", &c);
33
34         if (flag2 == 1)
```

```

35     {
36         fprintf(stdout, BLUE "%c" RESET, c);
37     }
38 }
39
40 fprintf(stdout, "\n");
41 return 0;
42 }

```

## Листинг 1.2 – Два потока

```

1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <pthread.h>
4
5 #define RESET    "\033[0m"
6 #define RED      "\033[1;31m"
7 #define BLUE     "\033[1;34m"
8
9 typedef struct args
10 {
11     FILE *fs;
12     char *color;
13 } arg_struct;
14
15 void *print_letter(void *arg)
16 {
17     arg_struct *now_arg = (arg_struct *) arg;
18     FILE *fs = now_arg->fs;
19
20     int flag = 1;
21     char c;
22
23     while (flag == 1)
24     {
25         flag = fscanf(fs, "%c", &c);
26
27         if (flag == 1)
28         {
29             fprintf(stdout, "%s%c" RESET, now_arg->color, c);
30
31             sleep(1);

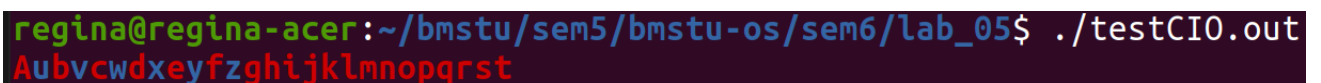
```

```

32     }
33 }
34
35     return NULL;
36 }
37
38 int main(void)
39 {
40     int fd = open("alphabet.txt", O_RDONLY);
41
42     FILE *fs1 = fdopen(fd, "r");
43     char buff1[20];
44     setvbuf(fs1, buff1, _IOFBF, 20);
45     arg_struct arg1 = {.fs = fs1, .color = RED};
46
47     FILE *fs2 = fdopen(fd, "r");
48     char buff2[20];
49     setvbuf(fs2, buff2, _IOFBF, 20);
50     arg_struct arg2 = {.fs = fs2, .color = BLUE};
51     pthread_t tid;
52     pthread_create(&tid, NULL, print_letter, &arg2);
53
54     print_letter(&arg1);
55
56     pthread_join(tid, NULL);
57     fprintf(stdout, "\n");
58
59     return 0;
60 }

```

## Результаты выполнения программ



```

regina@regina-acer:~/bmstu/sem5/bmstu-os/sem6/lab_05$ ./testCIO.out
Aubvcwdxeyfzghijklmnopqrst

```

Рисунок 1.1 – Результат работы первой программы: один поток

```
regina@regina-acer:~/bmstu/sem5/bmstu-os/sem6/lab_05$ ./testC10_threads.out  
Auvbwcdxeyfzghijklmnopqrst
```

Рисунок 1.2 – Результат работы первой программы: два потока

## Анализ полученных результатов

С помощью системного вызова `open()` создается дескриптор открытого файла `alphabet.txt` с правами доступа на чтение (`O_RDONLY`). Созданному дескриптору соответствует индекс в таблице дескрипторов файлов, открытых процессом, равный 3, так как 0, 1 и 2 заняты `stdin`, `stdout` и `stderr`, и другие файлы процессом не открывались. `fd[3]` указывает на `struct file`, которая связана с `struct inode`, соответствующей файлу `alphabet.txt`.

С помощью двух вызовов функции `fdopen()` стандартной библиотеки создаются две структуры `FILE` — `fs1` и `fs2`, поле `_fileno` которых содержит значение 3.

Функция `setvbuf` устанавливает буферы для каждой из структур `FILE`, параметрами функции являются указатель на начало, размер (20 байт) и тип буферизации (полная буферизация). Указатель на конец буфера устанавливается через указатель на начало буфера и его размер.

В цикле `while` реализован поочередный вызов функции `fscanf()` стандартной библиотеки для `fs1` и `fs2`. При первом вызове `fscanf()` для `fs1` буфер этой структуры будет заполнен полностью — в него запишутся 20 символов от 'А' до 't', так как была установлена полная буферизация. В структуре `struct file` поле `f_pos` установится на символ 'u', следующий за символом 't'. Переменная `c` будет инициализирована символом 'А' и будет выведена на экран. При первом вызове `fscanf()` для `fs2` буфер этой структуры будет заполнен символами от 'u' до 'z', так как `fs1` и `fs2` ссылаются на один и тот же дескриптор, поле `f_pos` которой указывает на символ 'u'. Этот символ запишется в переменную `c` и выведется на экран. В следующих вызовах `fscanf()` переменной `c` будут поочередно присваиваться символы из каждого буфера и выводиться на экран. Когда один из буферов будет прочитан полностью, на экран будут выводиться символы только из одного буфера, что показано на рисунке 1.1. Файл был полностью прочитан при первом вызове `fscanf()` для `fs2`, поэтому повторных

заполнений не будет.

В многопоточной реализации в зависимости от того, какой поток первым вызовет `fscanf()` возможны различные варианты, один из которых представлен на рисунке 1.2.

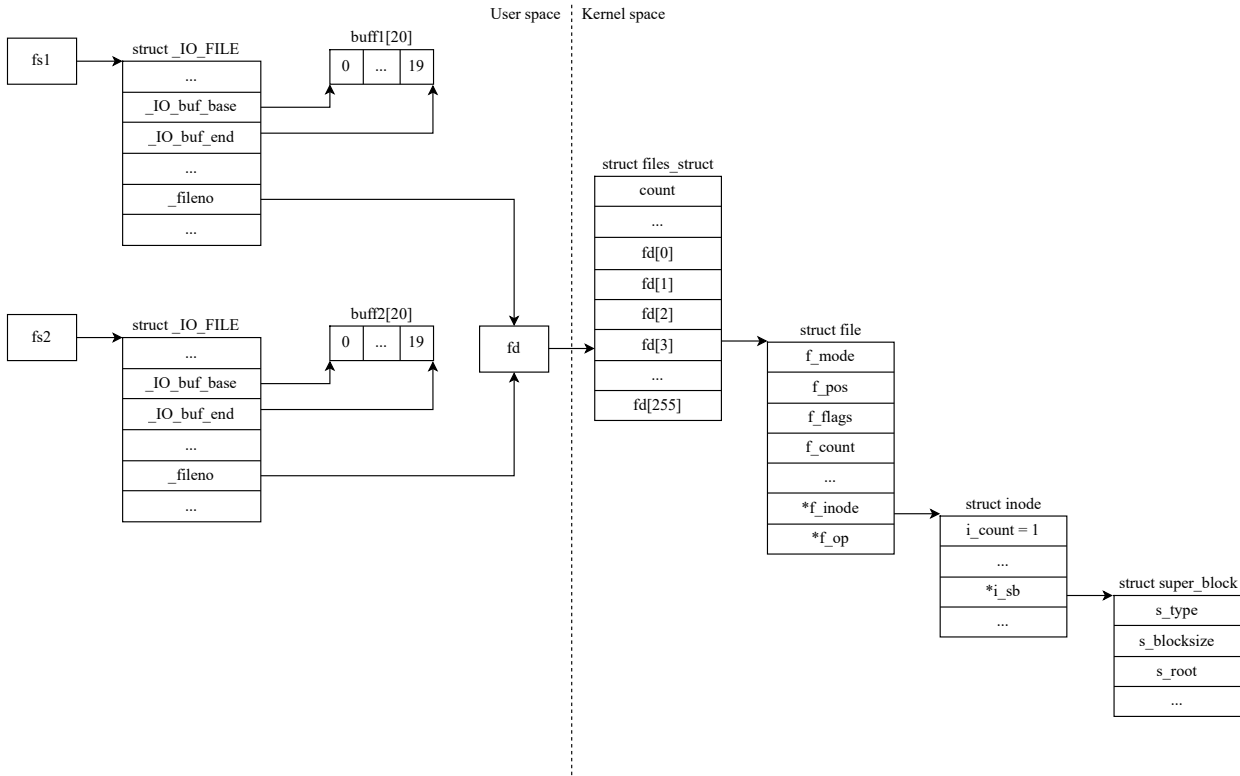


Рисунок 1.3 – Связь между созданными дескрипторами в первой программе

## 2 Вторая программа

### Коды программ

Листинг 2.1 – Один поток

```
1 #include <fcntl.h>
2 #include <unistd.h>
3
4 #define RESET    "\033[0m"
5 #define RED      "\033[1;31m"
6 #define BLUE     "\033[1;34m"
7
8 int main(void)
9 {
10     char c;
11
12     int fd1 = open("alphabet.txt", O_RDONLY);
13     int fd2 = open("alphabet.txt", O_RDONLY);
14
15     int flag1 = 1, flag2 = 2;
16
17     while (flag1 == 1 || flag2 == 1)
18     {
19         flag1 = read(fd1, &c, 1);
20
21         if (flag1 == 1)
22         {
23             printf(RED "%c" RESET, c);
24         }
25
26         flag2 = read(fd2, &c, 1);
27
28         if (flag2 == 1)
29         {
30             printf(BLUE "%c" RESET, c);
31         }
32     }
33
34     printf("\n");
35     return 0;
36 }
```

## Листинг 2.2 – Два потока

```
1 #include <fcntl.h>
2 #include <unistd.h>
3 #include <pthread.h>
4
5 #define RESET    "\033[0m"
6 #define RED      "\033[1;31m"
7 #define BLUE     "\033[1;34m"
8
9 typedef struct args
10 {
11     int fd;
12     char *color;
13 } arg_struct;
14
15 pthread_mutex_t mutex;
16
17 void *print_letter(void *arg)
18 {
19     pthread_mutex_lock(&mutex);
20     arg_struct *now_arg = (arg_struct *) arg;
21     int fd = now_arg->fd;
22
23     int flag = 1;
24     char c;
25
26     while (flag == 1)
27     {
28         flag = read(fd, &c, 1);
29
30         if (flag == 1)
31         {
32             printf("%s%c" RESET, now_arg->color, c);
33         }
34     }
35
36     pthread_mutex_unlock(&mutex);
37
38     return NULL;
39 }
40
```

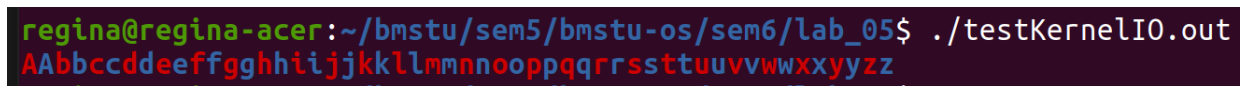


```

41 int main(void)
42 {
43     int fd1 = open("alphabet.txt", O_RDONLY);
44     arg_struct arg1 = {.fd = fd1, .color = RED};
45
46     int fd2 = open("alphabet.txt", O_RDONLY);
47     arg_struct arg2 = {.fd = fd2, .color = BLUE};
48     pthread_t tid;
49     pthread_create(&tid, NULL, print_letter, &arg2);
50
51     print_letter(&arg1);
52
53     pthread_join(tid, NULL);
54
55     printf("\n");
56     return 0;
57 }

```

## Результаты выполнения программ

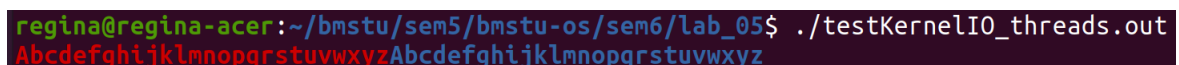


```

regina@regina-acer:~/bmstu/sem5/bmstu-os/sem6/lab_05$ ./testKernelIO.out
AAbbccddeeffgghhiijjkkllmmnnooppqrrssttuuvvwxxyzz

```

Рисунок 2.1 – Результат работы второй программы: один поток



```

regina@regina-acer:~/bmstu/sem5/bmstu-os/sem6/lab_05$ ./testKernelIO_threads.out
AbcdefghijklmnopqrstuvwxyzAbcdefghijklmnopqrstuvwxyz

```

Рисунок 2.2 – Результат работы второй программы: два потока

## Анализ полученных результатов

Два системных вызова `open()` открывают файл `alphabet.txt` для чтения (`O_RDONLY`) и создают два дескриптора открытого файла, которым присваиваются значения 3 и 4. При этом создаются две структуры `struct file`, которые ссылаются на одну и ту же структуру `struct inode`. Поля `f_pos` двух структур `struct file` изменяются независимо друг от друга, поэтому для каждого файлового дескриптора происходит полное чтение

файла, и каждый символ (от 'А' до 'z') будет выведен два раза, что показано на рисунке 2.1.

При многопоточной реализации символы дублируются, но порядок их вывода хаотичен. Добавление в программу мьютекса `pthread_mutex_t mutex` приводит к тому, что алфавит выводится полностью два раза, как показано на рисунке 2.2.

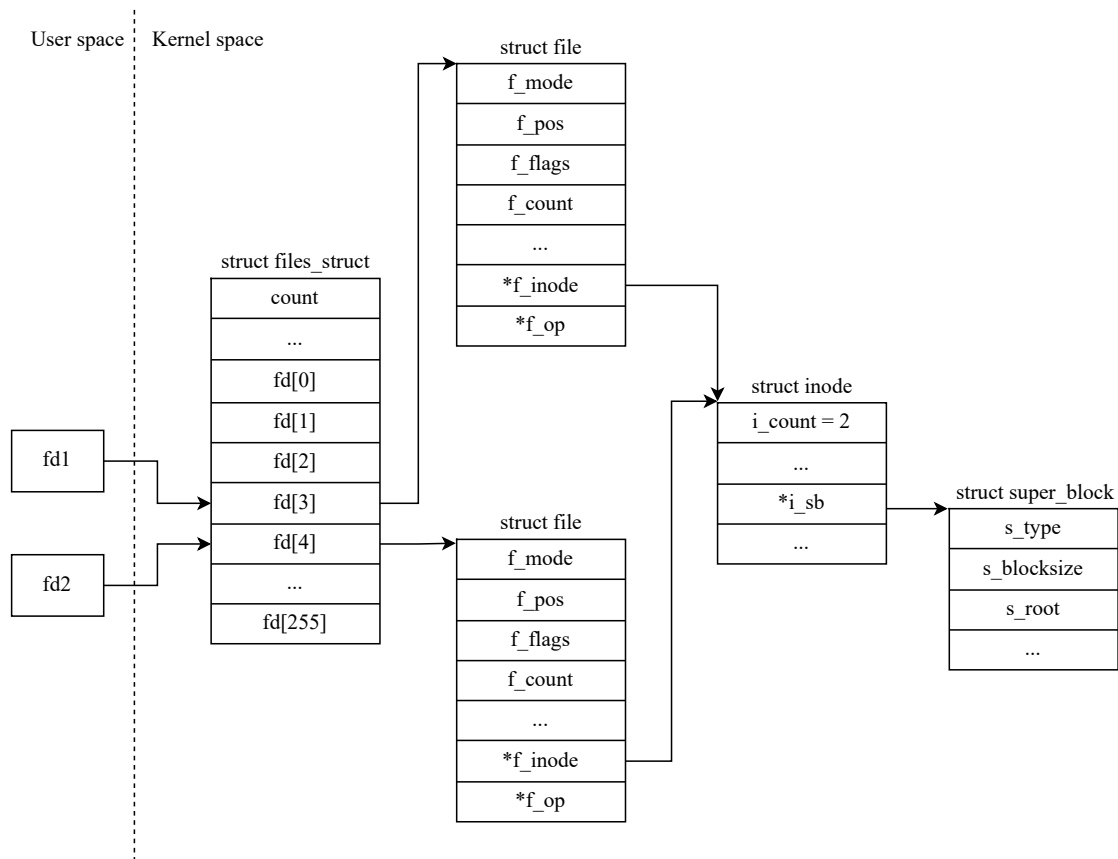


Рисунок 2.3 – Связь между созданными дескрипторами во второй программе

# 3 Третья программа

## Коды программ

Листинг 3.1 – Один поток

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

```

37     print_file_info(fs1);
38
39     print_file_info(fs2);
40     fclose(fs2);
41     print_file_info(fs2);
42
43     return 0;
44 }

```

### Листинг 3.2 – Два потока

```

1 #include <stdio.h>
2 #include <pthread.h>
3
4 void *print_letter(void *arg)
5 {
6     FILE* fd = fopen("results.txt", "a");
7     char *c = (char *) arg;
8
9     while (*c <= 'z')
10    {
11        fprintf(fd, "%c", *c);
12        (*c) += 2;
13    }
14
15    fclose(fd);
16
17    return NULL;
18 }
19
20 int main(void)
21 {
22     pthread_t tid;
23     char c2 = 'b', c1 = 'a';
24     pthread_create(&tid, NULL, print_letter, &c2);
25
26     print_letter(&c1);
27     pthread_join(tid, NULL);
28
29     return 0;
30 }

```

## Результаты выполнения программ

```
inode: 4597094
Размер в байтах: 0
Текущая позиция: 0

inode: 4597094
Размер в байтах: 0
Текущая позиция: 0

inode: 4597094
Размер в байтах: 0
Текущая позиция: 13

inode: 4597094
Размер в байтах: 13
Текущая позиция: -1

inode: 4597094
Размер в байтах: 13
Текущая позиция: 13

inode: 4597094
Размер в байтах: 13
Текущая позиция: -1
```

Рисунок 3.1 – Информация о файле

```
regina@regina-acer:~/bmstu/sem5/bmstu-os/sem6/lab_05$ cat results.txt
bdfhjlnprtvxz
```

Рисунок 3.2 – Результат работы третьей программы: один поток, последний вызов `fclose()` для `fs2`

```
regina@regina-acer:~/bmstu/sem5/bmstu-os/sem6/lab_05$ cat results.txt
acegikmoqsuwy
```

Рисунок 3.3 – Результат работы третьей программы: один поток, последний вызов `fclose()` для `fs1`

```
regina@regina-acer:~/bmstu/sem5/bmstu-os/sem6/lab_05$ cat results.txt
bdfhjlnprtvxzacegikmoqsuwy
```

Рисунок 3.4 – Результат работы третьей программы: два потока

## Анализ полученных результатов

Два вызова функции `fopen()` стандартной библиотеки открывают файл `results.txt` для записи ("`w`") и создают два дескриптора открытого фай-

ла, которым присваиваются значения 3 и 4. При этом создаются две структуры `struct file`, которые ссылаются на одну и ту же структуру `struct inode`. Так как по умолчанию используется полная буферизация, запись в файл происходит либо при полном заполнении буфера, либо при вызове `fflush()` или `fclose()`.

В реализации используется вызов `fclose()`. В цикле `for` с помощью поочередной передачи функции `fprintf` дескрипторов `fs1` и `fs2` в файл записываются буквы латинского алфавита (от 'a' до 'z') до вызовов `fclose()`. При вызове `fclose(fs1)` в файл записываются буквы, стоящие на нечетных позициях в порядке алфавита. При вызове `fclose(fs2)` запись в файл происходит с начала файла, и записанные ранее символы перезапишутся буквами, стоящими на четных позициях в порядке алфавита, так как поле `f_pos` соответствующей структуры `struct file` не менялось.

Так, если первым идет вызов `fclose(fs1)`, а затем — `fclose(fs2)`, в файл запишутся буквы, стоящие на четных позициях в порядке алфавита, как показано на рисунке 3.2. В обратном порядке — буквы, стоящие на нечетных позициях в порядке алфавита, что показано на рисунке 3.3.

В многопоточной реализации аргумент функции `fopen()` `mode` равен "a", поэтому каждая запись будет производиться в конец файла. Буквы алфавита перезаписываться не будут, что представлено на рисунке 3.4.

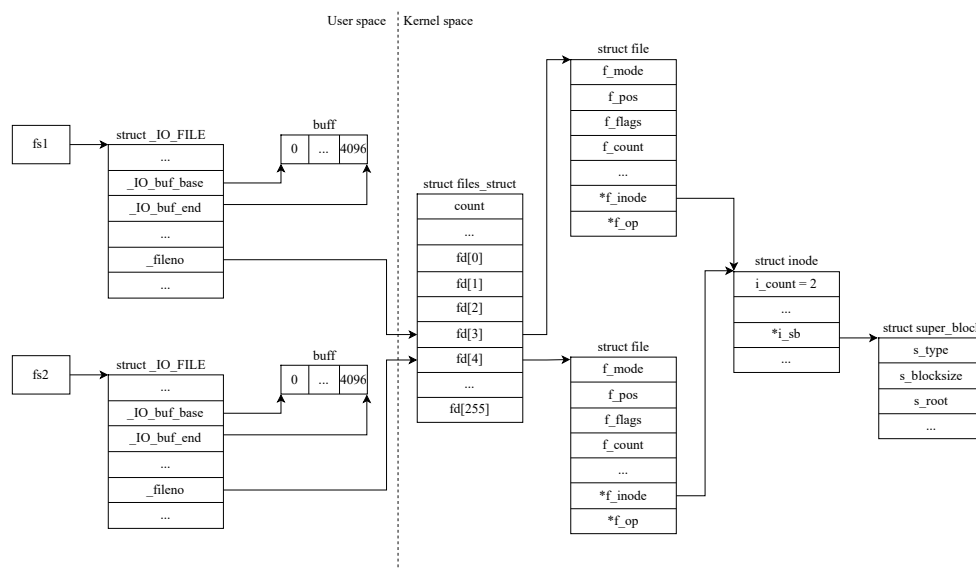


Рисунок 3.5 – Связь между созданными дескрипторами в третьей программе

## 4 Структура FILE

Листинг 4.1 – Описание структуры FILE в файле

/usr/include/x86\_64-linux-gnu/bits/types/FILE.h

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

Листинг 4.2 – Описание структуры \_IO\_FILE в файле

/usr/include/x86\_64-linux-gnu/bits/struct\_FILE.h

```
1 struct _IO_FILE
2 {
3     int _flags;          /* High-order word is _IO_MAGIC; rest is
4                          flags. */
5     /* The following pointers correspond to the C++ streambuf
6        protocol. */
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
16       undo. */
17    char *_IO_save_base; /* Pointer to start of non-current get
18                          area. */
19    char *_IO_backup_base; /* Pointer to first valid character of
20                           backup area */
21    char *_IO_save_end; /* Pointer to end of non-current get area.
22                        */
23    struct _IO_marker *_markers;
24    struct _IO_FILE *_chain;
25    int _fileno;
26    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 #ifdef _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 };

```



## 5 Вывод

При работе с буферизованным и не буферизованным вводом-выводом возникает ряд следующих проблем.

В первой программе появляется проблема буферизации: символы из файла выводятся не в том порядке, в котором они записаны в файле в связи с включением полной буферизации.

Во второй программе чтение файла и вывод символов производятся два раза, так как два файловых дескриптора связаны с одной структурой `struct inode`. Добавление мьютекса приводит к созданию разделяемой области памяти.

В третьей программе также возникает проблема буферизации, при этом символы в файле перезаписываются, то есть, происходит потеря информации. Открытие файла в режиме добавления приводит к тому, что запись в файл производится в конец, и потери данных не происходит.